AR Patterns: Event-driven Design Patterns in Creating Augmented Reality Experiences

Philipp Ackermann

Zurich University of Applied Sciences, School of Engineering
Winterthur, Switzerland
philipp.ackermann@zhaw.ch

Abstract. Augmented Reality (AR) and Mixed Reality (MR) enable superimposing digital content onto the real world. These technologies have now matured to a point where low-code/no-code editors for AR development have emerged. However, existing collections of design principles for AR often fall short, either being too generic or overly focused on low-level details. This makes it challenging to identify the essential patterns necessary for creating captivating AR experiences. This paper addresses this issue by introducing high-level AR design patterns encompassing fundamental concepts for crafting immersive AR experiences. Event-Condition-Action rules are leveraged as a generic abstraction from the reactive behavior of AR software systems to establish a unified framework. AR-specific behavioral patterns and augmentation patterns are presented in detail. Additionally, a uniform pattern diagram schema is proposed that ensures consistent presentation and technology-agnostic documentation of AR design patterns, facilitating their effective use in design and creation of AR applications.

Keywords: Augmented Reality · Design Patterns · Reactive AR · Scene Understanding · Active Rules · AR Pattern Diagram.

1 Introduction

1.1 Augmented Reality Design Patterns

Design patterns are a widely accepted concept for documenting proven solutions to recurring design problems. Initially adapted from building architecture to software development by Gamma et.al. [1] in the mid-90s the idea has become a central concept in software engineering. Design patterns have been extensively applied to various aspects of software development, including system architectures and user interfaces, and have proven to be an effective tool for improving software quality, reusability, and maintainability.

Design patterns for virtual, augmented, and mixed reality have recently gained momentum [2–4]. Over the last decade, augmented reality (AR) technology has experienced strong advancements, resulting in a growing number of successful AR solutions. To capture and share this knowledge, design patterns have been proposed for developing AR applications on different levels of abstraction, including *software components*, *system architectures*, and *user interfaces*.

Design patterns for AR software components focus on applying generic software design patterns, such as Composite, Iterator, and Chain of Responsibility, to capture the relationships and interactions between classes or objects. These patterns can be viewed as solution templates for solving common challenges in AR software development at the object-oriented programming level.

Design patterns for AR system architectures center around the coordination of various subsystems, such as computer vision, rendering, interaction, and networking. One example is the Augmented Reality Framework [5] developed by the European Telecommunications Standards Institute (ETSI) which is responsible for setting telecommunications and broadcasting standards. ETSI's Industry Specification Group (ISG) has developed a functional reference architecture that defines relevant components and interfaces to ensure AR components, systems, and services interoperability. Other efforts to establish similar standards can be found in references such as [6, 7].

Design patterns for AR user interfaces provide guidelines for implementing user interfaces and best practices for interactions in AR. These design patterns primarily focus on adapting existing design heuristics for layout, interaction, and usability to meet AR requirements. Recently, meta-analysis studies of existing UI design patterns for AR have emerged, which provide a collection of best practices [8, 9].

1.2 AR Experiences Driven by Understanding Spatial Context

Creating AR experiences poses additional challenges compared to designing virtual reality (VR) and 3D content (e.g., video games). When creating VR/3D scenes, designers are in control of the virtual world they are building (even if it's programmatically generated), thus taking a sort of "god role". AR experiences take place in the uncontrolled real world, and scene understanding algorithms detect the user's spatial context. The AR experience is then driven by elements detected in the real world, without having control over their occurrence and timing during the creation process. Consequently, adapting VR and 3D design patterns to AR do not fit well due to the loss of control in the real-world environment. It is therefore worth focusing on AR-specific design patterns that reflect the dynamic scenography of AR/MR experiences.

1.3 Motivation: Unifying High-level AR Patterns

The current generation of AR software libraries (ARKit [10], ARCore [11], MRTK [12]), WebAR toolkits (AR.js [13], 8th Wall [14]), and AR frameworks (OpenXR [15], Unity AR Foundation [16], Vuforia [17]) offer reusable system components in run-time environments that provide a working system architecture for AR applications. These tools reduce the complexity of developing AR software, allowing developers to focus more on creating engaging AR experiences.

As a result, the use of design patterns has shifted from low-level software programming and system architecture to more high-level best practices that focus on the creation of compelling AR experiences.

When tackling high-level topics related to AR it is an obvious choice to focus on design patterns for AR user interaction which are already well documented [8, 9]. However, while there is a large collection of UI design principles for AR/MR, they tend to be either very generic (e.g., responsiveness, consistency, personalization, learnability) or focused on low-level aspects (e.g., navigation, selection and manipulation in 3D). As a result, it is challenging to identify which of the patterns are essential for creating AR experiences. The use of large, unspecific design pattern collections are of limited help in the authoring process.

The methodological approach employed in this paper is based on the analysis of the features and functionalities offered by existing Augmented Reality Software Development Kits (AR SDKs). Special attention was given to examining generic features that can be accessed and utilized within low-code/no-code AR editors such as Apple Reality Composer [18] and Adobe Aero [19]. The assessment of these features involved identifying and validating common patterns based on the author's experience in developing a mobile AR browser [20] and its associated editor toolkits [21].

The AR patterns proposed in this paper are motivated by the following objectives:

- Focusing on high-level concepts that are relevant in creating AR experiences.
- Including only patterns that are specific to AR and related to AR (i.e., excluding generic software design patterns or VR UI patterns).
- Generalizing from specific device hardware (e.g., hand-held or head-mounted devices), AR toolkits, and programming languages, so that the patterns can be applicable in different contexts.
- Unifying the AR patterns so that they can be consistently documented and presented as diagrams, making them easy to understand and apply.

2 Event-Condition-Action Pattern as Abstraction from Reactive AR System Architectures

To design reactive systems, breaking down the system's behavior into discrete events, conditions, and actions provides a structured and modular approach. An event is a signal that something has occurred, such as the start of an AR session (`on:start`), a user tapping on an item (`on:tap`), or the detection of an image marker (`on:detect`). The no-code editors in [18, 19] are using a trigger-action mechanism to define the behavior of an AR scenario. We propose to use more flexible Event-Condition-Action (ECA) rules that perform an action in response to an event, provided that certain conditions are met. ECA rules are widely used in event-driven and reactive systems, such as active databases [22] and workflow systems [23]. In the context of AR patterns, ECA rules provide a generic abstraction of the reactive behavior of AR software systems.

Table 1: Event categories in AR applications.

Event Category	**Event Producer**	**Cause** → `Event Examples`
Session Event	AR Session	State change → `on:start, on:locating`
Invocation Event	Rule Initiation	Invocation → `on:command, on:call`
Detection Event	Installed Detector	Discovery of entity → `on:detect`
User Event	App User	User interaction → `on:tap, on:select`
Temporal Event	Time Scheduler	Elapsed time reached → `in:time`
Data-driven Event	Data Observer	Value change → `on:altered, as:steady`
Response Event	Remote Request	Response of REST call → `on:response`
Notification Event	Subscribed System	System change → `on:enter, on:leave`

2.1 Reactive AR using Active ECA Rules

AR systems utilize a variety of sensors, such as cameras, LiDAR, accelerators, gyroscopes, magnetometers, and microphones, along with various event producers in operating system and user interfaces. These sensors and producers generate events asynchronously, which are then handled by event-driven programs. Unlike traditional programs that make function calls to these event producers themselves (in an inner loop), an event-driven program relies on the execution environment to dispatch events to installed event handlers. Thus, control over the execution of program logic is inverted (inversion of control).

To address the reactive nature of AR applications we promote ECA rules as a means of loosely coupling AR patterns with the underlying system architecture, while also abstracting from implementation details. AR patterns are designed to be loosely bound to the specific run-time system, since events that trigger actions within AR applications are not necessarily aware of the consequences of their occurrence. As a result, creators of AR experiences are primarily responsible for defining a set of event handling rules that govern how the system responds to various arising signals and events.

2.2 Event Categories in AR Applications

In software systems, events are typically generated by a producer and triggered by various circumstances. These events can vary greatly in nature. Regarding AR systems, we organized typical events into distinct categories, as shown in Table 1. Appendix A provides a list of common events within each event category for AR applications. Compared to other interactive 3D applications, the event categories *detection events* and *data-driven events* tend to dominate in AR.

Detection events are fundamental for AR experiences. Computer vision and machine learning techniques are applied for detecting entities in 3D space and continuously track their pose [24]. The behavior of an AR application is primarily controlled by installing the necessary detectors to receive the corresponding

detection events. These events are produced when a particular type of object is detected by the specialized detector, including:

- *Location Detector*: Tracks world location and device pose in environment.
- *Feature Detector*: classifies feature as video stream label.
- *Segment Detector*: tracks feature as image segment in video stream.
- *Plane Detector*: detects plane in 3D space.
- *Image Detector*: recognizes and tracks image or marker in 3D space.
- *Text Detector*: detects text matched by regular expression in video or 3D.
- *Code Detector*: detects QR/barcode matched by regex in video or 3D.
- *Object Detector*: detects object by shape in image or 3D.
- *Face Detector*: tracks facial parts of humans in video or 3D.
- *Hand Detector*: tracks hand, fingers, and gestures in 3D.
- *Body Detector*: tracks body parts and joints of humans in video or 3D.
- *Speech Detector*: recognizes voice commands.
- *Transmitter Detector*: locates signal and position of wireless sender.

Data-driven events are events that are generated as an AR session progresses and the understanding of the scene improves. The AR system may detect various entities, such as horizontal planes, vertical planes, and recognized objects. These entities can be transformed into application-specific data models, such as a floor, table, wall, door, window, or collision environment for a physical simulation. The detection and tracking of such entities has significantly improved with the use of machine learning techniques. Data changes can be observed at the key-value level and then trigger an event. When using a state machine, both value changes and state transitions can generate data-driven events, taking into account previous values (see Appendix A.5). This dynamic triggering turns ECA to active rules.

2.3 Condition Evaluation within Spatial AR Context

The data model reflecting the AR context is exposed for condition evaluation for any ECA rule. This model typically includes the following types of data:

- *Session Data*: time, date, device data, temporary data variables, UI mode
- *Location Data*: longitude, latitude, address, country, ambience
- *User Data*: position, orientation (tilt, yaw), name, user settings
- *Detected Occurrences*: results of installed detectors
- *Augmentation Items*: model elements with visual representation (scene nodes)

Conditions are typically formulated by predicates as logical statements using an expression syntax that has access to key-values in the data model. Additionally spatial functions can be included in condition evaluations such as:

- *Existence*: does item with ID exist in AR world
- *Visibility*: is item with ID visible to user
- *Proximity*: distance in meters from user (virtual camera) to item with ID
- *Gazing*: is user gazing at item with ID
- *Geo-Distance*: distance in meters from user to place in latitude/longitude

2.4 Actions in AR Applications

Common actions in AR applications are listed by category in Appendix B. Many of these actions are concerned with manipulating augmentation items as data model (B.1) and their visual representation (B.2), or with audible feedback as well as user interface activities (B.3).

Augmentation items are visual and audible enhancements of the world that are presented in the AR application. They can be categorized as follows:

- *Visual Items*:
  - *2D View Overlay*: 2D graphics, images, and UI elements as flat overlay
  - *3D World Embedding*: spatial geometry as node in 3D scene graph
- *Audible Items*:
  - *Speech*: recorded voice or generated voice by text-to-speech system
  - *Sonification*: sound effect driven by data or by user interaction
  - *Music and ambient sound*: play-back of audio files
- *Haptic Items*: visual or non-visual items with haptic feedback on collision

Staging of augmentation items is based on the content composition principle (`do:add`). It involves assigning unique identifiers to visual and audible items and positioning them within the observed world relative to anchors of detected entities. By doing so, the AR experience becomes more immersive, as the virtual objects are seamlessly integrated into the user's physical environment.

2.5 AR Pattern Diagram using ECA Rule Blocks

In order to provide a compact representation of active ECA rules we developed a diagram consisting of rule-reaction blocks [25]. The first line of the diagram shows the active rule as an Event-Condition-Action triple. Following the rule is a blockquoted line that depicts the changed state as reaction (see Fig. 1). If no condition is defined, it evaluates to true, and the diagram shows an immediate execution arrow (→, Fig. 2). To illustrate the use of the diagram, consider the example shown in Fig. 3, which presents an active rule triggered by a temporal event (in 20 seconds). If no item is found in the current AR session (the condition), the action will execute voice feedback (the reaction) using a text-to-speech system.

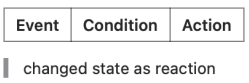


Fig. 1: ECA rule block

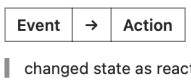


Fig. 2: Immediate EA

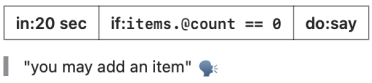


Fig. 3: Timed reaction rule

An action may dynamically load and run new rules. These rules are displayed as indented block quote (see Fig. 4) consisting of several sequential lines. All rules in a block are loaded and installed in sequence, yet not (all) executed at loading time, but triggered by their corresponding event.

Fig. 4: Consecutive loading of rules.

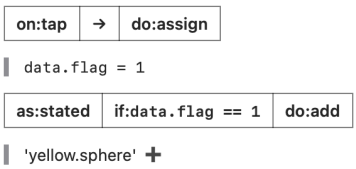


Fig. 5: Indirect reaction.

The proposed AR pattern diagram has been designed to be technology-agnostic but still enable easy transformation from existing program code or declarative scripts. As demonstrated in this paper, all examples have been transformed from declarative code to the Markdown language and rendered as styled text (see Fig. 17). The mapping to Markdown is defined in [25]. With the proposed solution, AR patterns may be seamlessly incorporated into the authoring process, ultimately improving both documentation and communication.

3 Behavioral Patterns

The real world context during an AR session can be seen as stage. The dynamic behavior of an AR experience is determined by its ECA rules, which are triggered by events occurring in the actual real-world context. Table 2 lists common behavioral patterns in AR that result from ECA rules.

3.1 Instant Reaction Pattern

An instant reaction is directly triggered by the invocation of a rule and causes the immediate, singular execution of the action (see Fig. 6). It is equivalent to a function call.

3.2 Timed Reaction Pattern

Timed reactions are ECA rules that are fired after a given interval to carry out their action. Once the timed rule is initiated, an internal job scheduler triggers the rule at the time interval specified (Fig. 7)

3.3 Conditional Reaction Pattern

An ECA rule's condition is tested on the data available in the current AR session at the time of evaluation. The outcome of the condition evaluation is either true or false. Only if true the action of the ECA rule will be executed. In addition, a conditional reaction can be combined with a timed reaction (Fig. 3 and 8). Together, these reactions form the core of the continuous evaluation pattern.

Table 2: Behavioral patterns in AR applications.

Behavioral Pattern	**Description**	**Examples**
Instant Reaction Pattern	Direct execution of action triggered by invocation of rule	Immediate command of action or call of function
Timed Reaction Pattern	Temporally executed action	Delayed action or sequence of timed actions
Conditional Reaction Pattern	Execute an action only when a condition is fulfilled after being triggered by event	State-driven, asynchronous programming logic
Continuous Evaluation Pattern	Continuous polling of state changes that will triggers rules	Continuous checks on value change, existence, visibility, proximity
Publish-Subscribe Notification Pattern	Receive notifications via a message queue from a subscribed system	From speech recognition system or from WebRTC system in collaboration session
Request-Response Pattern	Remote procedure call resulting in asynchronously receiving ECA rules or media assets	REST API call to a server via a Web URL to load rules or assets (images, 3D models)
Chain Reaction Pattern	Course of events processed as indirect reactions of running subsequenced rules	Rule changing data that will trigger a rule to update an item's visual as a follow-up
Complementary Reactions Pattern	Two active rules with opposite reactions	Reacting on toggling states with two complementary active rules
Detector Reactivation Pattern	Reactivate detector with a only-once reaction	Reactivate detector after resulting augmentation is no longer existing

Fig. 6: Instant reaction

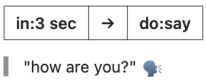


Fig. 7: Timed r.

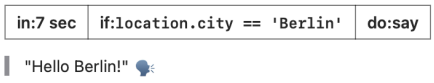


Fig. 8: Timed conditional reaction

3.4 Continuous Evaluation Pattern

The continuous evaluation of rules can be driven by a constant time interval (first rule in Fig. 9) or by each state change of the data model (second rule in Fig. 9).

A built-in state management is observing the data model and does dispatch the processing of rules according to the data-driven events (Appendix A.5) that are bound to the ECA rules.

Fig. 9: Continuous evaluation pattern.

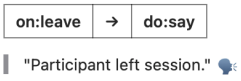


Fig. 10: Notification pattern.

3.5 Publish-Subscribe Notification Pattern

In a similar manner to monitoring changes in the data model, changes in subsystems can also generate events that trigger rules. Subsystems can be observed through a publish-subscribe notification mechanism. For instance, a speech recognition system might be initiated (by a `do:listen` action) and subscribed to. When a voice command is recognized, an `on:voice` event is published as message. Another example is the usage of a collaboration subsystem that sends notifications when a participant joins or leaves a session (Fig. 10).

3.6 Request-Response Pattern

The Request-Response Pattern is a method of communicating with remote services through non-blocking asynchronous communication. This pattern is commonly used to request media assets such as images, audio files, video streams, or 3D models, as well as scripts such as additional ECA rules (Fig. 4). Because of the temporal decoupling of request and response, an event is required to signal the arrival of the received data from the server. This allows an ECA rule to handle the result as soon as it is available.

3.7 Chain Reaction Pattern

A reaction of a rule can trigger a new event that invokes subsequent rules. In turn, these subsequent rules may have reactions that again invoke rules, leading to a chain of reactions. The Chain Reaction pattern consists of consecutive loaded rules (Fig. 4) and of indirect reactions (Fig. 5) which lead to a cascade of triggered rules with their corresponding reactions. To better illustrate this concept, consider the example shown in Fig. 16, which demonstrates how the Chain Reaction pattern can be used to detect and augment an image.

3.8 Complementary Reactions Pattern

Complementary reactions are constructed using two rules that have opposite conditions and actions with opposite results. When these two rules are evaluated continuously, they exhibit a toggling state by flipping their mutual rule execution (Fig. 11).

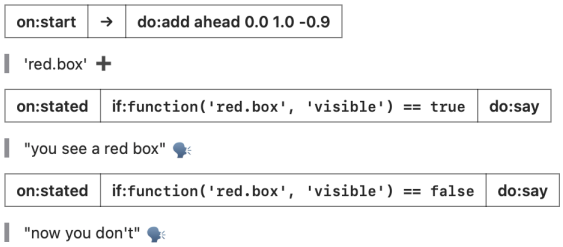


Fig. 11: Complementary reactions.

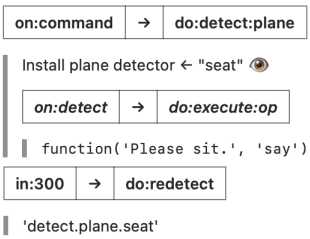


Fig. 12: Detector reactivation.

3.9 Detector Reactivation Pattern

Some detectors halt after capturing a first occurrence of an entity and need to be reactivated by a `do:redetect` action. The reactivation can be driven by a separate active rule, for instance, after a specific period of time (Fig. 12) or by assessing the existence or visibility of the item added by the detector based on a corresponding condition.

4 Augmentation Patterns

While a VR/3D designer is placing virtual objects using positions in a controlled world coordinate system, an AR content creator primarily specifies object placement intents relative to appearing anchors, which are dynamically produced by detectors. These spatial anchors serve as reference points for pinning objects. Generally in AR Patterns, the augmentation intents are formulated as ECA rules that are triggered by detector events. When a detector event occurs, ECA rule's reaction will add augmentation items to the AR scene.

Fig. 13 depicts a rule that invokes an image detection action if a command event is received. As reaction, the image named 'marker.png' is requested and downloaded. Once the image is downloaded, an image detector is installed and a subsequent rule is loaded to be triggered when the image is detected in the real world. Upon detecting the image, an instant reaction pins an item with the id 'scene.3D' to the anchor of the detected image.

Another example of scene augmentation is shown in Fig. 14. In this example a detector is installed that is capturing planes of type 'seat'. When detected, this results in indirect reactions that include the creation of audible and visible augmentations.

Table 3 outlines several common placement intents for event-driven augmentation patterns that can be used to stage AR experiences. In AR, the real world serves as the spatial context for the stage, making users both spectators and performers. Their movements and perspectives influence the firing of events, leaving limited control over time and space for AR scenography (in contrast to film, theater, and VR/3D/game design). The augmentation patterns differ by the *purpose* of the added augmentation items (depicted by title), where they are *placed* (position in space), and how they are *aligned* (orientation, see Fig. 15).

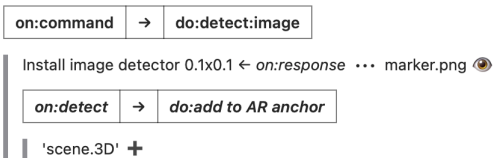


Fig. 13: Augmentation intent as ECA rule.

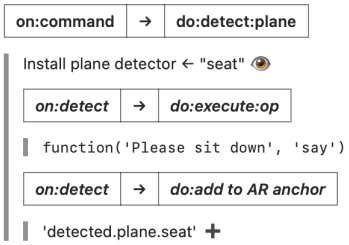


Fig. 14: Rules for adding audible and visual augmentations.

4.1 Geolocated Remark Pattern

A geolocated remark is a rule that is triggered by GPS location data (latitude and longitude) or by address data (i.e., country, city, street, building name). Typically, the reaction to a geolocated remark is presented as text in the 2D user interface or as audio feedback. Due to the precision restrictions of GPS signals, the reaction is not precisely placed in 2D or 3D space.

placed: *no*	aligned: *no*

4.2 Segment Overlay Pattern

By utilizing computer vision and machine learning techniques, a segment detector can recognize and track designated landmarks and image segments. The outcomes of the image segmentation process may include:

- *point (pixel)*: room corner, object corner, pupil center, ...
- *edge*: horizon line, wall-floor edge, ...
- *bounding box*: depicting rectangle border of detected text, image, face, ...
- *path (open path)*: eyebrow, body skeleton, ...
- *contour (closed path)*: face, mouth, eye, ...
- *image mask*: sky, grass, hair, ...

In the AR view, the overlay is positioned based on the pixels relative to the segment within the 2D image, on top of the video stream.

placed: *on screen at image segment*	aligned: *flat on top as overlay*

4.3 Area Enrichment Pattern

The Area Enrichment Pattern uses detected image segments to calculate the spatial 3D area resulting from perspective projection of the segment into depth of space. The calculated 3D area can then be used to populate the space with virtual items.

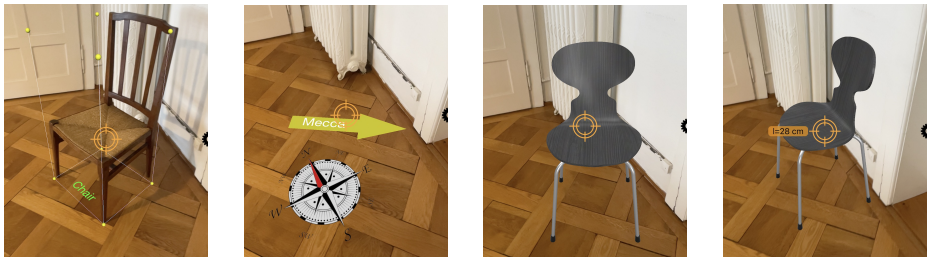
placed: *in 3D area at image segment*	aligned: *to area / towards user*

Table 3: Augmentation patterns in AR applications.

Augmentation Pattern	**Description**	**Examples**
Geolocated Remark Pattern	Triggering of action or of user feedback based on GPS location data or on address data	Visual or audio feedback about location-based point of interest
Segment Overlay Pattern	Presentation of 2D overlay on top of image segment detected in video stream	Attaching 2D text description to a detected image segment
Area Enrichment Pattern	Approximately placing 3D content at area of image segment	Presenting balloons in sky area
Captured Twin Pattern	Captured element of real world added to 3D data model	Captured walls and doors in an indoor AR session
Anchored Supplement Pattern	Presentation of 3D content aligned to detected entity for enhancement	Attaching visual 3D elements to a detected image (marker) or captured object
Superimposition Pattern	Presentation of 3D content replacing a detected entity	Cover a detected object with a virtual one
Tag-along Pattern	Presentation of 3D content within user's field of view while head-locked	Place 3D control panel that follows the user
Hand/Palm Pop-up Pattern	Presentation of 3D content on hand or palm while visible	Place 3D UI elements at palm of user's one hand
Ahead Staging Pattern	Presentation of 3D content ahead of user	Placing 3D item on floor in front of spectator
Pass-through Portal Pattern	Present partly hidden 3D content to force user to go through	Placing 3D scene behind a portal / behind an opening
Staged Progression Pattern	Ordered, linear story: temporal order or interaction flow of 3D presentations	Sequence of 3D content with forth and optionally back movements
Attention Director Pattern	Guide user's attention to relevant place	Use animated pointers to direct user's attention
Contextual Plot Pattern	Spatio-temporal setting that aggregates diverse AR patterns to form a non-linear plot	Scenography of dynamic, interactive, and animated AR

4.4 Captured Twin Pattern

A captured twin is a virtual replica of a physical element. It is created using data collected from sensors, cameras, and other sources. A captured twin can then, for example, be virtually visualized as a contour or as a transparent 3D bounding box to keep the real object recognizable. In Fig. 15a, a real-world chair has been detected as an object and is indicated with a virtual bounding box and a text



(a) with object (b) geo-referenced (c) towards user (d) to environment

Fig. 15: Alignment of augmentations in 3D.

label of the object type. A captured twin may also have no virtual representation but becomes available in the spatial data model (e.g., for collision detection or as a reference for spatial alignment).

placed: *on object*	aligned: *with object*

4.5 Anchored Supplement Pattern

Anchored supplements provide additional information that is aligned with a detected entity. For instance, a 3D info panel can be anchored beside a detected image, as demonstrated by the pattern diagram in Fig. 16. Similarly, a guiding sign can be anchored towards a detected door to assist with navigation.

placed: *relative to object*	aligned: *with object or towards object / user*

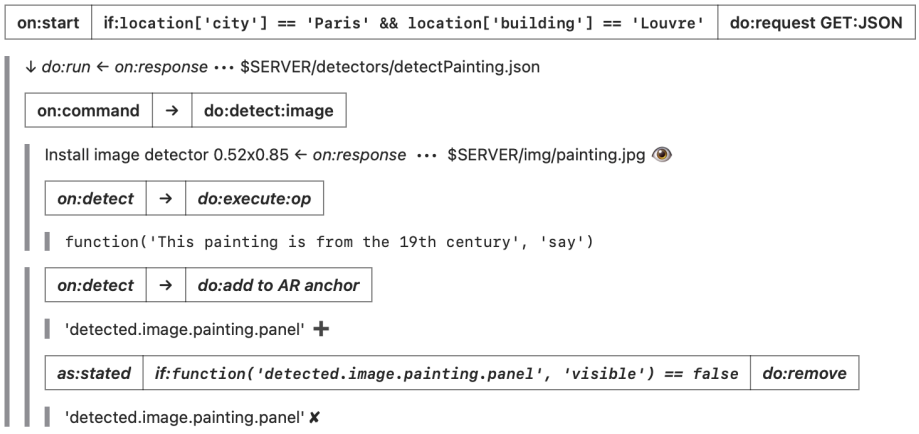


Fig. 16: Example of an anchored supplement using image detection.

In Fig. 16, an example is shown where an ECA rule is executed at the beginning of an AR session to determine whether the current location is suitable for detecting an image as a painting. If the location is a match, an image is loaded from a remote server and installed as active rule set which provides audible and visible augmentations in case of detecting the image in the real world. The visible augmentation takes the form of an info panel that is monitored by a rule which removes it when it is no longer visible from the user's perspective.

4.6 Superimposition Pattern

Superimposition aims to cover a detected entity with a virtual one rather than simply supplementing it. For example, if an image is detected, it could be covered with a downloaded image, or if an object is recognized, it could be replaced with a virtual 3D object that hides the real one.

placed: *on object*	aligned: *with object*

4.7 Tag-along Pattern

A tag-along augmentation attempts to always stay within the user's view range, which is constrained to their eye position. To achieve this, the augmentation is typically always oriented towards the user's face using billboarding techniques. The main advantage of tag-along augmentations is that they ensure the presented interaction elements are always visible and easily accessible to the user. In AR applications for head-mounted displays, tag-along augmentations are usually designed to fit within the user's arm-length range, ensuring they can interact with the elements without stretching or straining themselves. An example is the 'Near Menu' UX component of the Mixed Reality Toolkit [12].

placed: *constraint ahead of user*	aligned: *constraint towards user*

4.8 Hand/Palm Pop-up Pattern

The hand/palm pop-up is a prevalent design pattern used to present interaction elements for AR using head-mounted displays. Rather than constantly displaying these elements, they are shown only when the palm or back of the hand is visible (and optionally, when a hand gesture was detected beforehand). The other hand can then be used to interact with the presented 3D UI elements. An example is the 'Hand Menu' UX component of the Mixed Reality Toolkit [12].

placed: *constraint on palm*	aligned: *constraint towards user*

4.9 Ahead Staging Pattern

Ahead staging is a technique for presenting 3D content in a way that it is aligned to spectator's position and view direction. The scene is launched at a default

distance (1-2m) in front of the user using a world-locked anchor, often relative to the ground floor plane. The alignment can be directed towards geolocated references (e.g., as a guide as in Fig. 15b), towards the user (Fig. 15c), or can also consider nearby objects in the environment (e.g., a wall as in Fig. 15d or the room axis). After the initial staging, users can interact with the virtual scene from their current position or move toward and around the staged content.

placed: *initial ahead of user*	aligned: *initial towards user, object, or georef*

4.10 Pass-through Portal Pattern

A pass-trough portal is an augmentation that is initially occluded by a virtual object, which prevents the user from seeing the entire scene. This design is intended to encourage users to engage with the experience by requiring them to pass through a gateway to become fully immersed.

placed: *initial ahead of user*	aligned: *initial towards user or object*

4.11 Staged Progression Pattern

A staged progression refers to presenting a linear story in a structured and sequential manner in AR, with an explicit beginning. This typically involves the ordered presentation of 3D content, which can be unidirectional or bidirectional (going back and forth). The story usually starts staged ahead of the user, but it can also begin at the anchor of a detected entity. The story's progress is governed by rules triggered by user events, temporal events, or data-driven events.

placed: *initial ahead of user*	aligned: *initial towards user or object*

4.12 Attention Director Pattern

An attention director uses animated pointers (bubbles, arrows), light rays, or spatial sound to direct where users should pay attention if the relevant area is not visible or not in focus.

placed: *initial ahead of user*	aligned: *pointing towards point of interest*

4.13 Contextual Plot Pattern

A contextual plot combines a variety of augmentation patterns and behavioural patterns to create a non-linear and immersive experience. It builds a scenography that blends seamlessly with the spatio-temporal setting of the real world. A contextual plot reacts to diverse event types, triggering rules that will dynamically control interactive and animated reactions. One of the defining features of a contextual plot is its dependence on the real-world context. The experience is affected by how the user is interacting with and exploring the scenario.

placed: *multiple all over spatial context*	aligned: *diverse*

5 Using AR Patterns in Authoring Tool

5.1 Declarative Creation of AR Content

The proposed AR patterns were elaborated and validated during the development of the ARchi VR App [20]. Instead of using a programming language to algorithmically define how AR content should be created and behave, the app uses a declarative approach that focuses on specifying what needs to be accomplished with each AR asset. To achieve this, the app interprets declarations in JSON data structures that do not include conventional programming code, but instead use active ECA rules to define the behavior of the AR experience [21].

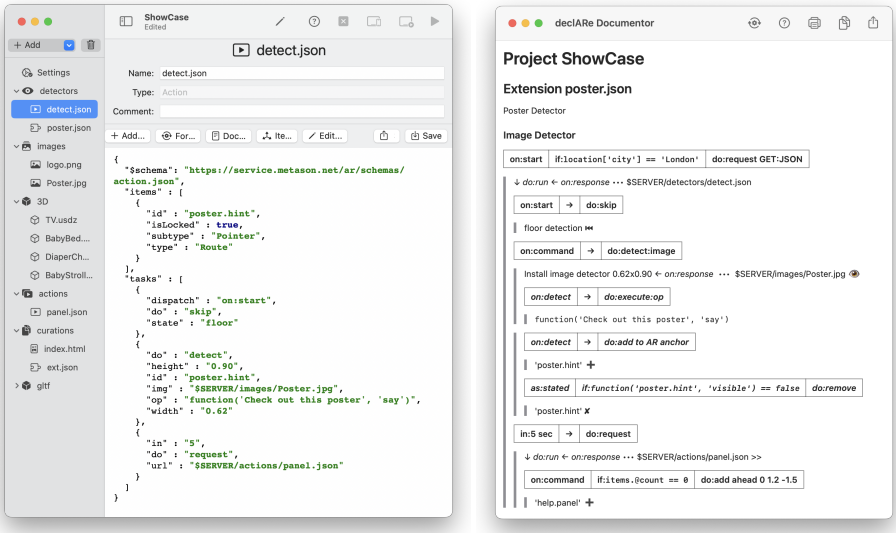


Fig. 17: Generated AR pattern diagram (right) from code (left) in an AR IDE.

5.2 Generation of AR Pattern Diagrams

When using such a declarative authoring approach many simple ECA rules are created and encoded (in case of the ARchi VR App in many separate JSON files). This helps to keep local focus when creating AR scenes, but it can be challenging to maintain an overview of the entire rule system that defines all the intents of the AR world. To address this challenge and better understand how rules are loaded and run in sequence, an integrated development environment (IDE) could support on-the-fly generation of AR pattern diagrams [25] from the encoded rules (see Fig. 17) to visualize an overall view of intended AR scenario.

6 Conclusions and Perspectives

6.1 Summary

The proposed event-driven AR design patterns have shown to be a convincing way of documenting high-level concepts in creating AR applications. AR patterns serve as a valuable means of communicating proven, reusable solutions to recurring design problems encountered during AR development. Once AR patterns are understood, learning new AR toolkits might become more accessible, as the used design patterns are familiar.

During the technical design and development process, AR patterns may also serve as a requirements catalog to clarify the specification of features and functions that an AR application should support. If an editor facilitates the creation of AR content, it might make sense to support the generation of AR pattern diagrams from code. In addition, AR design patterns may also be a useful organizational structure for providing solution templates, such as code snippets in AR authoring tools. This would allow developers to incorporate common patterns into their AR applications efficiently, improving development speed and overall quality.

6.2 Community-driven Validation

We are strongly convinced that AR patterns can be applied to hard-coded, scripted, and no-code implementations making them widely applicable. Yet the approach was showcased in only a few development, research, and student projects. Therefore the soundness and comprehensiveness of this proposal should be further elaborated and validated:

- Do the proposed AR Patterns resonate within the AR community?
- Are common high-level AR patterns missing?
- Are the proposed AR patterns useful in the AR design process?
- How can guidelines foster learning and applying of AR patterns?
- Is the Event-Condition-Action pattern a generic abstraction mechanism?
- Can AR Pattern diagrams be in fact used "technology-agnostic"?

One way to improve the expressiveness of the proposed catalog of high-level AR patterns would be to display rendered AR user interfaces as samples and to provide implementation-specific code examples. While this paper provides a starting point, future research should further expand the catalog besides the documented behavioral patterns and augmentation patterns. Other important AR-related design patterns should be explored, e.g., covering gesture- and voice-based interactions, human-human AR collaboration [27], and human-robot AR collaboration [28]. This paper addresses the need for AR patterns in the hope that it contributes to diverse innovations ahead of us within the AR community. Contributions are welcome and are planned to be coordinated via arpatterns.dev [29] and the AR Patterns catalog repository [30].

6.3 ECA Rules as Foundation for AR Interoperability

For exchanging AR worlds between applications from different providers, open standards are in demand. The Event-Condition-Action rule could serve as a common pattern in defining standardized exchange formats for augmentation and behavior of AR content and metaverse assets (as addressed by the Metaverse Standards Forum [26]). This requires further clarifications and feasibility studies.

6.4 Generating Code from AR Patterns

A visionary idea (mentioned by a paper reviewer) is the generation of code from AR patterns. Yet it has not even for software design patterns became a widespread approach in IDEs (although some papers appeared). Recent development in generative AI (e.g., Co-Pilot) could be a solution to fulfill the vision of generating code from design patterns. As a prerequisite for such a generative approach the AR community needs to establish a common understanding of AR design patterns and has to provide sample code organized according to AR patterns.

Acknowledgements. The author would like to thank Yanick Lukic and Stefan Schmidlin for their comments and proofreading.

A Common AR Event Types

A.1 Session Events

- `on:start`: immediately after start of AR session or after loading action
- `on:locating`: on locating in the world (by GPS, by SLAM device positioning)
- `on:stable`: when spatial registration of AR device gets stable
- `on:load`: after loading 3D item to AR view, e.g., to animate or occlude node
- `on:stop`: before AR session ends

A.2 Invocation Events

- `on:command`: on command initiation
- `on:call`: on function call

A.3 User Events

- `on:tap`: when tapped on item
- `on:press`: when long-pressed on item
- `on:drag`: when dragging an item
- `on:select`: when selected from options of pop-up menu
- `on:dialog`: when selected from options of pop-up dialog panel
- `on:poi`: when selected a point of interest in a map or minimap

A.4 Temporal Events

- `in:time`: when elapsed time in seconds is reached
- `as:always`: several times per seconds
- `as:repeated`: like `as:always`, but only triggered `each` seconds

A.5 Data-driven Events

- `on:change`: on each change of data value
- `as:stated`: like `as:always`, but action only is triggered once when if-condition result is altered from false to true
- `as:steady`: like `as:stated`, but action only is triggered when condition result stays true `for` a certain time in seconds.
- `as:activated`: like `as:always`, but action always is triggered when if-condition result becomes true
- `as:altered`: like `as:always`, but action always is triggered when if-condition result is altered from false to true or from true to false

A.6 Response Events

- `on:response`: on receiving response from request
- `on:error`: on error of handling request

A.7 Detection Event

- `on:detect`: on detecting occurrence of depicted type
- `on:track`: on tracked changes in occurrence of depicted type

A.8 Notification Events

- `on:voice`: on voice command from speech recognition system
- `on:enter`: on enter of participant in collaboration session
- `on:message`: on message from participant in collaboration session
- `on:leave`: on leave of participant in collaboration session

B Common AR Actions

B.1 Item-related Actions

Items are elements of the application model. An item can be represented as 3D object in the AR scene or as 2D overlay (sprite image, text label, UI element, ...) on top of the AR view.

- `do:add at`: add and anchor item at position
- `do:add onto`: add and anchor item onto another item
- `do:add to`: add item as child to another item
- `do:add ahead`: add and anchor item ahead to user position and orientation
- `do:add overlayed`: add 2D item flat on top of AR view
- `do:remove`: remove item from scene
- `do:replace`: replace item with another item at same position
- `do:move to/by`: move item absolute/relative to new position
- `do:turn to/by`: turn item absolute/relative to new orientation
- `do:tint`: set color of item
- `do:lock/unlock`: lock/unlock item to control allowed manipulation

B.2 Visual-related Actions

Visual-related actions do change the visual representation of items in the AR scene but are not reflected or stored in the application model.

- `do:hide/unhide`: hide/unhide visual representation of item
- `do:translate to/by`: move absolute/relative
- `do:rotate to/by`: rotate absolute/relative
- `do:scale to/by`: scale absolute/relative
- `do:animate key`: create animation of graphical parameter (key)
- `do:stop key`: stop animation of graphical parameter (key)
- `do:occlude`: set geometry of 3D node as occluding but not visible
- `do:illuminate`: add additional lightning to hot spot (from above or from camera)

B.3 UI-related Actions

Common actions to control the user interface of an AR application.

- `do:prompt`: show instruction in a pop-up panel
- `do:confirm`: get a YES/NO confirmation via an pop-up dialog panel
- `do:warn`: set warning or status label
- `do:vibrate`: vibrate device
- `do:play`: play system sound
- `do:stream`: play remote audio file
- `do:pause`: pause current audio stream
- `do:say`: say something using text-to-speech (TTS) system
- `do:listen`: start speech recognition system and subscribe
- `do:open service`: open service menu
- `do:open catalog`: open item catalog
- `do:install`: install item catalog entry or service menu entry
- `do:filter`: filter item catalog or service menu
- `do:screenshot`: take screen snapshot
- `do:snapshot`: take photo shot

B.4 Data-related Actions

- `do:assign`: set a data variable to a value
- `do:concat`: concatenate a string with an existing variable
- `do:select`: select a value from a menu and assign to data variable
- `do:eval`: set a data variable by evaluating an expression
- `do:fetch`: fetch data from remote and map to internal data
- `do:clear`: delete data variable

B.5 Process-related Actions

- `do:save`: save the AR scene / application model
- `do:exit`: end AR session
- `do:execute`: execute function(s)
- `do:service`: execute service action
- `do:workflow`: execute workflow action
- `do:request`: request action from remote server and execute

B.6 Detector-related Actions

- `do:detect type`: install detector for type (feature, text, plane, image, ...)
- `do:halt`: deactivate detector
- `do:redetect`: reactivate detector

References

1. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Pearson (1995)
2. Nystrom, Robert: Game programming patterns. Genever Benning, (2014).
3. Zollmann, S., Langlotz, T., Grasset, R., Lo, W. H., Mori, S., Regenbrecht, H.: Visualization techniques in augmented reality: A taxonomy, methods and patterns. In: IEEE transactions on visualization and computer graphics, 27(9), 3808-3825, (2020).

4. Koreng, R., Krömker, H.: User Interface Pattern for AR in Industrial Applications. Information (2021); 12:251. https://doi.org/10.3390/info12060251
5. ETSI GS ARF 003: Augmented Reality Framework (ARF) - AR framework architecture. www.etsi.org (2020)
6. MacWilliams, A., Reicher, Th., Klinker, G., Bruegge, B.: Design patterns for augmented reality systems. In: Proceedings of the International Workshop exploring the Design and Engineering of Mixed Reality Systems, Funchal, Madeira (2004)
7. Makamara, G., Adolph, M.: A Survey of Extended Reality (XR) Standards. In: ITU Kaleidoscope-Extended reality–How to boost quality of experience and interoperability, 1-11 (2022). https://doi.org/10.23919/ITUK56368.2022.10003040
8. Görlich, D., Akincir, T., Meixner, G.: An overview of user interface and interaction design patterns for VR, AR, and MR applications. In: Mensch und Computer, Workshopband, Darmstadt (2022). https://doi.org/10.18420/muc2022-mci-ws06-419
9. Börsting I, Karabulut C, Fischer B, Gruhn V.: Design Patterns for Mobile Augmented Reality User Interfaces - An Incremental Review. Information (2022); 13(4):159. https://doi.org/10.3390/info13040159
10. Apple ARKit: https://developer.apple.com/augmented-reality/arkit/
11. Google ARCore: https://developers.google.com/ar?hl=en
12. Microsoft MRTK: Microsoft Mixed Reality Toolkit; https://learn.microsoft.com/windows/mixed-reality/mrtk-unity/mrtk3-overview
13. AR.js - Augmented Reality on the Web: https://ar-js-org.github.io/AR.js-Docs/
14. Niantic 8th Wall: https://www.8thwall.com/products-web
15. OpenXR Standard: https://www.khronos.org/openxr
16. Unity AR Foundation: https://unity.com/unity/features/arfoundation
17. PTC Vuforia: https://www.ptc.com/en/products/vuforia
18. Apple Reality Composer: https://developer.apple.com/augmented-reality/tools/
19. Adobe Aero: https://www.adobe.com/products/aero.html
20. ARchi VR App: https://archi.metason.net
21. ARchi VR Content Creation: Technical Documentation; https://service.metason.net/ar/docu/
22. Paton, N. W. (ed.): Active rules in database systems. Springer Science & Business Media, (2012).
23. Müller, R., Greiner, U., Rahm, E.: AGENTWORK: A workflow system supporting rule-based workflow adaptation. Data & Knowledge Engineering. 51. 223-256, (2004). https://doi.org/10.1016/j.datak.2004.03.010
24. Rambach, J., Pagani, A., Stricker, D.: Principles of Object Tracking and Mapping. In: Springer Handbook of Augmented Reality (pp. 53-84). Cham: Springer International Publishing (2023).
25. AR Pattern Diagram: https://github.com/ARpatterns/diagram
26. The Metaverse Standards Forum: https://metaverse-standards.org
27. Pidel, C., Ackermann, P.: Collaboration in virtual and augmented reality: a systematic overview. In Augmented Reality, Virtual Reality, and Computer Graphics: 7th International Conference, AVR 2020, Lecce, Italy, Proceedings, Part I 7 (pp. 141-156). Springer International Publishing, (2020).
28. J. Delmerico et al.: Spatial Computing and Intuitive Interaction: Bringing Mixed Reality and Robotics Together; in: IEEE Robotics & Automation Magazine, vol. 29, no. 1, pp. 45-57, March 2022, https://doi.org/ 10.1109/MRA.2021.3138384.
29. AR Patterns: https://arpatterns.dev
30. AR Patterns Catalog: https://github.com/ARpatterns/catalog