Universität Stuttgart

Visualisierungsinstitut der Universität Stuttgart

Allmandring 19
70569 Stuttgart

Bachelorarbeit

Agile Situated Visualization

Till Mayer

Studiengang: Softwaretechnik

Prüfer: Prof. Dr. Daniel Weiskopf

Betreuer: Dr. Leonel Merino, Xingyao Yu

begonnen am: 01.05.2021

beendet am: 28.10.2021

Abstract

A critical feature of situated visualization toolkits is the ability to automatically place visualizations by detecting objects and surfaces in the physical environment. However, the few existing available toolkits lack this feature, which slows down development cycles and makes situated visualizations less practical. In this thesis, I present AVAR-X, a toolkit to create situated visualizations that boosts the agility in development cycles through spatial mapping and object recognition. To this end, I extended AVAR, an existing toolkit, with situating modalities as well as revised the user interface such that development of situated visualizations is user-friendlier. Using AVAR-X, users can situate visualizations by selecting targets in the environment, using two predefined modalities, eliminating the need to do this by hand. To demonstrate the practical application of AVAR-X, I present three usage examples, developed using the toolkit. They illustrate how AVAR-X may be used to quickly situate energy efficiency visualizations in a room, how situated visualizations can help in day-to-day scenarios such as managing the inventory of a supermarket, and how troubleshooting complex computer networks can be made easier. While AVAR-X indeed simplified the development and situating of visualizations, further improvements could be made. Exploring different methods of text entry, or integrating common programming tools, are only a couple of ways of helping the user with agile development of situated visualizations.

Kurzfassung

Eine essentielle Funktion jedes Toolkits zur Erstellung räumlich-situierter Visualisierungen ist es, diese nahe ihres Referenten positionieren zu können. Um diese Positionierung zu erleichtern, sollten dabei Objekte und Flächen in der Umwelt automatisch erkannt werden. Allerdings besitzen keine der wenigen existierenden Toolkits diese automatische Funktion, was die Entwicklung verlangsamt und die Nutzbarkeit situierter Visualisierungen einschränkt. In dieser Arbeit präsentiere ich AVAR-X, ein Toolkit für die Entwicklung situierter Visualisierungen, welches mit Spatial Mapping und Objekterkennung die Agilität des Entwicklungsprozesses erhöht. Dazu wurde das existierende Toolkit AVAR mit automatischen Positionierungsmethoden erweitert, und die Benutzerobefläche überarbeitet, sodass die Entwicklung von Visualisierungen benutzerfreundlicher wird. Mit AVAR-X können Visualisierungen automatisch durch zwei Methoden situiert werden, indem Positionen in der Umwelt direkt festgelegt werden. Um AVAR-X zu demonstrieren stelle ich drei Anwendungsszenarien vor, für welche ich mithilfe des Toolkits Visualisierungen entwickelt habe. Diese illustrieren wie AVAR-X verwendet werden kann um die Energieeffizienz eines Raumes zu visualisieren, und wie situierte Visualisierungen in der Verwaltung eines Supermarktes helfen können. Des Weiteren zeige ich, wie mit AVAR-X die Fehlerbehebung in einem Computernetzwerk erleichtert werden kann. Obwohl AVAR-X die Entwicklung situierter Visualisierungen erleichtert hat, könnten weitere Verbesserungen durchgeführt werden. Verwendung anderer Eingabemethoden, oder die Integration weit verbreiteter Programmierwerkzeuge, sind nur zwei Möglichkeiten um die agile Entwicklung situierter Visualisierungen zu vereinfachen.

Contents

List of Figures

List of Listings

Acronyms

API Application Programming Interface. 7

AR Augmented Reality. 7

CSV Comma-Separated Values. 17

GUI Graphical User Interface. 7

HMD Head-Mounted Display. 17

JSON JavaScript Object Notation. 17

MRTK Mixed Reality Tool-Kit. 25

VM Virtual Machine. 7

VR Virtual Reality. 17

WODEN World Dynamic Engine. 7

1 Introduction

Although toolkits can offer an agile approach to create situated visualizations for immersive AR, they are often application specific and usage limited. Additionally, state-of-the-art toolkits do not offer appropriate support to develop data visualizations in an iterative and agile manner while not breaking the immersion. That is, users are required to design and develop situated visualizations on a desktop computer and then use visualizations in immersive AR. To address the shortcomings of these existing toolkits, AVAR [MSY+20] was proposed. AVAR allows users to programmatically develop visualizations, evaluate, and improve them all from within the immersive environment itself in an iterative manner.

While AVAR represents a first step in the right direction, it lacks methods to establish a connection between the created visualizations and the environment. Matching the visualization to the related object in the real world is not just helpful to the viewer for association and providing tactile, physical feedback, it may also prevent misunderstandings if multiple visualizations are present. If this connection does not exist, reordering the physical referents will not reorder the visualizations, which will cause a false association by the perceived distance of the visualization to the referent. With AVAR, working in an iterative fashion with multiple visualizations in the environment is impractical, if not impossible, due to how limited the programming environment is. This however is desirable if a user wants to compare the performance of different visualization techniques, or if multiple visualizations are to be situated.

In this thesis I will introduce AVAR-X, which builds upon AVAR, with the goal to connect visualizations more to the environment and to improve the programming interface in a way to enable the user to more easily develop visualizations from within AVAR. I will give details on how I implemented my changes and why I changed AVAR the way I did, finally illustrating with practical examples the improvements of AVAR-X over AVAR.

Situated visualizations. Bruce H. Thomas *et al.* define a spatially-situated visualizations as [MSD+18]

> *(...) data representations that are related to and portrayed in their physical environment. Sensemaking is achieved through the combination of the visualization and the relationship of that visualization to the immediate physical environment.*

Take for example a person that wants to purchase a smartphone. Specifications of all the devices for sale could be looked up online to compare between the different models. Or, the store could *situate* a visualization of the specifications for each model right next to the relevant device, its so called physical *referent*. This way, the user is close to both the relevant visualization as well as the physical referent itself. In the smartphone example, having access to the devices helps in understanding the visualizations better, as they can help the viewer in, for example, comparing the weights of different models against each other. The distance of the visualization to the referent, real or perceived, is

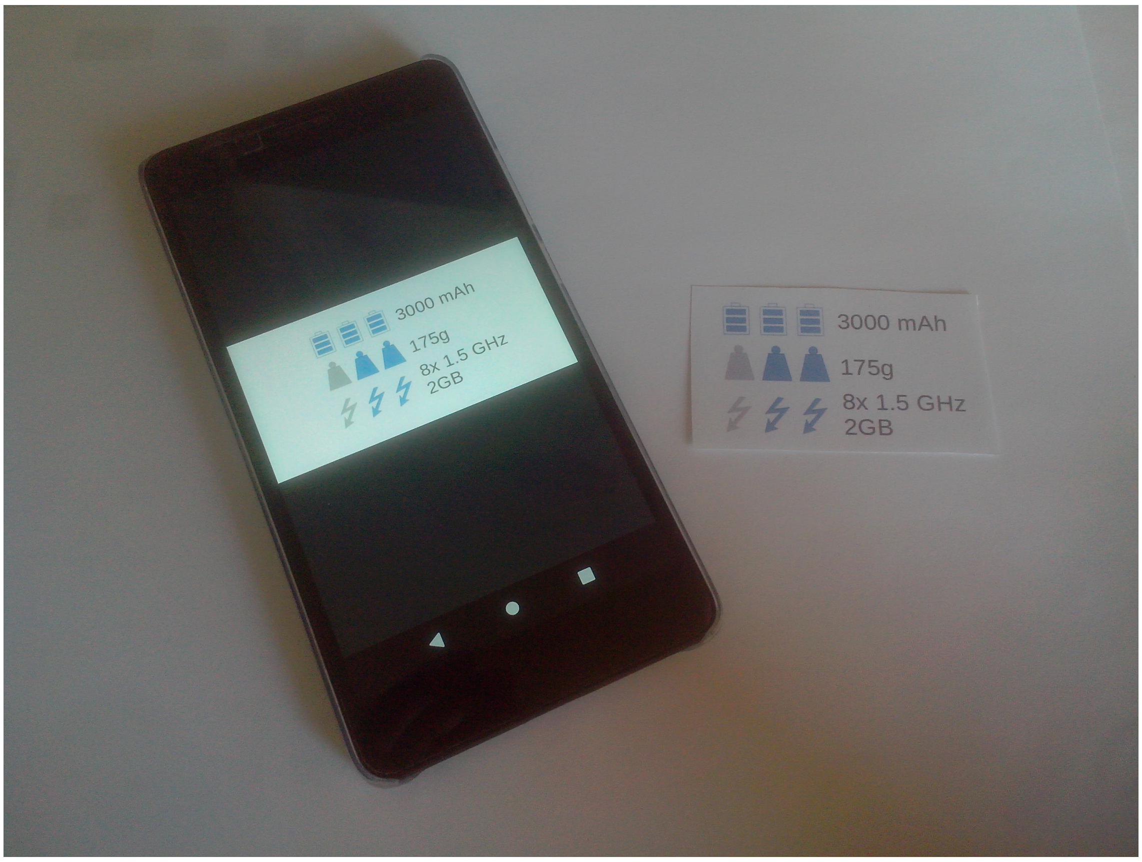


Figure 1.1: Two ways to situate a visualization: The referent, in this case the smartphone, can either show the visualization directly on the screen, or the visualization can be printed out and situated close to the referent.

what makes a visualization situated, however, there is no fixed threshold. Situated visualizations are not reliant on a specific display method, they can be shown on, for example, a display, or be printed out on a piece of paper, as seen in Figure 1.1.

A special case of situated visualizations are *embedded visualizations*. Embedded visualizations subdivide the referent into multiple sub-referents, to each of which a visualization is situated. In the smartphone example, stickers on the devices themselves next to the camera, the touchscreen or the microphone with a visualization of their specifications printed on them, would be examples of embedded visualizations.

A challenge when situating visualizations comes with physically large or movable referents. Take for example, a person wanting to visualize conduits hidden in a wall. While it is possible to print out a plan of the wall with the conduits visualized on it, establishing a mapping between the plan and the real referent can become very difficult. This becomes especially apparent if the precise location of a conduit on the surface of the wall needs to be known. The user would have to first find the conduit on the plan, and then, measure the distance from some reference point. Finally, the position can be marked on the wall. Possibly, a better approach would be to scale up the plan to the size of the wall, and overlay it directly on it. This way, the position of all conduits will become immediately apparent to the observer, and no additional measurements have to be made. Thanks to AR technology, this approach is now practical. AVAR-X aims to support the user in the *agile* development of such situated visualizations.

Agile development. According to Pekka Abrahamsson *et al.*, a development method is agile if [ASRW17]

> *(...) software development is* incremental *(small software releases, with rapid cycles),* cooperative *(customer and developers working constantly together with close communication),* straightforward *(the method itself is easy to learn and to modify, well documented), and* adaptive *(able to make last moment changes).*

If a method follows these principles, developers should be able to recognize and adjust to unexpected situations as they progress [All21]. These concepts can be applied to the development of visualizations as well. With agile visualizations, the focus lays closer at both the incremental as well as the adaptive aspect of agile development. Visualizations should be built in incremental steps, each of which should only last a couple of minutes [Ber16]. To make a toolkit suitable for agile visualization development, it has to make these short development cycles possible.

2 Related Work

In this chapter I discuss related works. Specifically, I introduce other immersive analytics toolkits and explain how these approaches to creating visualizations in immersive environments differ from my approach with AVAR-X.

MIRIA [BLD21]. The Mixed Reality Interaction Analysis toolkit (MIRIA) was created to help in analyzing spatial interaction data by the means of situated visualizations. Using a Head-Mounted Display (HMD), the user can collaboratively view the data in the same environment in which it was recorded. The study data to be analyzed as well as additional metadata are loaded onto a HoloLens device. After choosing the desired data set, the data is imported and processed, after which the user can place the visualization in the space. Seven fixed 3D and 2D visualization types such as trajectory plots or heat maps are available, which the user can select from within the AR environment. Different parameters of the visualizations can be changed without the need to remove the HMD. The user can, in-situ, start and stop the playback of the recorded data, as well as filter it by specifying a desired time interval. The user has to manually situate the 3D visualizations, 2D visualizations can also be situated by attaching them to, for example, walls, or to the virtual representation of the device that was tracked. MIRIA is an application specific toolkit. That is, it is intended to only visualize spatial interaction data. Similarly to AVAR-X, visualizations can be edited in the same immersive environment in which they are viewed, and interactions with an external editor is only required for the configuration of the study data. However, because MIRIA is intended for a specific use-case, it is not designed with the intention of users fundamentally changing or adding new visualizations in-situ. To do so, the user would have to use the Unity editor and program new visualizations in C#.

DXR [SLC+18]. Users of DXR are able to customize and interact with visualizations from within an AR or Virtual Reality (VR) environment. The toolkit is intended for a wide range of users, from non-programmers to experienced Unity developers. To describe the visualization, a custom grammar is used, similar to Vega-Lite. Input data is provided as Comma-Separated Values (CSV) or JavaScript Object Notation (JSON) files. Visualization parameters are loaded from a JSON file, which more experienced users can edit directly instead of using the GUI. Apart from using text to create a visualization, DXR provides templates for common visualizations that can be simply added into the Unity editor by drag-and-drop. Visualization parameters like the data set, or the mark type, are changeable from within the environment, however, the user is restricted to the GUI alone, in which the JSON file describing the visualization cannot be edited directly. Being able to do so would allow for more direct control over the visualization, including the filtering of the data. More advanced users can create their own visualizations using C#, however, this is not possible in-situ. With DXR, situated visualizations are possible, but not efficiently. In particular, the position of every referent has to be provided as an offset to some anchor which has to be measured manually by the user. Then, a visualization can be placed in the environment, however, this also means that the visualization is not attached to the referent. A notable difference to AVAR-X is that the user has to interact with the Unity editor to create a visualization, only some adjustments can be made without it. AVAR-X does not require any changes in the Unity editor to change a visualization, it is programmed directly in-situ.

IATK [CCB+19]. With IATK, visualizations can both be created from templates using the Unity editor or, if the user is more experienced, by creating new visualization templates using C# code. The visualization can then be shown in either VR or AR. Contrary to MIRIA and AVAR-X, IATK does not allow for filtering or basic changes to visualization parameters in-situ. The data for the visualizations is provided by CSV files, that are loaded into the Unity editor. Similar to the other toolkits mentioned, IATK requires the user to manually place the visualization in the environment. IATK does not have any special situating features that create a link between referent and visualization. As opposed to AVAR-X, IATK allows for the linking between multiple visualizations. For example, a user wants to visualize the same data set with both a 3D scatter-plot and a 3D bar-plot. After enabling the feature in the editor, the user can select specific data points from the scatter-plot, that are then highlighted in the bar-plot. A big difference to DXR and AVAR-X is, that ITAK is designed to be used with a large number of data points.

PapARVis Designer [CTW+20]. The PapARVis designer is a tool to create *augmented static visualizations*. Instead of being a stand-alone visualization, they are supposed to extend physical, static visualizations and provide additional, possibly dynamic information. A static visualization printout from a newspaper could, for example, be extended with real time data. As the name suggests, PapARVis Designer is only used to design the visualizations. The application creates the necessary output files, that are then hosted on a server. To view the visualization, the user has to provide an AR application. To create visualizations, PapARVis Designer uses an extension of the Vega grammar. The user describes the visualization using this grammar in a JSON file, data can be either entered directly or loaded from files. A preview of the designed visualization is provided as a prototype for evaluation. Once completed, the visualization is split into a *static* and a *virtual* part. From the static part, a target is created which can then be used by the AR viewer to situate the virtual part. The target along with the virtual part is uploaded to a public server, where the used AR application then combines the static visualization with the virtual component. Similar to the other toolkits previously introduced, the visualizations created with PapARVis Designer cannot be modified directly with the AR device. As the toolkits' focus lays on 2D visualizations, and it is intended to only extend existing visualizations, the need for testing on a real device is not as high as for 3D visualizations that are intended to be embedded in the environment.

Corsican Twin [PWE+20]. The Corsican Twin toolkit takes advantages of VR as well as AR to enable the development of situated and embedded visualizations. With it, visualizations can be created within the immersive environment using a graphical user interface in-situ. However, the development is expected to be conducted within VR instead of involving a real space. To do so, a model of the real location is used. To situate a visualization, the user can then place virtual markers in VR, which will later match up with physical markers in the real location. Once the user is satisfied with the result, an AR headset can be used to view and interact with the visualizations. Corsican Twin is limited to simple and template-based visualizations, where AVAR-X aims to give the user more control over how the data is visualized. Furthermore, AVAR-X is intended to be used close to the referent when developing a visualization, whereas a core goal of the Corsican Twin toolkit is to enable the development off-site. Very similar to AVAR-X are the situating features, helping the user create situated as well as embedded visualizations without the need to position the visualization manually.

3 AVAR-X

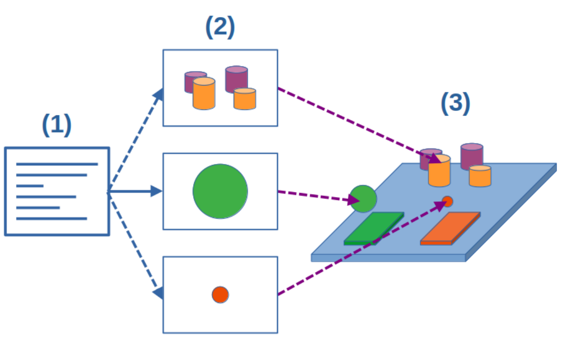
AVAR-X, is a situated visualization toolkit, which extends the AVAR toolkit. Specifically, AVAR-X includes support for spatial mapping and object recognition. In the following, I will give detailed information on how I implemented these extended features, and finally, give an architectural overview over the new AVAR-X toolkit.

3.1 AVAR

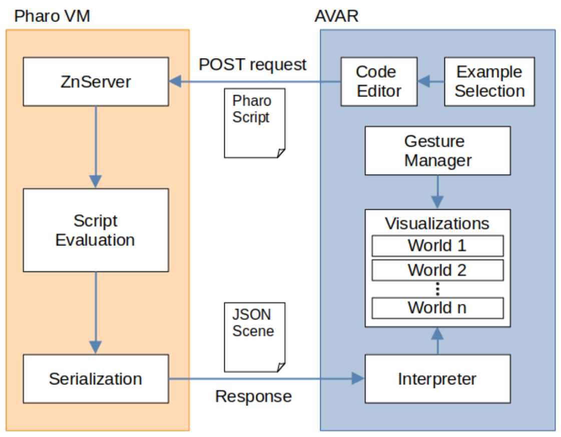
AVAR[1] is a toolkit aimed at enabling the user to create, evaluate, and improve situated visualizations all from within the immersive environment itself. It features a simple text editor with which the user can write and run a script and a number of example visualizations to choose from. The architecture of AVAR is shown in Figure 3.1 (b). AVAR employs the Pharo[2] scripting language and VM as a back-end to build the visualizations. After writing the code for the visualization, the code can be executed, which will send it to a Pharo VM. The script is executed in the VM, the resulting geometries are returned to AVAR using the JSON format, and the visualization will appear in the immersive environment. The user can manipulate the visualization and evaluate it, making changes to the code if necessary, all without breaking the perception of immersion. The code used to build a visualization can be temporarily saved to a clipboard, and later recalled. This way, a practically unlimited number of visualizations can be displayed [MSY+20].

Figure 3.1 (a) shows how AVAR is limited when working with multiple visualizations and binding them to objects in the real world. The code editor (1) is used to to build visualizations (2). After the code is successfully deployed, the visualization will appear in the environment, and the user can place the visualization relative to a fixed origin location in the room (3). This means that there is no connection between the referent and the visualization, if the referent was to be moved, the visualization would stay put. Furthermore, due to the fixed number of code editors (1), the user is limited to actively working on one visualization at a time. If another, previously created visualization was to be edited, the old code had to be recalled, and the code for the new visualization would be lost, as the other visualization would occupy the clipboard. In the next section, I explain how I want to address these shortcomings with a new toolkit, called AVAR-X.

[1] `https://github.com/bsotomayor92/AVAR-unity/`, visited October 27, 2021.
[2] `https://pharo.org/`, visited October 27, 2021.



(a) From code to situated visualization in AVAR: The single code window (1) has to be used to write the code for multiple visualizations (2). The user then places them manually in the AR environment (3).



(b) The code of the visualization is sent to the Pharo VM, which evaluates the script and returns the serialized scene. AVAR interprets the JSON response and adds the visualization to the environment. The user then can situate the visualization manually using gestures.

Figure 3.1: AVAR principle and architecture.

3.2 Design

The objective of AVAR-X, which builds upon AVAR, is to:

i) let the user situate visualizations by specifying target objects in the environment, and

ii) ease agile in-situ development of visualizations by providing a method for organizing multiple code editors.

In the following, I will detail my design decisions for AVAR-X, and how they fulfill these objectives.

3.2.1 Concept

Figure 3.2 shows the overall concept of AVAR-X. Like with AVAR, in AVAR-X visualizations are entirely programmable from within the immersive environment. The user should have an unlimited number of code editors available (1). Each of them should describe one visualization (2). The user then should have the option of choosing between different situating modalities, which are managed by trackers (3). A tracker is responsible for binding the visualization to a target in the real world (4), keeping track of the object's position, and updating it if necessary. The user should then be able to make fine adjustments to the position of the visualization relative to its target.

3.2.2 Software and Hardware

To help explain my design decisions, I will first introduce the software stack and hardware I used.

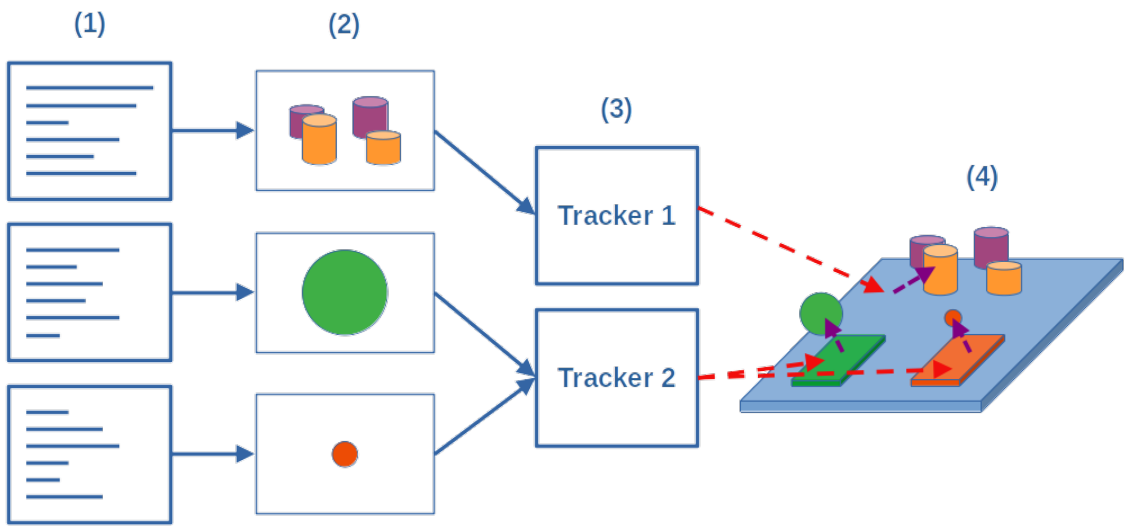


Figure 3.2: In AVAR-X, every code editor (1) is linked to a visualization (2). A tracker (3) is responsible for situating the visualization in the AR environment by binding it to a target (4).

Unity. Unity[3] is an engine designed for developing 2D and 3D applications for a wide variety of devices. A Unity application is divided into *Scenes*, which contain *GameObjects* arranged in a tree-like structure. This hierarchy determines the transformations of the GameObjects, this means that all transformations applied to a parent GameObject will also affect a child GameObject. The GameObjects are internally built of multiple *Components*. These components are scripts written in the C# language and define how a GameObject behaves.

Vuforia. Vuforia[4] is a library intended for the use in AR applications, and can be easily integrated into a Unity project. With it, an AR application can determine the orientation and position of an object or image printout by the means of a color camera. The tracking capabilities can be automatically extended by additional capabilities of hardware on the device, such as accelerometers. It also features extended tracking, the capability to track the position of an object by observing features of its surroundings, even if the tracked object is not in the view anymore.

Pharo. Pharo is a modern Smalltalk environment, which combined with toolkits such as Roassal[5] and WODEN[6], provide users expressive means for building visualizations. Because it is just-in-time compiled, it is very fast to execute newly written code, making it ideal for an agile development approach, in which users have to iterate on the code several times to develop a visualization script. The syntax is designed to be as simple as possible.

[3]https://unity.com/, visited October 27, 2021
[4]https://developer.vuforia.com/, visited October 27, 2021
[5]https://github.com/ObjectProfile/Roassal3/, visited October 27, 2021
[6]https://github.com/woden-engine/woden/, visited October 27, 2021

Microsoft HoloLens 2. The Microsoft HoloLens 2[7] device is the successor to the older HoloLens 1 HMD, featuring a resolution of 1440×936 pixels. Some of the features include a front-facing color camera, and multiple black and white cameras used for reconstructing the environment, and determining the location of the HMD in the room.

3.2.3 Situating Features

To attach a visualization to a physical object in the AR environment, the position of this object has to be known. If the object is movable, it may also be desirable that the position, rotation, and occlusion is updated at a reasonable rate. Methods for tracking a physical object generally fall into one of two different categories, *inside-out* or *outside-in* tracking. When using outside-in tracking, sensors are placed in the vicinity of the object that then calculate the position of it. With inside-out tracking, the object uses sensors to observe the environment around it, and determine the position of itself. With AVAR-X, two situating methods were to be chosen and implemented.

Method 1: Spatial Mapping. The first situating method makes use of the inside-out tracking capability of the HoloLens, called spatial mapping. The device already supports this as it has to know the approximate position of the device to correctly display the holograms. The surroundings are scanned using multiple cameras on the device, and the distinctive features of it are then used to build a 3D mesh that closely resembles the environment around the device [Fer20]. Because this mesh directly relates to the objects in the environment, it can be used to situate visualizations on any of them. The user would only have to choose the part of the mesh that relates to the object in the real world. Additionally, the second generation of HoloLens devices also support *scene understanding*. The experimental library divides the spatial mapping mesh up automatically into categories, such as, platforms, walls and ceilings. Using this library, the user could directly choose a recognized object instead of manually specifying it on the mesh.

With spatial mapping, the situating of a visualization close to its referent would be faster, as the manual adjustment of the position is not required anymore. Furthermore, the quick switching between positions in the room would enable the user to evaluate the visualization closer and further away from the user. When using scene understanding, the process of selecting the relevant target on the spatial mapping mesh can be simplified, as the mesh is already divided into sections.

As spatial mapping relies on the features of all objects around the device, attaching markers to objects is not required, which can be beneficial in some circumstances (such as attaching a visualization to a ceiling, or quickly situating the visualization in front of the user for development purposes). One big drawback of spatial mapping is that binding to smaller objects may not be possible, if the HoloLens does not recognize it. Furthermore, attaching a visualization to a portable object would require the user to change the spatial mapping target manually. The scene understanding library actually supports a method to give objects in the environment a unique id. After the surroundings are re-scanned, the library will try to give the same id to what is identified as the same object. When testing this feature however, I found it to be be not reliable enough.

[7] https://www.microsoft.com/en-us/hololens/hardware/, visited October 27, 2021

Method 2: Object Recognition. To mitigate the shortcomings of spatial mapping, a different situating method using physical markers was chosen. To do so, the Vuforia library is used. It uses the front-facing color camera of the HoloLens to track targets, making it an outside-in tracking method. One benefit of Vuforia is that it is intended to be used with the HoloLens, utilizing some of its other sensors to improve tracking quality. This also implies that Vuforia does not need to determine the position of the device itself . As such, visualizations bound to markers whose location has been determined once before, but are not in view anymore, will still be situated at the correct position, assuming the position of the marker has not changed.

The library supports the tracking of different types of markers, such as simple images, cylinder-shaped objects, or more complex targets, by loading a model of the real-world object into the target database. This way, the need for physical markers is eliminated, however, object models would have to be added into a database that then has to be imported into Unity, being very specific to the use case. Initially, when designing AVAR-X, I considered making targets addable from withing the immersive environment, however, due to limitations in the Vuforia API this idea was not realised. For changing or adding a target, the Unity editor has to be used, this should not, however, affect the development of visualizations.

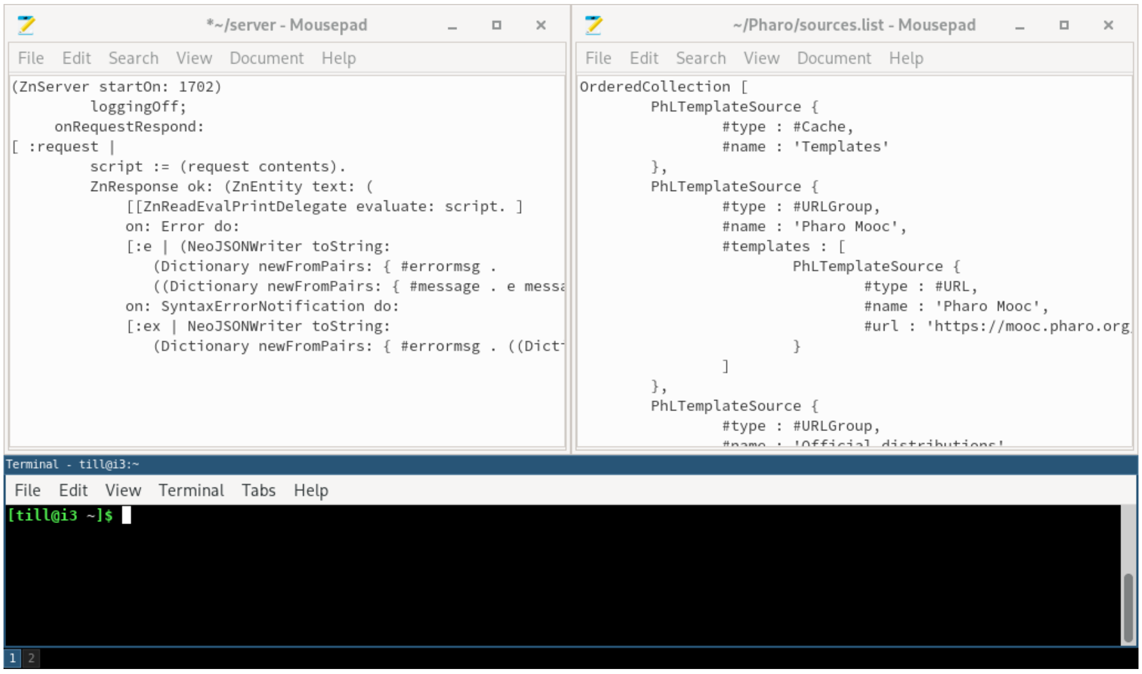
One of the goals for AVAR-X is to make the API as extensible as possible, such that new situating modalities can be added without changing any unrelated code. The interface with which the other parts of AVAR-X interact, should be kept as abstract and high-level as possible.

3.2.4 User Interface

AVAR has a simple user interface, controlled with a Bluetooth keyboard. It presents the user two window panels and a list of example visualizations to choose from. One panel is used by the user for writing code and one is used to show errors, between which the user can switch with the tab key. This is extremely limiting, especially when working with multiple visualizations. When designing AVAR-X, I decided to keep the keyboard-centric nature of the user interface. The reason behind this is, that while virtual keyboards inside the immersive environment could be used, writing large amounts of text on one would not be comfortable. A real keyboard gives the advantage of tactile feedback, and thanks to the nature of AR, inexperienced typists are still able to look at their hands and the keyboard. A big drawback of this approach is that the user has to be close to a surface that allows the keyboard to be placed on, and as such is somewhat restricted to where in the environment the simulations can be developed.

AVAR uses 2D windows that are always fixed in front of the users view. I decided to keep this approach in AVAR-X, however, to enable the user to work on multiple visualizations simultaneously, a method for arranging and switching between different code editor windows had be implemented. Because of the limited resolution of the HMD, and because the user already uses a physical keyboard for programming, I took inspiration from the *i3* window manager[8] and decided to implement a basic tiling window manger for AVAR-X. As opposed to a more common, floating window manager, a tiling window manager divides the available screen space into tiles, resizing them such that all available space is used. This has the effect that windows never overlap, and that the user does not need to manually control the position of the windows. An example can be seen in Figure 3.3, where

[8]`https://i3wm.org/`, visited October 27, 2021


*~/server - Mousepad
File Edit Search View Document Help
~/Pharo/sources.list - Mousepad
File Edit Search View Document Help
Terminal - till@i3:~
File Edit View Terminal Tabs Help
1
2

```
(ZnServer startOn: 1702)
         loggingOff;
     onRequestRespond:
[ :request |
         script := (request contents).
         ZnResponse ok: (ZnEntity text: (
             [[ZnReadEvalPrintDelegate evaluate: script. ]
             on: Error do:
             [:e | (NeoJSONWriter toString:
                (Dictionary newFromPairs: { #errormsg .
                ((Dictionary newFromPairs: { #message . e messa
             on: SyntaxErrorNotification do:
             [:ex | NeoJSONWriter toString:
                (Dictionary newFromPairs: { #errormsg . ((Dict
```

```
OrderedCollection [
        PhLTemplateSource {
                #type : #Cache,
                #name : 'Templates'
        },
        PhLTemplateSource {
                #type : #URLGroup,
                #name : 'Pharo Mooc',
                #templates : [
                        PhLTemplateSource {
                                #type : #URL,
                                #name : 'Pharo Mooc',
                                #url : 'https://mooc.pharo.org
                        }
                ]
        },
        PhLTemplateSource {
                #type : #URLGroup,
                #name : 'Official distributions'
```

```
[till@i3 ~]$
```

Figure 3.3: The i3 window manager divides all available screen space up into tiles that are arrangeable either vertically or horizontally. Windows are controlled mostly through the keyboard, although some operations can be performed with the mouse.

the available space is automatically divided into three tiles, one tile for each window. If a window is to be inserted, the focused window is resized either vertically or horizontally to half its original size, depending on what the user selected beforehand. The new window is then inserted and resized such that the created gap is completely filled. i3 also provides work-spaces, essentially virtual desktops, between which the user can switch rapidly.

The window manager for AVAR-X is intended to give the user more flexibility when programming visualizations. The user should be able to create, resize, close and freely arrange windows. Two types of windows should be implemented, a code editor for programming visualizations, and a console window for showing messages from the Pharo backend. Additionally, the code editor window should enable the user to select from the available situating modalities and their related targets. As illustrated in Figure 3.2, every editor should be responsible for only up to one visualization. Due to the relatively low resolution of the HMD, the number of simultaneously open windows is limited. With switchable work-spaces similar to i3 however, this number can be increased. Controlling the window manager should be, like with i3, possible only with key combinations.

The advantage of a tiling 2D window manager over a more traditional 2D floating window manager is that all available space will be used, which is limited on the HMD. However, learning key combinations to control windows might represent a certain difficulty for users that never used a tiling window manager. A different approach would have been to use 3D windows in the immersive environment itself, that the user could then manipulate with e.g. gestures or direct interactions. This would be less constrained by the resolution of the HMD, the user, however, would have the

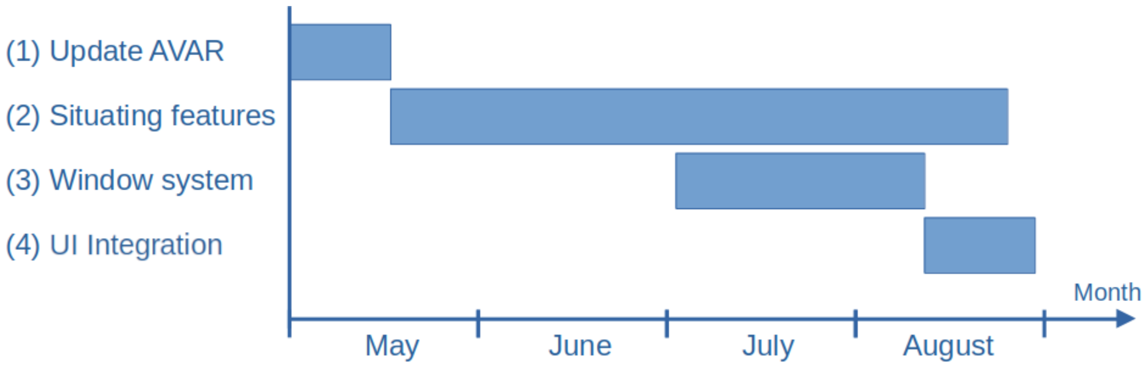


Figure 3.4: Most of the implementation was conducted during four months. In particular, (1) the existing AVAR code had to be ported to the newest Unity version; (2) while adding the situating features; (3) some problems where encountered, and instead development on the arrangeable window system started. Finally, (4) the existing situating code was then adapted to an extensible API, integrating it with the window system and the existing AVAR code.

responsibility to arrange the windows manually. Additionally, if multiple windows are shown in 3D, they have to be made either smaller, as is the case with the 2D window manager anyways, or located further away, which would make reading text harder.

In addition to improving the window system, the selection of example visualizations should be changed to a selection that provides a preview of the example to the user.

3.3 Implementation

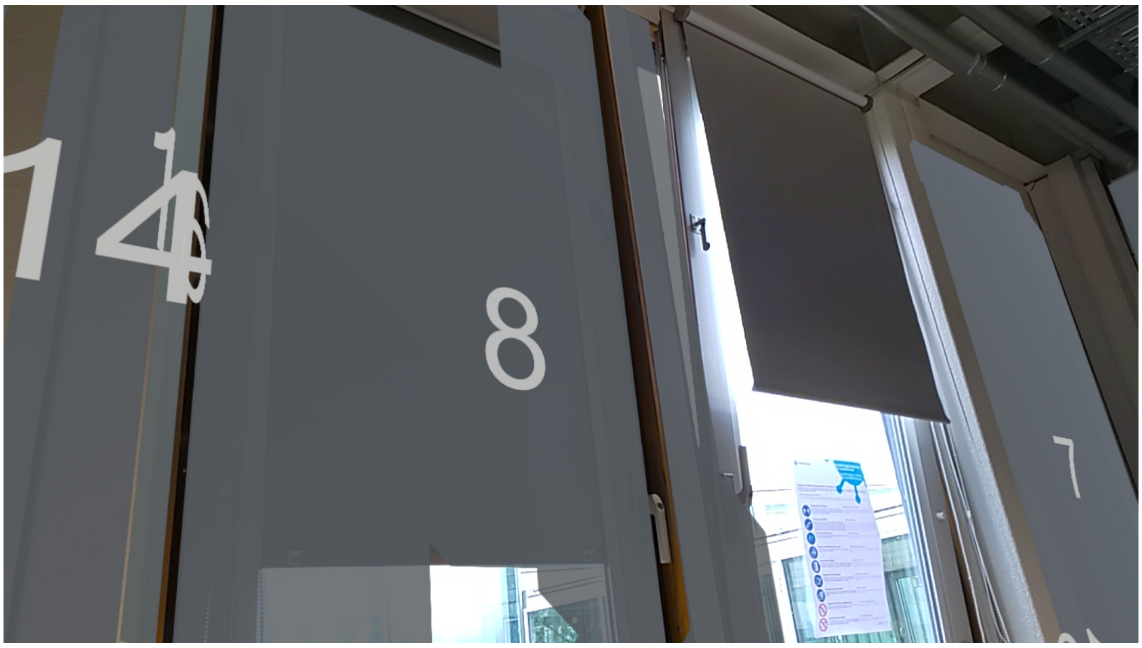
In this section, I present the development process I followed. AVAR-X uses the new LTS version of Unity, 2020.11.3f, and with it the newly introduced Mixed Reality Tool-Kit (MRTK).

3.3.1 Situating Features

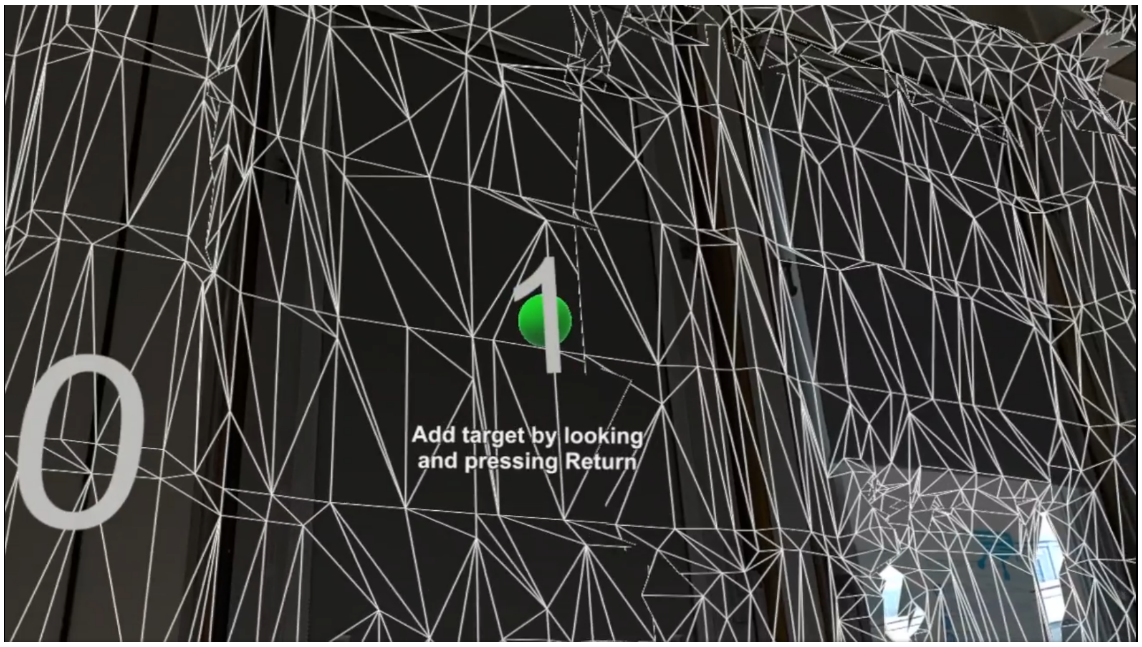
I did not have previous knowledge of Unity, Vuforia, and the HoloLens. In consequence, I decided that my highest priority for AVAR-X were the situating features. This turned out to be a good decision, as visible in Figure 3.4.

Method 1: Spatial Mapping

Because a more recent version of Unity was used, I initially chose to use the scene understanding API instead of the deprecated spatial understanding API to implement the spatial mapping feature. Unfortunately, this new API is currently experimental and not supported on the older HoloLens 1. The user would situate a visualization by first scanning the room with the device. After pressing a shortcut, the scene understanding API then processes the spatial mapping mesh and identifies flat surfaces like walls and platforms, returning identified objects to the tracker which then assigns a



(a) Target selection using scene understanding to recognize surfaces. Notable is that only the left window is recognized properly, as the right window is tilted.



(b) Revised target selection using the look direction of the user. To add a target, the user has to simply look at a position on the spatial mapping mesh. The sphere indicates the origin of the new target. If the sphere turns red, no spatial mapping mesh is available and no target can be added.

Figure 3.5: (a) The initial spatial mapping target selection method used scene understanding to recognize flat surfaces in the room and assigned each a target number the user could choose. (b) The revised target selection gave the user the ability to choose any position on the spatial mapping mesh.

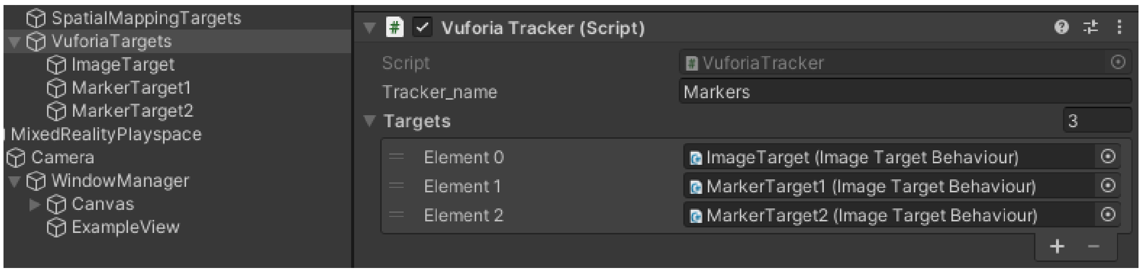
unique number to every surface. The number is situated itself at the origin of the surface, where a visualization would then be placed. The user can then specify this number to situate a visualization on this surface.

This implementation had numerous problems, the biggest of which was the re-identification of surfaces. Because the HoloLens does periodic scans of the environment, the spatial mapping mesh is periodically updated and changes constantly. The surface would then appear as a different one to the tracker, which then could not guarantee to give the target the same id. The user would then have to re-specify the target number for every visualization manually. While the scene understanding API has a mitigation for this very problem, by giving similar appearing structures unique identifiers, I found it to be too unreliable for real-world use in testing. Furthermore, not every surface can necessarily be identified. For example, tables with items on them, such as a keyboard and a monitor, would not be recognized as a platform. This problem is also visible in Figure 3.5 (a), where the user can situate a visualization on the left window, not on the right one however, as this window is tilted and as such not identified as a flat surface. To position the visualization anyways, the user would have to specify the left window (Target 8), and then manually adjust the position of the visualization, which would defeat the purpose of the situating feature. Additionally, the object recognition on bigger spatial meshes was computationally expensive, and the scene understanding API is experimental, which means that it only works on the newer HoloLens 2 devices.

This is why, very late in the development process, it was decided to completely scrap the idea of using scene understanding, and instead to use a simpler and more practical approach. The user can now add spatial mapping targets by pressing a shortcut. This will show the complete spatial mapping mesh, as seen in Figure 3.5 (b). The user can then look at any point on it, and a cursor will show where the visualization origin will be placed on the mesh. This is realized by casting a ray from the camera in the view direction, and colliding it with the mesh. By pressing return, this target will be added at the point of intersection, and can then be selected in the code editor. Just placing the visualization at this point will, however, not orient the visualization parallel to the surface, as the scene understanding API did. To mimic this behaviour, the normal of the point on the mesh is calculated, and the visualization is then oriented such that the positive Z axis of the visualization points to the opposite direction of the normal. This implementation has a small drawback however, as the normal of only a small portion of the mesh is used, instead of an average over the whole surface. If the visualization is situated at a rough surface, the user may have to manually correct the orientation.

Method 2: Object Recognition

Integrating Vuforia into AVAR-X was very straight forward. To track an object, first a target `GameObject` is added in the editor. Any visualization that is to be attached to this object is simply made a child of this object. If Vuforia then updates the position of the `GameObject`, the position of the visualization will automatically change as well, no additional code has to be written. In AVAR-X, Vuforia is only used with simple image markers, that the user can print out and attach to objects. Adding targets during runtime, i.e. not in the Unity editor, is limited by the Vuforia API to only simple image targets. This is why targets are not configurable from within AVAR-X, however, adding new targets using the Unity editor does not require any C# code to be written. If a new target has to be added, a new `GameObject` is created and then added to the list of object tracking

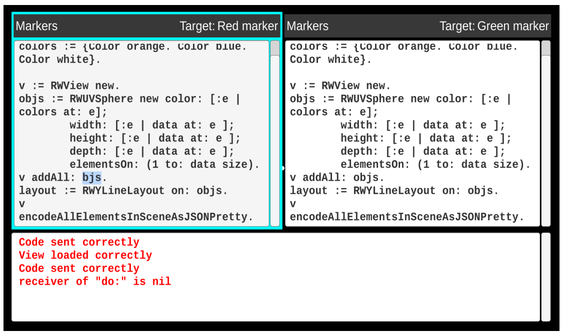


SpatialMappingTargets
VuforiaTargets
ImageTarget
MarkerTarget1
MarkerTarget2
MixedRealityPlayspace
Camera
WindowManager
Canvas
ExampleView

Vuforia Tracker (Script)

Script	VuforiaTracker
Tracker_name	Markers
Targets	3
Element 0	ImageTarget (Image Target Behaviour)
Element 1	MarkerTarget1 (Image Target Behaviour)
Element 2	MarkerTarget2 (Image Target Behaviour)

Figure 3.6: Adding new object tracking targets can be done using the GUI of the Unity editor, no additional code needs to be written.



Markers   Target: Red marker

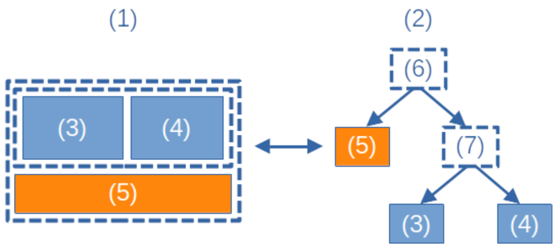
```
colors := {Color orange. Color blue.
Color white}.

v := RWView new.
objs := RWUVSphere new color: [:e |
colors at: e];
        width: [:e | data at: e ];
        height: [:e | data at: e ];
        depth: [:e | data at: e ];
        elementsOn: (1 to: data size).
v addAll: bjs.
layout := RWYLineLayout on: objs.
v
encodeAllElementsInSceneAsJSONPretty.
```

Markers   Target: Green marker

```
colors := {Color orange. Color blue.
Color white}.

v := RWView new.
objs := RWUVSphere new color: [:e |
colors at: e];
        width: [:e | data at: e ];
        height: [:e | data at: e ];
        depth: [:e | data at: e ];
        elementsOn: (1 to: data size).
v addAll: objs.
layout := RWYLineLayout on: objs.
v
encodeAllElementsInSceneAsJSONPretty.
```

```
Code sent correctly
View loaded correctly
Code sent correctly
receiver of "do:" is nil
```

(a) The active window is always highlighted. The user can switch the active window by holding down the meta key and pressing one of the arrow keys.



(b) Windows are managed using a tree structure. (6) and (7) are window groups, which arrange windows horizontally and vertically.

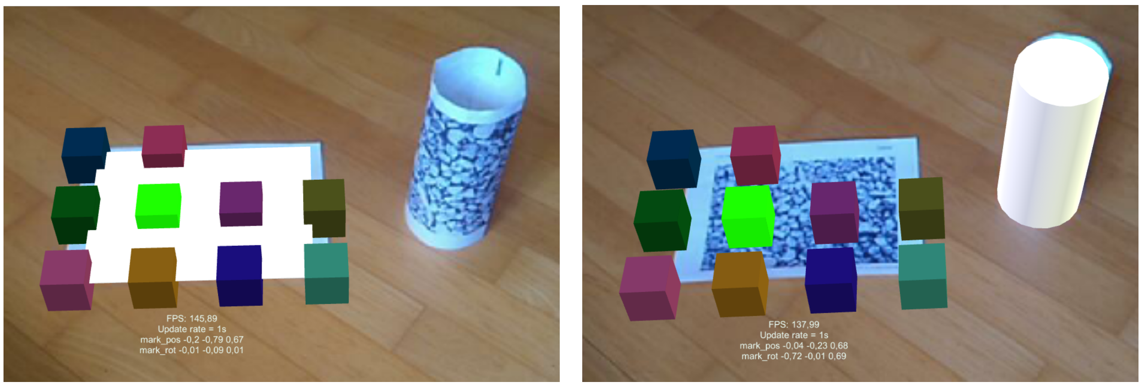
Figure 3.7: A practical window arrangement example: Two code editors are placed next to each other, with a console window showing errors in the scripts below them

targets as seen in Figure 3.6. Because some issues were encountered during the development of the spatial mapping feature, I intermittently worked on the window manager while waiting for access to a device.

3.3.2 User Interface

I deemed that implementing the window manager was of less priority than implementing the situating features, especially since the testing of it would be more straight forward.

Window manager. The window layout is managed by the window manager. Windows are organized in a tree structure, where every workspace is a separate tree. At the root of every tree is a special type of window, called the `WindowGroup`. This window is not visible to the user and only always contains other windows. These windows, which are the children to this node, are laid out based on the rules set for the window group. The user can change these rules, by specifying the arrangement of the windows to be either horizontal or vertical. Because window groups are also windows, they can be contained in other window groups, as can be seen in a typical arrangement shown in Figure 3.7 (b). In the example, the user placed two code editors, (3) and (4), next to each other. Below this, a console window (5) is placed, which will show possible error messages. (2) shows the tree that these windows form. Window group (6) is set to a vertical layout mode, to place the console window



```
FPS: 145,89
Update rate = 1s
mark_pos -0,2 -0,79 0,67
mark_rot -0,01 -0,09 0,01
```

```
FPS: 137,99
Update rate = 1s
mark_pos -0,04 -0,23 0,68
mark_rot -0,72 -0,01 0,69
```

(a) Visualization bound to image marker, no offset is needed.

(b) Visualization bound to cylinder marker. Offset is added in X and Z axes to make sure that the visualization appears outside the cylinder.

Figure 3.8: Screenshots of an early version of AVAR-X. The white plane/white cylinder indicate the marker position as recognized by Vuforia.

below the code editors. To place a second code window next to window (3), a window group has to be inserted with the layout mode set to be horizontal. Into this layout group, a second code window (4) can now be inserted. Should a window be closed, for example window (4), window group (7) will be redundant. To prevent these unnecessary window groups from accumulating, the window manager will check and remove unused window groups in the layout if necessary. In the example, window (3) would move into place of window group (7), which would be deleted.

The user has to specify the modality and the target to situate a visualization. Because every window describes exactly one visualization, this selection can be done from the editor describing the respective visualization. Every code editor shows the used situating modality as well as the target the visualization is situated at. The user can switch the modality and the target independently for every window.

Interaction with the window manager is realized with key combinations. Every command involves the combination of the so called meta key, in the case of AVAR-X the `Alt` key, and one or multiple command keys. To change the layout direction of a window to vertical, the user would press the combination `Alt+V`, and to open a new code editor the combination `Alt+Return` is used. All available shortcuts are listed in Table A.1.

Looking at the example shown in Figure 3.8 (b), it becomes obvious why the user may still have to adjust visualizations manually despite the situating features. To do so, a visualization selection was implemented that enables the user to choose a visualization, and then translate, rotate and scale it using the keyboard. I chose to do this by keyboard, as the manual adjusting of the offset should be kept to a minimum anyways.

Example selection. Next to improving the code editing functionality of the user interface, the example selection was improved. To archive this, the list of examples was removed and instead the user can now show a list of example visualization by pressing a shortcut. A preview for the selected visualization is shown, and by pressing a key, the code for this visualization will be copied into

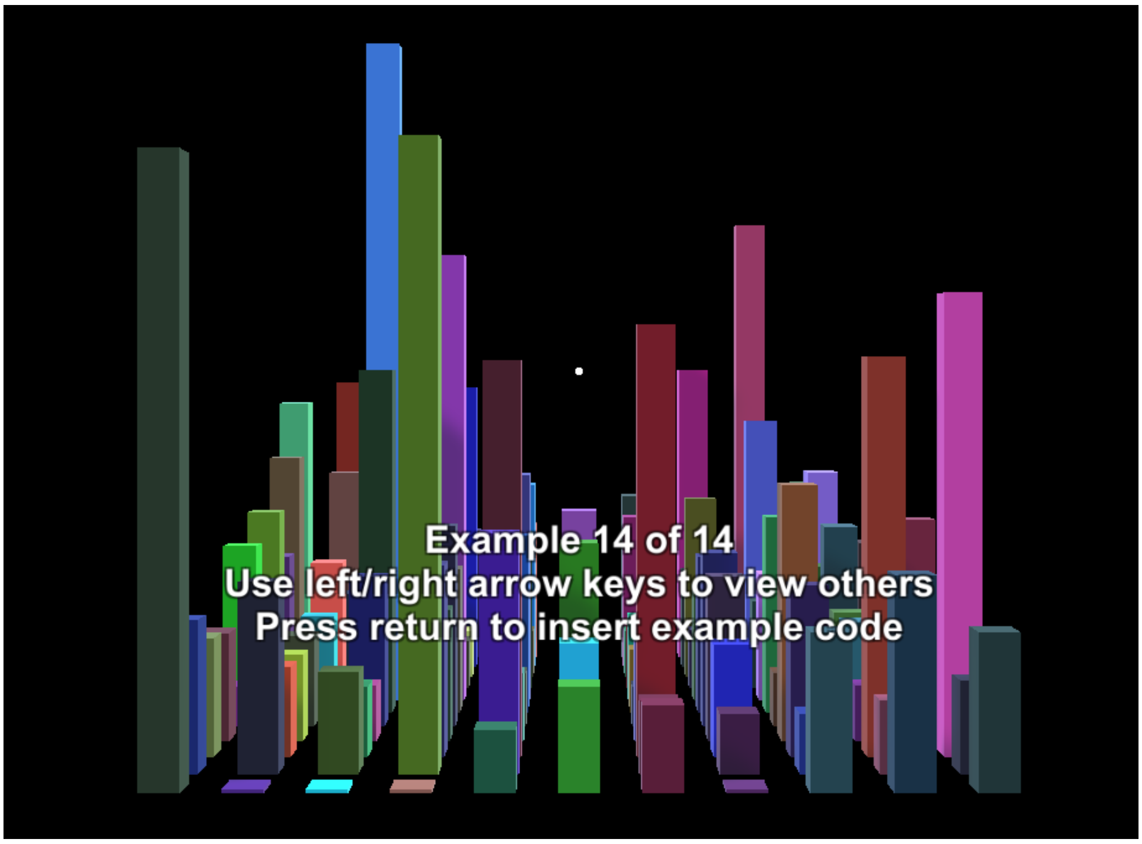

Example 14 of 14
Use left/right arrow keys to view others
Press return to insert example code

Figure 3.9: AVAR-X gives the user a preview of the example visualizations. Once the user presses the return key, a new window will be created, and the code inserted. The visualizations are scaled such that they fit into the users view and the aspect ratio is preserved.

a new code editor. The code for the previews is, like with AVAR, contained in a text file. When selecting a new preview, this code will be deployed to the Pharo instance, which will return the geometries.

Because the scale of the visualizations can vary a lot (e.g., a visualization placed on a wall may be bigger than a visualization placed on a table), the visualization has to be scaled appropriately. To do so, the bounding box of the visualization is calculated. Then, the visualization is scaled such that the longest edge in X, Y and Z axes fits into the users view. Finally, the scaled visualization is centered. An example of this new selection can be seen in Figure 3.9.

3.3.3 Tracker API

To make AVAR-X as extensible as possible, a C# API was created. The goal is to enable the easy integration of different situating modalities in the future, without the need to change other parts of AVAR-X. The implementation happened after both object recognition and spatial mapping have been implemented, this way I could foresee how the API should look like. This approach had the drawback that the code for both modalities and the window system had to be adapted to fit the API.

Essentially, a tracker exposes to the window system functions to retrieve the available count of targets, as well an interface to query their names. Binding a visualization (”world”) to a target that this tracker manages is then as easy as calling the `BindWorldToTarget` function with the world and the target number. The tracking manager manages the trackers themselves, this is where a user can add custom trackers by the means of the Unity GUI.

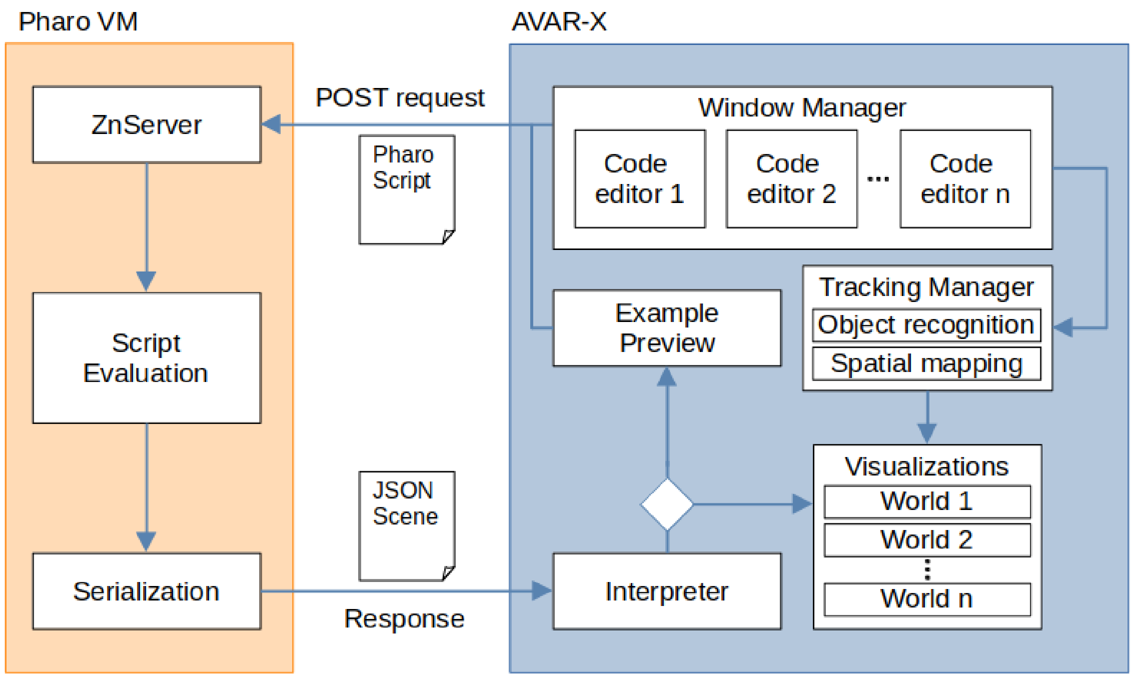


Figure 3.10: The final architecture of AVAR-X. The interface between the Pharo VM and AVAR-X is kept the same, so was the interpreter, except for some minor changes due to the newer WODEN version used. The gesture manager was removed, and all situating is now done with the tracking manager. Clearly visible is the newly implemented example selection, now using the Pharo VM to show a preview of the visualization.

3.4 Discussion

The final architecture of AVAR-X can be seen in Figure 3.10, class diagrams of the implemented features are in Appendix A.1. The interface between the Pharo back-end and AVAR-X is kept unchanged. The gesture manager was removed and replaced by the tracking manager, responsible for positioning the visualizations. Additionally, the old GUI code has been revised and the example selection now also uses the back-end to provide the user a preview of the visualization code.

Because access to a HoloLens 2 device was limited, a lot of development had to be done using an emulator. This posed a problem, as Vuforia does not work in the emulator and the program will refuse to start up. The tracking capabilities of Vuforia can be tested within Unity using a webcam, however, some components still had to be tested in the emulator. This limitation resulted in making sure to design the components such as the trackers and the window manager as modular as possible, to be able to e.g. simply disable the Vuforia tracking capability.

Additionally, MRTK offers two plugins to make a Unity project AR enabled. OpenXR, the recommended plugin, did not work with native keyboard input, which was required for AVAR-X. This problem as well as other problems such as crashes when debugging on a real device let me to choose the older Windows-XR plugin. A benefit was, over the absence of the problems with OpenXR, that AVAR-X can also be used on a HoloLens 1 device.

Some problems with the documentation of MRTK were encountered, with some misleading and conflicting information existing.

When deploying a Unity project to a HoloLens 2, it is important that it is build in ”Release” mode. This means, however, that no debug information can be printed. During development, it was helpful to show debug information in the immersive environment itself.

A Git repository[9] was used to help with versioning the project, and was useful if some mistake needed to be rolled back. Because a file size limit exists with the Git provider, I had to pay attention not to commit big binaries, and only commit code changes, along with some metadata.

[9]https://github.com/st155592/AVAR-X/

4 Usage Examples

In this chapter, I present selected usage examples [MSK+20] to demonstrate how AVAR-X can improve agile development and simplify the process of situating visualizations.

4.1 Inventory Management

The Ebeca supermarket needs to monitor its inventory closely. Otherwise, not enough products may be available, or some of them that are past their expiration date could be still on a shelf. Sarah, a supermarket manager, wants to know which aisles need to be stocked. Sarah also wants to know which aisle in the market is frequented more often and at what time of the day, such that placements of products may be optimized. First, the data has to be obtained. The supermarket already has an inventory management system, and every time an item is bought, put on a shelve or sold, it is registered in the system. The expiration dates of items is entered manually into the system when it is placed on a shelf. Of course when an item is sold, it is not known which item with what expiration date was sold. This is why aisles are checked periodically for expired products.

Sarah then builds a situated visualization with AVAR-X. The visualization is identical for every aisle, only the aisle identifier is changed in the code to visualize a different data set. First, her script queries the data for the specified aisle from the management system. Then it is determined if this aisle has to be checked for expired products by comparing the earliest expiration date of items placed on the shelf to the current date. If the expiration date is equal or later to the current date, a label will show in the visualization, indicating that an employee has to check the contents of the shelf. After the check, the management system is notified that all products on the shelf have an expiration date later than the current date, and items with an earlier date can be removed from the system. Next, a label should show if items have to be moved to the shelf, or if stock of an item is running low. This is simply accomplished by comparing the number of sold items to the number of shelved items, and checking if the number of items in the inventory is over a certain threshold. Using the number of items sold from the aisle at specific times during the stores opening, the script can then calculate the number of people buying items at this location. Sarah decides she wants to visualize the foot traffic as well as the average amount of items on the shelf for every hour the store is open. She wants to be able to compare this statistic to the average of the preceding quarters. To do so, she uses one of the layouts provided by WODEN to arrange twelve spheres in a circle, each sphere representing the average data over one hour the store was open. She uses the size of the sphere to indicate how many people bought an item in the aisle, and colors the sphere depending on how many items were on the shelf for this specific hour. Then she creates three copies of this visualization and arranges them in a grid layout. Every circle then represents the data of the three preceding quarters. To situate the visualization, a unique marker can be attached to each shelf. Then to deploy it, a new window

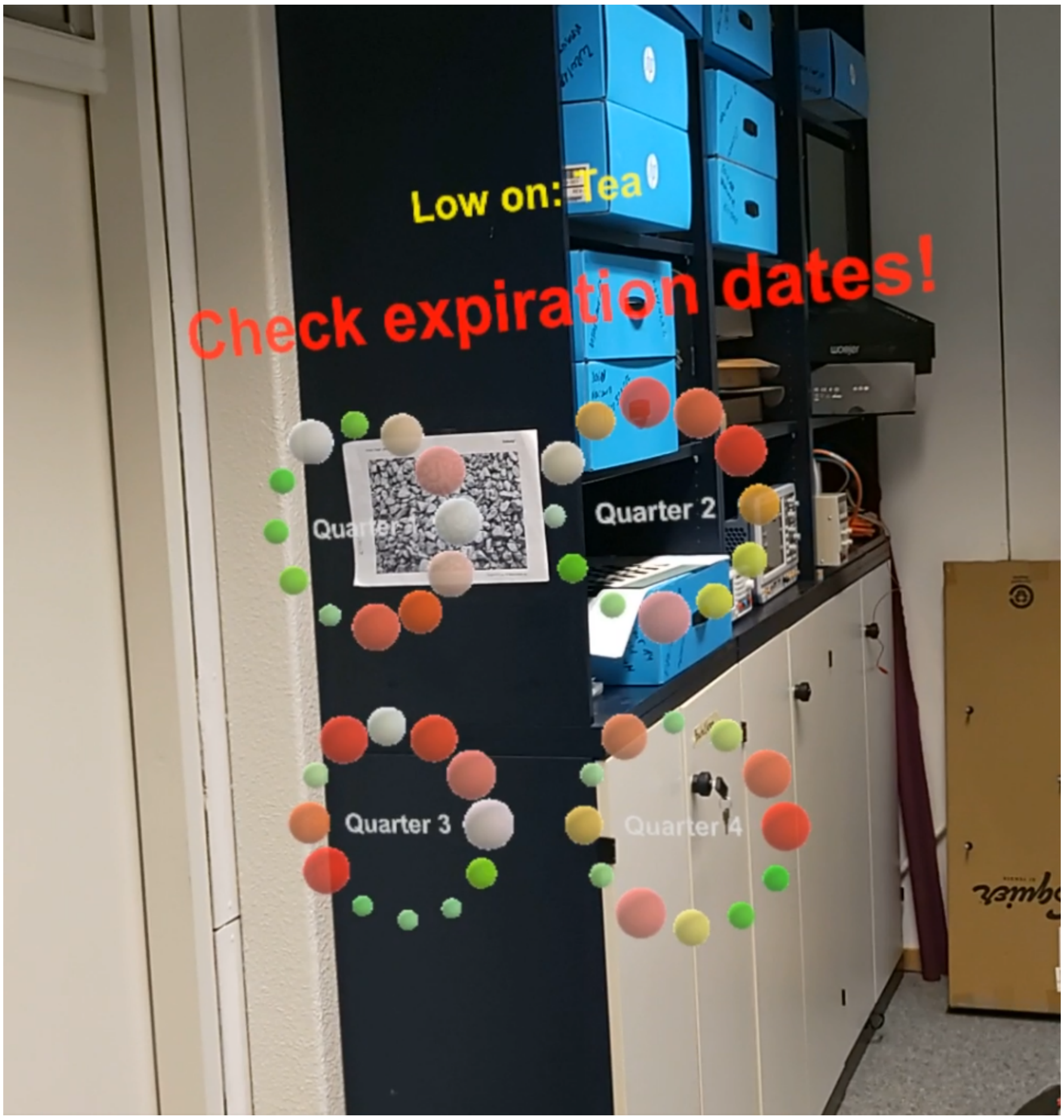


Figure 4.1: A visualization that helps with managing the inventory of a supermarket. Shown is if the staff has to check the expiration date of items and what items have to be put on the shelf. Also shown is the recent traffic through the specific isle at every store opening hour, compared to other quarters of the year. This way, item placement may be optimized. The visualization is situated using a marker, in an actual supermarket, every aisle would be marked with a unique marker at each end of the aisle .

with the copy of the visualization code is created. Sarah then selects the marker modality and the appropriate marker in the GUI. She specifies the correct aisle number in the code and executes it. The visualization is then automatically situated at the appropriate position.

A possible implementation of such a visualization can be seen in Figure 4.1. Here, all data was loaded from a CSV file[1] containing the synthetic dataset. The visualization shows Sarah directly if an item has to be restocked or if she has to check the expiry dates of the items. Because the visualizations are situated directly on the relevant shelf, there is no danger of associating a shelf with the wrong visualization. This could be a problem especially if the wrong shelf is checked for expired items. Because she is at the aisle when she views the visualization, she can inspect the shelves and comprehend the decision of the customers better.

4.2 Network Administration

Richard is a network administrator of an office building. He is responsible for the upkeep of the network itself, as well as the servers which store critical documents. To ensure a high availability of the servers, the network is organized in a semi-decentralized way. Multiple file servers as well as backup servers are located all over the facility. If an error occurs in the network, Richard is responsible for quickly diagnosing and fixing the error. To help with his job, he developed a software that constantly sends status updates from every single server to his computer, reporting different statistics about the server, such as network and processor load. If a report is not received within a certain time-span, it is assumed that the server had a failure which needs to be investigated.

Ideally, Richard would do this remotely, however this is sometimes not possible if the server e.g. crashed and needs to be restarted on-site or if there was a hardware failure. However, he noticed that due to the many different server arrangements in the office building, he has a hard time of remembering how the network is set up on each location, slowing down the troubleshooting process. Additionally, he does not have access to his computer to show the current status of the local network. To circumvent these problems, Richard decides to use AVAR-X to create a situated visualization for every server group located in the building. If a problem occurs that cannot be resolved remotely, he has to go to the location of the fault. There, all the statistics of the different computers as collected by his program should be visualized on the location of the fault. The network topology should be visualized, to help with understanding the problem and track down defective equipment more quickly. He wants to make the visualization as intuitive as possible, such that his co-worker who is not very familiar with the network is also able to address issues quickly. Richard decides to use a situated visualization for this purpose.

First, Richard has to create a visualization template, which can then be customized for every site. To make the use as streamlined as possible, he decides to add the visualization code of every location to the example selection of AVAR-X. This way, the needed visualization code can be quickly recalled when needed. He decides that every visualization should be situated with a marker that is permanently attached somewhere close to the servers. First, the status of every component in the local network is queried from his program. He then draws a graph of the network topology over the servers, where every node is a sphere colored either green if the server is responding, or red if the server is offline. The position a node relative to the marker is input directly into the code in a list. This is better than loading it from an external CSV file, as Richard has to adjust the position of the nodes by trial-and-error from within AVAR-X. Then, the nodes are connected by colored edges indicating the link status of the specific interfaces of a server. The name of every server is

[1] https://github.com/st155592/AVAR-X/tree/master/Examples/inventory.csv

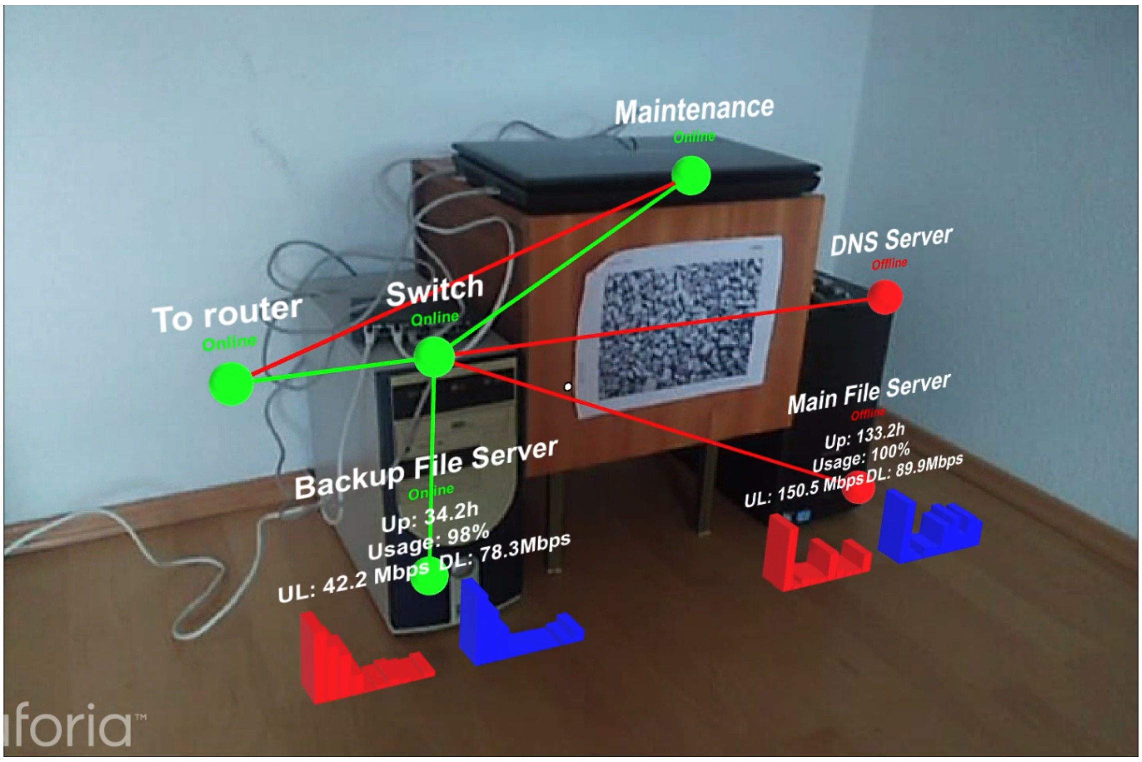


Figure 4.2: Visualizing the network topology of multiple servers, as well as their current status. The nodes visualize the status of the respective server, and the color of the edges visualize the link status.

shown over each node, making the status of every server immediately obvious. Over every node applicable, additional data such as uptime, CPU usage, as well as upload and download rate over time is shown. Once finished, Richard can copy the code from the HMD into the example text document for AVAR-X. If a fault should occur, he only has to select the visualization from the examples. He then has to choose the appropriate marker for the site and execute the code. Richard could optionally customize the name of the markers in the Unity GUI to simplify the selection process.

In Figure 4.2, a possible failure scenario is shown. Here, the node statistics were not loaded from an external server, but hard coded into the visualization[2]. Using libraries such as Zinc[3], Richard could connect directly to his already existing monitoring program to access the data. Looking at this visualization, it becomes immediately obvious even to people that are not familiar with the infrastructure, how the network is set up and where the faulty device is located. This will speed up the troubleshooting process and simplify Richard's job.

[2]https://github.com/st155592/AVAR-X/tree/master/Examples/NetworkMaintenance.st

[3]https://github.com/svenvc/zinc/, visited October 27, 2021

4.3 Energy Efficiency

Meet Dieter, an architect who needs to evaluate the energy efficiency of a building. Specifically, he needs to examine how much heat in a room is lost or gained through various surfaces. Heat loss of a surface is directly dependent on the area, material, and temperature differences across the surface.

This means that factors on the outside such as the exposure of a surface to the sun, which may depend on the time of the year, may affect heat lost or gained. Dieter observes that the room in question is heated in the winter and cooled in the summer. The room has three windows and a door, which are usually the biggest contributors to heat loss relative to surface area. Also of interest for energy efficiency is how often and for how long windows are opened, as this may indicate that the temperature in the room is not ideal. Other than that, Dieter wants to know the heat lost through the four walls. Three of the walls face to other rooms in the building, and one wall faces to the outside.

Dieter has collected a complete data set with over the course of a year. To do so, sensors are directly placed on the outside and inside of every surface. He accomplished this by using an ESP32 micro-controller[4], which sends the collected data over a wireless network to a central server. The central server then calculates the heat loss with the previously defined surface parameters. Additionally, for windows and doors a switch is used to determine if it is open or closed. For windows, an ambient temperature sensor has been placed close to them as well. This sensor then measures the temperature close to the inside of the window. If the temperature is closer to the outside temperature than the inside temperature, it could be a window may be not closing properly.

After the data has been collected, Dieter starts AVAR-X and builds visualizations that are then shown on every relevant surface. As the visualizations are mostly the same for every surface in the room, Dieter wants to build a single visualization which then only has to be changed slightly for every surface. Because it is inconvenient to place physical markers on every relevant surface, spatial mapping is used to situate the visualizations. This, however, also means that every visualization has to be manually matched to the correct surface, such that the correct data is displayed. To accomplish this task in an effective way, a variable in the visualization code is defined, which contains the name of the target, e.g.. ”Corner window”, ”Door”. When he ran the Pharo script, a query is sent to the server with the defined surface name. It returns the appropriate data, that is processed and visualized.

Now Dieter has to decide how the data should actually be visualized and program the visualizations. To make developing easier, a new spatial mapping target is created such that the created visualization is directly in front of the users view. First, the total energy lost through the surface over the duration of the year should be shown. As this is an absolute value, it makes sense to show it as a number directly. The energy loss should also be visualized as compared to the total energy lost over all measured surfaces. The initial approach was to visualize the total energy loss as the size of e.g. a sphere. However, because the visualizations are spread out over the whole room, perspective will make it hard to compare the size of objects. Instead, a visualization is chosen that shows both total energy loss and loss through the surface next to each other. Next, the average temperature difference for every month should be shown. Dieter decides to do this by placing twelve spheres in a grid.

[4]https://www.espressif.com/en/products/socs/esp32/, visited October 27, 2021

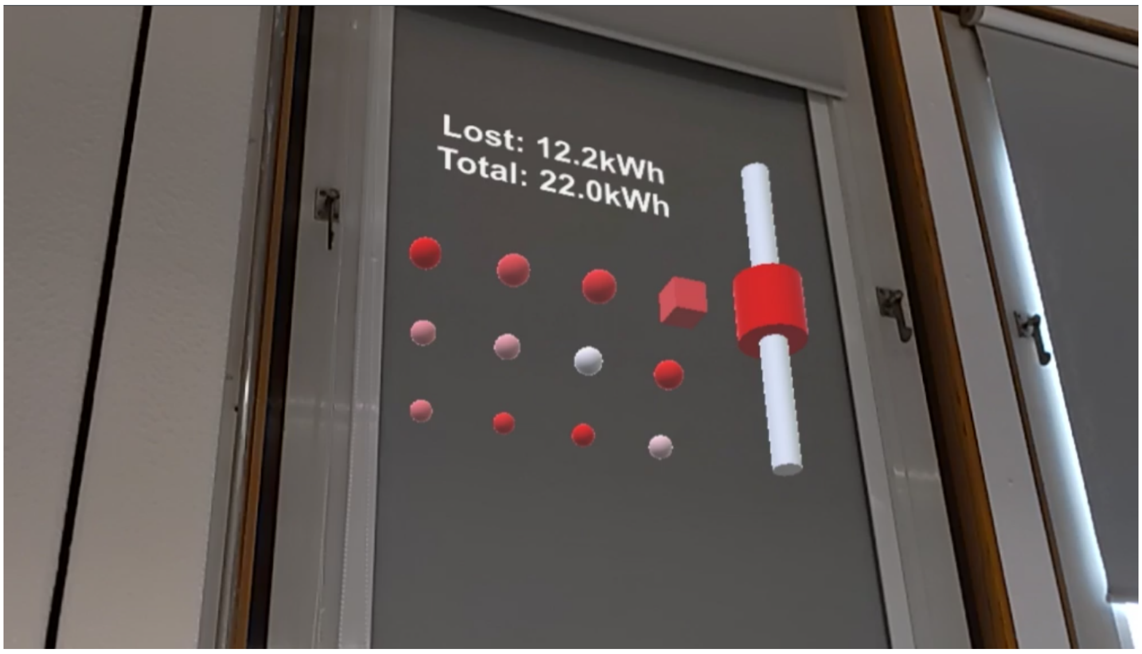


Figure 4.3: What a heat loss visualization may look like. The size of the spheres visualize the temperature difference, while the color shows how long the window was opened as compared to the other months. The cylinders on the right visualize the energy loss through this window as compared to the other surfaces.

Every sphere represents a month, and the size of the sphere represents the temperature difference. If the visualization is attached to either a window or a door, the spheres are color coded with the total time the window was open. The month with the longest open time is colored red, the month with the least time is colored white. To show excessively leaky windows, a cube is shown in place of the sphere.

Finally, the visualizations can be situated. For every surface of interest, Dieter first creates a new spatial mapping target. This can be rapidly done from within AVAR-X. For every target he now opens a new code window and selects the appropriate target number. The visualization code is copied into every window, the name of the surface is specified in the code, and the code is executed. An example for this visualization can be seen in Figure 4.3.

In this case, Dieter was not satisfied with how the visualization turned out. He thinks the visualization of the months is not very intuitive and can be improved. To make it easier to understand, he wants to visualize the data with text showing the month's name directly. The size of the text should show temperature difference, and if the month is colored red, the window may be leaky. The saturation of the text color still shows how long a window was open for. These changes can be made directly in AVAR-X, Dieter only has to copy the code of the new visualization into every window and re-run all scripts again.

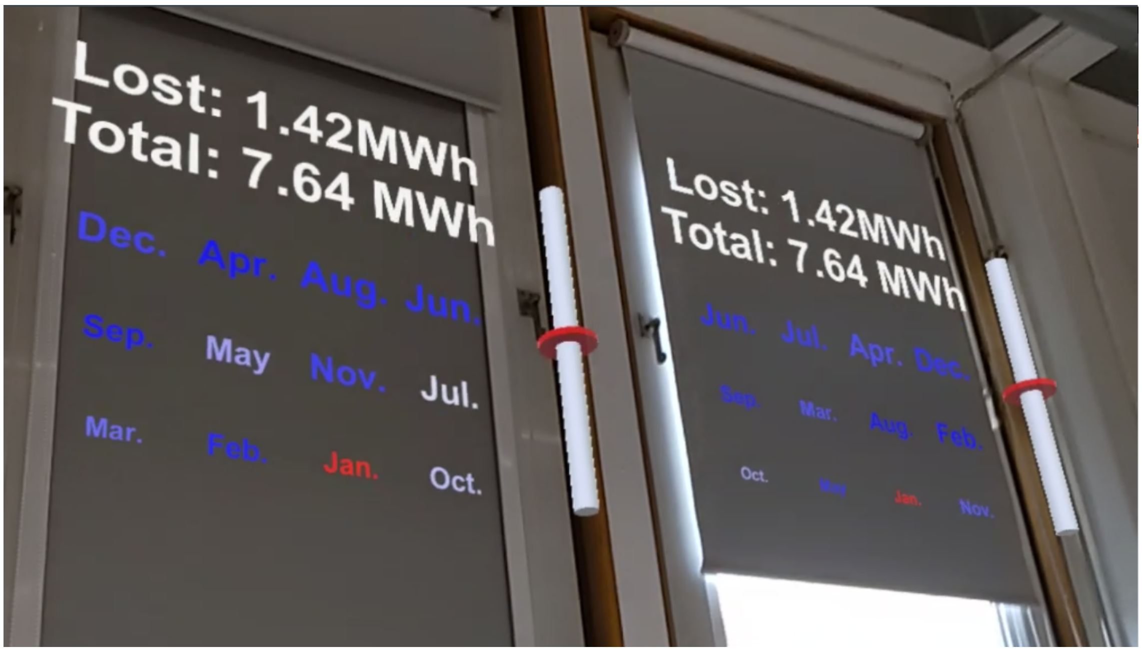


Figure 4.4: The final heat loss visualization. The size of the month names indicate the temperature difference. Months labeled red indicate that the window had a suspiciously low temperature difference and may leak. In this example, the window was open a lot in winter and summer, indicating too much heating or cooling

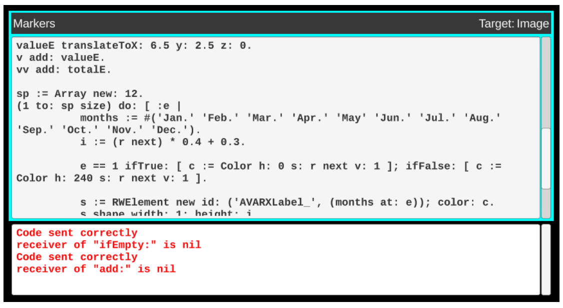
The revised version of the visualization is shown in Figure 4.4. Dieter took advantage of the quick development cycles he has with AVAR-X to determine the size and spacing between the different visualization elements. To illustrate this specific example, a synthetic data set was created and used. All the parameters are specified in a CSV file[5] located on the Pharo server.

4.4 Discussion

The presented usage examples illustrate how AVAR-X could be used in envisioned real-world applications. First of, visualizations are created completely in-situ. Unlike toolkits introduced in related works, the user does not have to use the Unity editor to create or fundamentally change visualizations. With the expressive Pharo programming language, users are not restricted to limited, generic visualizations. Thanks to the just-in-time compiled nature of Pharo, very short development cycles are possible, as opposed to other toolkits which need to be recompiled and redeployed onto the HMD.

While AVAR already had these benefits, additional improvements aim to shorten these cycles even further and make development easier. Placing visualizations in the environment is faster thanks to the situating features. Compared to AVAR, where every visualization would have to be placed manually after running the script, visualizations are now situated by either specifying a previously

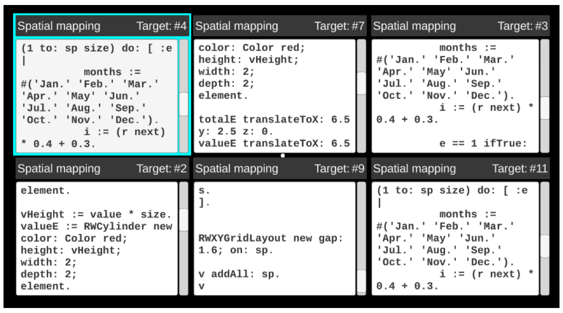
[5]https://github.com/st155592/AVAR-X/tree/master/Examples/windows.csv



```
valueE translateToX: 6.5 y: 2.5 z: 0.
v add: valueE.
vv add: totalE.

sp := Array new: 12.
(1 to: sp size) do: [ :e |
        months := #('Jan.' 'Feb.' 'Mar.' 'Apr.' 'May' 'Jun.' 'Jul.' 'Aug.'
'Sep.' 'Oct.' 'Nov.' 'Dec.').
        i := (r next) * 0.4 + 0.3.

        e == 1 ifTrue: [ c := Color h: 0 s: r next v: 1 ]; ifFalse: [ c :=
Color h: 240 s: r next v: 1 ].

        s := RWElement new id: ('AVARXLabel_', (months at: e)); color: c.
```

```
Code sent correctly
receiver of "ifEmpty:" is nil
Code sent correctly
receiver of "add:" is nil
```

(a) During development, having more space to program is desirable. A window showing errors in the code can be opened as needed, maximizing the usable space.



```
(1 to: sp size) do: [ :e
|
        months :=
#('Jan.' 'Feb.' 'Mar.'
'Apr.' 'May' 'Jun.'
'Jul.' 'Aug.' 'Sep.'
'Oct.' 'Nov.' 'Dec.').
        i := (r next)
* 0.4 + 0.3.
```

```
color: Color red;
height: vHeight;
width: 2;
depth: 2;
element.

totalE translateToX: 6.5
y: 2.5 z: 0.
valueE translateToX: 6.5
```

```
        months :=
#('Jan.' 'Feb.' 'Mar.'
'Apr.' 'May' 'Jun.'
'Jul.' 'Aug.' 'Sep.'
'Oct.' 'Nov.' 'Dec.').
        i := (r next) *
0.4 + 0.3.

        e == 1 ifTrue:
```

```
element.

vHeight := value * size.
valueE := RWCylinder new
color: Color red;
height: vHeight;
width: 2;
depth: 2;
element.
```

```
s.
].


RWXYGridLayout new gap:
1.6; on: sp.

v addAll: sp.
v
```

```
(1 to: sp size) do: [ :e
|
        months :=
#('Jan.' 'Feb.' 'Mar.'
'Apr.' 'May' 'Jun.'
'Jul.' 'Aug.' 'Sep.'
'Oct.' 'Nov.' 'Dec.').
        i := (r next) *
0.4 + 0.3.
```

(b) During deployment, a smaller window can be created for every visualization, as ideally larger edits are not required anymore. In this case, the code can be simply copied into every window only changing the visualized data set.

Figure 4.5: The arrangeable windows allow for different window configurations, depending on the current user requirements.

selected surface in the room or by choosing a marker that can be attached to any object. The biggest improvement over AVAR, however, is the composable window system. It enables the user to work on and improve upon multiple visualizations simultaneously, without having to save and recall code. As opposed to AVAR, the code of every visualization is accessible, as long the window is not explicitly closed by the user. During development of the usage examples, it became apparent that the isolated development of the different visualization sub-components was helpful in managing the code of larger visualizations. The window system was also helpful during deployment of the visualizations. Usually, the involved visualizations were designed such that the code could be reused for different referents with minimal changes, just by specifying a different data set. A new window was then created for every visualization, into which the code was copied and the relevant data specified. Thanks to the flexible window manager, the user can create an ideal environment for both the development and deployment scenario. In Figure 4.5, a possible arrangement for both scenarios is shown.

The biggest shortcoming of AVAR-X is the lack of interactability with the visualizations. Sometimes placement of the visualization relative to their target object has to be adjusted, which is currently only possible by keyboard. Selecting objects by e.g. pointing at them and then manipulating them using gestures would make sense, especially in the AR environment. AVAR supported translation, rotation and showing popup information of visualizations this way, however due to a change in the gesture recognition API in the newer Unity version, this would have meant that a lot of this code would have had to be rewritten. While the usage of the situating features makes the need for manually adjusting of the position somewhat infrequent, it is still required in some cases. During the deployment of the example visualizations, it became apparent that for the positioning of some visualizations, a close up inspection and correction was necessary. For this, the Bluetooth keyboard had to be re-positioned closer to the visualization, which was inconvenient in some situations. Interactions can also help with the better understanding of visualizations. Looking at Section 4.2 it becomes apparent that visualizations can quickly become overcrowded with information. The previously introduced Corsican twin toolkit mitigated this problem by letting the user show and hide visualization details by gesture, a feature that AVAR-X still lacks.

Listing 4.1 An excerpt of the visualization code used in Section 4.2 to create a label.

```
...
"Add name and status label"
labels := OrderedCollection new.
name := RWElement new id: 'AVARXLabel_', (nd at: 2).
name shape height: 0.4.
labels add: name.
...
```

Additionally, while the current window manager approach is very usable and quick to learn, it is very limited by the low resolution of the HMD. The window manager currently also lacks the ability to swap the position of windows, which would improve the user experience, however I found this to not be essential. Visualizations can currently not be saved in any way, and once AVAR-X is quit, all code is lost. This is not desirable, and some way to save and recall visualization to and from the HMD would be ideal. To help manage multiple visualizations, it might be desirable to access the target type and number from within the Pharo code. This way the user would not need to specify the target twice for every visualization, a problem which becomes apparent in Section 4.1, especially with many targets. When creating the example in Section 4.2, I initially attempted to situate a visualization on every device, which where then to be connected with the edges. However, with the current implementation, as every visualization is essentially isolated, this is not possible. Instead, the less-ideal method of creating and situating one big visualization had to be used.

To make the development of visualization faster, it would make sense to employ classes that make performing commonly performed tasks, such as normalization, easier. While the creation of helper classes during runtime is possible in the Pharo VM, it is not as practical to do this from within AVAR-X. One more shortcoming lays in the back-end of AVAR-X. The WODEN library used to create the 3D visualizations in Pharo has to serialize all elements in the scene as JSON. This JSON response contains the color and the position of the elements, the rotation, however, is not serialized. Due to the update to Pharo 9.0, and with it, the newer WODEN and Roassal versions, the previous method to insert labels into a visualization was not supported anymore. To avoid having to use a special Pharo version, a very simple method to add labels to visualizations was implemented, using the generic `RWElement` provided by WODEN, as shown in Listing 4.1. An element is considered a label by AVAR-X, if its *id* has the ”AVARXLabel_” prefix. All text following the underscore is used as the label text, including newline characters. While this is certainly not the cleanest approach possible, it has the benefit of being usable with an unmodified version of WODEN, and was used extensively in the illustration of the examples[6].

Overall, AVAR-X can be used to quickly develop and situate visualizations. While some practical limitations still exist, both design goals as laid out in Section 3.2 are fulfilled.

[6]https://github.com/st155592/AVAR-X/Examples/

5 Conclusion and Outlook

In this thesis I presented AVAR-X, an agile situated visualization toolkit. Initially, I defined situated visualizations and agile development. I illustrated the benefits of these concepts and identified challenges with these methods. Then I introduced toolkits that can be used to create situated visualizations and explained their shortcomings, merits and commonalities compared to AVAR-X. I introduced AVAR, the toolkit upon which AVAR-X builds, and identified its shortcomings as a missing connection between visualizations, their code and the real environment. Additionally, I came to the conclusion that multiple code editors are essential for agile development, especially when working with multiple visualizations. After I presented the employed hardware and software, I detailed how the various improvements were implemented. One of the two situating modalities used is Vuforia, that uses printable markers to determine the location and orientation of objects in the environment. The other modality is spatial mapping, where the geometry of the real environment is determined and certain objects in this environment such as platforms or walls can then be identified and used as a target. While the situating features were implemented, I also started to work on the user interface. I decided to keep the windows fixed in front of the user, and as such had to implement a way to arrange windows. Taking inspiration from i3, I implemented a tiling window manager usable from within the environment, controllable with only keyboard shortcuts. It, together with groups of windows arranged in switchable workspaces, enables the user to work on multiple visualizations simultaneously, and helping with the management of larger code-bases. After having basic situating features implemented and the window manager was working, I needed to create an easily extensible API for the situating modalities. I decided on a very high-level API, exposing a way to bind visualizations to *targets*, which were managed by *trackers*. This way, the user can select for every visualization the situating modality and target to use. I also extended the example templates, by providing the user with a preview of how the visualization will look like. Through selected usage examples, I demonstrated how AVAR-X can mitigate the shortcomings of AVAR identified earlier, and came to the conclusion that AVAR-X is well suited for agile development of situated visualizations.

Outlook. There are various user experience improvements that can be made to AVAR-X, such as fine positioning of a visualization by using gestures. An interesting possibility for future work would be the replacement of the physical keyboard with an input method more suited for the AR environment, as explored in a related study [Gär20]. Thanks to the modular design, new situating modalities could be added, providing improvements over the currently implemented ones. One such tracker could use cameras placed in the room to improve tracking of objects, as Vuforia is limited to tracking objects that are in the view of the camera in the HMD. For improving the programming experience, it might be helpful to integrate programming tools such as syntax highlighting, versioning support or find-replace functionality into the code editor.

Bibliography

[All21] A. Alliance. *Agile 101*. 2021. URL: https://www.agilealliance.org/agile101/ (cit. on p. 15).

[ASRW17] P. Abrahamsson, O. Salo, J. Ronkainen, J. Warsta. "Agile Software Development Methods: Review and Analysis". In: *arXiv preprint arXiv:1709.08439* (2017) (cit. on p. 15).

[Ber16] A. Bergel. *Agile Visualization*. 2016. URL: http://agilevisualization.com/AgileVisualization/Introduction/0001-Introduction.html (cit. on p. 15).

[BLD21] W. Büschel, A. Lehmann, R. Dachselt. "MIRIA: A Mixed Reality Toolkit for the In-Situ Visualization and Analysis of Spatio-Temporal Interaction Data". In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 2021, pp. 1–15 (cit. on p. 17).

[CCB+19] M. Cordeil, A. Cunningham, B. Bach, C. Hurter, B. H. Thomas, K. Marriott, T. Dwyer. "IATK: An Immersive Analytics Toolkit". In: *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE. 2019, pp. 200–209 (cit. on p. 18).

[CTW+20] Z. Chen, W. Tong, Q. Wang, B. Bach, H. Qu. "Augmenting Static Visualizations with PapARVis Designer". In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–12 (cit. on p. 18).

[Fer20] H. Ferrone. *Inside-out tracking*. 2020. URL: https://docs.microsoft.com/en-us/windows/mixed-reality/enthusiast-guide/tracking-system (cit. on p. 22).

[Gär20] N. Gärtner. "Exploring Interaction Modalities for Immersive Analytics and Situated Visualization". B.S. thesis. 2020 (cit. on p. 43).

[MSD+18] K. Marriott, F. Schreiber, T. Dwyer, K. Klein, N. H. Riche, T. Itoh, W. Stuerzlinger, B. H. Thomas. *Immersive Analytics*. Vol. 11190. Springer, 2018 (cit. on p. 13).

[MSK+20] L. Merino, M. Schwarzl, M. Kraus, M. Sedlmair, D. Schmalstieg, D. Weiskopf. "Evaluating Mixed and Augmented Reality: A Systematic Literature Review (2009-2019)". In: *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE. 2020, pp. 438–451 (cit. on p. 33).

[MSY+20] L. Merino, B. Sotomayor-Gómez, X. Yu, R. Salgado, A. Bergel, M. Sedlmair, D. Weiskopf. "Toward Agile Situated Visualization: An Exploratory User Study". In: *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–7 (cit. on pp. 13, 19).

[PWE+20] A. Prouzeau, Y. Wang, B. Ens, W. Willett, T. Dwyer. "Corsican Twin: Authoring In Situ Augmented Reality Visualisations in Virtual Reality". In: *Proceedings of the International Conference on Advanced Visual Interfaces*. 2020, pp. 1–9 (cit. on p. 18).

[SLC+18] R. Sicat, J. Li, J. Choi, M. Cordeil, W.-K. Jeong, B. Bach, H. Pfister. “DXR: A Toolkit For Building Immersive Data Visualizations”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (2018), pp. 715–725 (cit. on p. 17).

All links were last followed on October 20th, 2021.

A Appendix

A.1 Class diagrams

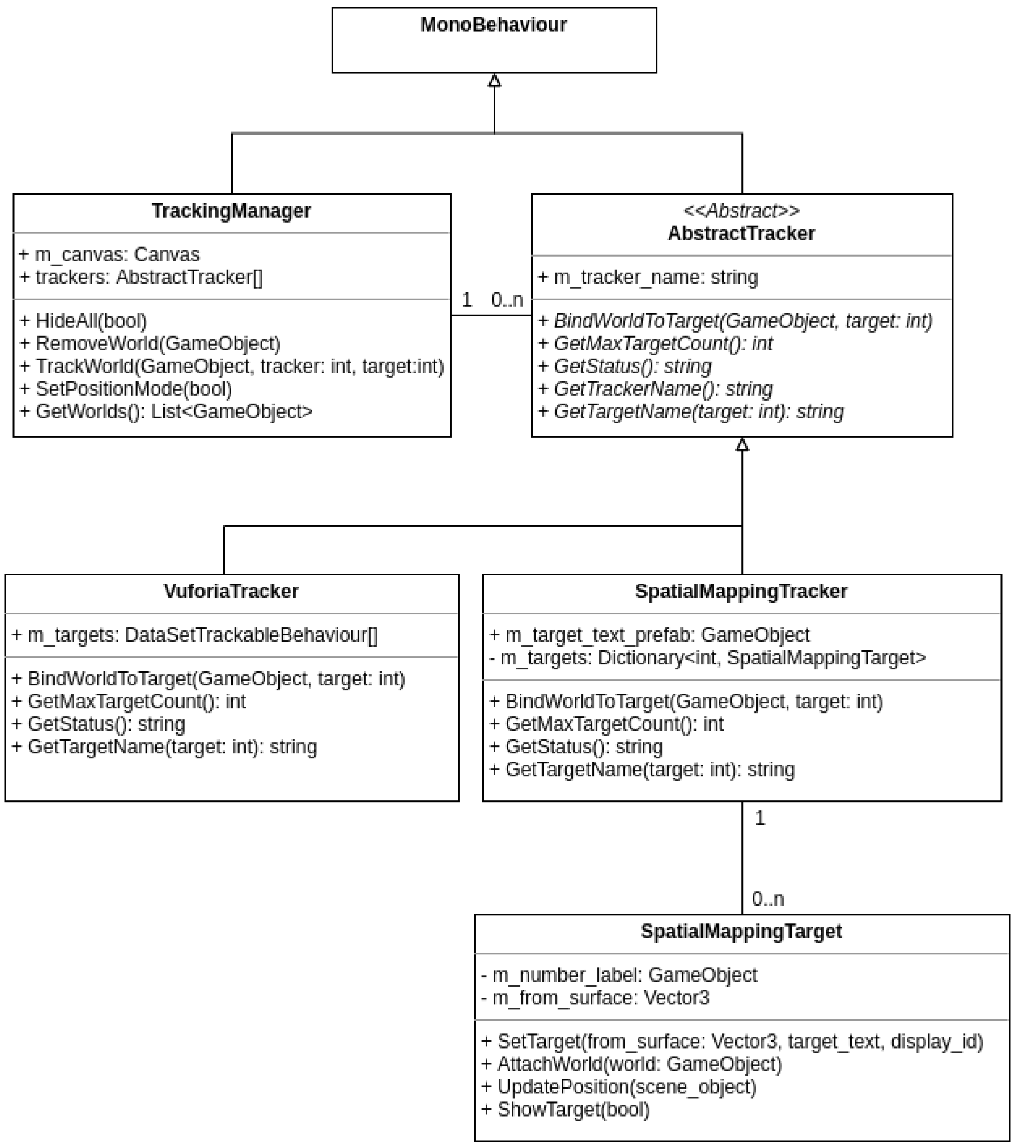

MonoBehaviour
TrackingManager
+ m_canvas: Canvas
+ trackers: AbstractTracker[]
+ HideAll(bool)
+ RemoveWorld(GameObject)
+ TrackWorld(GameObject, tracker: int, target:int)
+ SetPositionMode(bool)
+ GetWorlds(): List<GameObject>
1
0..n
<<Abstract>>
AbstractTracker
+ m_tracker_name: string
+ BindWorldToTarget(GameObject, target: int)
+ GetMaxTargetCount(): int
+ GetStatus(): string
+ GetTrackerName(): string
+ GetTargetName(target: int): string
VuforiaTracker
+ m_targets: DataSetTrackableBehaviour[]
+ BindWorldToTarget(GameObject, target: int)
+ GetMaxTargetCount(): int
+ GetStatus(): string
+ GetTargetName(target: int): string
SpatialMappingTracker
+ m_target_text_prefab: GameObject
- m_targets: Dictionary<int, SpatialMappingTarget>
+ BindWorldToTarget(GameObject, target: int)
+ GetMaxTargetCount(): int
+ GetStatus(): string
+ GetTargetName(target: int): string
1
0..n
SpatialMappingTarget
- m_number_label: GameObject
- m_from_surface: Vector3
+ SetTarget(from_surface: Vector3, target_text, display_id)
+ AttachWorld(world: GameObject)
+ UpdatePosition(scene_object)
+ ShowTarget(bool)

Figure A.1: Class diagram of the situating system.

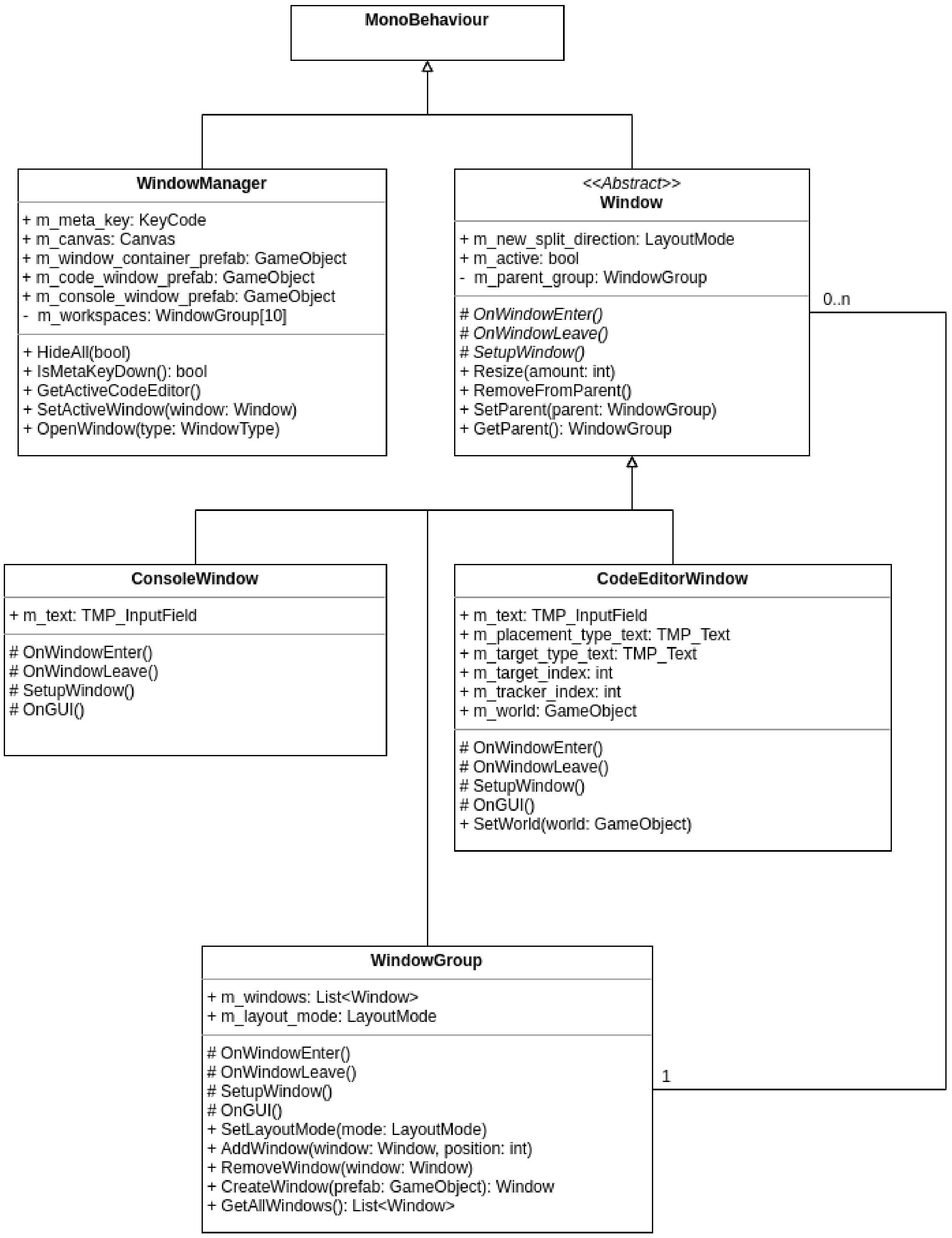

MonoBehaviour
WindowManager
+ m_meta_key: KeyCode
+ m_canvas: Canvas
+ m_window_container_prefab: GameObject
+ m_code_window_prefab: GameObject
+ m_console_window_prefab: GameObject
- m_workspaces: WindowGroup[10]
+ HideAll(bool)
+ IsMetaKeyDown(): bool
+ GetActiveCodeEditor()
+ SetActiveWindow(window: Window)
+ OpenWindow(type: WindowType)
<<Abstract>>
Window
+ m_new_split_direction: LayoutMode
+ m_active: bool
- m_parent_group: WindowGroup
# OnWindowEnter()
# OnWindowLeave()
# SetupWindow()
+ Resize(amount: int)
+ RemoveFromParent()
+ SetParent(parent: WindowGroup)
+ GetParent(): WindowGroup
0..n
ConsoleWindow
+ m_text: TMP_InputField
# OnWindowEnter()
# OnWindowLeave()
# SetupWindow()
# OnGUI()
CodeEditorWindow
+ m_text: TMP_InputField
+ m_placement_type_text: TMP_Text
+ m_target_type_text: TMP_Text
+ m_target_index: int
+ m_tracker_index: int
+ m_world: GameObject
# OnWindowEnter()
# OnWindowLeave()
# SetupWindow()
# OnGUI()
+ SetWorld(world: GameObject)
WindowGroup
+ m_windows: List<Window>
+ m_layout_mode: LayoutMode
# OnWindowEnter()
# OnWindowLeave()
# SetupWindow()
# OnGUI()
+ SetLayoutMode(mode: LayoutMode)
+ AddWindow(window: Window, position: int)
+ RemoveWindow(window: Window)
+ CreateWindow(prefab: GameObject): Window
+ GetAllWindows(): List<Window>
1

Figure A.2: Class diagram of the window system.

A.2 Keyboard shortcuts

Shortcut	**Function**
F1	Show/Hide all windows
F2	Show server configuration dialog and shortcut help
F3	Show/Hide debug information
Alt + Return	Open a new code editor
Alt + C + Return	Open a new console window
Alt + V	Split the selected window vertically if a new window is to be inserted
Alt + H	Split the selected window horizontally if a new window is to be inserted
Alt + 0 - 9	Select workspace
Alt + Arrow keys	Select a window
Alt + Ctrl + Arrow keys	Resize selected window
Alt + Ctrl + Q	Close the selected window
Alt + X	Switch tracker in code window
Alt + PageUp/PageDown	Select next/previous target in code window
F5	Run code of selected code window
Alt + E	Show example preview
Alt + M	Adjust position of visualization/remove visualizations
Alt + T	Add spatial mapping target

Table A.1: Keyboard shortcuts in AVAR-X

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature