APRIL: A High-level Framework for Creating Augmented Reality Presentations

Florian Ledermann* and Dieter Schmalstieg†
Vienna University of Technology

ABSTRACT

While Augmented Reality (AR) technology is steadily maturing, 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 paper we present APRIL, the Augmented Presentation and Interaction Languge, an authoring platform for AR presentations which provides concepts and techniques that are independent of specific applications or target hardware platforms, and should be suitable to raise the level of abstraction on which AR content creators can operate.

CR Categories: I.3.6 [Computing Methodologies]: Computer Graphics—Methodology and Techniques; D.2.2 [Software]: Software Engineering—Design Tools and Techniques; D.2.11 [Software]: Software Engineering—Software Architectures

Keywords: augmented reality, authoring, storytelling

1 INTRODUCTION

For Augmented Reality (AR) and Mixed Reality (MR) technologies to become exposed to a larger audience, we do not only need to build systems that can be used intuitively by untrained people, but also have to provide content that makes use of the special features this new media provides. Without elaborating on possible or sensible usage scenarios of AR systems, it can be said that the huge potential of these systems lies in the *presentation* of information: be it an outdoor tourist guide, a novel navigation system, a museum installation or an educational setting (to mention a few of the more common AR scenarios). The focus of these applications is on the presentation of information in a temporally and spatially structured manner, and on allowing the user to interact with this presentation in order to browse, filter and search according to her needs and interests. Obviously, the user interfaces to support these tasks should be as simple and intuitive as possible.

However, we want to make use of the full range of devices, tools and paradigms that AR research has produced and is continuing to produce, to support these presentations. Presentations should be able to address at least a large subset of established Mixed Reality technologies, including classical head-mounted displays, but also immersive projection technology or portable devices. Consequently, we are dealing with complex hardware setups, using non-standard displays, multi-modal input devices and customized interaction tools in networked multi-host setups, incorporating personal computers running different operating systems, but also handheld devices and even cellphones. And while part of this heterogeneity can be accounted to the transient nature of research prototypes, the increased efforts to provide ubiquitous Augmented Reality services and applications indicates that these hybrid systems will soon be more common than any controlled, single-user single-host setups.

*e-mail: ledermann@ims.tuwien.ac.at
†e-mail: schmalstieg@ims.tuwien.ac.at

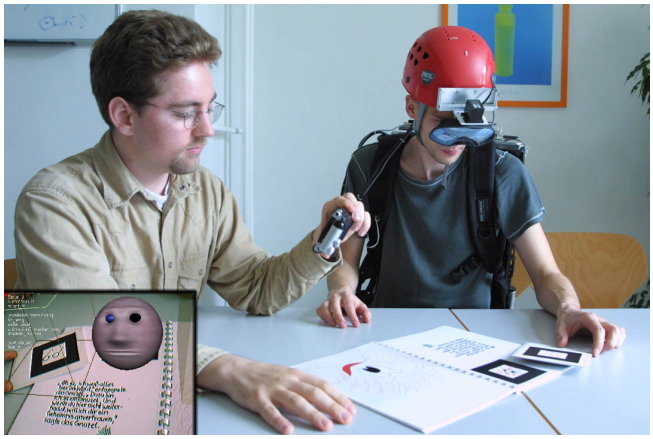

Figure 1: Two users with different AR platforms using the same application, a "Magic Book" created with the APRIL authoring toolkit.

Providing facilities for non-programmers to create presentations for such systems is a challenging task. The complexity of the underlying system should be hidden from the author, while at the same time allowing her to make use of the unique properties of such a system. Therefore, the first goal in the process of designing an authoring framework was to identify the key concepts that are needed by authors to create compelling AR presentations. These concepts had then to be implemented on top of our existing systems to make use of the technology already available.

The key contribution of this work is a set of high-level, AR specific concepts for authoring on hybrid, distributed projective AR systems, and a working implementation of these concepts. We did not concentrate on end-user support for a specific application (as by providing a GUI for a specific authoring scenario), but on raising the level of abstraction for AR presentations. Our approach (1) allows a description of content independent of a specific target hardware platform, and (2) provides templates and best practices that are independent of the actual presentation domain.

2 RELATED WORK AND CURRENT STATE OF THE ART

The first attempts to support authoring on early Virtual Reality systems were to provide ASCII-based file and scripting formats such as Open Inventor [20], VRML [22] or X3D [23]. While scriptable frameworks work well for programmers, who can create application prototypes without the need to compile code, they do not offer the necessary concepts and abstractions for controlling a presentation's temporal structure and interactive behaviour, and provide no built-in support for Augmented or Mixed Reality setups. Platforms like Avango [21] or Studierstube [18] add the necessary classes to such frameworks to support the creation of Augmented Reality appli-

cations, potentially distributed across several computers, but from the perspective of a presentation author this complicates matters further rather than providing the level of abstraction needed. The need for AR-specific authoring languages has been expressed by researchers [12], but little work has been done in this area.

The Alice system [5] 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 towards an educational setting. For creating "real world" applications and presentations, the reusability and modularity of Alice-based presentations is not sufficient. Also, Alice focusses on animation and behaviour control of single objects and does not offer any high-level concepts for presentation control.

An early system for the creation of presentations, the Virtual Reality Slide Show system (VRSS) [6], provides a set of high-level concepts for presentation authoring through a collection of Python macros. VRSS draws inspiration from conventional slide shows, and offers the necessary concepts to the user to create such slide shows for a VR environment. While VRSS seems to be a feasible solution for creating slide-show-like presentations, it was not developed further to allow a more complex structure of the presentation or sophisticated user interaction.

The increasing awareness of researchers of the problem of structuring narrative content led to more research activities that looked at literature and drama theory and conventional storytelling techniques to derive concepts suited for the creation of interactive and spatially structured presentations. The Geist project [9] incorporates a detailed analysis of classical and interactive storytelling and provides several runtime modules to support presentations based on these concepts. Using Prolog, authors can create semiotic functions that drive the story, and the virtual characters that appear are connected to an expert system to provide compelling conversational behaviour and emotional status.

Although the Geist project uses a mobile AR system as its output media, the focus lies clearly on the underlying storytelling framework. Generally, Geist and similar approaches can only unfold their potential in complex presentations, incorporating multiple real and virtual actors, and hence require a correspondingly high effort in content creation. At the same time, the possibilities of running the Geist system on different or hybrid Augmented Reality setups remain unclear.

More pragmatic approaches have focussed on the *tools* used by authors to create the content of their presentations. Powerspace [8] 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 presentation, 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 already existing content into the Augmented Reality domain. Other groups have presented prototypes for AR authoring built on top of existing modeling packages [2].

The Designers Augmented Reality Toolkit (DART) [11] is also built on top of existing software: DART extends Macromedia Director, an authoring tool for creating classical screen-based multimedia presentations and desktop VR presentations. DART allows design students who are already familiar with Director to quickly create compelling AR presentations, often using sketches and video-based content rather than 3D models as a starting point. Typical DART presentations are run in single-user video-see-through setups, and to our knowledge there is no or very limited support of distributed setups or non-standard display hardware. Supporting these systems often requires concepts that are difficult or impossible to implement as single extension classes, but require a modification of the underlying model or paradigm.

The Mobile Augmented Reality System (MARS) [7], developed at Columbia University, has also been extended by a visual editor for creating *situated documentaries*. These hypermedia narratives, located in outdoor environments, can be browsed by the user by roaming the environment, wearing the MARS system. In contrast to the projects mentioned so far, the MARS team has developed their own visual editor for presentations from scratch, allowing them to implement an authoring paradigm tailored to the needs of their system. While the visual editor of MARS looks very promising, the underlying hypermedia system is not sufficiently flexible to suit our content creation needs and the support of non-standard AR setups.

The need for an additional abstraction layer to support hybrid setups and AR-specific features has been recognized by some researchers. AMIRE [24] provides a component model for authoring and playback of AR applications. On top of the AMIRE system, an authoring tool for AR assembly instructions has been created, which is limited to the domain of step-by-step instructions for assembly tasks. Sauer and Engels [17] propose to model multimedia presentations using UML [13]. They use statecharts and sequence diagrams to create a model of a (conventional) presentation's behaviour, which can then be used as a basis for the implementation of the presentation. The spatial arrangement of content or any special aspects of VR or AR presentations are not considered in their work.

The alVRed project [1] picks up these ideas and uses UML statecharts to model the temporal structure of VR presentations. In their model, a state represents a *scene* of the presentation, while the transitions between states represent changes in the presentation triggered by user interaction. However, aspects of AR authoring are not considered in the alVRed project.

Recently, a discussion about *design patterns* for Augmented Reality has been started [14]. Design patterns try to grasp concepts that can not be easily modelled as entities or classes, but are rather a careful arrangement of such entities and their concerted behaviour. Up to now, this discussion has been a purely theoretical one, although some existing AR systems already make use of some of the patterns that have been discussed.

3 Our Approach

The first question we asked ourselves when designing our authoring solution was not *how* to author such presentations, but *what* we want to allow users to author. This implies that we had to identify the key concepts and processes that presentation authors would like to work with. The discussion about design patterns mentioned above is a first step towards the identification of such key concepts, but no implementation of an authoring system using these concepts exists. We believe that high-level patterns should be made available to presentation authors, relieving them of the burden of coping with the (often non-trivial) implementation details, while at the same time allowing them to use the very features that make Augmented Reality a unique media form.

3.1 Requirements

From our own experience with students and external collaborators, we could identify a set of requirements for our authoring solution. 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. Presentations and their components should be reusable in different setups, and a presentation developed for one system should run on another setup, with little or no modification.

This also opens up the possibility of cross-platform development. As most AR systems are prototypes, they are usually also a

scarce resource. It should therefore be possible to develop presentations in a (desktop-based) simulation environment, without having to occupy the target system for the whole time of the 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 only for fine-tuning.

Concerning content-creation, our goal was to support industry standards that are used by professionals, instead of providing our own tools and file formats. Generally, we tried to follow the policy to integrate available tools and practices wherever possible, instead of re-inventing existing solutions. By doing so, we could focus on the AR-specific aspects of the framework.

Finally, we did not want to start a new platform from scratch, but build 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 presentations created with our framework.

3.2 Practical Considerations

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 ASCII- 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. With the choice for XML as the base technology for our content format, the next step was to design the vocabulary of our intended AR authoring language.

4 The APRIL language

APRIL, the Augmented Reality Presentation and Interaction Language, covers all aspects of AR authoring defined in the requirements analysis. APRIL provides XML elements to describe the hardware setup, including displays and tracking devices, as well as the content of the presentation and its temporal organisation and interactive capabilities.

Enumerating all elements and features that APRIL provides is beyond the scope of this paper. Interested readers are referred to [10], 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.

4.1 Overview

The five main aspects that contribute to a presentation – hardware description, content description, temporal structure, dynamic behaviour and interaction – are encapsulated in four top-level elements – `setup`, `cast`, `story`, `behaviors`[1] and `interactions` – that can be easily exchanged, allowing for the customization of the presentation for various purposes.

The *story* is an explicit representation of the temporal structure of the presentation, composed of individual *scenes*. In each scene, a predefined sequence of *behaviours* 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*, possibly provided by interaction components.

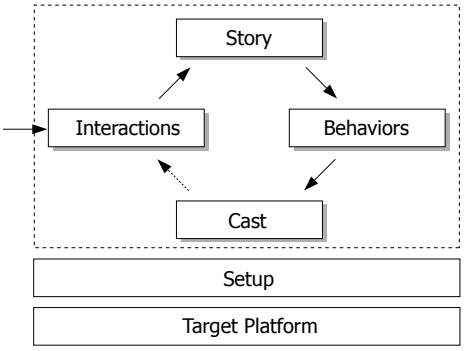

Figure 2: The main components of APRIL.

The decision to have a central storyboard controlling the presentation was made well aware of other, agent-centric approaches, where the overall behaviour of an application is the result of the individual behaviours of more autonomous agents. In contrast to other applications, for our AR presentations we want the results to be predictable and easily controllable by a human author, therefore having a single, central instance of a storyboard seemed best suited to model such an application.

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

4.2 Of Stages, Scenes and Actors

The two fundamental dimensions along which a presentation is organized have already been mentioned: the temporal organization, determining the visibility and behaviour of the objects of the presentation over time, and the 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 a presentation'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 behaviour of other actors. APRIL allows the nesting of actors, so one 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*, which allow reading and writing of typed values. Details of the APRIL component model will be explained in section 4.4.

By *behaviour* of an actor we mean the change of the fields of the actor over time. Parts of the behaviour can be defined by the author beforehand, by arranging field changes on a timeline, and parts of it will be dynamic, determined by user interaction at runtime.

We decided to use UML statecharts to model presentations, a tool that has been used successfully by other projects like alVRed. UML statecharts can be hierarchical and concurrent, meaning that a state can contain substates, and there might be several states active at the same time. 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, another one (the `do` timeline) is executed as long as the scene remains active (this means that behaviours 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

[1] Note that in this paper, we use the British spelling (*behaviour*), while in the APRIL Schema defining the names of the XML elements, the American English spelling (*behavior*) is used.

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.

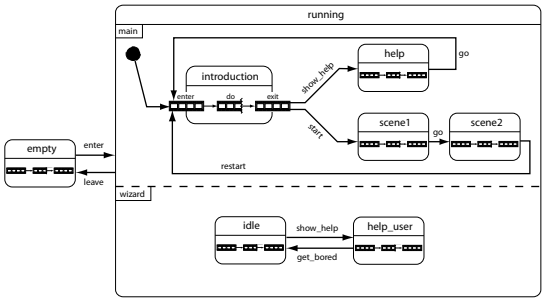

Figure 3: The storyboard of a simple APRIL presentation, modelled as a UML state diagram. For the "introduction" scene, the three timelines `enter`, `do` and `exit` are emphasized.

While for the temporal organization of presentations we could borrow an already existing concept previously used for virtual reality content, the spatial organization of content in an AR application differs from the approaches known so far. 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 of an AR system, there may be several "realities" that are simultaneously viewed and used: besides the real world and the corresponding registered computer generated overlays, there are several ways to display user interface elements, like head up displays (HUDs) or interaction panels, and there are possibilities to display worlds within worlds like the world in miniature approach [19] for navigation or the possibility of rendering a complete scene to a texture to be used as a 2D information display.

In APRIL, the top-level spatial containers for the content of a presentation are called *stages*. For each stage, authors can not only define the spatial relationship 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). Per default, actors appear on the main stage, the area that is aligned with and equally scaled as the real world. Interaction objects can be placed on interaction stages, where they will, depending on the setup the presentation is run, be rendered as a HUD or interaction panel.

Stages are one example of the concepts that require a coupling between the individual hardware setup and the presentation – the available stages are different for each setup, and therefore have to be defined in the hardware description section (which will be described in section 4.3). If these stages would simply be referenced in the presentation (e.g. by name), the portability of the presentation would be reduced, because the presentation could then only be run on similar setups, that provide the same number of stages with equal names. To overcome this problem, the concept of *roles* has been introduced: Each stage is assigned one (or multiple) out of a predefined set of roles, that describe the function of this stage. These roles are tokens, like, 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 presentations is the real world. Usually, for more advanced presentations, 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 modelling or by scanning the objects with a 3D-scanner.

4.3 Hardware abstraction

An important consequence of the requirements that have been analyzed is to separate the content of the presentation from all aspects that depend on the actual system that the presentation will run on. At the same time, a powerful yet flexible coupling mechanism between the hardware dependent layer and the presentation had to be found, to allow the presentation to make use of the individual features of a hardware setup, like tracking devices or displays.

APRIL allows to put all hardware description aspects into a separate file, and supports running a presentation on different setups with different 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 presentation.

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, amongst others. For configuring tracking devices, we use the already existing OpenTracker XML-based configuration language [15], 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, and instead of 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 presentation.

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 technique called ray-picking, using a "virtual laserpointer" 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 presentation'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 presentation, `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 2-dimensional 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-address="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-address="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.

4.4 Component model

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

As these components constitute the content of our presentations, 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. An APRIL-component is basically a template, using *any* existing, ASCII-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 presentation. The chosen content format has to 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. The alternative would have been to create a platform-neutral content definition language, that could only use a set of features supported by all platforms, which would preclude the creation of sophisticated content that uses state-of-the-art real time rendering features.

An APRIL component definition file contains 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 scross 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 presentation will choose the implementation that is best suited. 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 to implement the component's behaviour 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 presentation 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).

4.5 Presentation 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 presentation's actors. The two fundamental 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 timespan.

For more dynamic behaviour of the presentation, 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 behaviours are specified is active, so no `disconnect` or `uncontrol` elements are provided.

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` toplevel element, they can be easily exchanged. This process, called *interaction mapping*, can be used to derive different versions of the same presentation, suiting different needs. For example, a non-interactive version of a presentation, using only `timeout` transitions to linearly step through the presentation, can be provided for demo purposes, while a fully interactive version of the same presentation is run in user sessions.

5 Implementation

While it would theoretically be possible to implement a runtime platform that reads APRIL files directly and supports the APRIL concepts by a corresponding architecture, our goal was to make use of our existing systems and transform the APRIL presentation files into the necessary configuration files for the two frameworks we currently use: the *Studierstube* Augmented Reality system, and a lightweight AR system that runs on PDAs, called *StbLight*. From this approach, we get support for the high-level concepts of APRIL for both of our platforms, with very little need to actually implement these features natively in C++.

In some areas the two runtime systems, which are both based on Open Inventor, had to be extended to support APRIL presentations. An implementation of a generic state-engine that controls the presentation 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 [4].

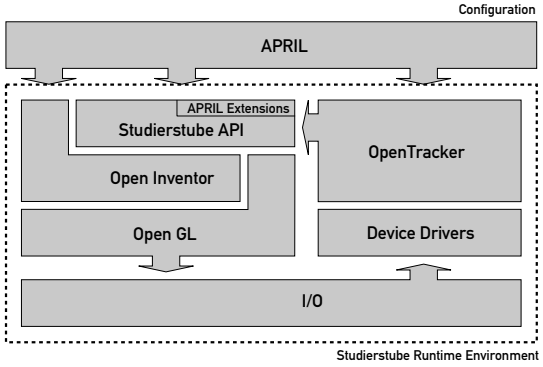

Figure 6: The overall architecture of the Studierstube system, using APRIL as a high-level authoring language.

XSLT is a template-based language for transforming XML documents into other, ASCII-based document types. One or multiple input files can be processed in a non-linear fashion, generating arbitrary numbers of output files. XSLT is used most often 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 the XSLT transformation, using the information that is stored in the APRIL file in a well-designed and consistent manner.

A schematic overview of this transformation process is shown in figure 7. The story specification together with the corresponding interaction and behaviour definitions constitutes the core of the APRIL presentation. Components, defined in separate files for reusability, are included in the presentation, 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 7 also shows the places of human intervention in the APRIL authoring process. APRIL transforms the view on 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 presentation. This workflow is also much more scalable from single individuals who create a whole presentation to entire teams of collaborating professionals, using the storyboard as a central artifact for communication to contribute at different levels to the final result.

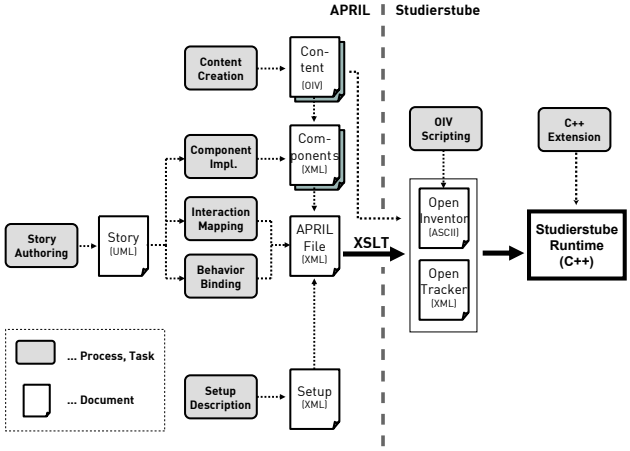

Figure 7: Schematic view of the APRIL transformation process.

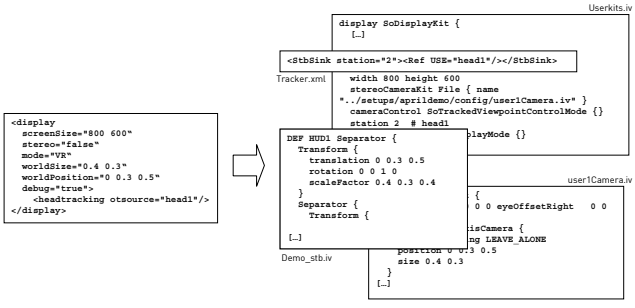

Figure 8: This figure illustrates which files in the output are affected by a single element in the hardware description file. Using conventional tools, authors would have to keep the information of these files in sync manually.

6 RESULTS

An early version of APRIL was used by the students of our Augmented Reality lab lecture to create different presentations for a broad spectrum of AR setups. There were 25 undergraduate students participating, organized in 9 groups, each assigned a different task. The setups for presentations included a virtual showcase system [3], a mobile AR backpack system for indoor and outdoor use [16] and desktop-based AR setups. The setup description files were provided by the lecturers, and for the virtual showcase and mobile AR setups an additional desktop emulation setup file was provided, to allow students to develop their presentations on a desktop computer. For the mobile AR setups, this simulation environment contained a component to simulate tracking data, so that walk through scenarios could be tested without the need for physical roaming.

Following the APRIL workflow, students first had to come up with a storyboard for their presentation, accompanied by research concerning the subject of their presentation and the gathering of raw and inspirational content. This encouraged the participants to think about the intended presentation early and come up with a proposal of its temporal structure in the form of a storyboard. For modelling the storyboard, a third party UML tool was used, which could save the diagrams in an XML format that would later be converted automatically to the APRIL syntax. Mapping all interactions to `buttonaction` interactions, the storyboard immediately gave students an executable prototype of their presentation, that could be used for testing the consistency of the story.

The raw content found in the research phase was then added to the presentation to give a first impression of the content of the individual scenes. MacIntyre et. al. [11] underline the important role of informal, "sketchy" content for exploring a design space, which allowed our students to experiment with variations of the storyboard and different interactions to trigger the transitions. From then on, students would also specialize to be able to make use of their individual skills – some students focusing on implementing new components, while others specialized in content modelling or the scripting of animations and interactions.

The results of the students work was impressive. In previous years, students would typically implement small, usually stateless applications with little content, using Open Inventor scripting and C++ to create custom extension nodes. With APRIL, they could implement much more complex application logic, while at the same time focusing more on content creation to fill their application with life. Results ranged from AR-enhanced Lego-robots and interactive furniture assembly instructors over multimedia presentations for the virtual showcase to indoor and outdoor tourist guide applications of near-professional quality. Some images of the students results are presented in Figures 9-11.

From the experience of these early application results, some of the concepts of APRIL were refined. Originally, we planned more "built-in" interaction techniques (similar to the `buttonaction` element), but in practice we discovered that a lot of the requirements for interactive presentations could be fulfilled with the very simple basic interactions APRIL provides. More sophisticated interaction techniques can always be added by implementing or reusing a custom component, and the attempt to categorize these interactions in advance and attempting to provide a generic set of hardcoded interactions needlessly limits creativity.

Another improvement that emerged from our initial experiences was the concept of stages to structure the spatial arrangement of actors. Originally, the only alternatives provided were the insertion of content in the world or in the HUD of all users. The demands of the users soon indicated that a more flexible concept for spatial structuring of the presentation was needed.

During the development and refinement of APRIL, we also verified the feature set that we developed against existing projects that were scripted manually, to see if similar things could be implemented with APRIL. It was interesting to observe that, especially in museum applications and other scenarios involving public exposure of AR technology, most discussions and prototyping sessions focussed on details that are supported by and much easier to realize and change with APRIL ("Instead of making the user click on this object, we want a timeout to trigger the animation"). In these scenarios, APRIL can support the rapid prototyping of different ideas, shortening the delay between conception and implementation, hence allowing more user study to take place.

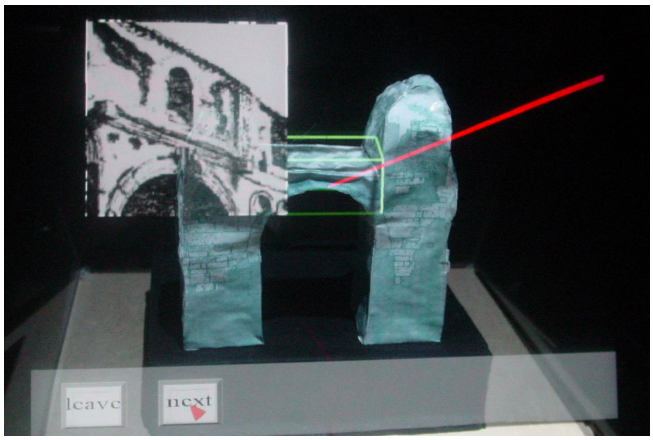

Figure 9: An archaeological ruin inside the Virtual Showcase. In this setup, a raypicker is used to select parts of the real object for retrieving further information. A projector is used to cast shadows onto the object.

Figure 10: An indoor tour guide application, running on the desktop developer setup. A world-in-miniature view on the model of the building is shown in the background, and location-dependent HUD overlay graphics is presented to the user as she roams the building.

7 CONCLUSIONS AND FUTURE WORK

With our work on APRIL we hope to have provided a starting point for identifying key concepts, patterns and techniques for Augmented Reality authoring. By focusing on the requirements of authors and developers and the properties of the target systems rather than starting with the creation of a GUI for specific authoring tasks, we could consequently introduce and refine the features needed for sophisticated applications, without being constrained by an existing framework or practice.

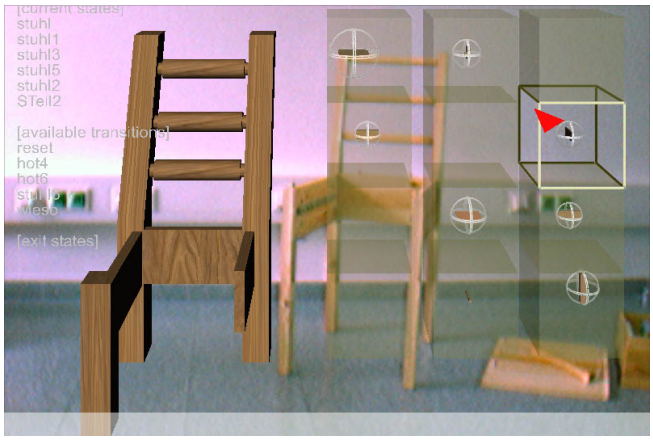

Figure 11: Interactive furniture construction with APRIL. The construction process is modelled with the state engine, and possible parts for the next step are shown to the user.

The XSLT-based reference implementation, based on our *Studierstube* runtime system, provides templates for the implementation of the APRIL features on top of the widely used Open Inventor scripting API. These templates do not only provide a working implementation of APRIL, but implicitly document *best practices* for implementing common AR patterns on top of that framework.

Having developed the set of features to cover our authoring needs, we will now focus on providing tools for the interactive visual creation of presentations. Such a visual authoring tool would provide the user with an intuitive interface to the APRIL concepts, without limiting the possibilities for collaborative work and distribution of authoring tasks that are the key features of the workflow developed.

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

8 Acknowledgements

The authors would like to thank Daniel Wagner for his work on our AR framework for handheld devices, Gerhard Reitmayr for valuable suggestions throughout the development of APRIL and Joe Newman for proofreading this paper. 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 the bm:bwk contract #TUWP16/2001.

References

[1] S. Beckhaus et al. alVRed – Tools for storytelling in virtual environments. Technical report, Fraunhofer IMK, Sankt Augustin, 2002.

[2] Rodney Berry, Naoto Hikawa, Mao Makino, Masami Suzuki, and Takashi Furuya. Authoring augmented reality: A code-free approach. In *Proceedings of ACM SIGGRAPH 2004*. ACM, August 8–12 2004.

[3] O. Bimber, B. Fröhlich, D. Schmalstieg, and L. M. Encarnação. The virtual showcase. *IEEE Computer Graphics and Applications*, 21(6):48–55, November 2001.

[4] James Clark. XSL transformations (XSLT) version 1.0 – W3C recommendation. http://www.w3.org/TR/xslt, 1999.

[5] Matthew Conway, Randy Pausch, Rich Gossweiler, and Tommy Burnette. Alice: A rapid prototyping system for building virtual environments. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 2, pages 295–296, April 1994.

[6] A. Fuhrmann, J. Prikryl, R. Tobler, and W. Purgathofer. Interactive content for presentations in virtual reality. In *Proceedings of the ACM Symposium on Virtual Reality Software & Technology*, 2001.

[7] Sinem Güven and Steven Feiner. Authoring 3D hypermedia for wearable augmented and virtual reality. In *Proceedings of the 7th International Symposium on Wearable Computers*, pages 118–126, White Plains, NY, October 21-23 2003. IEEE.

[8] Matthias Haringer and Holger T. Regenbrecht. A pragmatic approach to Augmented Reality authoring. In *Proceedings of ISMAR 2002*, Darmstadt, Germany, 2002. IEEE.

[9] Ursula Kretschmer, Volker Coors, Ulrike Spierling, Dieter Grasbon, Kerstin Schneider, Isabel Rojas, and Rainer Malaka. Meeting the spirit of history. In *Proceedings of VAST 2001*, Glyfada, Athens, Greece, November 28–30 2001. Eurographics.

[10] Florian Ledermann. An authoring framework for augmented reality presentations. Diploma thesis, Vienna University of Technology, May 2004.

[11] Blair MacIntyre and Maribeth Gandy. Prototyping applications with DART, the designer's augmented reality toolkit. In *Proceedings of STARS 2003*, pages 19–22, Tokyo, Japan, October 7 2003.

[12] U. Neumann and A. Majoros. Cognitive, performance, and systems issues for augmented reality applications in manufacturing and maintenance. In *Proceedings of the Virtual Reality Annual International Symposium*, page 4. IEEE Computer Society, 1998.

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

[14] Thomas Reicher, Asa MacWilliams, and Bernd Bruegge. Towards a system of patterns for Augmented Reality systems. In *Proceedings of the International Workshop on Software Technology for Augmented Reality Systems (STARS 2003)*, October 2003.

[15] Gerhard Reitmayr and Dieter Schmalstieg. OpenTracker – an open software architecture for reconfigurable tracking based on XML. In *Proceedings of IEEE Virtual Reality 2001*, pages 285–286, Yokohama, Japan, March 13–17 2001.

[16] Gerhard Reitmayr and Dieter Schmalstieg. Collaborative augmented reality for outdoor navigation and information browsing. In *Proceedings of the Symposium on Location Based Services and TeleCartography*, Vienna, Austria, January 2004.

[17] S. Sauer and G. Engels. Extending UML for modeling of multimedia applications. In *Proceedings of the IEEE Symposium on Visual Languages (VL'99)*, pages 80–87, 1999.

[18] Dieter Schmalstieg, Anton Fuhrmann, Gerd Hesina, Zsolt Szalavári, L. Miguel Encarnação, Michael Gervautz, and Werner Purgathofer. The Studierstube augmented reality project. *PRESENCE - Teleoperators and Virtual Environments*, 11(1), 2002.

[19] Richard Stoakley, Matthew J. Conway, and Randy Pausch. Virtual reality on a WIM: interactive worlds in miniature. In *Conference proceedings on Human factors in computing systems*, pages 265–272, Denver, Colorado, USA, 1995. ACM Press, Addison-Wesley.

[20] P. Strauss and R. Carey. An object oriented 3D graphics toolkit. In *Proceedings of ACM SIGGRAPH '92*. ACM, 1992.

[21] H. Tramberend. Avocado: A distributed virtual reality framework. In *Proceedings of IEEE Virtual Reality 1999*. IEEE, IEEE Press, 1999.

[22] VRML Consortium. VRML97 specification. Specification 14772-1:1997, ISO/IEC, 1997.

[23] Web3D Consortium. X3D specification website. http://www.web3d.org/x3d/specifications/.

[24] Jürgen Zauner, Michael Haller, and Alexander Brandl. Authoring of a mixed reality assembly instructor for hierarchical structures. In *Proceedings of ISMAR 2003*, pages 237–246, Tokyo, Japan, October 7–10 2003. IEEE.