Towards Automating the Construction of Augmented Reality Interfaces for Information Systems

Rubén Campos-López
Universidad Autónoma de Madrid
Madrid, Spain *Ruben.Campos@uam.es*

Esther Guerra
Universidad Autónoma de Madrid
Madrid, Spain *Esther.Guerra@uam.es*

Juan de Lara
Universidad Autónoma de Madrid
Madrid, Spain *Juan.deLara@uam.es*

Abstract

Information systems typically provide a graphical user interface running on the desktop, web clients or mobile devices. However, we are witnessing a steady increase in the capabilities of mobile phones and tablets, and the proposal of sophisticated head-mounted devices. All these devices enable the construction of augmented reality (AR) applications, where virtual objects are overlayed over real ones using the device camera. AR-based user interfaces can be beneficial for many applications in domotics, smart factories, emergency management, or transportation, to name a few. However, they are difficult to build as they require specialized knowledge.
To alleviate this problem, we propose a model-driven engineering approach to automate the construction of AR-based user interfaces for information systems accessible via REST APIs. The feasibility of the proposal is demonstrated on a prototype implementation atop iOS/ARKit, and a case study of an inventory information system.

Keywords: Augmented Reality, Information Systems, REST APIs, Model-Driven Engineering.

1. Introduction

According to recent studies [5], a large percentage (48.3%) of the world population owns a smartphone. The capabilities of these devices are rapidly increasing in terms of computing power and the variety of embedded sensors and components they are equipped with (e.g., camera, GPS, microphone, accelerometer, gyroscope, lidar). This has triggered the possibility of novel ways of interaction, for example based on augmented reality (AR) [4].

AR opens the door to new ways to interact with information systems. By blending reality and virtual objects, AR allows displaying virtual information close to the object of interest, as well as manipulating real objects via their virtual digital twin. This interaction modality may be suitable for applications in areas like smart factories (to display real-time data about the factory's machines and processes on-site) [10], Internet of Things (to configure and connect devices) [23], interior design (to overlay virtual furniture on a real room) [13], transportation (to show directions overlayed with the reality) [12], emergency management (to overlay relevant changing information on the emergency scenario) [8], and museums (to augment the information about the exhibited art works), among others [7]. However, building AR-based applications is time consuming and requires specialized, highly technical knowledge.

To facilitate the creation of AR-based user interfaces for information systems, we propose to automate their construction by the application of software language engineering and model-

driven engineering principles [6]. Our approach assumes an information system available via a REST API. Then, the software designer needs to build a domain meta-model with the concepts managed by the information system. Each concept can be annotated with the following information: (i) its AR visualization; (ii) the mechanism for anchoring its virtual occurrences, either based on their real-world position, or on the recognition of barcodes and quick response (QR) codes; and (iii) the API calls to be performed upon certain user interactions (e.g., creating, updating or deleting an occurrence of the concept) or events (e.g., time triggers).

We demonstrate the feasibility of our approach by making available a prototype for iOS devices (iPhones, iPads) which is able to render AR-based user interfaces out of concept meta-models so annotated. Moreover, we illustrate the approach and the tool using an inventory application as a running example. The next section introduces this running example.

2. Running Example

As a running example, we are using a simple inventory information system, inspired by the one at our University. The inventory needs to tag different computer equipment (CPUs, monitors, printers) using barcodes and QRs, and record data about each element. The current system has the drawback that the PC technician cannot retrieve easily the data associated to a barcode when he/she is on-site (e.g., when checking the barcodes in a lab). In addition, the equipment changes places frequently, making it difficult to trace the monitors and printers that belonged to each CPU originally in the inventory.

To solve these issues, we are creating an AR-based user interface that will: (a) display the inventory information overlayed on the physical equipment upon reading its barcode, (b) enter data of the equipment whose barcode is not yet in the system, (c) display AR connections between each CPU and its associated monitors and printers. Fig. 3(a) shows a screenshot of the targeted AR interface, and the next section describes our approach to automate its synthesis.

3. Approach

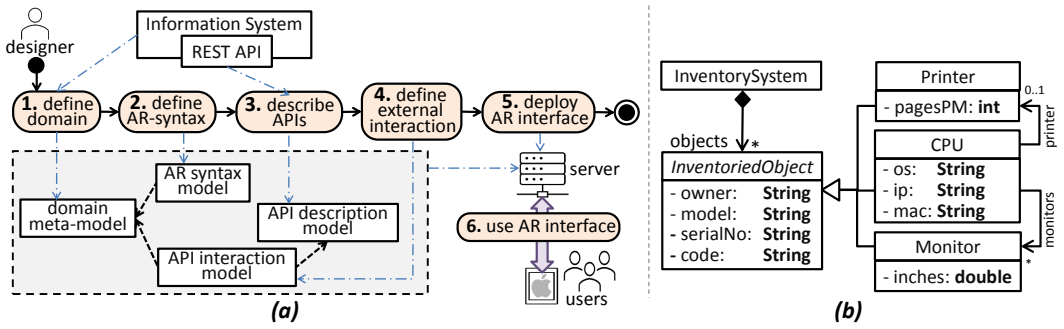

Fig. 1. (a) Overview of our approach. (b) Meta-model of the inventory system.

Fig. 1(a) gives an overview of our approach. It is model-based [6], as the designer uses models to describe the domain concepts, every aspect of the AR interface, and the interaction with the existing information system. First, the software designer creates a meta-model of the domain (step 1). This is a structural model (a class diagram) declaring the concepts tackled by the information system, together with their properties and relations. Then, the designer annotates the meta-model elements to specify the AR syntax used to represent them (step 2). Next, he/she builds a model-based description of the information system's API (step 3). There might be one or several APIs. Next, the designer defines when (e.g., on creating a virtual object) and how (i.e., where the data values are taken from) the API will be invoked (step 4). Finally, the designer uploads all defined artefacts in a server (step 5). At this point, end-users can use an iOS

tool that we have developed to interact with the information system and visualize it according to the defined AR-based syntax (step 6).

In the following, we detail the different steps on the basis of the running example.

1. Defining the domain. In the first step of our method to create an AR interface, the designer needs to describe the domain by means of a meta-model [6]. Fig. 1(b) shows the meta-model for the inventory system. It contains InventoriedObjects of three types: CPUs, Printers and Monitors. All of them have an owner, a model, a serial number, and a code that physically corresponds to a barcode sticked to the real, physical object. CPUs have an operating system (os), an IP and a MAC address, and refer to their connected Printer (if any) and Monitors. Printers specify their speed in pages per minute, and Monitors indicate their size in inches.

2. Defining the AR syntax. The next step is to design the AR representation of the elements in the domain meta-model. For this purpose, we have created a meta-model based on the proposal in [7]. This meta-model permits annotating the domain classes with 2D and 3D objects, and the domain associations with AR lines having different styles and decorators. A class may define several representations, which the user can change at run-time. We also allow tagging the objects of a domain class with QR codes and barcodes, and selecting the attributes to display in the AR syntax. The way to create virtual objects is also customizable, either by selecting their type from a palette (cf. lower part of Fig. 3(a)), or using QR and barcodes exclusively.

In our running example, CPU, Printer and Monitor objects are to show the value of all their attributes but the code, and Printers and Monitors will not show their owner either. The associations will be represented as solid blue lines decorated with the association name. Finally, objects will be created via QR and barcodes, and for illustration purposes, it will be possible to create Printer virtual objects using a palette button (cf. Fig. 3(a)).

3. Describing the APIs. To enable the interaction between the AR interface and the information system, the designer needs to describe the API of the latter. While several approaches for API description exist, like OpenApi [21], we have created a much simpler description model, shown in Fig. 2(a). In this model, an APIDescription has a name, a protocol, a base URL, optionally an authentication mechanism, and resource Paths with input and output Parameters. Implementation-wise, we currently assume JSON for the output parameters, and use JSON-Path [14] to specify how to retrieve the output parameters from the JSON document.

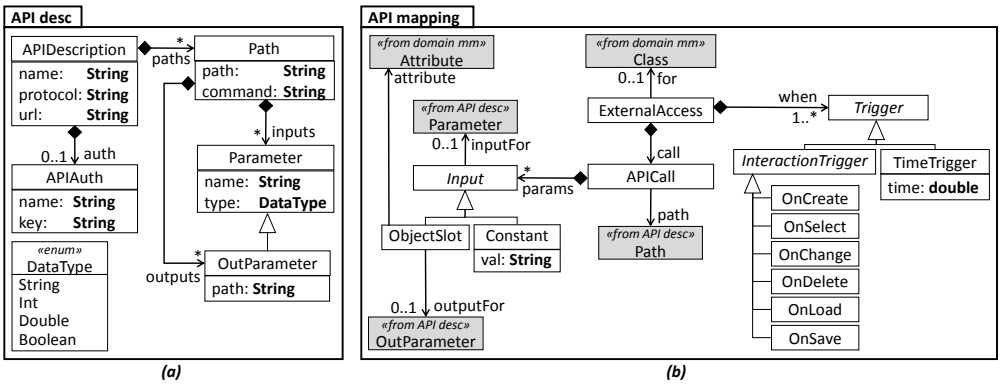

Fig. 2. Meta-models for: (a) API description; (b) Mapping actions in the AR interface to API calls.

The inventory API defines nine paths, one for each element type (CPU, Monitor, and Printer) and command (GET, POST, and DELETE). Command GET expects the code of an element as input parameter, and returns the element's information as output. POST receives the data of an element, and updates the database with the received data. DELETE expects an element's code, and deletes the element from the database.

4. Defining the external interaction. As the next step, the designer can map actions in the AR

interface to API calls. For this purpose, he/she uses the meta-model of Fig. 2(b). Specifically, the definition of an ExternalAccess includes: (a) the source of the Input data for the API call, which can be either a Constant value, or the value of an ObjectSlot (e.g., the content of the code slot of the CPU object selected in the AR interface); (b) the ObjectSlot where the output of the call is to be stored (e.g., in slot os of the selected CPU); and (c) the Triggers of the API call. In Fig. 2(b), the possible triggers are given by the subclasses of Trigger. If the trigger is OnCreate, the call is performed whenever an object of the specified class is created, either explicitly via a button in the palette, or via a QR code or barcode. OnSelect is activated upon the selection of a virtual element, and OnChange upon changing the value of its attributes. OnDelete triggers the API call when an object of the given class is deleted. While the previous triggers are local to a specific object, OnSave and OnLoad are global, and the API call is done when the model is saved and loaded, respectively. Finally, time triggers that invoke the API periodically are also possible. This is useful if the API serves dynamic information, like in the case of the fabrication process of smart factories.

In the case of the inventory, the GET commands would be called upon creating objects or loading the model, POST when there are object modifications or the model is saved, and DELETE whenever an object is deleted.

5, 6. Deploying and using the AR interface. As the last step, the designer needs to upload the defined artefacts to a server. To use the defined AR interface, we have created an iOS tool that allows listing the AR interface definitions uploaded to the server, selecting one of them, and using it to access the information system according to the defined interaction. The next section gives details of this tool, and a video is available at `https://youtu.be/UPfQomkFYG4`.

4. Tool Support

Based on the ideas in [7], we have created a prototype tool implementing the presented concepts, which is available at `https://alter-ar.github.io/`. It works for iOS devices (iPhones and iPads), and uses the Swift ARKit library [3] to handle AR on the devices. The tool overlays the virtual objects using the device camera. It has a client/server architecture, where the server stores the definition of the information systems and their AR interfaces (domain meta-model, AR representation, and API description and mapping models). The client can be used for different information systems just by loading different definition models. Upon the user selects a specific information system, our tool interprets its definition to provide the defined AR syntax and the external interaction with APIs. For the latter, the tool incorporates an API broker service that is in charge of connecting with the different APIs, as specified in the API description model.

Fig. 3(a) shows a screenshot of the tool, after loading the definition models for the inventory. The figure shows a CPU and a connected Monitor, together with their attributes. The bottom part of the tool includes a palette to create virtual objects. In the inventory, any kind of object can be created upon recognizing QR codes and barcodes, while for illustration, Printers can be created using the palette as well.

Fig. 3(b) shows the dialog to change the attribute values of an object and create references between objects. When the user confirms the changes, the OnChange event is triggered, which in our running example invokes the API of the inventory to update the information system. Saving and loading an AR model is challenging, since objects need to be recovered in the same physical place where they were saved. Fig. 3(c) shows our solution. On saving, the tool makes a photo of the location, which then is displayed as a hint (top-right of the figure) when the model is loaded.

5. Related Work

Our proposal to define user interfaces is model-based [18, 19]. Compared to the CAMELEON reference framework [9], our methodology permits modelling the domain, the abstract user

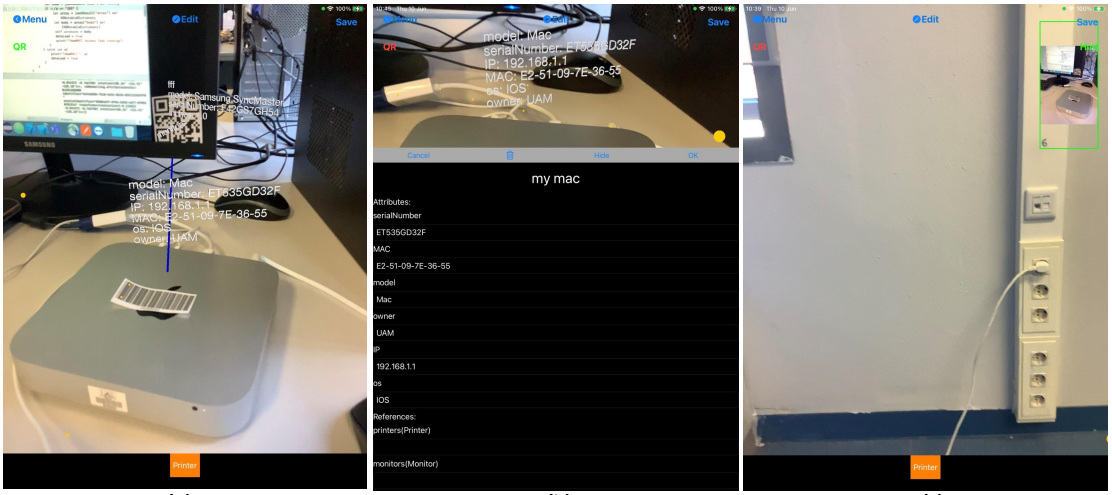

Fig. 3. (a) Screenshot of the AR-based user interface for an inventory information system. (b) Attributes of virtual object. (c) Persistence location hint.

interface model is fixed, and the designer can configure several representations for the domain elements. Our approach is heavily influenced by how graphical domain-specific languages are specified in model-driven engineering [6], [16]. Our task model is implicit, based on CRUD operations over virtual objects, which can attach API calls. While this is sufficient for simple information systems, our plan is to extend our approach towards an explicit task model [22].

Some approaches exist for the generation of 3D user interfaces for information systems [11]. Regarding the construction of AR applications, many development tools have been proposed, like DART [17] or ARtoolkit [15]. Some of them – like AR.js [2] or Argon4 [1] – offer special browsers to display the AR representations, and follow a web development style. Platforms like Unity AR Foundations [24], Wikitude AR SDK [26] or Vuforia Studio [25] offer dedicated environments for building multi-platform AR applications. At low level, some programming libraries for AR are device-specific (e.g., ARKit for iOS devices), while others are multi-platform (e.g., ARCore). Finally, some approaches, like ProtoAR [20], target the creation of 3D content and its placement over markers. While these are all generic tools to create AR applications or parts of them, our proposal enables the creation of AR-based user interfaces for information systems accessible via REST, without coding.

6. Conclusions and Future Work

In this paper, we have presented the first steps towards an approach to automate the construction of AR-based interfaces for information systems. The approach is based on model-driven and software language engineering principles. This permits the specification and realization of the AR interface by means of models, with no need for coding.

In the future, we would like to extend the tool to support objects with no virtual representation, but editable via the AR tool (e.g., in our example, to be able to describe Persons who own inventoried objects). We would also like to consider dynamic virtual objects, which goes beyond our current support for objects that change their values with a time trigger. Finally, even though our approach relies on an existing information system, we'd like to develop a modelling notation for a richer specification of tasks [22]. On the tool side we are working on more API protocols, and on a cloud-based environment for the specification of our AR description models.

Acknowledgements. This work has been funded by the Spanish Ministry of Science (RTI2018-095255-B-I00), and by the R&D programme of Madrid (P2018/TCS-4314).

References

1. Argon4, https://app.argonjs.io (last access on 2021).
2. AR.js, https://ar-js-org.github.io/AR.js-Docs/ (last access on 2021).
3. ARKit, https://developer.apple.com/augmented-reality/ (last access on 2021).
4. Azuma, R. T. A survey of augmented reality. *Presence Teleoperators Virtual Environ.*, vol. 6, no. 4, pp. 355–385, 1997.
5. bankmycell.com, https://www.bankmycell.com/blog/how-many-phones-are-in-the-world (last access on 2021).
6. Brambilla, M., Cabot, J., and Wimmer, M. Model-Driven Software Engineering in Practice, 2^{nd} Edition. USA: Morgan & Claypool (2017).
7. Brunschwig, L., Campos-López, R., Guerra, E., and de Lara, J. Towards domain-specific modelling environments based on augmented reality. In *ICSE-NIER* (2021) pp. 56–60.
8. Campos, A., Correia, N., Romão, T., Nunes, I. L., and Simões-Marques, M. Mobile augmented reality techniques for emergency response. In *MobiQuitous*. ACM (2019), pp. 31–39.
9. Cavalry, G., Coutaz, J., Thevenin, D., Bouillon, L., Florins, M., Limbourg, Q., Souchon, N., Vanderdonckt, J., Mauricci, L., Paternò, F., and Santoro, C. The CAMELEON reference framework. CAMELEON R&D Project IST-2000-30104, Tech. Rep. (2002).
10. Coscetti, S., Moroni, D., Pieri, G., and Tampucci, M. Factory maintenance application using augmented reality. In *APPIS*. ACM (2020) pp. 22:1–22:6.
11. González-Calleros, J. M., Vanderdonckt, J., and Arteaga, J. M. A method for developing 3D user interfaces of information systems. In *CADUI*. Springer (2006), pp. 85–100.
12. Google maps live view, https://blog.google/products/maps/new-sense-direction-live-view/ (last access on 2021).
13. IKEA room planner, https://apps.apple.com/us/app/ikea-place/id1279244498 (last access on 2021).
14. JSONPath, https://jsonpath.com/ (Last access on 2021).
15. Kato, H., and Billinghurst, M. Developing AR applications with ARToolKit. in *ISMAR* (2004), p. 305.
16. Kelly, S., and Tolvanen, J. Domain-specific modeling - Enabling full code generation. Wiley (2008).
17. MacIntyre, B., Gandy, M., Dow, S., and Bolter, J. D. DART: a toolkit for rapid design exploration of augmented reality experiences. *ACM Trans. Graph.*, vol. 24, no. 3, p. 932 (2005).
18. Meixner, G., Calvary, G., and Coutaz, J. Introduction to model-based user interfaces. https://www.w3.org/TR/mbui-intro/, W3C Working Group Note 07, Tech. Rep. (2014).
19. Meixner, G., Paternò, F., and Vanderdonckt, J. Past, present, and future of model-based user interface development. *i-com*, vol. 10, no. 3, pp. 2–11 (2011).
20. Nebeling, M., Nebeling, J., Yu, A., and Rumble, R. ProtoAR: Rapid physical-digital prototyping of mobile augmented reality applications. In *CHI*. ACM (2018), p. 353.
21. OpenAPI specification, https://www.openapis.org/, (last access on 2021).
22. Paternò, F., Santoro, C., Spano, L. D., and Raggett, D. MBUI - task models. https://www.w3.org/TR/task-models/, W3C Working Group Note 08, Tech. Rep. (2014).
23. Seiger, R., Gohlke, M., and Aßmann, U. Augmented reality-based process modelling for the internet of things with holoflows. In *BPMDS*, ser. Lecture Notes in Business Information Processing, vol. 352. Springer (2019) pp. 115–129.
24. Unity, https://unity.com/unity/features/ar (last access on 2021).
25. Vuforia Studio, https://www.ptc.com/en/products/vuforia (last access on 2021).
26. Wikitude AR SDK, https://www.wikitude.com (last access on 2021).