Design Patterns for Augmented Reality Systems

Asa MacWilliams, Thomas Reicher, Gudrun Klinker, Bernd Bruegge
Lehrstuhl für Angewandte Softwaretechnik
Institut für Informatik, Technische Universität München
(macwilli|reicher|klinker|bruegge)@in.tum.de

ABSTRACT
For documentation and development of augmented reality (AR) systems, we propose the use of design patterns. Patterns are structured descriptions of successfully applied problem-solving knowledge. They describe a specific design problem, the particular design context, and a generic solution scheme. The solution scheme specifies the involved components, their responsibilities, relationships and the way they cooperate.

We see design patterns as the right abstraction for the documentation of AR systems. A common language based on design patterns allows the use of common, well-known terms among developers of AR systems. Examples are terms such as *scene graph-based rendering* and *networked trackers*.

With a catalogue of patterns, developers of AR systems can more easily identify existing solutions to design problems. For this, we must identify relations and dependencies among individual patterns. The goal is the systematic ordering of the individual patterns into a system of patterns. Such a pattern system shows the interdependencies among the patterns. For example, a *VRML web plugin* enables easy embedding of AR content into web documents, but in turn requires a *web server*, which is unsuited for autonomous systems.

This paper is the continuation of a discussion we started at the STARS 2003 workshop [13]. We have already identified some 30 design patterns for AR. Most of the patterns will also be relevant for mixed reality systems. In this paper, we present our existing catalog of design patterns and discuss the findings. In particular, we consider it as important to find the right name for each pattern, to bring forward discussion in the developer community.

INTRODUCTION
The discussion and comparison of different software architectures for Augmented Reality (AR) is often difficult because of the different ways the developers document them. They use different notations, different abstraction levels, and have different intentions.

Often a specific application belongs to a class of applications, the *domain*. For each domain, there are specific functional and non-functional requirements which are mapped to common functions. In each architecture, these functions are implemented by subsystems. And for the implementation of a subsystem, a developer uses a particular *approach*. In a given domain such as augmented reality, similar or identical approaches are used by various developers. Often this stems from the common use of software components or libraries that implement the same functionality, e.g. *OpenInventor*. The result is a vocabulary of common terms that are understood by most augmented reality developers. This enables discussion and comparison of software architectures. To classify the approaches, we extracted an abstract generic architecture for augmented reality systems [14] from the descriptions of existing systems. A software architecture for AR can be described by the set of approaches used in the system.

While a set of *approaches* allows us to discuss existing augmented reality systems, it is only of little use for the design of new systems. For this, we must measure each approach within a certain context. The catalogue of known approaches then can mature to a system of known *patterns*. Each pattern must state the context where it is used, the problem it solves and the solution. In software architecture, patterns are structured descriptions of successfully applied problem-solving knowledge. Each pattern is described by name, goal, motivation, description, usability, consequences, and known use, as in [7, 16].

In this paper, we present the current state of our catalog of AR design patterns. In order to fulfill its goal of improving communication and collaboration between AR researchers, this system of patterns requires extensive discussion within the AR research community. We have set up a web site for this purpose[1] and invite members of the AR research community to join us in improving and extending the patterns.

AN AUGMENTED REALITY REFERENCE MODEL
A study of the various augmented reality systems [12] revealed that in spite of being quite different in detail, most augmented reality systems share a common basic architectural structure. In addition, many basic components and sub-

[1] `http://wwwbruegge.in.tum.de/projects/lehrstuhl/twiki/bin/view/ARPatterns`

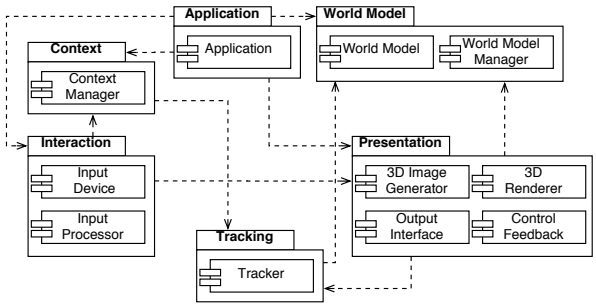

Figure 1: Generic augmented reality reference architecture

systems can be found in many different systems, e.g. various trackers or a scene graph. This is not surprising, as all augmented reality systems are interactive systems and the core functionality of augmented reality is the same for all systems: tracking the user's position, mixing real and virtual objects, and processing and reacting to context changes and user interactions.

This commonality of the core functionality among the different AR systems allows us to specify a descriptive reference model for augmented reality systems. We decompose an augmented reality system into six core subsystems. Each subsystem provides a particular functionality for the whole system.

Application subsystem. The abstract Application subsystem is a placeholder for all application specific code.

Interaction subsystem. The Interaction subsystem gathers and processes any input that the user makes deliberately. We distinguish it from other input such as by changing position.

Presentation subsystem. The Presentation system displays output for the user. Besides 3D augmentation, this also includes other media such as 2D text or speech.

Tracking subsystem. Tracking the user's pose is another key functionality of augmented reality systems. The tracking subsystem is responsible for tracking the user's pose and providing it to other subsystems such as the presentation subsystem.

Context subsystem. The Context subsystem collects different types of context data and makes it available to other subsystems. Examples include user preferences and the current user task. Although the user's pose can be considered as part of the user's context, tracking is so essential to augmented reality that we separate the Tracking and Context subsystems.

World model subsystem. In augmented reality, the user moves in the real world and obtains information linked to real-world objects or user positions. Information about the world is stored in a world model. Similar to tracking, the world related information is of a particular interest for augmented reality systems design.

Each subsystem consists of several components. Figure 1 gives an overview of the identified subsystems and the main components. Subsystems are shown as UML packages which contain components. Each component in turn may be realized by other components. There are dependency relationships between the subsystems, illustrated with dashed lines. A dependency shows that a subsystem relies on interfaces of of another subsystem.

The abstract reference model describes the general components and structure of augmented reality systems. However, depending on the functional requirements of a particular system, some of the components may be left out. For example, a video mixer component is not required for optical see-through augmented reality.

The abstract structure that we use for augmented reality systems is similar to the Model/View/Controller (MVC) pattern [8]. The MVC pattern separates interactive systems into subsystems for the Model, Views of the Model, and a Controller for the data flow. The Model encapsulates application data, provides access and manipulation methods, and processes input from the controller; the View represents user information, updates on changes in the model, and creates the Controller; the Controller is related to a View, provides user input methods, forwards events to the Model, and initiates changes in the View. To conform to terms used in HCI research we call the View *Presentation*, and the Controller *Interaction*. We add specific extensions for augmented reality, in particular abstractions for *Tracking*, a *World Model*, and *Context*. The similarity of our structure and the MVC pattern is not surprising, as AR systems are interactive systems by definition [1].

PATTERNS FOR AUGMENTED REALITY SYSTEMS

On a subsystem level, the developers of existing augmented reality systems use different *approaches* to implement the subsystems, e.g. tracking or presentation. An analysis of existing systems reveals that several approaches recur in various existing systems—sometimes explicitly, such as when two systems use a common library, and sometimes implicitly, when different developers apply the same basic techniques. The selection depends on the systems' non-functional requirements and the design goals.

Successfully applied approaches can be extracted from the descriptions of existing systems and then be described as abstract reusable *patterns* for augmented reality systems design. This is heavily based on the idea of design patterns in software architectures. Patterns are structured descriptions of successfully applied problem-solving knowledge: *A software architectural pattern describes a specific design problem, which appears in a particular design context, and presents a generic solution scheme. The solution scheme specifies the involved components, their responsibilities, relationships and the way they cooperate* [2, pp 8].

In a first step, we have collected patterns from existing systems. We are currently in the process of ordering this list into a *system of patterns*, as discussed below.

A Scheme for the Description of Patterns

We describe each pattern by name, goal, motivation, a description, usability, consequences, collaborations, and known use. This follows the scheme of describing architectural or design patterns, e.g. as used by Gamma et al. [7].

Name The name of a pattern should be descriptive and in the best case in use in several systems.

Goal The goal is a short description for the target use of the pattern.

Motivation This section describes why the pattern was developed.

Description We describe each pattern informally by its tasks and structure. We do not yet have formalisms such as UML static and dynamic diagrams for each pattern.

Usability Describes when and how each pattern can or cannot be used.

Consequences The advantages and disadvantages of the pattern.

Collaborations Other patterns that can or must be used in combination to this pattern.

Known use Projects and systems that use the pattern.

As examples we describe the patterns *Scene Graph Node* and *Scripting* for the Application subsystem, and the *Scene Graph* pattern for the Presentation subsystem.

Scene Graph Node pattern (application subsystem)

Goal: Embed application in world model.

Motivation: In augmented reality, user interaction is connected with the physical environment. Consequently applications are often linked to places in the real world. With this pattern, the application is seamlessly embedded in the environment.

Description: A scene graph models the world around a user as a tree of nodes. Each node can be of any type, usually graphical objects such as spheres. But there are also non-graphical objects that include control code.

Usability: In combination with the Scene Graph pattern for rendering.

Consequences: The scene graph-based approach for an application hands the control flow to the underlying scene graph platform, e.g. Open Inventor. Some scene graph platforms allow shared scene graphs, enabling application sharing among several users.

Collaboration: Uses scene descriptions in scene graph format, may be implemented with in a scripting language, may be implemented as event call-back.

Known use: Studierstube [15], Tinmith [11]

Scripting pattern (application subsystem)

Goal: Quickly develop new applications.

Motivation: The real-time constraints of a user application are often not very strong, so that it is possible to quickly develop new applications in a scripting language supported by a powerful environment.

Description: For the development of an application, there is a scripting wrapper around all components that have performance constraints. These components are written in compiled languages such as C++ and offer scripting interfaces.

Usability: The development of scripted applications allows rapid prototyping but demands powerful components that implement important functionality. The disadvantage is that the scripting approach is not suited for very complex applications.

Consequences: A script interpreter is needed, as well as (possibly) a special scripting language for AR.

Collaboration: Can implement the Scene Graph Node pattern.

Known use: ImageTclAR [10], Karma [5], Coterie [9], MARS [4], EMMIE [3]

Scene Graph pattern (presentation subsystem)

Goal: Use a rendering component that allows more complex and dynamic scenes.

Motivation: For the representation of 3D environments, scene graphs have shown to be a reasonable choice. The level of abstraction is higher than for OpenGL, but they are much more powerful and flexible than VRML browsers with a limited application programming interface. Most scene graph components can read VRML descriptions of scenes.

Description: Examples are (Open) Inventor, OpenSG, Open Scene Graph.

Usability: Use a scene graph if you don't need the flexibility and low-level graphics access that OpenGL provides but want to render more complex scenes and need more dynamic access that a VRML browser offers.

Consequences: Can restrict the possibilities for modeling the application.

Known use: ARVIKA [6], Studierstube [15]

Collaboration: Scene Graph Node pattern for the application.

A Catalogue of Patterns

We classify the patterns into six problem categories which correspond to the six subsystems. Here we follow Buschmann's [2, pp 362] approach to specify categories that support the search process of developers and use the subsystem decomposition as the base for the problem categories.

	Augmented Reality Patterns
Application	Central Control Scripting Scene Graph Node Tracking-Rendering-Loop Web Service Multimedia Flow Description
Interaction	Handle in Application Use Browser Input Functions Networked Input Devices Modality Fusion DoF Adaption Operating System Resources
Presentation	3D Markup Low-level Graphics Primitives Scene Graph Video Transfer Multiple Viewers Proprietary Scene Graph 2D/3D Mapping UI Interpreter Filtering View Manager Remote Rendering Error Adaptation
Tracking	Inside-out Tracking Outside-in Tracking Tracking Server Networked Trackers Direct Access Black-box Fusion Tracker-Filter-Pipeline Interdependent Trackers
World model	Example Class Scene Graph Stream Object Stream Marker File Dynamic Model Loading
Context	Blackboard Repository Publisher/Subscriber Context Pull

Table 1. A collection of augmented reality patterns.

Table 1 gives an overview of the identified patterns classified by subsystems.

We cannot describe each of the identified patterns in detail here. Instead, we restrict ourselves to listing them here with a short description.

Central Control pattern (application)
Write the application in a high-level programming language, explicitly describing what happens when.

Tracking-Rendering-Loop pattern (application)
To simplify the development of AR applications, some libraries provide the needed low-level functionality to update the user's view regularly. The application's task is to provide hooks that can be called within the update loop and can modify the scene to be presented.

Web Service pattern (application)
The control flow is situated on a web server and implemented within a web service. This web service is published under a particular web address and the answer of the service is rendered on a web client. If the answer contains Augmented Reality content then an AR viewing component is activated to display the given AR content.

Multimedia Flow Description pattern (application)
A high-level markup language provides domain specific components and concepts that help create new content quickly. For example, to support a training scenario for unskilled workers, the AR system should visualize a sequence of AR scenes and other documents. To describe such a scenario, the content creator has to combine workflow steps and add content to each step. An execution engine for workflows reads such a description and controls the presentation of the current working step.

Handle in Application pattern (interaction)
Include input handling code in the application code, with explicit references to the types of input devices.

Use Browser Input Functions pattern (interaction)
VRML Browsers can send out events through the EAI interface when the user clicks on on-screen objects with the mouse or when the gaze direction coincides with certain objects. Other browsers provide similar functionality.

Networked Input Devices pattern
Provide an abstraction layer for input devices and a description of how the user input can be combined; interpret this description using a controller component. Use middleware to find new input devices dynamically.

Modality Fusion pattern (interaction)
Individual input modalities such as gesture and voice are combined to fire one single control event.

DoF Adaption pattern (interaction)
The degrees of freedom in an input modality are mapped upon the degrees of freedom of the application's expected input.

3D Markup pattern (presentation)
Use a viewer for high-level graphics description languages such as VRML to display 3D information. Use an API such as the External Authoring Interface (EAI) that is part of the VRML standard to modify the scene and set the viewpoint based on tracking data.

Low-level Graphics Primitives pattern (presentation)
Libraries for 3D graphics such as OpenGL provide components for rendering of low-level 3D constructs. The application developer creates new objects and tells the render to display them. With the information from the trackers, the scene can be rendered with the correct viewing direction and distance.

Proprietary Scene Graph pattern (presentation)
Use an own scene graph for graphics rendering on top of a graphics library such as OpenGL combined with an own concept for object access through an own addressing schema. Each node of the scene graph has the same abilities to serialize and address them as the other objects in the system.

Video Transfer pattern (presentation)
A thin client gathers videos through one or two head-mounted cameras, encodes them (e.g. MPEG 2), compresses them, and transfers them to a server. The server uncompresses the video images, processes them (calculates the camera position and orientation), augments, encodes and compresses the images. The images are sent to the client, decompressed and shown on the head-mounted display.

Multiple Viewers pattern (presentation)
Provide an abstraction layer for different types of viewers (AR, speech, text etc.) that can handle certain document types. Then provide the viewers with the appropriate documents.

2D/3D Mapping pattern (presentation)
Map output windows of 2D desktop applications onto output windows in a 3D environment. This allows the integration of legacy 2D applications in a spatially registered environment.

UI Interpreter (presentation)
Use an abstract description language for the specification of the user interface and render it with an interpreter.

Filter (presentation)
Reduce the complexity of the user interface by suppressing output requests of individual applications.

View Manager (presentation)
Control the user interface with a central manager component that filters or queues output requests.

Remote Rendering (presentation)
Use a remote rendering server to offload computationally intensive tasks such as rendering complex models from a CAD database.

Adaptation to Error Level (presentation)
Adapt the presentation to the level of error from other subsystems, in particular tracking. For example, if the tracking inaccuracy exceeds a certain threshold then switch from a 3D to a 2D presentation.

Inside-out Tracking pattern (tracking)
Inside-out tracking is a technique where a tracking system on the user side tracks the position and orientation. Example is optical tracking by a camera mounted on the user's head that tracks markers in the environment.

Outside-in Tracking pattern (tracking)
Tracking devices in the user's environment track the user from outside and send the information to the user system.

Tracking Server pattern (tracking)
A tracking server in the user's environment performs resource intensive computations and returns the results to the client.

Networked Trackers pattern (tracking)
For each tracking device, provide a wrapper that uses middleware concepts such as CORBA. The wrapper provides an interface to the tracker and registers itself in the network. Components that need a tracker (consumer) look for them through middleware services and connect to them. The components search for the trackers by name, not by address. Once connected, the tracker and the consumer communicate transparently.

Direct Access pattern (tracking)
The tracking devices are accessed through drivers for the operating system.

Black-box Fusion pattern (tracking)
Use a tracking fusion component that uses a fixed set of simple trackers as input and provides improved tracking data as output.

Tracker-Filter Pipeline pattern (tracking)
An application of the general pipes-and-filter pattern. Connect several trackers and or filters to a pipeline that sequentially processes the input of a tracking sensor.

Interdependent Trackers pattern (tracking)
Use a combination of trackers to process the input. The trackers are mutually connected and use the input from other trackers to modify their own output.

Example Class pattern (world model)
The developer creates a class with the constructs for a geometric body. An example is OpenGL code that calls the OpenGL rendering engine to display it. For correct registration with the user's pose, the position and angle of the virtual camera that looks at the scene can be changed. This is usually done in rendering-tracking-update loop.

Scene Graph Stream pattern (world model)
With an authoring tool a content developer creates the model of a virtual scene. In an industrial context, scenes created with CAD tools can be simplified and reused. The scene description is saved in the file system and given the AR system for processing. Scene graphs are usually stored on the file system.

Object Stream pattern (world model)
The runtime environment allows serialization of objects to disk. The next time the application is started the objects are recreated by deserializing them. Recursively, a whole scene graph can be loaded from disk.

Marker File pattern (world model)
At system startup or any time the system comes into a new environment, the trackers have to know what to look for and how to interpret it. The tracking component reads a file that describes the markers or any natural features it has to look for.

Dynamic Model Loading pattern (world model)
Instead of loading a particular scene from a file, the system has access to a database system. This system contains information about the environment, e.g. in a geographical schema. Part of the information are graphical information and marker information. The system queries for the graphical information that belongs to a discrete database object and passes it on to the rendering component. The same is true for marker information (to the tracking component) and real world objects (e.g. for occlusion).

Blackboard pattern (context)
Information producers write information to the Blackboard, a central component. Information consumers read data from the Blackboard, process them and may write new, higher abstract information to the Blackboard.

Repository pattern (context)
Components that produce context information write to the repository. Components that are interested into context information read from the repository. The repository uses an addressing schema to manage the information. Each kind of data is written and read by providing its address.

Publisher/Subscriber pattern (context)
Context providers connect as publishers to a central messaging service, context consumers as subscribers. The context providers write the new context information to a particular channel which distributes it to the connected subscribers.

Context Pull pattern (context)
An interested component directly queries the context producer component or it registers itself as subscriber. The subscriber list is managed by each component privately.

A SYSTEM OF PATTERNS

One of the goals of software patterns is to provide a common vocabulary for system designers to discuss and compare the different approaches they use. Similar to words in a vocabulary, patterns do not exist in isolation; there are interdependencies among them. Patterns can be integrated into a system of patterns: "*A system of patterns for software architecture is a collection of patterns for software architecture, combined with rules for their implementation, combination, and practical application for software development.*" [2, pp 360] Buschmann et al. [2] formulate several requirements on a system of patterns: it must contain a sufficient number of patterns, each pattern should be described consistently, the pattern system should show the relationship between patterns, the patterns should be ordered adequately, the pattern system should support the construction of new systems, and support its own evolution. We support these requirements by the catalogue of patters and the schema for the descriptions of individual patterns.

To give an overview of the relationships between the individual patterns we use a directed graph. Each pattern is part of this graph along with labelled arrows indicating direction and type of the relationships. Figure 2 shows the identified patterns and their relationships. This illustration is similar to the one used in Gamma et al. [7]. To support the locating of patterns we show the associated subsystems.

The list of patterns can be separated into two types of patterns: First, patterns that are specific to augmented reality and describe good practices for the design of augmented reality systems. Second, general patterns that we found in several existing augmented reality systems with modifications for AR. These patterns complete the pattern system for a high-level description of augmented reality systems. So although a pattern might already be well known as a general pattern for system design, for example the Blackboard pattern, we present its application in the augmented reality context.

RESEARCH ISSUES

We have identified nearly fifty patterns for augmented reality systems. However, there are several open issues left.

A system of patterns for subsystems is a practical way to discuss the software architectures of existing applications and prototypes. It provides a common vocabulary of well-known approaches for developers in the augmented reality domain. Of course, this requires an agreement among the developers on the chosen names for the patterns. With this work we present the names we chose for the identified patterns as a first proposal.

Patterns can appear on different abstraction levels. In this work, we have only considered architectural patterns. In several subsystems such as tracking, it would be worthwhile to establish more technical or algorithmic patterns for complex tasks such as sensor fusion. But these are on a different abstraction level. We could extend our current system of patterns to support more abstraction levels with technical patterns as sub nodes of architectural patterns.

The current scheme for the description of individual patterns lacks a description of their structure. We consider this as an important step for the usability of a system of patterns.

Also, existing virtual reality systems can provide a rich source for additional patterns that can be applied not only to VR, but to AR as well.

We hope that a lively discussion of patterns for augmented reality systems will benefit the AR research community and encourage collaboration between different research groups.

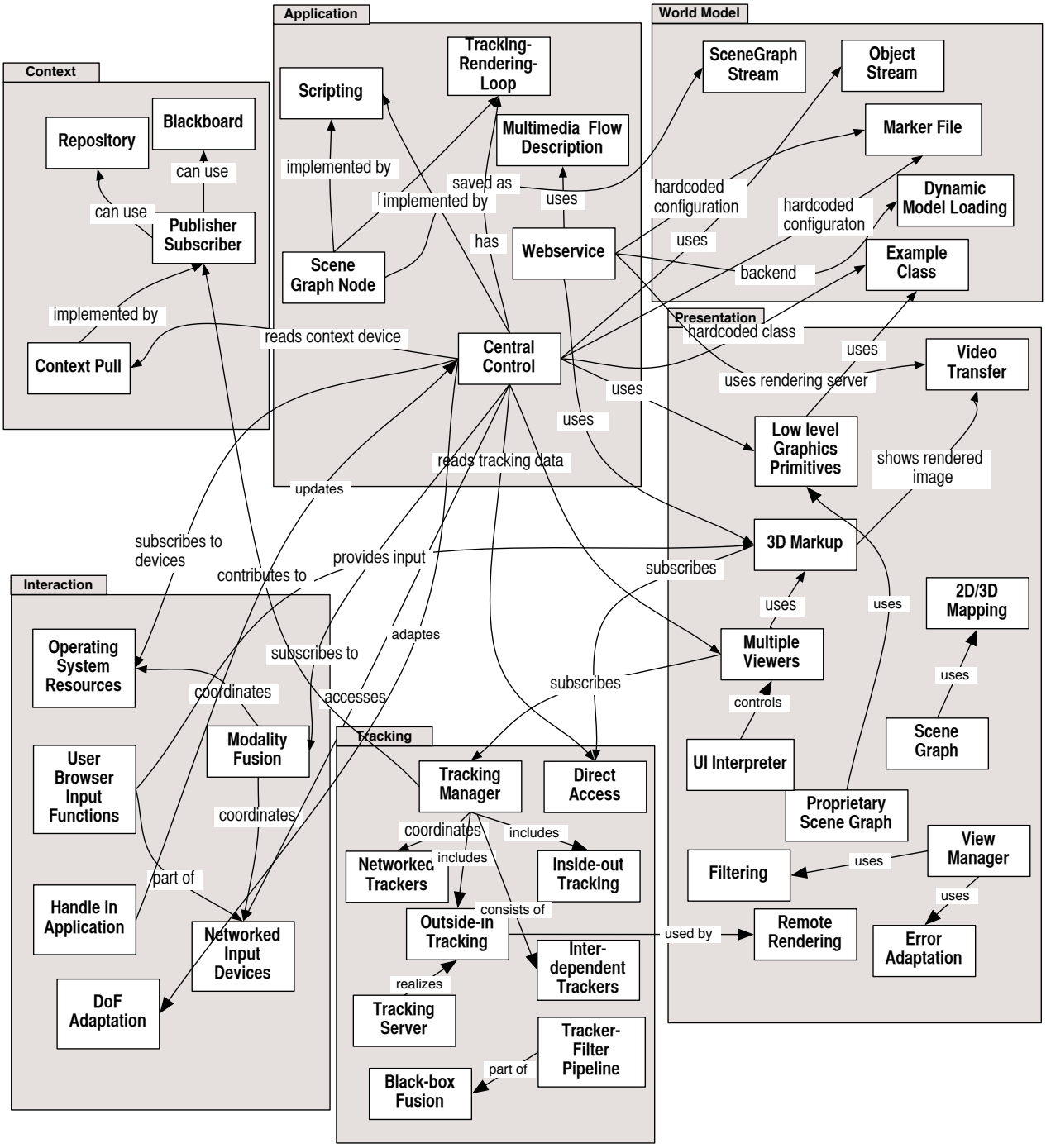

Figure 2: Relationships between the individual patterns for augmented reality systems. Several approaches are used in combination within an augmented reality system. One approach might require the use of another approach or prevent its usage.

Acknowledgments

This work was supported by the High-Tech-Offensive Zukunft Bayern of the Bavarian Government, and the compound project Forsoft 2 within the Softnet subproject supported by the Bavarian research foundation.

REFERENCES

1. R. T. AZUMA, *A Survey of Augmented Reality*, Presence, 6 (1997), pp. 355–385.
2. F. BUSCHMANN, R. MEUNIER, H. ROHNERT, P. SOMMERLAD, AND M. STAL, *Pattern-Oriented Software Architecture. A System of Patterns*, John-Wiley & Sons, 1996.
3. A. BUTZ, T. HÖLLERER, S. FEINER, B. MACINTYRE, AND C. BESHERS, *Enveloping Users and Computers in a Collaborative 3D Augmented Reality*, in Proceedings of IWAR '99 (Int. Workshop on Augmented Reality), San Francisco, CA, USA, pp. 35–44.
4. S. FEINER, B. MACINTYRE, T. HÖLLER, AND T. WEBSTER, *A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment*, in Proc. ISWC '97 (First Int. Symp. on Wearable Computers), Cambridge, MA, USA.
5. S. FEINER, B. MACINTYRE, AND D. SELIGMANN, *Knowledge-based augmented reality*, Communications of the ACM, 36 (1993), pp. 52–62.
6. W. FRIEDRICH AND W. WOHLGEMUTH, *ARVIKA - Augmented Reality for Development, Production and Service*, in The International Workshop on Potential Industrial Applications of Mixed and Augmented Reality, Tokyo, Japan.
7. E. GAMMA, R. HELM, R. JOHNSON, AND J. VLISSIDES, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1995.
8. G. E. KRASNER AND S. T. POPE, *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System*, tech. rep., ParcPlace Systems, Inc., Mountain View, USA, 1988.
9. B. MACINTYRE AND S. FEINER, *Language-level support for exploratory programming of distributed virtual environments*, in Proc. UIST '96 (ACM Symp. on User Interface Software and Technology), Seattle, WA, USA, pp. 83–95.
10. C. OWEN, A. TANG, AND F. XIAO, *ImageTclAR: A Blended Script and Compiled Code Development Systems for Augmented Reality*, in International Workshop on Software Technology for Augmented Reality Systems (STARS 2003), Tokyo, Japan.
11. W. PIEKARSKI AND B. THOMAS, *An Object Oriented Software Architecture for 3D Mixed Reality Applications*, in Proceedings of the International Symposium on Mixed and Augmented Reality.
12. T. REICHER AND A. MACWILLIAMS, *Study on Software Architectures for Augmented Reality Systems, report for the ARVIKA consortium*, tech. rep., Technische Universität München, 2002.
13. T. REICHER, A. MACWILLIAMS, AND B. BRUEGGE, *Towards a System of Patterns for Augmented Reality Systems*, in International Workshop on Software Technology for Augmented Reality Systems (STARS 2003), `http://wwwbruegge.in.tum.de/cgi-bin/pub/info.pl?publications/includes/pub/reicher2003patterns`.
14. T. REICHER, A. MACWILLIAMS, B. BRUEGGE, AND G. KLINKER, *Results of a Study on Software Architectures for Augmented Reality Systems*, in Poster Session of IEEE and ACM International Symposium on Mixed and Augmented Reality ISMAR 2003, Tokyo, Japan.
15. D. SCHMALSTIEG AND G. HESINA, *Distributed Applications for Collaborative Augmented Reality*, IEEE Virtual Reality, (2002).
16. D. C. SCHMIDT, M. STAL, H. ROHNERT, AND F. BUSCHMANN, *Pattern-Oriented Software Architecture, Vol. 2: Patterns for Concurrent and Networked Objects*, Wiley, New York, NY, 2000.