Model-Driven Engineering for Augmented Reality

Rubén Campos-López*, Esther Guerra*, Juan de Lara*, Alessandro Colantoni†, and Antonio Garmendia*
*Universidad Autónoma de Madrid, Spain
†Johannes Kepler University, Linz, Austria

ABSTRACT The steady increase of the capabilities of mobile devices and the appearance of novel head-mounted widgets has triggered the interest in developing Augmented Reality (ARs) applications. In these applications, virtual objects can be overlaid over the real ones, and the user can interact with the virtual objects. AR applications open the door to innovative scenarios both for industrial use and leisure. However, their construction requires substantial effort and specialised knowledge.
In this paper, we report on a model-driven approach to build AR applications that eliminates the need for manual coding. It is based on the definition of a domain meta-model, enriched with AR representations for the domain concepts and a description of the interaction with external information systems and devices. This paper presents the concepts, the technical realisation in an iOS tool called AlteR, an evaluation on five case studies, and a user study demonstrating its usefulness and usability.

KEYWORDS Model-Driven Engineering, Augmented Reality, Software Language Engineering, Mobile Computing.

1. Introduction

The increasing capabilities of mobile devices and the emergence of head-mounted widgets has enabled the use of Augmented Reality (AR) (Azuma 1997) as the interface for all sorts of applications (Ling 2017). These range from industrial applications – e.g., in combination with Digital Twins (Böhm et al. 2021), as training assistants (Heinz et al. 2019), or for remote collaborative work (Schäfer et al. 2022) – to education (Hincapié et al. 2021) and leisure activities like gaming in mobile devices (Pierdicca et al. 2019). The promotion of AR in platforms like SnapChat (Snap AR), or the upcoming Metaverse (Metaverse), has risen the usage of AR applications and the expectation of all sorts of AR-based usage scenarios in the short term.

Extended reality (XR) is a recent term coined to refer collectively to immersive technologies, including virtual reality (VR), augmented reality and mixed reality (MR). Augmented reality can be explained in terms of a *virtuality continuum*, as defined by (Milgram & Kishino 1994). As Figure 1 shows, *real environments* consisting only of real objects are at one end of this continuum, and *virtual environments* consisting solely of virtual objects are at the other end. *Mixed reality* environments can be anywhere between the extrema of this continuum, as they present both real objects and virtual objects together within a single display. AR is one of the best known types of mixed reality environments, where the display of a real environment is augmented by means of virtual (computer graphic) objects. This way, while users in a virtual environment are immersed in a synthetic world and do not perceive the real world around them, users in an AR environment perceive a combination of the virtual and real worlds.

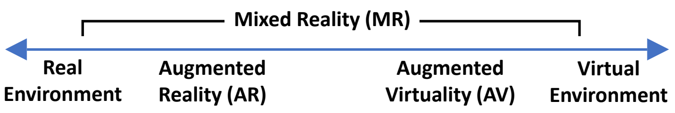


Figure 1 The reality-virtuality continuum according to (Milgram & Kishino 1994).

Azuma expands the definition of AR to systems having three characteristics (Azuma 1997): combining real and virtual objects in a single space, allowing user interaction in real-time, and registering virtual and real objects in the 3D space accurately. Additionally, the realisation of AR requires specific hardware devices, such as headset devices (like Microsoft HoloLens), smart

JOT reference format:
Rubén Campos-López, Esther Guerra, Juan de Lara, Alessandro Colantoni, and Antonio Garmendia. *Model-Driven Engineering for Augmented Reality*. Journal of Object Technology. Vol. vv, No. nn, 2023. Licensed under Attribution - NonCommercial - No Derivatives 4.0 International (CC BY-NC-ND 4.0) http://dx.doi.org/10.5381/jot.2023.vv.nn.aa

glasses, or the camera of hand-held displays (like smartphones and tablets).

The capability of AR technologies to display virtual objects overlaid on physical ones can be used to show information about the real object (e.g., working data of a machine in a factory) or interact with the real object via the virtual one (e.g., configure the machine using the AR overlay). This way, AR-based interaction can be suitable in application domains such as smart factories (to display real-time data about the factory's machines and processes on-site) (Coscetti et al. 2020), Internet of Things (to configure and connect devices) (Seiger et al. 2019), interior design (to overlay virtual furniture on a real room) (IKEA room planner), transportation (to show directions overlaid with the reality) (Google maps live view), emergency management (to overlay relevant changing information on the emergency scenario) (Campos et al. 2019), and museums (to augment the information about the exhibited art works), among others (Brunschwig et al. 2021). However, building AR applications is time-consuming and requires specialised, highly technical knowledge (Ashtari et al. 2020; Nebeling & Speicher 2018).

To tackle this problem, we propose an approach that eliminates the need for coding to build specific classes of AR applications. It is founded on model-driven development and software language engineering principles (Brambilla et al. 2017). The method requires defining a meta-model describing the domain (i.e., the concepts, properties and relations tackled by the application), as well as the AR representation of the meta-model elements, which can include 3D objects, images and videos. This defines an application supporting the *C*reation, *R*ead, *U*pdate and *D*eletion (CRUD) of the AR objects representing the meta-model concepts and relations. The AR objects can be placed (*anchored*) in the real world using image recognition, barcodes, QR codes and Bluetooth Low Energy (BLE) beacons. Finally, it is possible to enrich the application specification to make it interact with external information systems via REST APIs.

Overall, our approach permits creating AR applications with no need for coding. These range from CRUD applications that provide an AR representation for domain-specific concepts (i.e., AR-based modelling environments (Brunschwig et al. 2021)), to applications that communicate with devices like BLE beacons, and AR front-ends for information systems.

The approach is supported by a prototype tool for iOS devices (iPads and iPhones) called ALTER (https://alter-ar.github.io/), which is freely available in Apple's App Store (https://apps.apple.com/us/app/alter-ar/id1574875872). After introducing the approach and the tool in detail, we report on an evaluation based on five case studies that illustrate the capabilities that AR applications created with our approach can have. These case studies comprise an AR-based domain-specific language for home networking; an application to augment the user experience in a museum; an augmented home application where users can add virtual decorations, including a weather station that reads data from a web service; an AR front-end for an inventory information system that overlays data about equipment when detecting their QR codes, and permits modifying the data; and an AR-based social network that displays user messages upon the detection of BLE beacons. The case studies are complemented by a user study that evaluates the usability of one of the resulting AR applications.

This paper extends our preliminary work (Brunschwig et al. 2021), where we envisioned domain-specific languages with AR syntax synthesised from high-level descriptions and rendered within mobile devices. While that paper only considered the abstract syntax of languages and their 2D/3D representation, now we support different kinds of anchors for the virtual objects in the real world, the recognition of barcodes and QR codes, and the interaction with external APIs and information systems. This paper also expands our incipient work in (Campos-López et al. 2021) by providing a comprehensive presentation of our proposal and tool, an editor – called ALTER designer – to define AR application specifications, support for video content and BLE beacons, a user study that shows the usability of the AR applications created with our tool, and a catalogue of case studies that illustrate both the range of AR applications that can be constructed and the effort reduction that our model-based solution entails.

The remainder of this paper is organised as follows. First, Section 2 overviews our approach. Next, Section 3 details our model-driven solution for specifying AR applications, and Section 4 describes our tool support over iOS devices. Section 5 evaluates the approach on the basis of five case studies and a user study. Finally, Section 6 analyses related research and Section 7 ends with the conclusions and open research lines.

2. Approach

Figure 2 shows a schema of our proposal. It involves two roles: the AR application designer, and the AR application users.

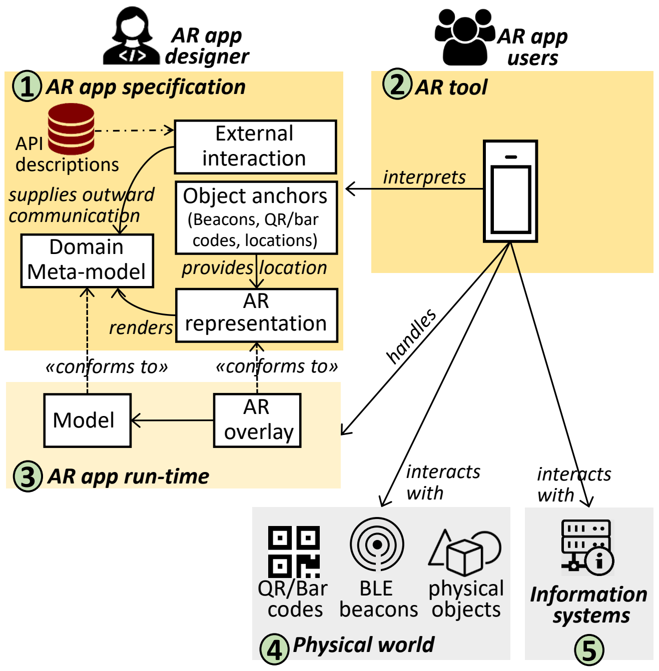


Figure 2 Schema of our approach.

Firstly, the AR application designer specifies the target AR application. We follow a model-driven engineering approach whereby the specification of an AR application consists of a domain meta-model (Brambilla et al. 2017) enriched with complementary models describing different aspects of the application (label 1 in Figure 2). The domain meta-model is a class diagram that contains the relevant concepts within the domain, along with their properties and relations. This meta-model is annotated with a model defining the AR representation of the meta-model elements. Each AR representation comprises a 3D object, its configuration (e.g., size, rotation) and its possible interactions (e.g., movements, overlapping). Another model specifies the anchoring mechanism to overlay the virtual objects on the real world. This can be based on a physical location, QR/barcodes, or BLE beacons. The last part of the specification describes how the AR application interacts with existing information systems and external devices via their APIs. For this purpose, a model defines which events in the AR application invoke, retrieve or send information from/to these APIs. Section 3 explains in more detail all these models.

Instead of monolithic specifications, our AR applications follow the principles of separation of concerns (Parnas 1972). This way, the specifications are split into several models which become more cohesive (i.e., they deal with just one concern of the application), reusable (e.g., external interaction definitions may serve different domain models), and flexible (e.g., a domain model can be assigned several AR representations), overall facilitating system evolution (e.g., changes in the AR representation meta-model do not impact existing domain models). As Section 3 will show, we relate the different (meta-)models by cross-references (i.e., references from an element in one (meta-)model to an element in another (meta-)model).

Once the AR application has been specified, it becomes immediately available to users. Our approach revolves around an AR application interpreter (label 2 in Figure 2) that implements the notion of models@runtime (Blair et al. 2009). This tool interprets the given AR specification, instantiating the domain elements and creating the specified AR overlays (label 3). The tool interacts with the physical world for placing the virtual objects as specified in the anchoring model (i.e., associated to QR/barcodes, BLE beacons, or in suitable locations, label 4). We use a notion of AR based on camera-enabled mobile devices, in particular from the iOS family (iPhones and iPads). This way, the virtual objects are visualised together with the physical ones in the screen of the mobile device, and the tool uses the camera as a canvas, as depicted in Figure 3.

The tool also captures events of interest (e.g., user events, time events), which may trigger exchanges with the specified information systems and devices (label 5 in Figure 2). The application state is a model, which can be saved and retrieved accordingly, and be updated at runtime. Section 4 will provide more information about the runtime behaviour of AR applications as supported by our AR tool, including the user interaction.

3. Model-Based AR Specifications

In the following, we describe the models used to specify AR applications (label 1 in Figure 2). Figures 4 to 8 show relevant excerpts of the meta-models to which these models must conform. They cover different concerns of the AR applications, specifically: the domain virtual concepts to be displayed (Section 3.1), their AR representation (Section 3.2), their anchors to real-world positions (Section 3.3), and the event-based interaction of the virtual objects with external systems (Section 3.5) via the external systems' exposed APIs (Section 3.4). These concerns together enable modelling all characteristics that, according to (Azuma 1997), AR systems must support: representation of virtual objects next to real objects, accurate positioning of virtual objects in the real world, and interaction in real-time.

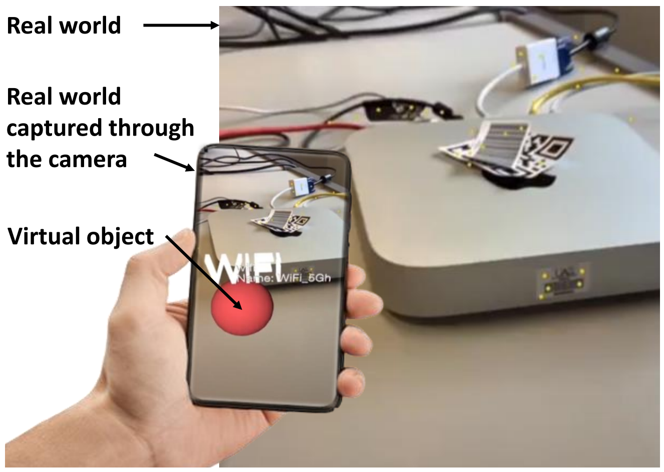


Figure 3 Using the AR application (sketch).

3.1. Domain meta-model

We provide a simple meta-modelling language to describe the domain meta-models (cf. Figure 4). It is based on the OMG's meta-object facility (MOF 2016), and permits describing domain concepts via classes with attributes and references. In addition, it supports special attribute data types for the AR domain, like Image and Video (see enumeration DataType).

Just like in MOF, classes have a name, a (possibly empty) set of superclasses, and may be abstract. They may also need to define a *key* attribute, so that the instances of a class can be identified uniquely at the model level (e.g., when presented in menus and data input forms). The attribute autoID permits configuring whether the tool must generate this identifier automatically when instantiating a class, or the user must explicitly provide a value for it.

For convenience, classes DomainMM and Class define some derived attributes that facilitate the technical realisation of the behaviour of the AR application. Specifically, /bluetooth indicates whether the objects of a class are to be anchored to BLE beacons; /onlyCodes if they are to be anchored just to QR/barcodes; and /noSCN if they have no virtual representation; whereas /externalAPI specifies whether the application needs to access external systems while the user interacts with the model; and /firebase states if the model uses non-local multimedia content, which we store in the Firebase cloud (https://firebase.google.com/). The next subsections introduce

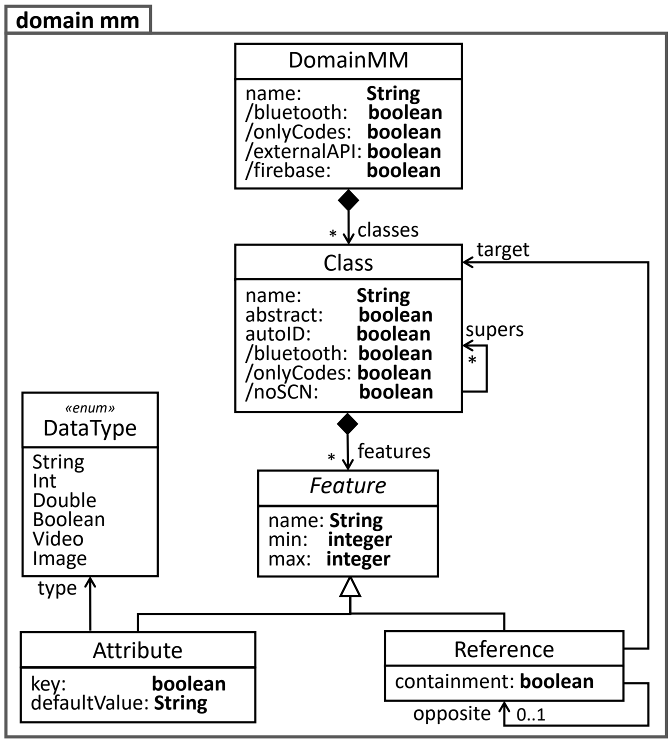


Figure 4 Meta-model for specifying the application domain.

the other meta-models of our proposal, from which the value of these attributes is derived.

3.2. AR representation

The AR representation of the domain meta-model is defined according to the meta-model in Figure 5. This permits assigning an ARNode representation to domain classes, and an ARConnection representation to domain references.

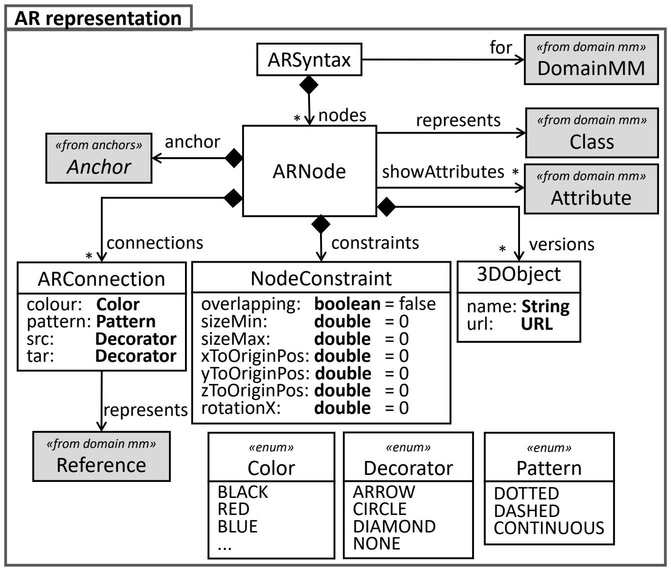


Figure 5 Meta-model for specifying the AR representation.

ARNodes point to zero, one or more 3DObject files containing the actual representation. 3D objects are stored at a URL, and implementation-wise, they are expected to conform to SceneKit's SCN format (Apple Inc.). Assigning several 3D objects to a class enables the end users to change between the different representations dynamically while interacting with the final AR application. Classes with no AR representation are allowed, in which case, the creation, update and deletion of their instances is performed through traditional menus. In addition, ARNodes can display the value of a set of attributes of the domain class (reference showAttributes).

The interaction of users with a given ARNode is specified in class NodeConstraint, and can be customised by means of the following attributes: overlapping controls whether two AR nodes can intersect with each other; sizeMin and sizeMax specify the maximum decrease or increase allowed in the node default size (if they are set to zero, the node cannot be resized); xToOriginPos, yToOriginPos and zToOriginPos establish the perimeter where an AR node can be moved once positioned in the world (zero to disallow movements); and rotationX applies an initial rotation (through the axis perpendicular to its position plane) on the node. The next subsection will explain how to establish and restrict the allowed positions of ARNodes using anchors.

ARConnections assign an AR line[1] representation to references. The colour, line pattern (dotted, dashed or continuous) and decorators on both line ends (arrow, circle, diamond or none) are customisable. At runtime, the position of ARConnections is determined by the source and target ARNodes, so they do not require anchoring mechanisms.

3.3. Anchors

The meta-model in Figure 6 permits assigning anchors to each ARNode. An anchor describes a real-world position where it is allowed to place an AR node.

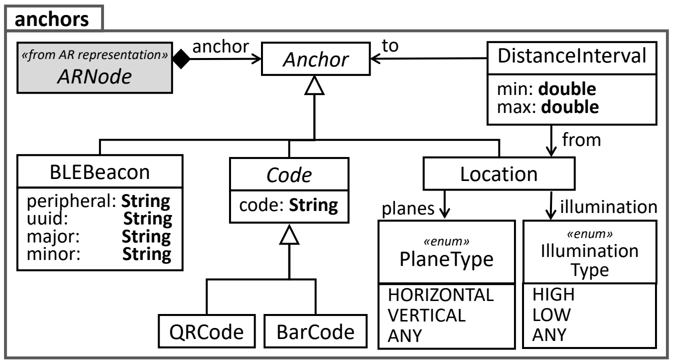


Figure 6 Meta-model for specifying anchors for virtual objects.

The meta-model supports three types of anchors: physical Location, Code, and BLEBeacon. In the first case, the location of an AR node can be restricted to be just on a horizontal or vertical physical plane, to be on a spot with a given light intensity (attribute illumination), as well as to be within a certain

[1] More precisely, a cylinder, which is a 3D line.

distance interval of the anchor of another AR node with same or different type (class DistanceInterval).

In the case of using a QR code or a barcode as the anchor, the AR interpreter will create the AR node when the camera of the device scans the specified code.

Finally, if using BLE beacons as anchors, the AR node will appear when the Bluetooth sensor detects the defined beacon. The identity of each beacon must be registered in the AR application by providing a value for its attributes peripheral, uuid, major and minor.

3.4. API descriptions

To enable the interaction of the defined AR applications with external services, our approach needs to have access to the description of the services' interface. Several approaches for API description exist, like OpenApi (OpenAPI specification). However, to keep our AR specifications as simple as possible, we have decided to create a much simpler description meta-model for the definition of REST-based APIs.

Figure 7 shows our API description meta-model, whereby an APIDescription has a name, a protocol, a base url, a collection of Paths and, optionally, authentication data (APIAuth). Each path needs to declare an HTTP command (e.g., GET, POST), the required input Parameters, and the expected OutputParameters.

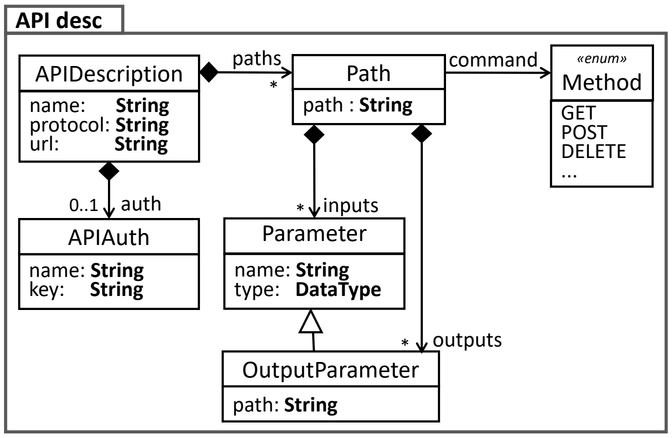


Figure 7 Meta-model for specifying REST-based APIs.

3.5. External interaction

The last meta-model, shown in Figure 8, allows configuring the invocation of the defined API paths (cf. Section 3.4) upon certain events. The triggering events can be operations performed on domain objects (e.g., their creation, change or deletion), on their AR representations (e.g., their selection), on the complete model (e.g., loading or saving the model), or they can also be time triggers that invoke an API recurrently. The latter is useful if the API serves dynamic information, like in the case of the fabrication process of smart factories; a traffic service that provides real-time information about road conditions, jams or car accidents; or a weather API. A model may define any number of triggers for the invocation of the same or different REST APIs.

When a trigger fires, the value of the Input parameters to the associated API call can be either the value of an object attribute

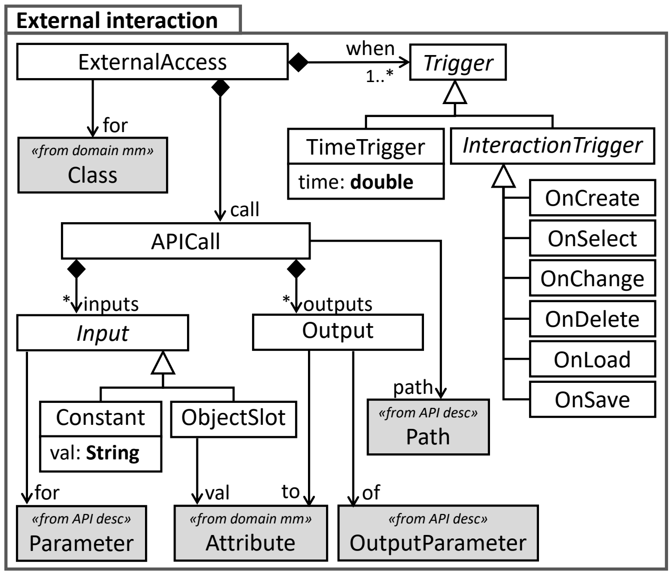


Figure 8 Meta-model for specifying the event-based interaction of AR applications with external systems.

(class ObjectSlot) or a constant (class Constant). The Output of an API call can be stored in object attributes.

4. Architecture and Tool Support

We have created a tool, called ALTER, which is an interpreter of model-based AR applications specified with our approach. The tool is freely available at the App Store (https://apps.apple.com/us/app/alter-ar/id1574875872). More information, including demonstration videos, can be found at https://alter-ar.github.io/.

In the following, Section 4.1 presents the tool architecture, Section 4.2 introduces our editor of AR applications, and Section 4.3 gives details on the client side.

4.1. Architecture

The tool has a client-server architecture, as depicted in Figure 9.

The AR application designer specifies the models describing the AR application (domain meta-model, AR representation, anchors, API description and external interaction) using the ALTER designer (see Section 4.2). These models are stored in a server that uses *mongoDB* and *Node.js* technology. The server includes an API service broker that delivers the necessary API calls to the external information systems. A *Firebase storage* stores the multimedia that the AR applications use, and permits loading and saving new videos and images. For the definition of APIs, we currently assume JSON for the output parameters, and use JSONPath (JSONPath) to specify how to retrieve the output parameters from the JSON document.

The client is a native iOS app by which users can run the defined AR applications. Section 4.3 describes this client.

4.2. ALTER designer

To facilitate the definition of the AR application models, we provide the tool ALTER designer. This is an Eclipse plugin that enables the AR application designer to edit and validate the

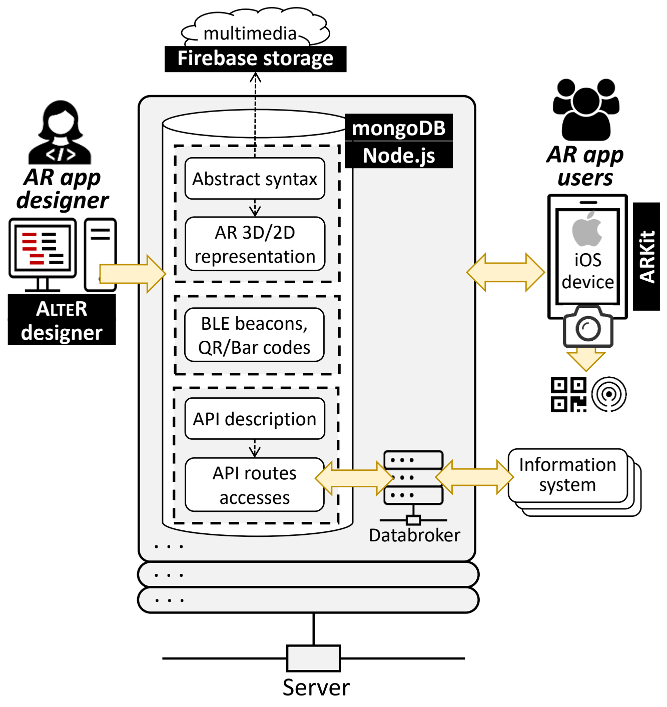
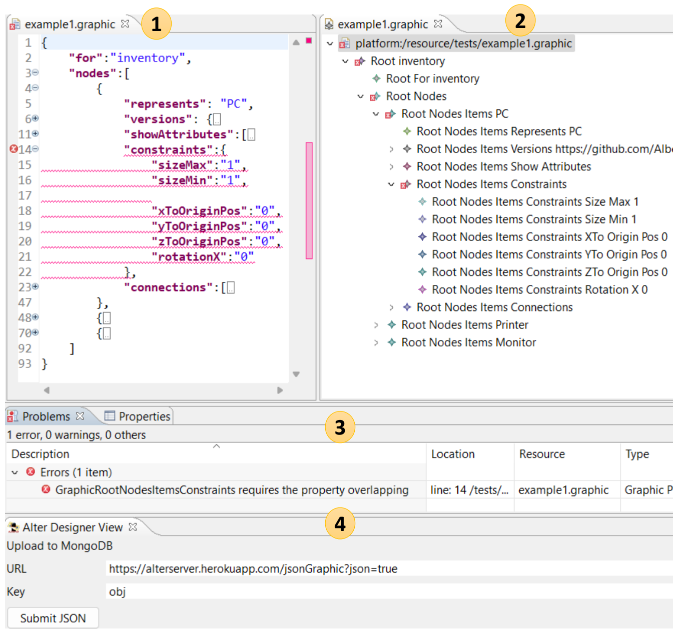


Figure 9 Architecture of ALTER.



example1.graphic

```
{
    "for":"inventory",
    "nodes":[
        {
            "represents": "PC",
            "versions": {..
            "showAttributes":[..
            "constraints":{
                "sizeMax":"1",
                "sizeMin":"1",

                "xToOriginPos":"0",
                "yToOriginPos":"0",
                "zToOriginPos":"0",
                "rotationX":"0"
            },
            "connections":[..
        },
        {..
        {..
    ]
}
```

example1.graphic

- platform:/resource/tests/example1.graphic
  - Root inventory
    - Root For inventory
    - Root Nodes
      - Root Nodes Items PC
        - Root Nodes Items Represents PC
        - Root Nodes Items Versions https://github.com/Alb
        - Root Nodes Items Show Attributes
        - Root Nodes Items Constraints
          - Root Nodes Items Constraints Size Max 1
          - Root Nodes Items Constraints Size Min 1
          - Root Nodes Items Constraints XTo Origin Pos 0
          - Root Nodes Items Constraints YTo Origin Pos 0
          - Root Nodes Items Constraints ZTo Origin Pos 0
          - Root Nodes Items Constraints Rotation X 0
        - Root Nodes Items Connections
      - Root Nodes Items Printer
      - Root Nodes Items Monitor

Problems | Properties

1 error, 0 warnings, 0 others

Description	Location	Resource	Type
Errors (1 item)			
GraphicRootNodesItemsConstraints requires the property overlapping	line: 14 /tests/...	example1.graphic	Graphic P

Alter Designer View

Upload to MongoDB

URL https://alterserver.herokuapp.com/jsonGraphic?json=true

Key obj

Submit JSON

Figure 10 JSON editor of AR representation models in ALTER designer.

AR models, represented as JSON documents. We use JSON Schema (JSON Schema) to define the structure of the JSON documents as, according to (Colantoni et al. 2021), the JSON Schema initiative is the major approach to validate JSON documents. We used an existing DSL for JSON Schema (JSONSchemaDSL) (Colantoni et al. 2021) to create the schemas that ALTER designer needs to validate the JSON documents (i.e., the models). JSONSchemaDSL allows defining valid schemas based on JSON Schema, and generates a fully-fledged textual language editor based on Xtext (Bettini 2016).

Figure 10 shows the environment of AR representation models in action, using the AR representation meta-model shown in Figure 5. The editor features are syntax highlighting, content assistance, validation and parsing. On the left side of the figure (label 1), we can see the failing in-line validation for the keyword "constraints" as it requires the definition of the property "overlapping" that is missing, as reported in the *Properties* view (label 3). The error message is also displayed when passing the mouse over the text underlined in red. The right part of the figure contains the same JSON document parsed in the abstract syntax tree (label 2).

Once the models have been defined and validated using ALTER designer, they can be uploaded to our dedicated server, becoming available to the client of ALTER. The upload can be automatically performed using the ALTER Designer View (cf. Figure 10, label 4), which just requires providing the request URL and the data API key.

4.3. Client

The client (i.e., the interpreter of model-based AR applications) is a native iOS app that uses the ARKit library to handle the AR overlay on the device with the camera. It communicates with the server to retrieve the AR application descriptions, to save and load models (i.e., application states), and to access the external systems via the service broker.

When opening the tool, the user of ALTER can either load an existing model, or select an AR application specification and start a new model from scratch. Next, the main user interface of the tool is presented overlaid on the device camera. Figure 11 shows a screenshot of ALTER running on an iPad mini.

ALTER displays a toolbar at the bottom (label 1), with buttons to create the domain objects defined by the selected AR application. In the figure, the toolbar enables the creation of objects of a home network (routers, computers, smartphones and the Wi-Fi), and the camera shows the AR overlay for a router and a Wi-Fi object (represented by 3D objects together with some attributes of the router and Wi-Fi). The 3D objects can only be created in the specified anchors. In this case, the router needs to be in a horizontal plane, and the Wi-Fi is required to be close to the router.

Once an object is created using the toolbar, the user can perform the following actions:

– Resize, rotate and change the position of the AR object, if permitted by the specification. As label 2 in the figure hints, ALTER recognises gestures on the screen for the following actions: *pan* to move the object to another position in the world; *rotate*; *pinch* to resize; and *swipe* to change the visualisation of the object if the specification defines several ones.
– Edit its attributes (label 3) and references according to the domain meta-model. Pressing the *edit* button (label 4) opens a table view with the value of all features of the selected object, which can be edited using a virtual keyboard. As an example, Figure 12 shows the editing of the features

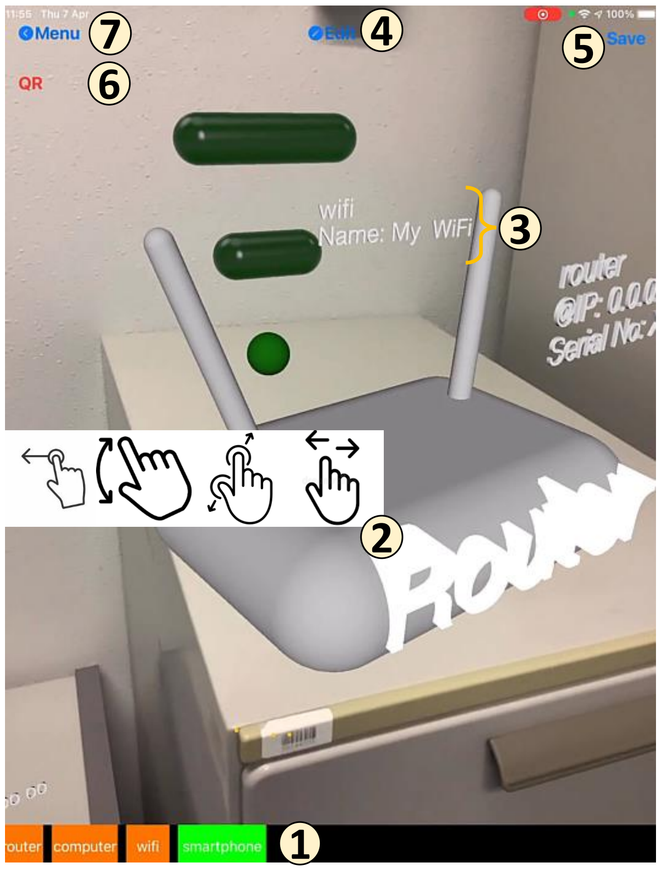
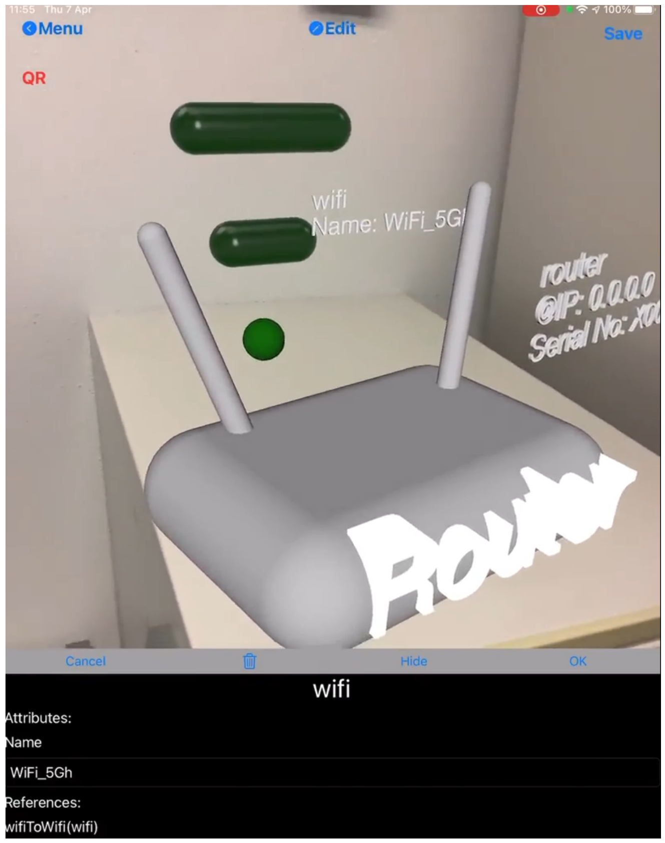


Figure 11 ALTER user interface.



wifi

Attributes:

Name

WiFi_5Gh

References:

wifiToWifi(wifi)

Figure 12 Editing features of objects within ALTER.

of the Wi-Fi object (attribute name and reference wifiToWifi). This view also enables deleting the object using the trash icon. To give value to a given reference, the system shows a menu that presents every compatible object in the model. Upon selecting one object in this menu, the tool displays the connection in the canvas.

– Save the model (the current AR application state) on the device (label 5 in Figure 11). Pressing the *Save* button records the virtual map and takes a photo of the current physical location. The screenshot will be displayed as a hint when the user wants to load the same model. This allows recovering the application state in the same location where it was saved, since some 3D objects may have been anchored in particular locations.
  Figure 13 illustrates a scenario in which the user wants to recover a previous model. For this purpose, the camera needs to focus close to the position where the model objects were last anchored, and so, ALTER shows the photo of the last location in the top-right corner as a hint.

The *QR* button (label 6 in Figure 11) enables or disables the scanning of QR/barcodes using ALTER. If enabled, scanning a QR/barcode may trigger the creation of an AR object associated to the code, as given by the specification. This way, ALTER supports three ways to create AR objects: by means of the toolbar as explained above, by scanning a registered QR/barcode which has an associated AR object, or when the mobile device is close to a registered BLE beacon with an associated AR object.

Finally, the *Menu* button (label 7 in Figure 11) closes the current AR application.

5. Evaluation

Next, we evaluate the range of AR applications that can be effectively defined with our approach. For this, we use five case studies, each illustrating different features of ALTER with increasing level of sophistication. Our goal is to answer the following research question (RQ):

RQ1 *What's the typical specification size of AR applications built with* ALTER*?*

In addition, we are interested in evaluating the resulting AR applications, to answer the following RQ:

RQ2 *What's the user perception of AR applications built with* ALTER*?*

To answer this second question, we conducted a user study with 11 participants who evaluated one of the applications built in the case studies.

5.1. RQ1: Case studies

Next, we report on the five case studies (subsections 5.1.1 to 5.1.5) and answer RQ1 (subsection 5.1.6).

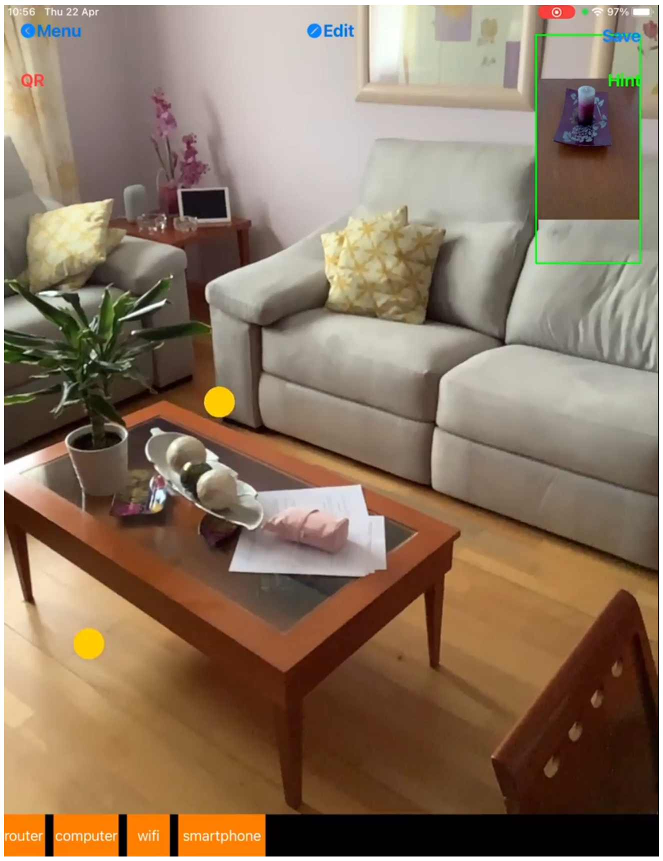


Figure 13 Recovering a saved app state.

5.1.1. Home networking Our first case study showcases the use of ALTER as a domain-specific modelling environment supporting models with an AR concrete syntax. Specifically, it is a CRUD application to model home networks using AR. A video showing the use case is available at https://youtu.be/4oFyIcNT-x0, and Figure 14 presents a screenshot (Figures 11–13 are also part of this case study).

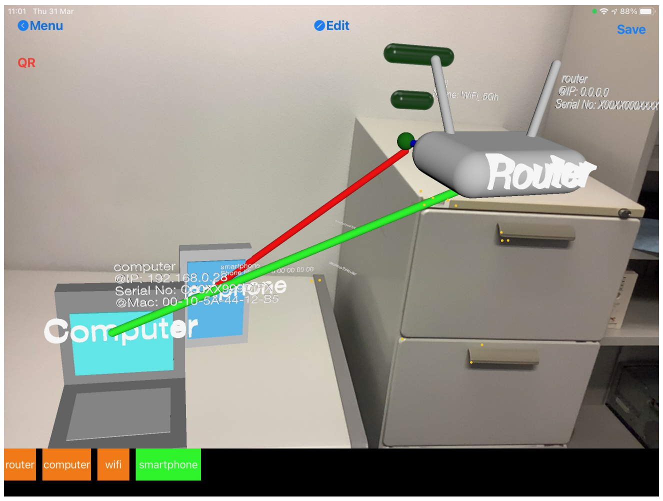


Figure 14 AR networking modelling environment.

The application specification considers devices like router, smartphone, computer and Wi-Fi. They can be inter-connected as defined in the domain model shown in Figure 15. The figure shows an instance of the meta-model in Figure 4 using a concrete syntax that resembles standard meta-models. Classes and references are decorated with icons depicting their AR representation, and attributes are marked with an *eye* icon if they are displayed on the AR application canvas.

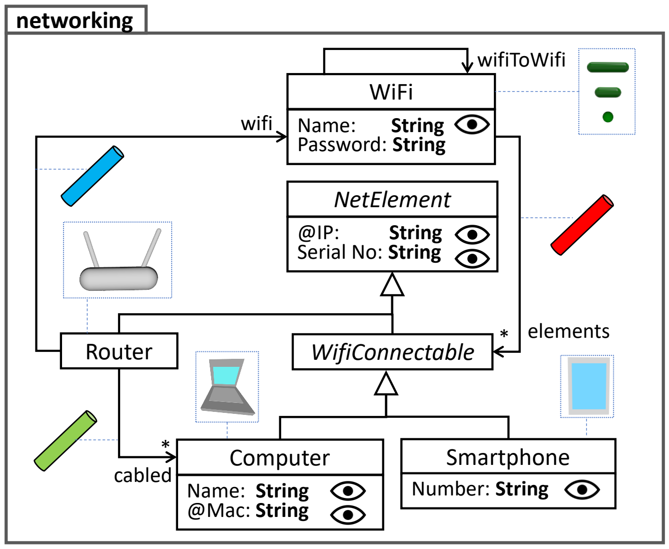


Figure 15 Domain model for the networking case study, annotated with AR representation.

The palette at the bottom of Figure 14 permits creating the devices. All can be anchored in any horizontal plane except the Wi-Fi, which can be free floating anywhere close to the router. The 3D objects display and allow editing the attributes of the represented domain objects, like the IP, MAC address, and serial number. This application is useful as a way to record the configuration of a network, where the AR objects may be placed in correspondence with the real objects they represent.

5.1.2. Museum The goal of our second application is to enhance the user experience in a museum by augmenting the exhibited items with AR overlays that display additional information (videos) about the items on demand. A video of the application is available at https://youtu.be/qV6fZDXJTFE, and Figure 16 shows a screenshot.

This application uses QR codes stuck to the museum items as anchoring mechanism for the AR objects. In this case, the toolbar is empty because the user does not create the AR objects by drag-and-drop, but the tool creates them automatically when the corresponding QR code is scanned. In the figure, the museum has historical computers in exhibition, and the AR objects associated to the QR codes display a video with explanations about the particular computer models. The AR object is a representation of a museum guide. Since one of its attributes is of type Video, the AR object displays AR buttons to play and pause the video. These videos are stored in *Firebase*.

5.1.3. Augmented home The third case study is an application to augment a home with virtual furniture (tables, chairs,

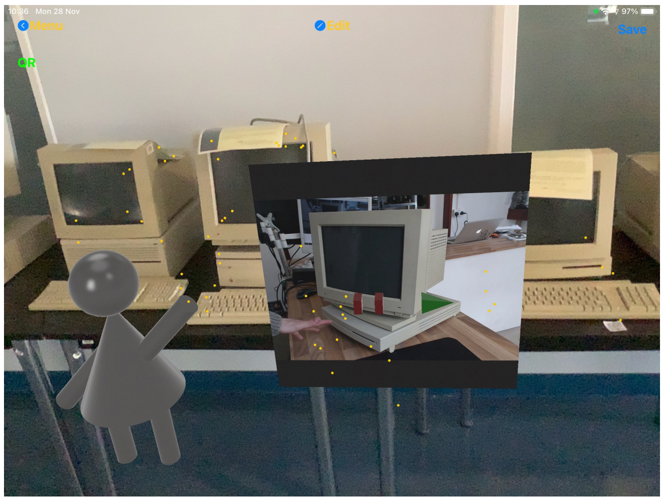


Figure 16 Enhancing a museum's visitor experience with AR.

sofas), decorations (lamps, vases), virtual pictures to be hanged on the walls, and a weather station which displays weather information about the current location. There is a video showcasing this application at https://youtu.be/Mob5sHp4pHo, and a screenshot in Figure 17.

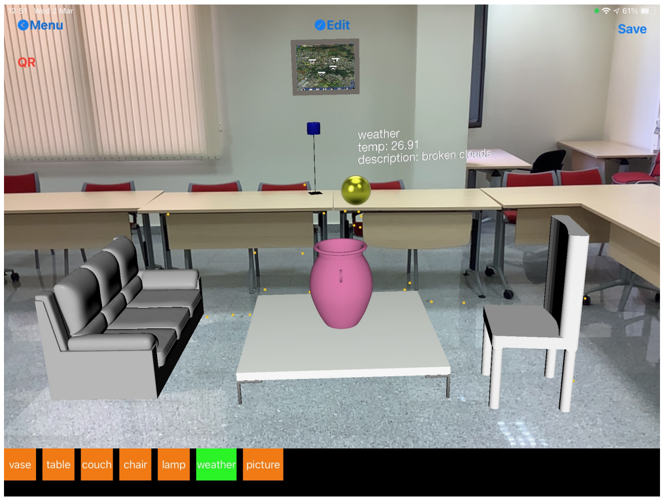


Figure 17 Augmented home.

The toolbar at the bottom enables placing the virtual objects overlaid in the room where the user is located. We have set some constraints so that pictures can only be placed on vertical planes (walls), the other objects need to be placed in horizontal planes (i.e., the ground, or on top of other virtual or physical objects), and chairs must be close to tables. Only pictures are editable, to select or change the image in display. This image can be taken from the mobile device, and then, the tool stores it in our *Firebase storage* server. The weather station has a time trigger that regularly invokes the OpenWeather API (https://openweathermap.org/api), stores the result in the attributes of the station, and displays the returned weather information.

Each domain object (except pictures and weather stations) has three AR representations assigned (e.g., different types of chairs or tables), which can be chosen by swiping on the AR object. Figure 18 shows how users may change between two chair representations. This way, users can design more personalised decorations, and furniture vendors can present the catalogue of articles they sell, as in (IKEA room planner). Moreover, users may try how different (virtual) furniture fits in a given room with existing physical furniture. Finally, our augmented home application supports a model-based AR ecosystem, where users can upload and share the home decorations they have created.

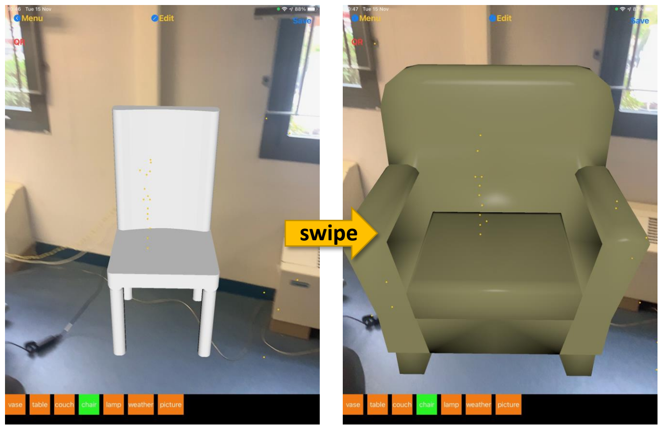


Figure 18 Changing between different types of chairs.

5.1.4. Inventory This case study illustrates how to create an AR interface for an existing information system. This case is an improvement of the one presented in (Campos-López et al. 2021), which targets an information system for an inventory of computers, monitors and printers. However, it can be easily adapted to other domains (e.g., hospitals or factories). A video showcasing the application is available at https://youtu.be/UPfQomkFYG4, and Figure 19 shows a screenshot.

First, we physically tag the real objects in the inventory using QR and barcodes. This way, the tool presents an AR overlay with information of the inventoried elements (e.g., serial number, IP and MAC address, computer model, monitor size, etc.) when their QR/barcode is scanned. The information is retrieved from an external information system using API calls, which are triggered when the AR object is created (when its QR/barcode is scanned). The outputs of the call are stored in the attributes of the domain object, as well as displayed by the AR object. The flow of information is bidirectional, since the user can modify or delete the data of the object, which triggers an API call sending the changes to the information system. In our case, we are only interested in displaying the objects' attributes, and therefore, the 3D representation of the objects is empty. The visualisation also displays graphically which pieces of equipment belong together (e.g., the monitors of a computer) using AR connections. For inventory purposes, this is useful to quickly locate the parts of a group of jointly registered items.

Technically, ALTER enables registering new QR/barcodes by using the camera to scan the code, and selecting a class (e.g., printer, computer, monitor) and an identifier by means of the

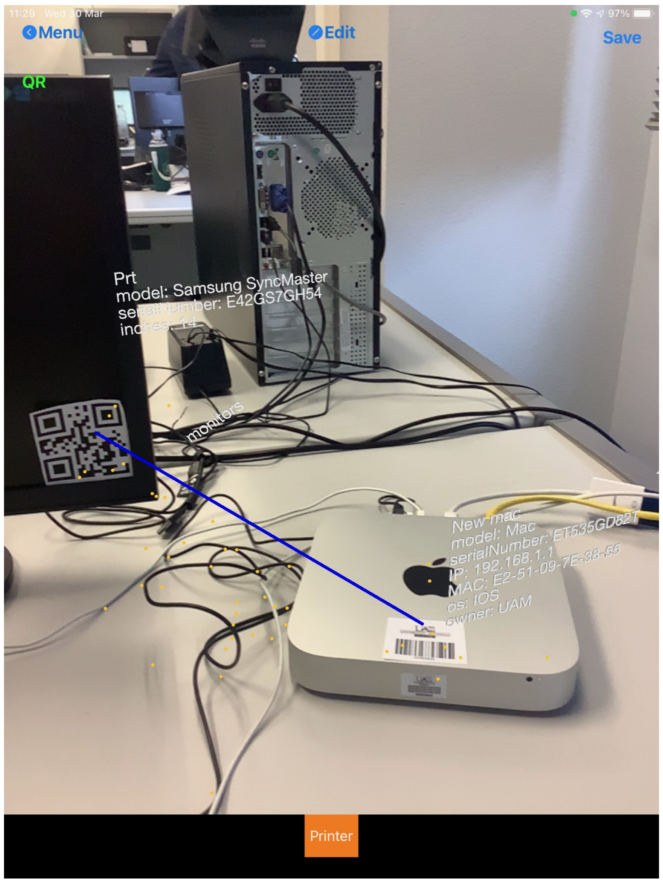
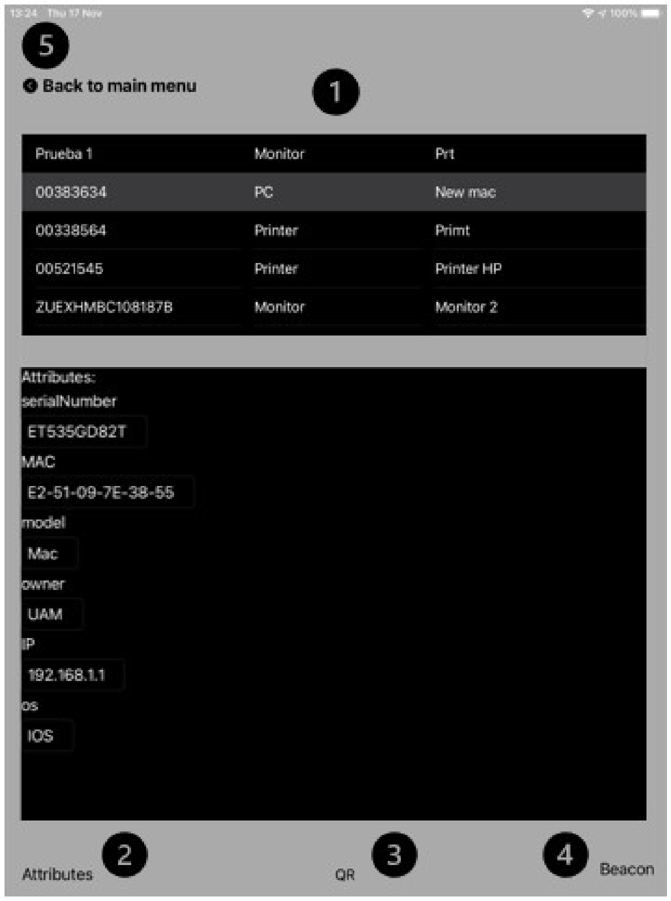


Figure 19 AR interface for an inventory information system.



Prueba 1	Monitor	Prt
00383634	PC	New mac
00338564	Printer	Primt
00521545	Printer	Printer HP
ZUEXHMBC108187B	Monitor	Monitor 2

Attributes:
serialNumber
ET535GD82T
MAC
E2-51-09-7E-38-55
model
Mac
owner
UAM
IP
192.168.1.1
os
IOS

Figure 20 Registering QR/barcodes in ALTER.

configuration menu in Figure 20. The menu shows the list of QR/barcodes that are already registered, displaying the code identifier, class and object identifier (label 1). Tapping on a QR/barcode of the list enables the editing of its attributes.

The configuration menu has three buttons at the bottom: *Attributes* (label 2) allows modifying the attributes of the selected object as explained in Section 4.3; *QR* (label 3) displays a camera view to scan a new QR/barcode and register it (cf. Figure 21); and *Beacon* (label 4) displays a form to register BLE beacons, as Section 5.1.5 will illustrate. The application disables the buttons not supported by the current model (e.g., registering BLE beacons if the specification of the current AR application does not include them). Finally, the *Back to main menu* button (label 5) closes the configuration menu.

5.1.5. AR-based social network In this final case study, we have developed an AR-based messaging system where users can send and receive multimedia messages through the use of BLE beacons. Hence, the app increases the physicality of a style of application (a social network) which traditionally has been purely virtual. There is a demonstration video at https://youtu.be/jAMf-2ZyYjE, and a screenshot in Figure 22. The figure shows a beacon, and two virtual messages, one replying to the other.

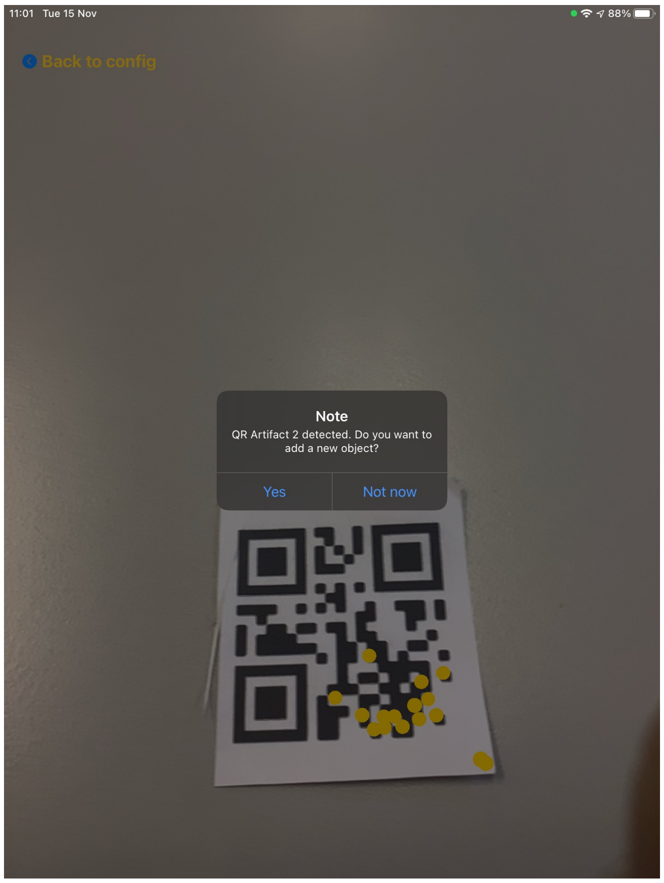
In this application, beacons are hot spots where displaying AR messages from given users. Messages are stored in a traditional information system, which provides a REST API to retrieve, create and update messages from the AR application. This API is called whenever AR messages are created or updated. Messages are multimedia and may include text, images and videos. The creation of messages can be local (when close to a registered beacon) or remote (using the information system). Moreover, previous messages are displayed when users are close to the registered beacons. The application also supports replies to messages, which are displayed as connected messages. The user profile of the message author is a (non-AR) object, which can be edited using the button in the toolbar.

BLE beacons need to be registered before they can be used within a model. Like in the inventory case study (cf. Section 5.1.4), ALTER permits registering new beacons by providing their data (peripheral, UUID, major, minor) and mapping them to classes in the domain meta-model (message in our case). Figure 23 shows the configuration menu for registering new BLE beacons in ALTER.

Altogether, the AR application converts BLE beacons into virtual notice boards, where multimedia messages and replies can be posted. The envisioned usage scenarios include social networks for organisations, where BLE beacons can be distributed across offices; for shops, where clients can provide opinions on-site on beacon-tagged items; or in restaurants and bars, as a way to share customer experiences, through videos.

	Domain meta-model			AR representation				Anchors		External
Case study	**#Classes**	**#Attribs**	**#Refs**	**#ARNodes**	**#ARConns**	**#Versions**	**Firebase**	**Beacons**	**Codes**	**#API calls**
Home networking	6	7	3	4	3	4	No	No	No	0
Museum	1	3	0	1	0	1	Yes	No	Yes	3
Augmented home	7	5	0	7	0	17	No	No	No	1
Inventory	3	14	2	1	2	3	No	No	Yes	9
Social network	1	6	1	1	1	1	Yes	Yes	No	3

Table 1 Summary of the specifications required to create the case studies.

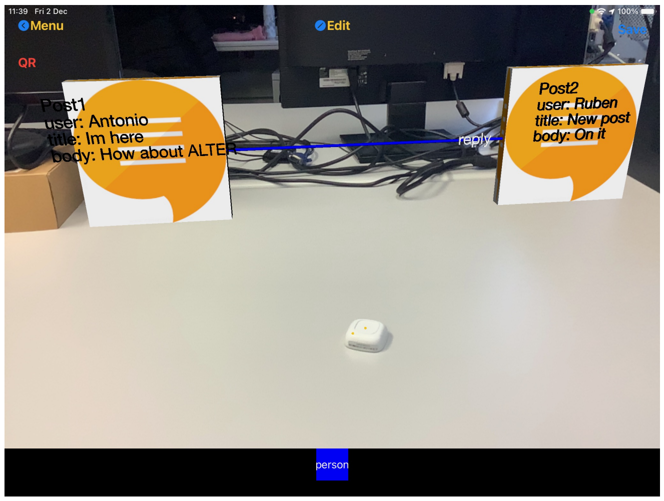


Note

QR Artifact 2 detected. Do you want to add a new object?

Yes Not now

Figure 21 Scanning a new QR code for its registration.



Post1
user: Antonio
title: Im here
body: How about ALTER

Post2
user: Ruben
title: New post
body: On it

Figure 22 AR-based messaging via BLE beacons.

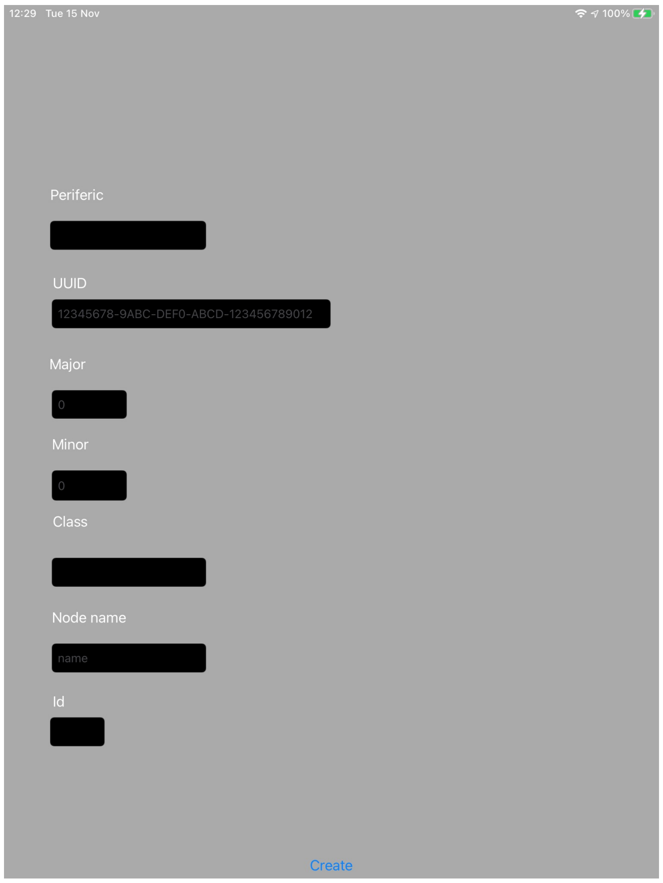
5.1.6. Discussion and answering RQ1 Table 1 summarises the size of the specifications required to create each case study. After the name of the case study, the next three columns show the size of the domain meta-model in terms of the number of classes, attributes and references. The following four columns display some size metrics of the AR representation model, specifically, the number of ARNodes, ARConnections, versions of 3DObjects, and whether there are multimedia elements (images, videos) that require *Firebase storage*. The following two columns describe whether the cases use anchor mechanisms based on BLE beacons or QR/barcodes. Finally, the last column shows the external access interactions, measured as the number of different API calls.

We can observe that the case studies cover all functionality offered by the tool. The size of the specifications is relatively small: the biggest domain meta-model (Inventory) has 19 elements, while the smallest one (Museum) has just 4. This shows that the specification effort is rather low. Actually, we were able to build the models for each case study within a couple of hours each. However, in our experience, most effort is dedicated to finding or creating 3D objects (17 in the Augmented home case study) or multimedia contents (videos in the Museum case). It should be stressed that creating these case studies did not require a single line of code.

As a limitation, our approach does not currently support dynamic virtual objects. These would be needed in applications – like games – where virtual objects are controlled via the mobile device, e.g., using the accelerometer or the gyroscope. Moreover, our approach does not support interaction between the virtual and the real-world objects (e.g., overlay a virtual button on top of a real button to allow concurrent registration in both the virtual and the real world). Extending our approach to support these aspects is left as future work.

5.2. RQ2: User study

We have conducted a user study to understand the perception of AR applications built with ALTER by potential users. After a short presentation and demo of ALTER, the participants had to use the augmented home application (https://youtu.be/Mob5sHp4pHo) to perform typical tasks: create, move and resize some AR elements; change their version and attributes; and store and load a model. All participants were provided



12:29 Tue 15 Nov 100%

Periferic

UUID

12345678-9ABC-DEF0-ABCD-123456789012

Major

0

Minor

0

Class

Node name

name

Id

Create

Figure 23 Registering BLE beacons with ALTER.

with an iPad mini to perform the tasks. After the experiment, participants filled in a System Usability Scale (SUS) questionnaire (Brooke et al. 1996), which is a standard means widely used to measure the usability of software tools.

Overall, the study involved 11 participants with an average of 33.5 years old (ranging between 24 and 47). Most of them had a computer science background (60%) and were male (60%). We asked them their level of use of mobile devices and AR applications, from 1–scarce to 5–intensive. Generally, participants used mobile devices intensively (average 4.5) and AR applications moderately (average of 2.4).

The SUS questionnaire consists of ten questions, each with five response options from *Strongly disagree* to *Strongly agree* (a Likert-scale from 1 to 5). Five of the questions have a positive tone and their best score is 5, while the other five have a negative tone and their best score is 1. Overall, ALTER obtained a SUS score of 79.3% which, according to (Bangor et al. 2009), can be qualified as *Good* (i.e., range [71.4–85.5)), close to be *Excellent* ($\geq$85.5). The anonymised raw data of the SUS questionnaires is available at https://alter-ar.github.io/ecmfa-eval.

We also included some open-ended questions asking participants to list three positive and three negative aspects of the tool, as well as improvement suggestions. Participants valued that the app was easy to use (9 out of 11) and the use case was useful to furnish a house. However, they noticed that placing AR elements in the desired position might be a bit tricky, and reported small glitches when loading a model. Concerning improvements, some participants mentioned the possibility to preview the position of the AR objects when being created, or the recommendation of possible actions for objects. We will incorporate these suggestions in future versions of ALTER.

Altogether, we can answer RQ2 positively: participants had a good perception of the tool, with a SUS of 79.3%. However, they noticed a few issues and suggested improvements.

5.3. Threats to validity

Next, we analyse the threats to the validity of our evaluations.

External validity refers to the extent to which the results can be generalized. Regarding RQ1, we used five case studies built by ourselves. While the specification size of these cases was moderate, other more complex case studies might require larger specifications. Regarding RQ2, the generalization of the results is restricted by the low number of participants and their somewhat uniform background. A replica of the user study involving more participants with a wider range of ages and backgrounds, and considering more use cases, would provide stronger evidence.

Construct validity is concerned with the relationship between the theory of the experiment and what is observed. Regarding RQ1, we measured specification size, however substantial effort is required also to prepare the AR objects and the additional multimedia materials.

Internal validity concerns confounding factors that may affect the results of the study. In particular, we only used the augmented home application for the user study designed to answer RQ2. However, using other case studies may yield different results. For uniformity, all participants used an iPad mini to perform the study. Using other devices, either smaller, like iPhones, or bigger, like iPads, may result in a different user experience.

6. Related Work

In this section, we review AR development tools for their comparison with our proposal (Section 6.1). Since our work can be used to create AR-based domain-specific modelling environments, we also analyse works exploiting XR for software engineering and modelling (Section 6.2).

6.1. AR development tools

Many tools have been proposed for the development of AR applications, like DART (MacIntyre et al. 2005) or ARtoolkit (Kato & Billinghurst 2004). Some of them, like AR.js[2] and Argon4[3], follow a web development style and supply special browsers to display the AR representations independently of the device. Widely known platforms like Unity AR Foundations[4], MAXST AR SDK[5], Wikitude AR SDK[6] or Vuforia Studio[7] offer dedicated environments for building multi-platform AR applications. At low level, some programming libraries for AR are

[2] https://ar-js-org.github.io/AR.js-Docs/
[3] https://app.argonjs.io/
[4] https://unity.com/unity/features/ar
[5] https://maxst.com/en/#/en/arsdk
[6] https://www.wikitude.com/
[7] https://www.ptc.com/en/products/vuforia

device-specific (e.g., ARKit for iOS devices), while others are multi-platform (e.g., ARCore). Finally, some approaches, like ProtoAR (Nebeling et al. 2018), focus on the creation of 3D content and its placement over markers. While these are all powerful, generic tools to create AR apps, they require deep expertise on AR technologies. Instead, our proposal enables the creation of AR applications without resorting to coding.

Like us, some authors recognise that building AR applications with most current authoring tools is complex and time-consuming (Ashtari et al. 2020; Nebeling & Speicher 2018), so various means to simplify their development have been proposed. For instance, tools like SimpleAR (Apaza-Yllachura et al. 2019) and AR Scratch (Radu & MacIntyre 2009) use block-based visual programming to construct AR apps. These tools are directed to children and non-technical users, so their focus is on the AR representation, lacking other advances concepts such as external interaction. The authoring tool presented in (Rau et al. 2022) provides small, self-contained, ready-to-use AR applications (called AR nuggets) which authors can execute, test and customise without requiring programming knowledge. AR nuggets include objects with parameters, which are configurable via buttons and other graphical elements. While this tool is very suitable for laypersons, it supports a close set of predefined AR applications and restricted customisation capabilities. In the commercial side, Simplifier[8] is a low-code tool for enterprise applications, which has been recently extended with AR integration (via HoloLens). This permits, e.g., providing an AR interface to SAP applications. We can integrate AR with existing information systems (non-SAP), and since we focus on mobile-based AR, we can target applications like the AR-based modelling environment in Section 5.1.1.

Another way to facilitate the development of AR applications is by targeting specific domains. For instance, ZeusAR (Marín-Vega et al. 2022) defines an architecture and a process to automate the generation of serious games based on AR, ExposAR (Lunding et al. 2022) is an authoring tool for the creation of AR games by children, and there are also many proposals oriented to education, like VEDILS (Mota et al. 2018) (see (Dengel et al. 2022) for a recent survey). In contrast, our proposal is domain-agnostic so it can be applied to any domain.

More generally, Börsting and collaborators (Börsting et al. 2022) identify open questions and challenges in all development phases of AR applications. Specifically for the software modelling phase, they find diagram-based modelling (e.g., state diagrams) to be a useful vehicle for understanding the AR application. However, they deem models as static documents, while we propose to leverage them to be the AR application itself.

Overall, our proposal is novel in that it is model-based, centred around a domain meta-model. This permits a succinct description of the concepts of the application domain and their connection with external information systems and devices, like BLE beacons, without the need for coding. Moreover, different from all the analysed works, it follows a models@runtime approach, meaning that the AR application itself is a model which can be modified at runtime to adapt its operation.

[8] https://simplifier.io

6.2. Extended reality for software modelling

Since the last decade, there have been proposals to use XR to tackle software engineering tasks. For example, Elliot and collaborators (Elliott et al. 2015) argue that VR environments leverage on *affordances* of natural human perception, like visual cognition, spatial memory, motion, manipulation and feedback. Since traditional environments do not provide support for them, they propose VR environments for live coding of VR scenarios and code review. Mehra and collaborators (Mehra et al. 2020) also propose immersive IDEs, arguing that (configurable) virtual workplaces may induce positive emotions and avoid distractions. Beyond code development, immersive VR systems have been used for visualising software as cities via metrics (Romano et al. 2019; Moreno-Lumbreras et al. 2023) and for software comprehension (Mehra et al. 2019).

Sharma and collaborators (Sharma et al. 2018) claim that the new generations of software engineers may expect different interaction styles when building software, and their familiarity with VR environments may make XR solutions appropriate. Hence, they propose an AR system based on smart glasses to help project managers track the progress of developers without disrupting their work.

Closer to our approach, XR has been applied to software modelling as well. For instance, VmodlR (Yigitbas et al. 2021) is a collaborative UML modelling environment in VR. It is built for the Oculus Quest VR headset, and supports a speech-based chat. A user study with students reported an increment of motivation and a more natural interaction than with traditional modelling editors. Targeting education, VmodlR was later added gamification elements (e.g., shooting at incorrect parts of a UML model) (Yigitbas et al. 2022). Another example is HoloFlows (Seiger et al. 2021), which exploits AR and a head-mounted device for creating IoT processes by "drawing" connections between devices. Instead of smart glasses, interaction in our approach is via mobile devices, and is not specific to IoT. While HoloFlows was programmed from scratch, a framework like ALTER could have been used for its construction.

In previous work, we experimented with a position-based modelling approach (Sebastian-Lombraña et al. 2020) using mobile devices. In that approach, the placement of elements within a model could be assigned via BLE beacons or using the GPS of the device. However, AR was not supported.

ALTER can be used to create AR-based modelling environments for DSLs. In this respect, Devil3D (Wolter 2012) can generate environments for 3D visual modelling languages, but it runs on desktops and does not support AR.

Overall, to the best of our knowledge, domain-specific modelling based on AR is a novel contribution of our approach.

7. Conclusions

In this paper, we have presented a model-driven approach to build AR applications, which eliminates the need for coding. The approach is based on the construction of a domain meta-model, decorated with models specifying the AR representation of the domain concepts, their anchoring, and their interaction with external devices and information systems. We have illustrated the versatility of our supporting tool ALTER on five case studies, and evaluated its usability with a user study.

We are currently working to incorporate a configurable physics model to ALTER specifications. This will allow adding dynamics to the AR objects, enabling the design of a wider range of AR applications, like games. We would also like to extend object behaviour with conditional styles, to change their appearance dynamically, or even make them appear/disappear. At the moment, our AR applications are for single users, so we would like to extend ALTER to support collaborative AR applications. We plan to build a dedicated web-based low-code environment to facilitate the creation of AR application specifications. This would enable synchronous collaboration (Google Docs-like) to specify AR applications. Our intention is to integrate this environment with facilities to create and store 3D object models, like the one proposed in (Jovanovikj et al. 2020). Finally, we are planning a user study from the point of view of AR application designers.

Acknowledgements

Work partially supported by the Spanish Ministry of Science (PID2021-122270OB-I00) and the R&D programme of Madrid (P2018/TCS-4314). We would like to thank the reviewers for their useful comments, and the participants of the user study for their availability.

References

Apaza-Yllachura, Y., Valderrama, A. P., & Delgado, C. C. (2019). SimpleAR: Augmented reality high-level content design framework using visual programming. In *SCCC* (pp. 1–7). IEEE.

Apple Inc. (Last access on 2022). *SceneKit.* https://developer.apple.com/documentation/scenekit.

Ashtari, N., Bunt, A., McGrenere, J., Nebeling, M., & Chilana, P. K. (2020). Creating augmented and virtual reality applications: Current practices, challenges, and opportunities. In *CHI* (pp. 1–13). ACM.

Azuma, R. T. (1997). A survey of augmented reality. *Presence Teleoperators Virtual Environ.*, *6*(4), 355–385.

Bangor, A., Kortum, P., & Miller, J. (2009). Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *J. Usability Studies*, *4*(3), 114–123.

Bettini, L. (2016). *Implementing domain-specific languages with xtext and xtend.* Packt Publishing Ltd.

Blair, G. S., Bencomo, N., & France, R. B. (2009). Models@run.time. *IEEE Computer*, *42*(10), 22–27.

Börsting, I., Heikamp, M., Hesenius, M., Koop, W., & Gruhn, V. (2022). Software engineering for augmented reality - A research agenda. *Proc. ACM Hum. Comput. Interact.*, *6*(EICS), 155:1–155:34.

Brambilla, M., Cabot, J., & Wimmer, M. (2017). *Model-driven software engineering in practice, 2^{nd} edition.* USA: Morgan & Claypool.

Brooke, J., et al. (1996). SUS-a quick and dirty usability scale. *Usability evaluation in industry*, *189*(194), 4–7.

Brunschwig, L., Campos-López, R., Guerra, E., & de Lara, J. (2021). Towards domain-specific modelling environments based on augmented reality. In *ICSE-NIER* (pp. 56–60). IEEE.

Böhm, F., Dietz, M., Preindl, T., & Pernul, G. (2021). Augmented reality and the digital twin: State-of-the-art and perspectives for cybersecurity. *Journal of Cybersecurity and Privacy*, *1*(3), 519–538.

Campos, A., Correia, N., Romão, T., Nunes, I. L., & Simões-Marques, M. (2019). Mobile augmented reality techniques for emergency response. In *Mobiquitous* (pp. 31–39). ACM.

Campos-López, R., Guerra, E., & de Lara, J. (2021). Towards automating the construction of augmented reality interfaces for information systems. In *ISD.* Universitat Politècnica de València / Association for Information Systems.

Colantoni, A., Garmendia, A., Berardinelli, L., Wimmer, M., & Bräuer, J. (2021). Leveraging model-driven technologies for JSON artefacts: The shipyard case study. In *MODELS* (pp. 250–260). IEEE.

Coscetti, S., Moroni, D., Pieri, G., & Tampucci, M. (2020). Factory maintenance application using augmented reality. In *APPIS* (pp. 22:1–22:6). ACM.

Dengel, A., Iqbal, M. Z., Grafe, S., & Mangina, E. (2022). A review on augmented reality authoring toolkits for education. *Frontiers Virtual Real.*, *In press*.

Elliott, A., Peiris, B., & Parnin, C. (2015). Virtual reality in software engineering: Affordances, applications, and challenges. In *ICSE - Volume 2* (pp. 547–550). IEEE Press.

Google maps live view. (Last access on 2022). https://blog.google/products/maps/new-sense-direction-live-view/.

Heinz, M., Büttner, S., & Röcker, C. (2019). Exploring training modes for industrial augmented reality learning. In *PETRA* (pp. 398–401). ACM.

Hincapié, M., Diaz, C., Valencia, A., Contero, M., & Güemes-Castorena, D. (2021). Educational applications of augmented reality: A bibliometric study. *Comput. Electr. Eng.*, *93*, 107289.

IKEA room planner. (Last access on 2022). https://apps.apple.com/us/app/ikea-place/id1279244498.

Jovanovikj, I., Yigitbas, E., Sauer, S., & Engels, G. (2020). Augmented and virtual reality object repository for rapid prototyping. In *HCSE* (Vol. 12481, pp. 216–224). Springer.

JSON Schema. (Last access on 2022-11-24). http://json-schema.org/.

JSONPath. (Last access on 2022). https://jsonpath.com/.

Kato, H., & Billinghurst, M. (2004). Developing AR applications with ARToolKit. In *ISMAR* (p. 305).

Ling, H. (2017). Augmented reality in reality. *IEEE Multim.*, *24*(3), 10–15.

Lunding, M. S., Grønbæk, J. E. S., Bilstrup, K. K., Sørensen, M. S. K., & Petersen, M. G. (2022). ExposAR: Bringing augmented reality to the computational thinking agenda through a collaborative authoring tool. In *CHI* (pp. 131:1–131:14). ACM.

MacIntyre, B., Gandy, M., Dow, S., & Bolter, J. D. (2005). DART: A toolkit for rapid design exploration of augmented reality experiences. *ACM Trans. Graph.*, *24*(3), 932.

Marín-Vega, H., Alor-Hernández, G., Colombo-Mendoza, L. O., Bustos-López, M., & Zataraín-Cabada, R. (2022). ZeusAR: A process and an architecture to automate the development of augmented reality serious games. *Multim. Tools Appl.*, *81*(2), 2901–2935.

Mehra, R., Sharma, V. S., Kaulgud, V., & Podder, S. (2019). XRaSE: Towards virtually tangible software using augmented reality. In *ASE* (pp. 1194–1197). IEEE.

Mehra, R., Sharma, V. S., Kaulgud, V., Podder, S., & Burden, A. P. (2020). Immersive IDE: towards leveraging virtual reality for creating an immersive software development environment. In *ICSE Workshops* (pp. 177–180). ACM.

Metaverse. (Last access on 2022). https://www.meta.com/.

Milgram, P., & Kishino, F. (1994). A taxonomy of mixed reality visual displays. *IEICE Transactions on Information and Systems*, *E77-D*(12), 1321–1329.

MOF. (2016). http://www.omg.org/spec/MOF.

Moreno-Lumbreras, D., Minelli, R., Villaverde, A., González-Barahona, J. M., & Lanza, M. (2023). Codecity: A comparison of on-screen and virtual reality. *Inf. Softw. Technol.*, *153*, 107064.

Mota, J. M., Ruiz-Rube, I., Dodero, J. M., & Sánchez, I. A. (2018). Augmented reality mobile app development for all. *Comput. Electr. Eng.*, *65*, 250–260.

Nebeling, M., Nebeling, J., Yu, A., & Rumble, R. (2018). ProtoAR: Rapid physical-digital prototyping of mobile augmented reality applications. In *CHI* (p. 353). ACM.

Nebeling, M., & Speicher, M. (2018). The trouble with augmented reality/virtual reality authoring tools. In *ISMAR* (pp. 333–337). IEEE.

OpenAPI specification. (Last access on 2022). https://www.openapis.org/.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM*, *15*(12), 1053–1058.

Pierdicca, R., Frontoni, E., Zingaretti, P., Mancini, A., Loncarski, J., & Paolanti, M. (2019). Design, large-scale usage testing, and important metrics for augmented reality gaming applications. *ACM Trans. Multimedia Comput. Commun. Appl.*, *15*(2).

Radu, I., & MacIntyre, B. (2009). Augmented-reality scratch: A children's authoring environment for augmented-reality experiences. In *IDC* (pp. 210–213). ACM.

Rau, L., Döring, D. C., Horst, R., & Dörner, R. (2022). Pattern-based augmented reality authoring using different degrees of immersion: A learning nugget approach. *Frontiers Virtual Real.*, *3*, 841066.

Romano, S., Capece, N., Erra, U., Scanniello, G., & Lanza, M. (2019). On the use of virtual reality in software visualization: The case of the city metaphor. *Inf. Softw. Technol.*, *114*, 92–106.

Schäfer, A., Reis, G., & Stricker, D. (2022). A survey on synchronous augmented, virtual and mixed reality remote collaboration systems. *ACM Comput. Surv.*. (In press) doi: 10.1145/3533376

Sebastian-Lombraña, A., Guerra, E., & de Lara, J. (2020). Positioning-based domain-specific modelling through mobile devices. In *SEAA* (pp. 150–157).

Seiger, R., Gohlke, M., & Aßmann, U. (2019). Augmented reality-based process modelling for the internet of things with HoloFlows. In *BPMDS* (Vol. 352, pp. 115–129). Springer.

Seiger, R., Kühn, R., Korzetz, M., & Aßmann, U. (2021). Holoflows: modelling of processes for the internet of things in mixed reality. *Softw. Syst. Model.*, *20*(5), 1465–1489.

Sharma, V. S., Mehra, R., Kaulgud, V., & Podder, S. (2018). An immersive future for software engineering: avenues and approaches. In *ICSE (NIER)* (pp. 105–108). ACM.

Snap AR. (Last access on 2022). https://ar.snap.com/.

Wolter, J. (2012). DEViL3D - A generator framework for three-dimensional visual languages. In *DMS* (pp. 171–176). Knowledge Systems Institute.

Yigitbas, E., Gorissen, S., Weidmann, N., & Engels, G. (2021). Collaborative software modeling in virtual reality. In *MODELS* (pp. 261–272). IEEE.

Yigitbas, E., Schmidt, M., Bucchiarone, A., Gottschalk, S., & Engels, G. (2022). Gamification-based UML learning environment in virtual reality. In *MODELS (Companion Proceedings)* (pp. 27–31). ACM.

About the authors

Rubén Campos-López is a researcher at the modelling & software engineering research lab of the Universidad Autónoma de Madrid, Spain. His research interests include augmented reality, mobile development and model-driven engineering. You can contact the author at Ruben.Campos@uam.es.

Esther Guerra is Professor at the Universidad Autónoma de Madrid, where she leads the modelling & software engineering research lab (http://www.miso.es) together with J. de Lara. She is interested in model-driven engineering, flexible modelling, meta-modelling, domain-specific languages and model transformation. You can contact the author at Esther.Guerra@uam.es.

Juan de Lara is Full Professor at the Computer Science Department of the Universidad Autónoma de Madrid, Spain. Together with E. Guerra, he leads the modelling & software engineering research lab. His interests are in automated software engineering, model-driven development, chatbots and augmented reality. You can contact the author at Juan.deLara@uam.es.

Alessandro Colantoni is a PhD student at the SE department of Johannes Kepler University of Linz, Austria. His research interests include DevOps, model-driven engineering and low-code engineering platforms. You can contact the author at Alessandro.Colantoni@jku.at.

Antonio Garmendia is Assistant Professor at the Universidad Autónoma de Madrid, Spain. Before he was a postdoctoral researcher at the WIN-SE department, JKU Linz. His research interests are in scalability in model-driven engineering and the construction of graphical modelling environments. You can contact the author at Antonio.Garmendia@uam.es.