UNICA

UNIVERSITÀ
DEGLI STUDI
DI CAGLIARI

Ph.D. DEGREE IN
MATHEMATICS AND COMPUTER SCIENCE

Cycle XXXVII

TITLE OF THE Ph.D. THESIS

End-User Development for Extended Reality

Scientific Disciplinary Sector(s)

INF/01

Ph.D. Student: Valentino Artizzu

Supervisor Lucio Davide Spano

Final exam. Academic Year 2023/2024
Thesis defence session: April 2025

Abstract

In recent years, consumer interest in Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR) is increasing, thanks to improved technology and wider availability. These XR technologies are not just for gaming; they're also being used effectively in education and vocational training. Literature suggests that as XR becomes more accessible, users will increasingly require more control over the content they create and experience within these immersive environments, mirroring the evolution seen with technologies like the Internet of Things and web content management. Creating and customizing content for Immersive XR environments is often limited to technically skilled individuals. This is similar to how the creation of web content was limited before the development of Content Management Systems, such as WordPress.

In this thesis we propose End-User Development (EUD) as a way to empower non-programmers to design and build functional XR experiences. This set of techniques "allow users of software systems, who are acting as non-professional software developers [...] to create, modify, or extend a software artifact" [LPKW06]. Similarly, research has been conducted to explore methods for guiding users without prior XR experience in effectively interacting with these systems, further supporting the goal of making XR development accessible to a broader audience.

The first work proposes ECARules4All, a solution for supporting end users in configuring Virtual Reality environments by exploiting reusable templates created by experts. In the work they are defined the roles participating in the environment development and the means for delegating part of the behaviour definition to the end users. The focus is in particular on enabling end users to define the environment behaviour. The solution exploits a taxonomy defining common virtual objects having high-level actions for specifying event-condition-action rules readable as natural language sentences. End users exploit such actions to define the environment behaviour.

The second part presents XRSpotlight, a system for assisting novice developers in learning how existing VR environments have been developed, so they can exploit these notions to adapt and implement their own systems. The proposed approach also employs a model abstraction that harmonizes existing XR toolkit implementations, offering an interface that facilitates toolkit-agnostic interaction comprehension, particularly for novices.

The third part focuses on ViRgilites, a meta-level user interface designed to guide users in multimodal interactions within VR environments. The system enhances task dis-

coverability, provides feedforward on interaction results, and dynamically recommends interaction modalities based on application context, user preferences, and designer inputs.

The fourth part presents BeatriXR, an adaptation to the previous system, geared towards showing direct feedforward representations of the interactions described in the user interface. These representations can be adapted through a dedicated user interface, or through alternative options made by the creator of the procedure. These adaptations can be also suggested at design time, and triggered to be set up at runtime by specialised LLM models.

Finally, the last part presents a summary of the thesis, drawing conclusions and linking the findings back to the research questions.

Acknowledgements

First of all I would like to thank my supervisor prof. Lucio Davide Spano for his big support during these years. I strongly think that a big part of this goal has been made possible thanks to him, and I can't be thankful enough for his availability, patience and strong support that definitely made the journey as smooth as possible. I'm honoured to have had a great mind and a great person as a mentor, and I truly appreciate the experience that I've had thanks to him.

I would like to extend my thanks to the whole CG3HCI/BatCave lab, that has been my home away from home for many years, even before starting my PhD career, and all the people that came and went through those rooms for working (but also have some fun moments as well). You have been a significant part of my academic life, and I'm pretty sure that I will always bring very fond memories of that period with me.

A very warm thanks goes to prof. Kris Luyten and his team at UHasselt EDM. I don't think I have truly expressed my gratitude and my respect towards you. I am really thankful for the welcome I've received since day one, and the kindness and availability of everyone was well received and appreciated. I will always bring the experience I had in Belgium with me, as a formative part of my career and my life. Thanks for everything, really. I hope I'll be able to visit you again to say hi to everyone!

Out of academia, I want to thank my parents, for being supportive and understanding during these years. If I am where I am right now is also because of you. I hope this achievement will make you proud once more of what I have become.

Also thanks to my late auntie Maria, for being a second mother, for the love and kindness you always gave me, I'll never forget that. You saw me getting my bachelor's degree, you saw me getting my master's degree, you could've seen me closing this PhD chapter as well, but unfortunately you went away too soon. I hope having you in this paragraph will bring you joy, wherever you are right now. Thank you so much for being the wonderful auntie you were.

I want to thank Moonisa for joining my journey mid-career, providing an amazing support and encouragement where she thought she was needed. I really am grateful for your help, and I will never forget it.

Last but absolutely not least I want to give my heartfelt thanks to my friends for being a huge beautiful part of my life, for sticking with me this far (and hopefully even more), for letting my mind take a pause from work and supporting me through this journey, either directly or indirectly.

Finally, one special thanks goes to the Italian Ministry and University and Research (MIUR) who supported this PhD under the “National Operational Program Research and Innovation 2014-2020 (PONRI)” program.

Contents

I Introduction ix

Introduction and Motivation xi
1.1 Structure of the thesis . . . xiv
1.2 Achievements . . . xv

II Related Work and Literature Review 1

2 End-User Development and Extended Reality 3
2.1 End-user definition of VR environments . . . 3
2.2 Rule-based approaches in End-User Development . . . 6
2.3 Discussion . . . 7

3 Empowering Novice XR Developers Through Example-Driven Design 9
3.1 Authoring Tools for Novice XR Developers . . . 9
3.2 Example-Driven Design and Programming with Examples . . . 11
3.3 Toolkits for XR Development . . . 12

4 Feedforward and Multimodality in Virtual Reality 15
4.1 Multimodal Interaction in Virtual Reality . . . 16
4.2 Plastic User Interfaces . . . 17
4.3 Avatar representations for education . . . 18
4.4 Large Language Models for Decision Making . . . 18

III End User Development for Extended Reality 21

5 End User Development in Configurable Environments 23
5.1 An End-User Development Approach for Configuring Virtual Reality Environments . . . 24
5.1.1 The VR Object Taxonomy . . . 25
5.1.2 The EUD Rule Language . . . 28
5.1.3 The Rule Execution Support . . . 30

5.2 Proof-of-Concept Implementation . . . 31
5.2.1 The ECALibrary . . . 32
5.2.2 The Rule Engine . . . 34
5.2.3 Adding Configuration Points to a Scene . . . 35
5.2.4 Configuring a Template . . . 36
5.3 Development Case Studies . . . 39
5.3.1 Case Study 1: Virtual Shop . . . 39
5.3.2 Case Study 2: Virtual Museum . . . 43
5.4 Developer Evaluation . . . 46
5.5 Demonstrating Rules using ECARules4All . . . 50
5.5.1 Demonstrating Rules in VR . . . 50
5.5.2 Preliminary Validation . . . 52
5.6 Discussion . . . 53

6 XRSpotlight **57**
6.1 XRSpotlight Walkthrough . . . 57
6.2 XRSpotlight overview . . . 60
6.3 Toolkit Analysis and modelling . . . 62
6.3.1 Modelling Toolkits and Interactions in the Scene in XRSpotlight . . . 65
6.4 XRSpotlight Implementation . . . 67
6.4.1 Rule Inference Algorithm . . . 67
6.4.2 Rule Similarity Algorithm . . . 69
6.4.3 Copy-Paste Implementation . . . 70
6.5 Toolkit Model Validation . . . 72
6.6 User Evaluation . . . 73
6.6.1 Calibration with XR Development Experts . . . 73
6.6.2 Study with Novice XR Developers . . . 77
6.7 Discussion . . . 81

7 ViRgilites **83**
7.1 Multi-Level Interaction Guidance for Virtual Reality . . . 83
7.2 The Design of ViRgilites . . . 85
7.3 Anatomy of ViRgilites . . . 87
7.3.1 Exploring and Selecting the Current Interaction . . . 89
7.3.2 ViRgilites Placement . . . 89
7.3.3 Navigation aids . . . 90
7.4 Meta-level Interactions: Triggers, Progress and Adaptation . . . 90
7.4.1 Triggering the interaction guidance . . . 90
7.4.2 Adaptation and Prioritisation of the Steps . . . 91
7.5 Architecture of the system . . . 94
7.5.1 Example Scenarios . . . 96
7.5.2 Chemistry Lab Scenario . . . 96
7.5.3 Kitchen Scenario . . . 98

7.6 Formative and User Study . . . 99
7.6.1 Method . . . 100
7.6.2 Results . . . 101
7.6.3 Discussion on Feedback . . . 104

8 BeatriXR **107**
8.1 Direct Feedforward implementation for VR environments . . . 107
8.1.1 Feedforward Guidelines for Virtual Reality . . . 108
8.1.2 Example Scenario . . . 110
8.2 System Design . . . 111
8.2.1 The User Interface . . . 112
8.2.2 The Avatar Representations . . . 113
8.2.3 BeatriXR Capabilities . . . 114
8.2.4 Artificial Intelligence Dynamic Settings Change . . . 115
8.3 Implementation . . . 118
8.4 Validation . . . 121
8.4.1 Validation of BeatriXR . . . 121
8.4.2 Validation of LLM Answers . . . 121

IV Conclusions and Future Works **133**

9 Conclusions **135**
9.1 Future Works . . . 137

Bibliography **139**

A Appendix: VR Object Taxonomy for ECARules4All **157**

B Appendix: System Prompt for BeatriXR LLMs **163**

List of Figures **167**

Part I

Introduction

Introduction and Motivation

In recent years, consumer devices within the domain of Extended Reality (XR), encompassing Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR), have have become more accessible and popular [Vv.]. They offer immersive and fun experiences, mainly for gaming purposes, but they also proved to be effective in other fields, ranging from educational and learning experiences to vocational training environments [SMZW13, RSB13].

As XR technologies advance, it is important to address the barriers that hinder both the creation and seamless use of XR content. As with other technologies in the past, such as the Internet of Things (IoT) and web content management, the more widely adopted a technology becomes, the more users demand better control over creating content [Pat13]. However, the XR field has yet to achieve this level of accessibility.

On the creation side, XR content development remains a highly specialized process, requiring experts in 3D modelling, programming, and design [uni, unr]. Ashtari et al. highlighted the fact that novice developers often struggle with knowing how to start, including selecting appropriate tools or frameworks that align with their goals [ABM+20]. This dependence on professionals not only limits the diversity and volume of XR experiences but also restricts the ability of non-experts to iterate quickly based on their evolving needs [ACF+22, LPKW06].

On the interaction side, once XR experiences are developed, users often face significant barriers to engagement. Muresan et al. [MMH23] identified a series of barriers that users face when approaching a XR environment. These include interacting with objects that have a low perceived interactivity, performing interactions that require a specific sequence of actions (possibly involving multiple controls), setting expectations for interactivity (e.g., for unfamiliar or different types of objects), and discovering action possibilities within VR.

Letting end-users create XR contents is important for a multitude of reasons. The first one sees a democratisation of the access to the technology, allowing more people in the process of content customisation. This fosters innovation and creativity to those who may have unique ideas that cannot put in action by themselves [NOM19]. Another reason is that having tools for faster application allows users to create self-developed solutions that are likely to meet their desired outcomes [PS19].

The research material provided in this document aims to find solutions to these problems using End-User Development techniques. End-User Development “can be defined as

a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact" [LPKW06]. During the early stages of my career, we opted to adopt this paradigm due to its demonstrated success in various domains, such as IoT [MFP+19] and content creation for cultural heritage [BCF+21, FST+19]. This strategic choice has notably contributed to achieving the objectives set over the course of my three-year tenure.

As a result, in the last three years I have developed ways to support users in two main categories: the former sees a number of methods of helping end users in creating XR experiences, without requiring programming expertise, 3D modelling skills, or both; the latter sees methods for allowing users to properly understand how to interact in virtual environments. The general idea was to create systems that allowed the creation of immersive configurations that show the user what they can do. This is achieved through the use of "what you see is what you get" techniques, which can enable the definition of behaviours for Extended Reality experiences, through the use of interfaces that can help the user in creating the environments. The results of this process have been validated through field testing to verify their effectiveness. In particular, the approach chosen for verifying the quality of the outcome was to perform thorough investigation of various potential solutions after researching current literature, identifying crucial points of improvement, and developing novel approaches for filling the gaps found in the field.

The main research challenge of my thesis was to identify meta-design processes [FGY+04, CFMP05, ABC+15] for XR environments, while creating usable tools for supporting the users on the creation of a XR environment, and its behaviour. These approaches envision the participation of end users in a hybrid role of designers and consumers of software artifacts. It defines three hierarchical levels in the creation and evolution of the software, which exchange information among them, exploiting different languages and tools at each level. At *meta-design* level, software engineers, professional developers, or content creators define the core aspect of the software using languages characterized by a high computational power (e.g., Turing Machine equivalent). End users cannot manage such development tools, so we have the highest computational power at this level but the lower usability. At the *design level*, domain experts participate in the design of software, using languages having less computational power (e.g., permitting a limited set of operations), which are usable by non-technical people. The *use* level includes the final users performing the well-defined activities that the software must support. It basically exploits a domain-oriented language with the lowest computational power but the highest usability for users. Just as meta-design principles help end-users be co-designers by providing them tailorable systems, an EUD approach for XR should involve systems where the core XR experience is created by expert developers, while the customization itself is left to end-users, that can use user interfaces or simple scripting for achieving it.

During my PhD career I have been exposed to different perspectives to the same problem. During my stay at UHasselt EDM, we expanded our target user group to domain experts - users with high expertise in their field but without any programming skills. The collaboration with the Belgian colleagues made me able to delve into their research top-

ics, namely *feedforward*. This allowed me to blend their knowledge with my previous research up to that point, to investigate the idea of helping the domain experts becoming End-User Developers, by bridging the gap between the expert developers, which have programming skills but no expertise in the field, and end users, which are trainees that need to learn a new skill, in this case helped through XR environments.

My doctoral research has been driven by the following research questions:

RQ1 What EUD methodologies and tools are effective in enabling individuals without prior experience in XR development to design and construct fully functional virtual environments?

RQ2 Are EUD techniques able to assist novice developers for facilitating them into learning existing examples, so they can apply them to their own XR environments?

RQ3 How can domain experts configure existing XR environments to provide the proper guidance to novices in the process of learning a task?

This thesis provides the following research results:

- ECARules4All, an End-User Development approach for supporting users without programming skills in configuring VR environments, with a Unity Engine [uni] environment as proof-of-concept (see RQ1);
- XRSpotlight, a system for assisting novice developers in learning how existing VR environments have been developed, so they can exploit these notions to adapt and implement their own systems. The proposed approach also employs a model abstraction that harmonizes existing XR toolkit implementations, offering an interface that facilitates toolkit-agnostic interaction comprehension, particularly for novices (see RQ2);
- ViRgilites, a meta-level user interface designed to guide users in multimodal interactions within VR environments. The system enhances task discoverability, provides feedforward on interaction results, and dynamically recommends interaction modalities based on application context, user preferences, and designer inputs (see RQ3);
- BeatriXR, an adaptation of the previous system, geared towards showing direct feedforward representations of the interactions described in the user interface. These representations can be adapted through a dedicated user interface, or through alternative options made by the creator of the procedure. These adaptations can be also suggested at design time and triggered to be set up at runtime by specialised LLM models (see RQ3).

The remainder of this chapter will describe the achievements reached during my PhD career and the structure of the thesis.

1.1 Structure of the thesis

This thesis is based upon the works in the papers [ACF+22, BAFS23, FSAN23, ALRS24, ALRRS24]. These works will be presented in a logical way, so their findings will be clear to understand. This document is organised as follows:

- Part I (this part) introduces the topic of this thesis, the motivations behind my PhD career, the achievements obtained during the last three years and the structure of this thesis;
- Part II provides a background for the works of this thesis, providing a broad view of previous research that eventually led to the contributions shown in part III;
- Part III showcases all the works of the past three years of my PhD career that made me answer the research questions described in the previous section. These works have been peer-reviewed and published in *Engineering Interactive Computing Systems* (EICS) and *Proceedings of the 15th Biannual Conference of the Italian SIGCHI Chapter* (CHItaly '23). One paper is under process of peer review.
- Part IV provides final remarks on the works described before, while also pointing out future directions that could be explored or taken.

1.2 Achievements

The main results and contributions to the state of the art are the following:

- [ACF+22] Valentino Artizzu, Gianmarco Cherchi, Davide Fara, Vittoria Frau, Riccardo Macis, Luca Pitzalis, Alessandro Tola, Ivan Blecic, and Lucio Davide Spano. 2022. Defining Configurable Virtual Reality Templates for End Users. Proc. ACM Hum.-Comput. Interact. 6, EICS, Article 163 (June 2022), 35 pages. https://doi.org/10.1145/3534517

- [BAFS23] Nikita Bellini, Valentino Artizzu, Vittoria Frau, and Lucio Davide Spano. 2023. Poster: Programming Rules by Demonstration in Virtual Reality. In Proceedings of the 15th Biannual Conference of the Italian SIGCHI Chapter (CHItaly '23). Association for Computing Machinery, New York, NY, USA, Article 42, 1–3. https://doi.org/10.1145/3605390.3610819

- [Art24] Artizzu, V. (2024). End User Development for Extended Reality. In: Harrison, M., et al. Engineering Interactive Computer Systems. EICS 2023 International Workshops and Doctoral Consortium. EICS 2023. Lecture Notes in Computer Science, vol 14517. Springer, Cham. https://doi.org/10.1007/978-3-031-59235-5_13

- [FSAN23] Vittoria Frau, Lucio Davide Spano, Valentino Artizzu, and Michael Nebeling. 2023. XRSpotlight: Example-based Programming of XR Interactions using a Rule-based Approach. Proc. ACM Hum.-Comput. Interact. 7, EICS, Article 185 (June 2023), 28 pages. https://doi.org/10.1145/3593237

- [ALRS24] Valentino Artizzu, Kris Luyten, Gustavo Rovelo Ruiz, and Lucio Davide Spano. 2024. ViRgilites: Multilevel Feedforward for Multimodal Interaction in VR. Proc. ACM Hum.-Comput. Interact. 8, EICS, Article 247 (June 2024), 24 pages. https://doi.org/10.1145/3658645

- [ALRRS24] Valentino Artizzu, Kris Luyten, Gustavo Alberto Rovelo Ruiz, and Lucio Davide Spano. 2024. Direct Feedforward Techniques for the ViRgilites System. In Companion Proceedings of the 16th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '24 Companion). Association for Computing Machinery, New York, NY, USA, 87–88. https://doi.org/10.1145/3660515.3662834

- Valentino Artizzu, Kris Luyten, Gustavo Alberto Rovelo Ruiz, and Lucio Davide Spano. BeatriXR: A Comprehensive and Adaptive Feedforward Support for Guidance in Virtual Reality (submitted to IEEE ISMAR 2025).

These publications align with the research questions discussed in the previous section. I contributed in the development, analysis, and writing of these works, while also developing the necessary skills to perform these tasks autonomously.

My work over the past three years allowed me to develop expertise in End-User Development methodologies, Extended Reality technologies, and principles of user-centred

design. This newfound knowledge was essential for the successful results shown in the rest of this document.

Part II

Related Work and Literature Review

Chapter 2

End-User Development and Extended Reality

My PhD research aligns with End-User Development (EUD) approaches, as defined by Lieberman et al. in their work: “EUD can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact” [LPKW06]. This concept is important because it brings the end user to a position where he can be empowered “to develop and adapt systems themselves” [LPKW06]. In general, the main advantage for users consists in developing and adapting systems at a level of complexity that is adequate to their practices, background, and skills [PS19]. On the other hand, professional software developers can rely on a broader adoption, impact, and diffusion of their applications. In this thesis EUD has been applied in the context of XR. XR is an umbrella term that includes VR (“technology that provides almost real and/or believable experiences in a synthetic or virtual way” [Fur08]), AR (“a system that enhances the real world by superimposing computer-generated information on top of it”[Fur08]), and MR (“a continuum that includes both VR as well as AR”[Fur08]). In the following, we summarise the techniques enabling end users to create VR environments and the relevant applications of rules in EUD.

2.1 End-user definition of VR environments

Different commercial and research tools support end users in creating VR environments. Most of these tools are limited to the definition of static scenes or multimedia content overlay. For instance, Hubs by Mozilla [Moz22a], now discontinued, allowed people to create shared rooms for VR communities and groups, while also being able to modify their configuration through multimedia content and appearance customisation. Its focus is on gathering people, so the behaviour of the environment is a second-class concern. A specific tool for Hubs, called Spoke [Moz22b], supported the composition of 3D models for setting up a static representation of a scene through user-friendly tools. It exploited

well mature techniques for end user VR modelling in a desktop setting. It contained tools for positioning, scaling, and rotating 3D models, an extensive catalogue of 3D models ready to be included in the scene, and features for seamless result sharing on the web. Its main drawback was the lack of support for the definition of behaviours beyond the animation, the need for build and test iterations, and the long time required for getting a good quality result.

The research literature focused for a long time on techniques for editing the VR environment while the user is immersed in the virtual world. Steed et al. [SS96] proposed the first environment targeting both the scene configuration and the behaviour definition in an immersive setting. It used a dataflow paradigm and aimed at speeding up the development process by technical people. Lee et al. [LKB05, LK09] evolved such a research thread defining the so-called "immersive authoring", where users created the scenes while immersed into them. Again, such an environment supported the behaviour definition but targeted developers. Takala [Tak14] introduced a toolkit for simplifying the creation of VR, providing a set of building blocks that allows hobbyists and students to create VR experiences rapidly. While the tool lowers the barrier for editing VR, users require technical knowledge for assembling the building blocks through dedicated code. More recently, tools like FlowMatic [ZO20a] evolved such ideas adapting to the newly available devices. It is an immersive authoring tool that allows programmers to specify reactions to discrete events (e.g., user actions and system timers). All these tools have a purpose similar to our project, but their goal is to increase developers' efficiency, reducing the build-test-fix cycle. EntangleVR [CPS21] proposes an innovative representation of the VR scene behaviour inspired by the entanglement phenomenon in quantum physics. The approach is interesting since it lowers the barrier for modelling such a phenomenon, but it also limits the definition of behaviour to a specific case.

We can also find VR tools designed explicitly for end-user development in the literature. For instance, XOOM [GGM+17] is a tool designed for non-ICT-specialists to create web-based immersive VR applications in the cultural heritage field. It contains a simplified behaviour model, defining a restricted set of experiences. The approach we propose in this work raises the ceiling of the possible interaction, also supporting the calibration of the behaviour definition on the domain needs. VR GREP [ZDAM16] is a tool dedicated to end users for the design and development of VR applications. The tool runs in two different modes, one for authoring, and one for running the resulting VR environment. The authoring environment offers means for creating the navigation in the environment, inserting, and manipulating 3D objects. While it provides standard interaction and manipulation techniques for statically defining the environment configuration, the support for behaviour definition is limited to navigation and reaction to button clicks.

There are also attempts to build commercial tools for end-user development. Ottifox [Ott18] is a tool for creating generic VR environments. The tool supports an entirely visual development of the VR environment, and it exploits a simplified version of rules we use in chapter 5, without support for conditions and working only with animations. Other commercial or open-source tools offer support for end-user storytelling or game development. Fungus [Gam20] is an open-source visual storytelling tool designed as a

Unity3D extension. It allows creating visual novels through flowcharts, but the visualization is demanding in terms of screen space, limiting the end user's comprehension [GP96].

We can find examples of tools targeting specifically the end-user development of VR behaviour in *serious game* literature. For instance, Manestrina et al. [MDA17] discuss the definition of an actor programming environment for creating and modifying the behaviour of non-playable characters in serious games. The aim is to shift the changes in the game behaviour with the evolving needs of the game. While we share some goals with this thread in the literature, we aim to cover a more generic representation of VR environments.

Relevant behaviour definition techniques are also available in the literature targeting the editing of interactive 360° videos. While we cannot consider them as full VR experiences, they share the immersion and the need to define how to react to the user's interactions. In this field, Blečić et al. [BCF+21, FST+19] propose the solution closest to the one we present in chapter 5. They created an authoring tool for designing point-and-click games limited to 360° videos, defining the behaviour through rules. We share the same rule-based approach with this work, but we extend it to more complex interactions in full VR environments. Torres et al. [TCW+20] propose a more limited tool, supporting the editing of 360° videos by adding interactive content panels containing information, quizzes, pictures, and 3D models. Mendes et al. [MGM+20] introduce a similar tool working in a different domain. They support teachers and instructors in adding multimedia content for learning purposes. It shares with the previous tool also the limitations in defining the behaviour. Adao et al. [APF+18] introduce a more complex behaviour modelling but is still limited in terms of generic VR interactions. It supports both interaction and time-triggered reactions in the environment, controlling multimedia content and spatial sound. A tool providing support to specific aspects of the experience is Culture4All [MOM20]. It focuses on supporting the accessibility of VR content, providing a platform explicitly designed for cultural heritage, allowing the content creator to use accessibility-related services while authoring the VR content.

Another relevant field for our work is the support for creating VR prototypes. On the one hand, such tools require simplifying the interaction definition and the environment modelling to support rapid design and test cycles. On the other hand, they target specifically designers, who have a higher knowledge of the VR environments' structure than end users. Nebeling and Madier [NM19] support design teams in creating rapid prototypes of both VR and AR. The fast-paced cycle of creating and editing the sketches requires avoiding complex content editing and behaviour programming. The solution proposed in the paper offers a smart combination of pictures and hand-drawn sketches, together with "Wizard of Oz" simulations of the environment behaviour. In a follow-up work called XRDirector [NLC+20], they expand the approach for supporting multiuser authoring using a filmmaking metaphor that assigns different roles to the participants (director, actor, camera). While the result is again focused on producing prototypes and is still a simulation of the object behaviour, it introduces an interesting idea for supporting collaboration between end users in producing VR and AR content. Tvori [Tvo20] is a VR and AR tool for

prototyping environments and interfaces while being inside an immersive environment and with the possibility to collaborate on the same project in real-time.

2.2 Rule-based approaches in End-User Development

Rule-based environment configurations for end users are widely adopted in research work. There are various rule programming styles documented in the literature. One of the most adopted is the Trigger-Action Programming (TAP), in particular in the Internet of Things (IoT) field, both at the academic level [GMPS17, DAM17, CDM19] and in successful commercial tools [UPYHB+16, Inc22c].

Trigger-Action programming addresses users without strong IT skills, so rules assume the following simple pattern: "*if* ⟨a trigger occurs⟩, *then* ⟨an action is executed⟩". The first part describes the event that fires the rule, and the second specifies the action reacting to the trigger. Events arise when an object in the environment changes its state (e.g., the light turns on) or there is a change in the context (e.g., the temperature is below 21°). The action describes a command sent to the same object that triggered the rule (e.g., turn the light off) or another device (e.g., close the door). The rule's action may trigger another rule, leading to a chaining effect. This is useful for creating complex behaviours, but it can also cause unexpected effects [CDRMR19].

Because of their simplified format, TA rules lend themselves well to wizard-style visual interfaces, where the user can specify the desired trigger and action, resulting in the final rule. This metaphor causes problems when the environment has many devices or, in general, when the complexity of the ecosystem becomes high [AMPS22].

An interesting variant of Trigger Action rules includes an *else* block, which provides an alternative set of actions to execute if the event is not satisfied. Coutaz and Crowley [CC16] proposed it in a EUD environment designed to empower people with tools to control their home. Unfortunately, they only provide a preliminary test in their home environment, so the variant requires further research.

Event-Condition-Action (ECA) rules add a third, intermediate part between the event and the action part: a condition defining a guard preventing the rule from being executed if not verified. This part usually checks the state of the environment (e.g., "if the light is on", "if the timer elapsed more than 3 seconds"), or other context-related information. ECA rules distinguish the trigger and the condition by the adverb introducing the rule part: "*when* [something happens] *if* [condition] *then* [action]". Usually, such a format allows triggering more than one action in the *then* part. ECA rules are more expressive than simple TA rules since they may associate different actions to the same trigger event, and they support filtering through the condition part. The higher expressiveness may also lead to the increased complexity of the behaviour definition, raising the ceiling for end users.

The study by Brackenbury et al. [BDR+19] shows that one of the most challenging issues for users in adding the condition part to an ECA or TAP rule consists in understanding the difference between events and conditions. One may exchange them in many situa-

tions, but sometimes there are subtle differences in the semantics that are difficult to grasp for end users. Instead, an erroneous understanding of the distinction between events and states could lead to inconsistencies, loops, and redundancies [CDRMR19]. The research also focused on supporting fixing or detecting bugs in rules through model checking [LBL+16, HCH19, CMT18, ZHM+19]. Currently, the results are limited to enforcing different safety checks (for instance, the fridge's temperature should never exceed 5°C). The solution proposed in [AMPS22] introduces a hybrid ECA-TAP approach, where users can introduce rules through both *where* and *if* adverbs, depending on the result they want to achieve. However, some participants misunderstood the difference among them, showing that we require more than changing the syntax for solving this problem.

ECA rules support the configuration of the environment in different domains. For instance, Barricelli and Valtolina [BV17] exploit them in an interactive visual system for the collaborative management of IoT sensors for improving the quality of life and promoting wellness awareness. Blečić et al. [BCF+21, FST+19] created an authoring tool for point-and-click games, whose logic is defined through a set of ECA rules following a natural language representation. The environment supports the editing through a set of drop-down lists, guiding the user in creating the rules.

2.3 Discussion

End User Development for Extended Reality has been explored through many techniques, and adopted in previous work. While these tools aim to empower users in the creation of XR content, they present some key shortcomings. Tools would either excel at helping define static scenes, but lack of robust support for defining dynamic ones [Moz22a, Moz22b], require technical knowledge for achieving their goals (despite simplifying the definition of behaviours in the virtual environment) [Tak14, ZO20a], present different levels of limitations in functionalities for the definition of the behaviours and environments [GGM+17, ZDAM16, Ott18, Gam20], or be tailored for specific domains and not targeting general environments [GGM+17, MDA17]. ECARules4All, shown in chapter 5 addresses these limitations by offering a novel immersive authoring interface supporting end-user developers in creating customised behaviour of configurable virtual objects in natural language.

Several challenges impede the effectiveness of rule-based approaches in VR authoring. Trigger-Action Programming, although user-friendly, faces difficulties in handling intricate VR interactions, such as comprehensibility in chaining rules and limited expressive capability. ECA rules, despite offering greater flexibility in defining complex behaviors (e.g., associating different actions with different triggers while applying conditions), may complicate the user experience due to inherent complexity, thus reducing user-friendliness. In Chapter 5, we adopted the Event-Condition-Action rules format within a VR environment for this study. This decision aligns with the recommendations in [PS19]: trigger-action paradigm is not enough for managing complex scenarios and automation the end users need to define for configuring VR environments. Moreover, leveraging

natural language and providing authoring support through a user interface that guides rule construction, as exemplified by Bleči'c et al. [BCF+21, FST+19], enables the system to combine the expressive power of ECA rules with the user-friendly nature of natural language.

Chapter 3

Empowering Novice XR Developers Through Example-Driven Design

XR holds high potential for creating engaging end enjoyable environments, but working on an XR system from scratch may be difficult, especially for novice developers or experts in other fields than software engineering. Authoring tools are software applications specifically designed to simplify the process of creating digital content, such as XR experiences. They provide user-friendly interfaces that prioritise accessibility, allowing users without advanced programming skills to design experiences, and this is particularly important when users like instructors may not have a background in computer science [AA24, Gab24]. On the technical side, platforms like Unity and Unreal [uni, unr] help developers in bootstrapping their ideas. The work shown in chapter 6 adds to a recent stream of research on authoring tools to empower novice XR developers, adapts existing streams of research on example-driven design to XR, and analyses the proliferation of new XR development toolkits.

3.1 Authoring Tools for Novice XR Developers

The entry threshold [MHP00, KP05] is one main barrier to the development of XR environments. This was confirmed by two separate studies [ABM+20, KBOR21], which also agreed to identify the usage of online resources and the lack of concrete guidelines and examples as open problems for increasing the number of XR developers. The existing research mainly focuses on creating authoring tools for lowering such threshold, usually paying the increased simplicity with a lower ceiling. These tools are mainly designed for two categories of users: end-user developers and XR designers. In the real world, such distinctions may blend, and often the same person assumes different roles in an XR project [KBOR21]. These profiles have limited development skills in common but differ in terms of the artefacts they each want to produce. While end-user developers typically aim to configure or adapt an existing XR experience, designers usually aim to create a prototype of a new XR experience for experimenting or communicating design ideas.

We can find different examples of authoring tools for developers in the literature. We distinguish between *immersive* authoring tool, allowing XR development while immersed in the environment, and *desktop* authoring, applying the build-test-fix cycle common to development environments. In the immersive authoring [LKB05, LK09], developers have the advantage of editing XR interactions *in-situ*, i.e., directly in the environment where the interaction happens. From the early [SS96] to the latest attempts [ZDAM16, ZDMA16], such approaches support navigation or manipulation interactions having limited consequences in terms of changes in the XR environment state. Desktop authoring tools use a variety of representations for the interaction logic, ranging from graphs [Tec22c, Gam22], block-based programming [CoS22], event-condition-action rules [YKGE21, BCF+21, FST+19], or domain-dependent representations [GGM+17, Gam20]. More recent immersive tools try to increase expressiveness by adapting desktop visualizations in VR. Flowmatic [ZO20b] adapted the graph-based representation [Tec22c, Gam22] reaching a ceiling equivalent to 2D authoring tools. Such notation is very close to scripting and difficult to use for novices. In ECARules4All (See Chapter 5) we proposed an immersive interface to configure *template* environments, specifying the behaviour of specific objects through natural language rules. XRSpotlight employs similar rules for showing to the novice developer how objects behave in an XR environment, but do not use them at runtime. All tools create a different (simpler) representation of the XR environment behaviour, lowering the threshold and the ceiling. This simplified representation enables developers to create new XR experiences, but it also defines the boundary for the types of XR experiences they are able to create. The representation showcased with XRSpotlight is an interaction-focused view on top of a game engine (Unity), guiding developers in finding and understanding the relevant information for building full-fledged XR environments.

Related attempts to lower the threshold are available in prototyping tools for designers. Such tools often try to reuse existing practices, like sketching [NNYR18, NM19, GJSW19, SKW+20], or film-making roles [NLC+20] for obtaining interactive prototypes. Pronto [LNKA20] supports creating AR interaction through video-capturing and supporting 3D manipulations through a tablet to fine-tune the prototype. In follow-up work on Rapido [LGK+21], the authors enhance the tool with a programming by demonstration metaphor for prototyping interactions in AR. The principles are based on video sketching, but the underlying technique uses ARKit to capture the interaction sequences in 6DOF with respect to the environment. Designers may revise the interaction sequences through a state machine representation to author the flow. The opportunity for creating low to middle-fidelity prototypes of the environments is also their main limitation, as creating high-fidelity prototypes will require abandoning the prototyping tool and switching to more advanced tools like Unity [KBOR21]. In this phase of their work, they would take advantage of adapting existing examples and focusing on creating the interactions without the need to master specific interaction toolkits.

Resnick et al. [RMN+05] identify “wide walls” as a desirable property of user interface development tools, besides low threshold and high ceiling. Having wide walls is a metaphor for identifying tools able for a wide range of projects, even different from those

they were designed for. None of the previously discussed tools explicitly position their approach on this aspect. However, prototyping tools [NNYR18, NM19, LNKA20] leverage existing practices for fostering the designers' creativity, and the papers include diverse design examples, showing a good potential support for wide walls. Instead, authoring tools for end-users [YKGE21, BCF+21, ZO20b] and ECARules4All (See Chapter 5) focus more on discussing the significance of the interactions and behaviours they simplify for addressing non-programmers. This does not allow us to position such work on their wall wideness. Even though XRSpotlight is designed mainly for inspecting examples, it supports creative variations through the copy-paste feature. Developers can select and copy interactions from one object in a scene or a project to another, using our proposed interface through the editor (see Figure 6.2), easing the migration of a behaviour. While pasting an interaction on a different object, the tool suggests reconfiguring it to the target scene and even the toolkit. This supports novices in creating variations of the same interaction.

3.2 Example-Driven Design and Programming with Examples

One of the main barriers novice XR developers face is knowing where to start [ABM+20]. They need to understand which toolkit better supports the interaction they have in mind and the effort required for reproducing them. In this regard, finding relevant examples is crucial, but there are few experiences they can draw upon compared to web and mobile development [ABM+20]. Example-based programming focuses on supporting the development by adapting relevant examples [Edw04]. However, finding a relevant example is not an easy problem for novices. Usually, they lack the knowledge to establish patterns and identify the similarities, which complicates the search problem of online resources [FMH+10].

Prior literature for desktop applications presented tools to identify relevant examples for fostering the learning of a language or technology. For instance, Blueprint [BDWK10] supports code search directly in the development environment, augmenting queries with code context, and retaining a link between copied code and its source. Similarly, Rehearse [BPC+10] enable programmers to adapt example code by identifying lines of code relevant to a particular interaction. Hartmann et al. [HWCK07] presented a tool called d.mix, which creates web mashups using already-established websites as examples. Ghiani et al. [GPSP16] allowed to graphically select snippets from webpages and paste them into a mashup application. Zhang et al. [ZG18] further advanced this approach by supporting developers in glueing together the snippets. Ply [LHZO18] supports novice web developers in identifying CSS patterns inside complex stylesheets, hiding rules that are not relevant to the visual feature selected on the current page. Kumar et al. [KST+13] used knowledge discovery techniques to build a design-oriented website repository. AI-based tools appeared recently in this area through embeddable agents suggesting the implementation of entire functions in real-time directly in the code editor [Inc22b]. To

our knowledge, there is no specific support for finding relevant sample interactions in XR, which motivates the research made in chapter 6.

Even though novices can find a relevant example through any search, they need to understand them, i.e., to discover the crucial concept in the example for solving their problem. Ichinco and Kelleher [IK15] identified different obstacles that prevent the correct usage of the example: how to apply it in the destination context, the programming environment hurdle, and the code comprehension. So, novice developers need support in finding an example and a focused representation of the information that makes it relevant. XRSpotlight displays XR interactions as natural-language rules, helping novices identify and understand the information relevant to interactions in an XR example environment. We ease the transfer of its implementation from the example to the experience under development through the copy-paste feature mentioned earlier.

3.3 Toolkits for XR Development

While research is currently experimenting with techniques for targeting different prototyping stages and different levels of development expertise, the current practice in creating XR experiences mainly involves commercial AR/VR platforms such as Unity and Unreal [KBOR21]. These platforms depend on additional toolkits for handling XR interactions, including components, APIs, debugging tools, documentation, and samples. Considering the proliferation of new XR platforms and devices, there is now a large variety of XR toolkits, which makes it difficult for novice XR developers to know about their features and limitations [NS18]. We provide an overview of the currently available toolkits in Table 3.1, considering the supported game engine (Unity or Unreal), if they support AR, VR, or both, and which devices they target. Novice developers need to mature knowledge about XR interactions to orient themselves in this complex toolkit landscape, e.g., to select the most capable and convenient toolkit for their needs and experience.

In chapter 6 we propose a toolkit abstraction extracting the relevant information for supporting novice developers in understanding the interactions in an XR scene. In the development of our tool, we focused on two representative toolkits: MRTK [Mic22b], which covers different types of XR devices (e.g., VR and AR headsets), and SteamVR [Sof22a], which focuses on VR headsets and interactions.

Table 3.1: A list of toolkits currently available for XR development.

Toolkit	**Game Engine**	**XR Support**	**Supported Devices**
ARCore [Goo22]	Unity, Unreal	AR	mobile
ARF [Tec22b]	Unity	AR	AR headsets, mobile
ARKit [Inc22a]	Unity, Unreal	AR	mobile
MRTK [Mic22b]	Unity	VR, AR	VR and AR headsets, mobile
Oculus Integration SDK [Met22]	Unity, Unreal	VR	VR headsets
SteamVR [Sof22a]	Unity, Unreal	VR	VR headsets
VirtualGrasp SDK [AB22]	Unity	VR	VR headsets
VIU [Sof22b]	Unity	VR	VR headsets
VRTK [Ltd22]	Unity, Unreal	VR	VR headsets
XR Interaction Toolkit [Tec22a]	Unity	VR, AR	VR and AR headsets, mobile
XRTK [XRT22]	Unity	VR, AR	VR and AR headsets, mobile

Chapter 4

Feedforward and Multimodality in Virtual Reality

Feedforward is a relatively unexplored concept that it goes on the other side of the temporal spectrum with the concept of feedback. Indeed, we can define feedforward as "what the result of their action will be" [VLvdHC13], while feedback refers to the ability to provide information about what an action did (or is doing). As extensively described by Vermeulen et al. [VLvdHC13], feedforward is grounded on the ability to couple available actions and the possible reactions in terms of time, location, direction, dynamics, modality, and expression [WDO04] and it provides information on what actions are available and how to perform them (*inherent*), the features they activate or their purpose (*functional*), or it may supplement the action possibilities through labels, icons, and other UI elements (*augmented*).

In the literature, we can find different research work covering both theoretical aspects of feedforward [VLvdHC13, WDO04, BEDF15, Nor99, Nor08], and practical applications of feedforward techniques for 2D interfaces [TM02, BM08, GF10, CLV+19]. Considering 3D interaction, Muresan et al. [MMH23] provide a design space on the feedforward design and modelling through a comprehensive analysis of the techniques available for VR, and they illustrate how to design feedforward for specific VR interactions. The design space covers three key stages of feedforward in VR (*triggering*, *previewing actions and outcomes* and *exiting the preview*) and, for each stage, it provides a list of parameters the designer can set for identifying the right technique to use in the considered application. Chapters 7 and 8 show different implementations of the ideas of Muresan et al. In chapter 7 we leverage the categorisation for identifying techniques to be applied in training procedures in XR and supporting procedural adaptation from a simple text-based procedure definition. The goal is to provide automatic feedforward support rather than exploring the design space. In chapter 8 we expand the previous idea by implementing on top of a similar interface direct feedforward techniques. These techniques employ avatars that guide users through interactions via animated demonstrations. The feedforward principles proposed by Muresan et al. serve as potential settings for visualizing these animations. Furthermore, the integration of specialized large language models (LLMs) enables

domain experts to define customized parameters for each action, which can subsequently be suggested to end users during the execution of the experience, where applicable. By accessing both indirect and direct feedforward, we aim to be as comprehensive as possible, regarding user guidance hints.

The application of feedforward techniques in VR environments usually covers the guidance for motor tasks, i.e., showing how to perform movements for completing an interaction. For instance, Fennedy et al. [FHR+21] proposed to use a 3D version of the *Octopocus* technique [BM08] showing a dynamic trace of the available interactive gestures and a label providing information on the effect. *Just Follow Me* shows the motion to be completed to the user as a superimposed ghost that anticipates the movements. Similarly, Lilija et al. [LKH21] limit the ghosting technique to the representation of the virtual hand. *LightGuide* [SBW12] uses superimposed arrows for suggesting the movement direction. Besides the guidance for motor tasks, traces, and arrows have also been used for suggesting the interaction targets, i.e., 3D objects the user can interact with [BPC19]. ViRgilites uses arrows and traces to guide the user towards the interactive object in the scene required for completing the current action. It also allows an automatic teleportation towards the target when far away.

Other studies focused on particular aspects and desired effects requiring the design of a feedforward support. For instance, Chauvergne et al. [CHP23] surveyed the techniques applied during the onboarding in VR applications (i.e., the phase where the application teaches the user how to interact with it). Barathi et al. [BFF+18] studied the effect of the interactive feedforward for enhancing the user's physical performance through a VR video game. We focus on a particular set of VR environments supporting goal-driven tasks, and we generate the feedforward to facilitate the interaction by exploiting the description of the procedure steps. Such environments are often dedicated to "Vocational training", a term referring to "education programmes that are designed for learners to acquire the knowledge, skills and competencies specific to a particular occupation, trade, or class of occupations or trades" [fS12]. Previous research has demonstrated the effectiveness of immersive learning environments for vocational training both using VR [BJSM17] and AR [CSQ22], in different domains such as welding [SMZW13] and construction management [RSB13]. However, these existing systems do not provide learners with the means to discover and select the appropriate interactions and modalities for completing the tasks. ViRgilites eases the introduction of such guidance, considering the designer's preferences, the context of use and, above all, the learner's choice, using plastic user interfaces to show this information.

4.1 Multimodal Interaction in Virtual Reality

The most relevant trait of the interaction in VR is the supported level of immersion in a synthetic environment. While other types of media target the dominant visual and auditory channel [BRP+05, SLVdS20], VR integrates visual, auditory, haptic and, in the most recent work, even olfactory and gustatory [RJTNT+18, MKK+18] channels. We refer

to Martin et al. [MMG+22] for an in-depth survey on the effects of multimodality in VR on perceived realism, user attention, and performance.

This work focuses on the different modalities available for interacting with virtual objects in VR. Techniques relying on the visual channel are the most researched in the literature [LJKM+17, STA+21, Pie18a, Pie18b], and they are quite established and standardised in professional toolkits for VR development. For instance, we adopted the Mixed Reality Toolkit (MRTK) by Microsoft [Mic22b], which offers support for effectively implementing far interactions (based on hand or remote-based raycasting), near interactions (based on poke interactions with nearby objects), object manipulation, locomotion and navigation techniques based on keyboard, controllers combined with the head-mounted display input. Even though these are standardised techniques, not all the objects in a VR environment respond to all the interactions. In addition, we cannot assume in VR the same level of familiarity as in mobile or desktop applications, and a learning phase is usually required [CHP23]. In such a context, interfaces supporting feedforward techniques are needed to guide the user towards proper interactions.

Besides techniques relying on the visual channel, VR hardware supports gesture-based input through remotes or using additional devices such as the Leap Motion or the Microsoft Kinect [YHF+19, JGH+08, WWQ+19]. Another source of input related to the movement of body parts is eye tracking, which is increasingly integrated into VR headsets and allows for pointing through the user's gaze [HGG+19, BMH+20, KKS+19, YLS+21]. In different toolkits (e.g., MRTK [Mic22b]), gaze pointing is usually supported also by head orientation. Finally, voice input allows users to interact with VR environments through natural language sentences or keyword-based utterances [Bil13, HWSW19, FFG+17].

Our work contributes to enhancing the usability of goal-based VR environments by generating feedforward that guides users in selecting the appropriate interaction considering all the available modalities.

4.2 Plastic User Interfaces

We found inspiration in the field of plastic user interfaces, "the capacity of a user interface to withstand variations of both the system's physical characteristics and the environment while preserving usability" [TC99], since we aim to provide a tailored guidance interface according to a 3D environment and the tasks that are supported in that environment. For example, 3DPlasticToolkit [LDA+19] is a toolkit that leverages plasticity to create adaptive 3D user interfaces. It utilises three models to represent the context of use, including hardware, task, and user configurations. The adaptation process is based on a scoring algorithm that considers the context of use to create the most appropriate 3D user interface. We share with this project the idea of using the user environment as a judgement point for deciding which element to present to the user, and the idea of using a scoring mechanism for evaluating the best outcome for a given scenario.

Lacoche et al. present a plastic interaction technique model to use and create interaction techniques that will fit to the needed tasks of a 3D application and to the input and

output devices available [LDA+15]. In ViRgilites we aim to make this type of work more directly accessible by the end-users, and guide users in what interaction techniques are available, and even allows users to control what interaction technique is used.

4.3 Avatar representations for education

Avatars in the context of education environments exploit the use of virtual immersive representations in order to foster personalization, student engagement, and interactivity. These mostly humanoid avatars allow the student to be guided in the learning of a new concept, and literature shows that this form of representation increases engagement [FRE24]. Avatars (and their ancestors, the pedagogical agents, "pieces of educational software with human characteristics that facilitate social learning" [MdW14]) can also support diverse learning needs by providing support and feedback depending on user action, like Herman the Bug, proposed by Lester et al. [LCK+97]. Additionally, 3D avatars can promote active learning through role-playing and scenario-based exercises, where students can practice communication [PD12, BSS+13], decision-making [CL22, HDP+07], and critical thinking skills in simulated real-world environments [CL22]. One interesting example is the first of the two studies conducted in the paper by Bailenson et al. [Tai], where the authors utilized a 3D avatar, specifically a recording of an instructor, to teach participants a selection of Tai Chi movements. The findings indicated that the presence of the avatar enhanced the overall experience of the participants, particularly when they observed themselves performing the movements in conjunction with the avatar, as opposed to solely observing the instructor. This suggests that providing a direct visual reference for participants to align their movements with those of the instructor significantly contributed to the learning outcomes. In recent times, with the advent of LLMs, these avatars have also been improved with the possibility to interact with the user in a way that provides individualized and contextualized instruction and the ability to answer questions promptly and accurately [FRE24]. Despite this type of interactions can be beneficial to the overall experience, LLMs still suffer from issues that may hinder their qualities, namely LLM "hallucinations", where provided data is not accurate, and LLM bias, where provided data is impartial (since LLMs are trained on vast and diverse data that let them inherit "toxic, offensive, misleading, stereotypical, and other behaviors that are harmful or discriminatory")[LGZ+24].

4.4 Large Language Models for Decision Making

LLMs to help users make informed decisions has been an emerging topic since the adoption of Artificial Intelligence (AI)-powered conversational agents. The role of LLMs in these contexts has been increasingly explored across various fields, showcasing their potential to enhance strategic decisions, predictive modelling, and even clinical support. One study highlights the utility of LLMs in clinical decision-making, where models like

ChatGPT support healthcare professionals by analysing multimodal data inputs, such as patient images, to provide real-time insights. However, concerns about safety and ethical alignment persist, suggesting a need for cautious integration into medical workflows [SDL+24]. Another investigation into LLMs, specifically focusing on game-theoretic decision-making, examines GPT-4 and LLaMa 2 in simulated strategic contexts. Findings reveal that these models can adapt strategies based on context and game structures, with LLaMa 2 excelling at understanding nuanced game rules, whereas GPT-4 shows a generalized approach, prioritizing structural over contextual factors [LH24]. In an emergency sensible context, E-KELL [CTT+24] aims to enhance decision support using knowledge graphs and LLMs. The authors proposed a system that stores emergency-related text in a standardised knowledge base, and once queried, it can assist emergency managers in decision support within standards and regulations. They have demonstrated that a system like E-KELL as enhanced the information quality for emergency decision support. Chen et al. [CLSZ23] investigate the economic rationality of large language models using revealed preference analysis to assess the consistency of risk, time, social, and food choices with strategic schemes used to achieve the highest level of satisfaction from economic decisions, finding a high degree of economic rationality, exceeding even human performance in a parallel experiment. Their work suggests that GPTs could have the potential in assisting human decision-making. Brookings and DeBacker [BD23] compared the decisions of LLM models with respect to human ones on two classic games, the dictator game and the prisoner's dilemma, finding that The LLM demonstrated tendencies toward fairness and cooperation, often exceeding the results of human participants. Strachan et al. [SAB+24] tested the performance of large language models on a variety of established theory of mind tests and found that GPT-4 displayed human-level or better performance on tests of false beliefs, indirect requests, and misdirection but struggled with detecting faux pas. Faux pas, however, was the only test where LLaMa 2 outperformed humans.

In the realm of End-User Development for XR, LLMs have been studied for finding ways to include them in systems that allow natural interactions between the application and the user. VRCopilot [ZPG+24] is a mixed-initiative system that integrates pre-trained generative AI models into immersive authoring to facilitate human-AI co-creation in VR. It allows assisted creation of rooms using an AI model that can suggest layouts and furniture, while also giving user agency for further customising the AI choices. This approach can be related to BeatriXR, which also leverages AI to generate alternative feedforward settings and provide feedback to procedure designers, facilitating the creation of more effective tasks. Manesh et al. [AMZO+24] studied through a prototype called Ostaad how users prompt generative AI system while asked to create a scene in VR. They showcased five key lessons, derived from their explorations: i) Programming agents should have embodied knowledge of the environment; ii) Programming agents should understand embodied (e.g. with body language like positioning and gesturing) prompts; iii) Programming agents should retain a conversational memory of their interaction with the user and the scene; iv) Programming agents should have a common sense knowledge of objects in the scene; v) Programming agents should acknowledge and provide timely feedback of users' prompts while users are prompting. LLMR [DLTFH+24], is a framework that enables the

real-time creation and modification of interactive Mixed Reality experiences using LLMs. This framework leverages the abilities of multiple distinct and specialized LLM modules, orchestrated in a way that enhances their individual and collective performance on both coding and reasoning. A similar project, DreamCodeVR [GNGS24] uses AI for generating behaviours inside a VR scene. The system enables the creation of components that modify the VR scene through the use of voice commands, that generate actual code that is attached to the environment after a process of compilation.

These works provide different perspectives that, in the context of this thesis, helped shape the characteristics for the implementation of AI in BeatriXR (see chapter 8), where the AI model is tasked for suggesting different options for the domain expert (the End-User Developer of the project) at design time and, when the end user is interacting with the system at use time, it decides whether to suggest the modification of the user settings or not, leaving the final choice to the user.

Part III

End User Development for Extended Reality

Chapter 5

End User Development in Configurable Environments

In this chapter, we propose an End-User Development approach [LPKW06] for supporting the configuration of a VR environment by users without skills in programming and/or 3D modelling. The approach includes three roles inspired by meta-design [FGY+04, CFMP05, ABC+15], collaborating for transforming the template of a VR environment created by an expert into a peculiar VR experience configured by an end user. From other works and systems, it is possible to configure an environment through scene builders or inspectors, which allow placing and orienting 3D models that constitute the virtual world in both immersive or desktop mode. We can find different solutions for this task, both in the literature and in commercial products [ZDAM16, ZDMA16, Goo17]. The most challenging part is still the definition of the dynamic behaviour of an environment, i.e., responding to the interaction with a user or other objects. We detail the engineering of a solution dedicated to expert users for including configurable objects in a VR environment, whose behaviour is defined by end users through Event-Condition-Action rules expressed in natural language. The solution is adaptable to different VR engines. It supports i) the rule language specification, ii) the definition of configuration points inside a VR environment, and iii) the execution of the rules at run-time, without requiring further builds of the VR environment. We report on a proof-of-concept implementation of the solution in Unity3D, the design of two sample templates (a virtual shop and a virtual museum), and their configuration for supporting two different experiences each. Finally, we discuss the results of a user study assessing the utility and the usability of the proof-of-concept in defining a template.

We organised the chapter as follows. First, we discuss the related work on EUD approaches for VR, and we summarise rules applications in EUD (Section 2). Then, we introduce the general concepts and roles in the EUD approach, detailing the components of the proposed solution (Section 5.1). After that, we describe a proof-of-concept implementation (Section 5.2), the development of two sample templates and their configuration through rules (Section 5.3). Then, we report on the results of a user study involving expert VR developers (Section 5.4), and we discuss the contributions and the limitations

in our approach (Section 5.6).

5.1 An End-User Development Approach for Configuring Virtual Reality Environments

Applying the principles of meta-design described in section 6, requesting an end user to build an entire VR environment is unfeasible. Instead, it is reasonable to apply a solution similar to Content Management Systems for publishing web content: end users download website templates, working out of the box but including dummy content. End users exploit specific languages and interactions for configuring it for their purposes. Therefore, we expect end users to start from predefined VR environments to configure and adapt them to their needs. In the proposed workflow, the environment prepared by experts represents a solution adaptable to different settings by configuring its behaviour and adding content in predefined points. They will not resemble a complete, final version of an interactive VR environment, but rather a *template*, which end users can tailor to their needs. Therefore, the same template may fit the need of many end users.

The main problem is defining the language at the *design level* for configuring such templates. As already pointed out in Section 2.1, while we already have usable solutions for moving the content around an existing VR environment, we do not have an equivalent for its behaviour. In this chapter, we propose to borrow the knowledge developed in the last years of research in rule-based EUD, especially in the IoT domain (see Section 2.2). We use Event-Condition-Action (ECA) rules for defining 3D objects' behaviour in isolation and their interaction with other objects. The main contribution of this work is the definition of components enabling rule-based configuration and the rule runtime support for VR.

In summary, our solution foresees three different roles:

- **Template Builders** (TB), which represents users with good skills in both 3D modelling and game programming. We assume that they are proficient with game programming platforms and can build complex VR environments. TBs create the *templates*, which are almost-complete VR environments, open to end-user configurations. They represent the experts of the *meta-design* level.

- **End-User Developers** (EUDevs), which represent users without skills in 3D modelling and game programming, but having an average familiarity with computer use (e.g., proficient in using standard office programs) and with VR environments (e.g., they played 3D videogames or have VR experience as users). They represent people who may require creating VR content for their business or leisure (e.g., touristic promotion, content advertisement, etc.). They are supposed to have a limited budget for creating the content or, even if they have the budget, they need to modify the content by themselves (e.g., they update the VR content frequently). They download or buy templates to configure for getting the final version of the VR environment. They represent the hybrid designer/consumer at the *design* level.

- **Users**, which represent the final consumer of the VR contents, either created by professionals or by EUDevs. They represent the final user of the *use* level.

In Table 5.1, we introduce a sample configuration scenario for concretely explaining the concepts in our approach. It partially covers one of the case studies outlined in Section 5.3. It will ease the understanding of the general solution in the current section and the proof-of-concept implementation in Section 5.2.

> *We suppose having a template representing a virtual clothing shop, including products (e.g., t-shirts, hats, shoes etc.) and multimedia elements for displaying additional information (e.g., text panels, virtual monitors for videos etc.). The EUDev wants to configure the environment for displaying a text panel containing the price when the user picks a pair of shoes.*

Table 5.1: A simple template configuration scenario.

5.1.1 The VR Object Taxonomy

For creating a usable configuration mechanism, EUDevs need a language for specifying the VR environment behaviour they can understand and manipulate. Even if we take inspiration from the rule-based configuration in IoT, there is a point that marks a substantial difference between the two domains. In an IoT environment, the capabilities of the physical devices define the actions they can perform. Two instances of the same device have the same associated actions. In VR, this is not true. The Entity Component System architecture [Mar07], used by most VR engines, allows fine-grained control of the behaviour for each object instance in an environment. The component list defines what objects can do, not their physical appearance. So, we can have objects in the environment that look the same (e.g., they share the same 3D model) but behave differently, as typically happens in games.

Therefore, before introducing a rule language defining *when* objects should do something, a TB needs a way for specifying *what* an object in the scene can do. Such a definition must include actions understandable by EUDevs, so it must avoid technical concepts and jargon, and rely on a general naïve understanding of the world and knowledge of the particular domain the VR environment models. On the one hand, we expect different domains to require performing different actions. On the other hand, if TBs need to map all the necessary internal components to high-level actions, adopting the solution would be unpractical. Therefore, we opted for providing TBs with a taxonomy of reusable high-level types for building VR environments templates. The taxonomy does not claim to completely represent all the possible VR object categories, but it defines a reusable starting point. Its effectiveness depends on an appropriate coverage of common VR concepts, but also on domain-dependent extensions, which we must support at the implementation level (see Section 5.2.1). For creating the concept list, we analysed the literature on

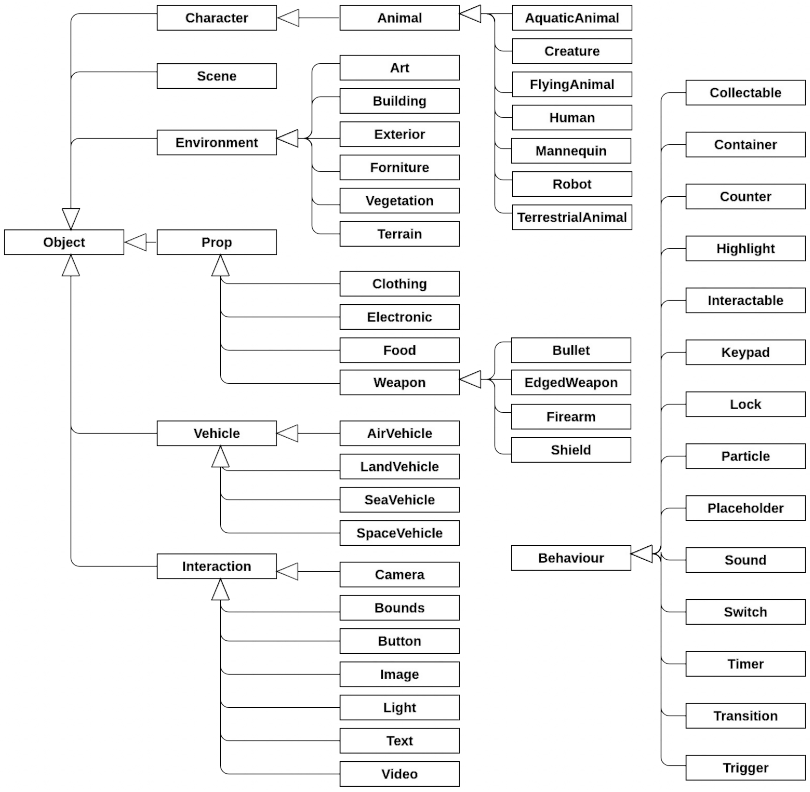


Figure 5.1: Class diagram for the object categories in the VR templates.

model-based approaches for VR development (e.g., [GCVMA09]), and the categorisation used by 3D models and assets repository, such as SketchFab, TurboSquid, Unity Asset Store etc. We briefly introduce the main concepts in this section, and we provide a more detailed description in Appendix A.

Figure 5.1 shows a class diagram depicting the proposed taxonomy. Its types define a coarse representation of virtual objects categories. In our approach, TBs assign such categories to each virtual object instance, while the EUDev can exploit the actions associated with each category for configuring their behaviour. Such assignment provides the following pieces of information to both the EUDev and the runtime support: i) a semantic tag for the game object, specifying the category it belongs to; ii) the specification of a set of status variables, whose control allows changing the object's state dynamically; iii) the specification of a set of actions associated with the object, which are the building blocks the EUDev uses for programming the VR environment behaviour.

In our taxonomy, we make a distinction between *Objects* and *Behaviours*. The difference between the two categories is related to their physical appearance, which drives the assignment: the former are the things a user can perceive, while the latter are common behaviours we can identify across the different categories of objects. We categorised perceivable objects according to an ontological criterion: each object instance belongs to a

single category, and we cannot assign it to multiple ones. For instance, a model whose shape resembles a chair belongs only to the *Furniture* category, and we cannot assign it, for instance, to *Food*. However, we can add the *Container* behaviour to the same chair, meaning that we can put other objects in it. If we consider a sandwich, we can assign it to the *Food* category, but we can consider it also as the *Container* of its ingredients. Therefore, *Food* and *Furniture* are *Objects*, the *Container* is a *Behaviour*.

The object categorisation is coarse. We limited the hierarchy's depth to high-level categories, stopping when we considered that going further would not add useful information for the EUDev. For instance, we assign an object representing a cat to the *Terrestrial Animal* category. We did not create a specific sub-category for the concrete species. The reason is that a hypothetical *Cat* category would have the same status description and actions of a *Terrestrial Animal*. However, such simplification on the object taxonomy would cause usability issues for EUDev in creating the behaviour rules: they will struggle in reasoning about objects belonging to abstract categories. So we provided TBs with means for specifying EUDev-friendly aliases for a category name, which will appear on the rule representation in natural language (see Section 5.1.2). If a TB defines an alias, the EUDev can refer to the hypothetical *Cat Tom*.

The *Behaviour* category models typical interactive behaviours we can assign to different objects, independently from how they appear. It is possible to assign more than one *Behaviour* to a virtual object, composing the set of possible actions the EUDev can program. In general, a configurable virtual object is associated with one *Object* category and a list of *Behaviours*. For instance, a TB can assign a *Collectable* and *Sound* interaction to a piece of *Furniture*. In this way, the EUDev can define rules that, for example, collect the piece and play a sound when the main character bumps on it. The final set of actions that a EUDev can configure on a single object consists of the union of all the actions defined by the *Object* and the different *Behaviour* components.

We have categories representing dynamic attributes of the 3D scene (*Camera*, *Lights*, *Transitions*), simple techniques for defining environment manipulation and automation (*Bounds*, *Container*, *Collectable*, *Counter*, *Keypad*, *Switch*, *Lock*, *Timer*, *Trigger*) and simple representations of media content (*Image*, *Sound*, *Text*, *Video*). For instance, considering a *Container*, it holds the information about the capacity and the number of objects currently it contains. It has the actions for inserting and removing objects or for emptying it.

A behaviour requiring a particular mention is the *Placeholder*. We introduced it to support TB in explicitly defining extension points in the template where the EUDev can insert custom content. We include a sample using such a category in Section 5.3.

Considering the sample in Table 5.1, the environment contains at least a *Character* representing the shop visitor (i.e., the user of the VR environment), *Clothing* objects representing the shoes and all the other products, *Text* objects representing the information panels for showing the price. In addition, all products are associated with a *Placeholder* and an *Interactable* behaviour. The former allows the EUDev to set a custom 3D model for each product. The latter allows the users (i.e., the shop visitors) to pick and release the products.

5.1.2 The EUD Rule Language

The rules EUDevs use for configuring the VR environment behaviour have a structured definition, i.e., they follow a precise syntax scheme, but are readable as natural language (English) sentences for grasping their meaning.

The complete grammar for the rule language is available on in the GitHub repository[1], defined using AntLR[2]. In the following, we report some excepts using uppercase words for identifying terminal tokens (e.g., `DEFINE` for the token "define"). Listing 5.1 shows the grammar for a single rule, including two sample structures: a rule including an event and a single action, and one including an event, a condition and multiple actions.

```
// rule grammar
rule : WHEN action (IF condition)? THEN (action)* ;

// sample structure: no condition, single action
when [action1] then [action2]
// sample structure: single condition, multiple actions
when [action2] if [condition] then [action 3] [action 4] ... [
    action n]
```

Listing 5.1: Declaration of a rule

A rule follows the event-condition-action (ECA) structure. The `when` keyword introduces the *event* part, described through an `action`. An action is associated with every method supported by classes in Figure 5.1, or domain-dependent extensions. So, a EUDev can specify rules that trigger whenever a managed object in the VR environment does something. The syntax representing the action that triggers a rule is the same for describing further actions the EUDev enters to define the environment response, in the `then` part. This eases the identification of actions as commands and event sources. The `then` part of a rule corresponds to the *action* part in the ECA schema. EUDevs can insert multiple actions in the `then` part, including multiple changes in response to a given event. All these actions, in turn, will raise the associated events, possibly triggering other rules. One important trait for the language is the incorporation of object categories directly into the rule definition to guarantee comprehension of their functions (e.g. "The vehicle car" instead of "car"). This measure was implemented to mitigate the risk of ambiguous identifiers rendering the rule unclear, upon inspection through their human-readable form. We report in Listing 5.2 six sentence schemes we identified, together with an example for each of them.

While the first five are straightforward, the last one requires some explanation. Passive actions are associated with the object type that undergoes the action, but they increase the available actions for the subject in the rule language. This happens when, considering a `[subject] [verb] [direct object]` structure, we have the implementation of the verb logic on the class representing the direct object. This sometimes makes sense for separating the concerns from an engineering point of view. We have an exam-

[1] https://github.com/cg3hci/ECARules4All
[2] https://www.antlr.org

ple of this type of action in the *wear* method of the *Clothing* object, which requires a *Character* parameter. By keeping its implementation in the *Clothing* class, we decouple the basic actions of a character from those involving other objects. Otherwise, given that users in the VR environment are *Character*s, it would become the class defining the large majority of all the possible actions. In contrast, from a EUDev perspective, the best description in natural language is "the character wears the clothing" and not "the clothing is worn by the character". By using a passive action, the owner is the type of the direct object, while it uses as a parameter the subject of the sentence in natural language.

```
// schema 1: [subject] [verb]
THE LANDVEHICLE IDENTIFIER STARTS
the land vehicle car starts;

// schema 2: [subject] [verb] [direct object]
reference: THE object IDENTIFIER;
THE character IDENTIFIER LOOKS AT reference
the human visitor looks at the art painting;

// schema 3: [subject] [verb] [value]
THE character IDENTIFIER JUMPS ((TO position) | (ON path)) ;
the human visitor jumps to the position entrance;
the human visitor speaks "Hello world!";

// schema 4 (state change): [subject] changes [property] to [value]
THE ART IDENTIFIER CHANGES AUTHOR TO STRING_LITERAL;
the art painting changes author to "Picasso";

// schema 5 (number increase/decrease): [subject] increases [
   property] by [value]
THE character IDENTIFIER (INCREASES | DECREASES) LIFE (BY
   floatLiteral)?
the human player1 increases life by 2;

// schema 6 (passive action): [subject] [verb] [direct object]
THE character IDENTIFIER WEARS THE CLOTHING IDENTIFIER;
the human player1 wears the clothing tShirt;
```

Listing 5.2: Different types of action specification

Each rule has an optional *condition* (the `if` part), which defines the conditions for executing the `then` part. End users usually avoid writing conditions, or they use a simple check on a single property [UPYHB+16]. Given this fact, many ECA languages for end users limit the condition to a single predicate. We support the definition of composite expressions in the language specification, but in the implementation of the rule-editing interface we support only one level of *and/or* on simple checks to avoid clutter. The solution resulted in a useful compromise between expressiveness and the interface usability already validated in other tools [BCF+21, FST+19].

Listing 5.3 shows the rule defining the custom behaviour required by the scenario in Table 5.1. The event triggering the rule is the *pick* action, whose subject is the character

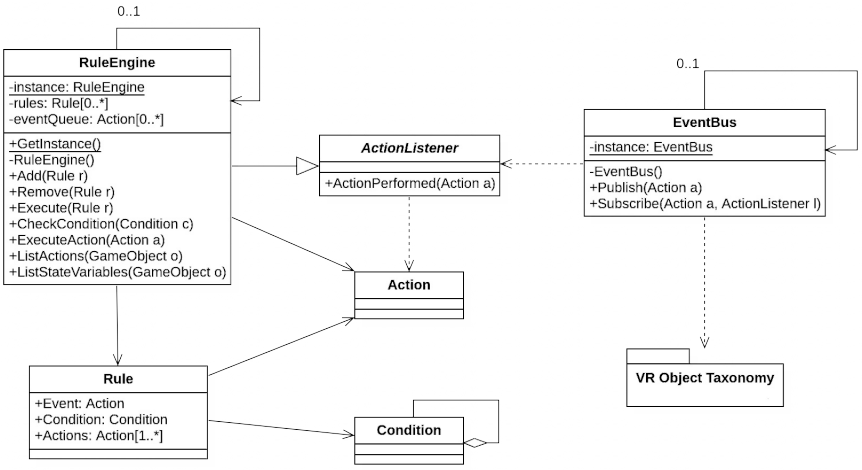


Figure 5.2: Class diagram for the ECA rule engine.

representing the user of the VR environment. Its direct object is *sneakers4* one of the *Clothing* objects representing the selling products. There is no need to specify further conditions for triggering the rule, so we omit the condition part. The reaction of the environment to the event is showing the information panel, called *infoPanel2*, which is a *Text* object.

```
when    the character visitor picks the clothing sneakers4
then    the text infoPanel2 shows
```

Listing 5.3: A rule specifying the behaviour required by the scenario in Table 5.1

5.1.3 The Rule Execution Support

The last component in our solution is the support for executing the rules at run-time. Since it is not feasible for EUDev to build the environment each time they change the rule definition, they need a specific run-time component inside the VR template, which we call *Rule Engine*. At a high level, it has two tasks: triggering and executing the rules. We provide here an abstract description (i.e., independent from the VR engine) of the classes we used for solving this problem, depicted in Figure 5.2.

We start from an object-oriented description of the rules, consisting of three classes: `Rule`, `Condition` and `Action` (see Figure 5.2). The class `RuleEngine` is responsible for managing and executing such a rule description. We have a single instance of such class in the system (singleton pattern [GHJ+95]). The class has methods for adding and removing rules from the configuration set (`add` and `remove`).

A `Rule` contains the `event` field, specifying the action that triggers the rule. A `Condition` is an object describing either a simple predicate or a tree composing such predicates through boolean operators. The predicates need to be tested against the values contained in the state variables. Both action and the condition description contain

references to the VR objects in the environment (either memory references or identifiers to lookup in the object repository). Executing a rule means first checking the condition on the current state of the VR objects and then invoking the method corresponding to each action, passing references to objects or the values specified in their definition.

Besides defining how to execute the rules, the *RuleEngine* also defines *when* to execute them. The triggers are the same actions the EUDevs use for specifying the behaviour. This means that each action in a rule definition can potentially trigger other rules. So, after finishing the execution of an action, the rule engine needs to check whether there are rules to trigger or not. The rule engine requires a notification each time an action completes. If such action triggers a rule, the engine enqueues it and continues the execution loop until the queue is empty. In Figure 5.2, we adopted the *publish-subscribe* pattern (a variant of Observer [GHJ+95]) to avoid the proliferation of interfaces defining callback methods. The `EventBus` class represents a channel abstraction where it is possible to publish messages on different actions and to register for receiving them. The publishers are the classes implementing *Objects* and *Behaviours*, which will send a message on the `EventBus` each time an action they implement is completed. The `RuleEngine` class represent the subscriber, which registers for notifications from each action in the *when* clause contained in the environment rules. The reaction to the notification enqueues the rule corresponding to the message and starts the execution loop if needed.

It is worth pointing out that there are two main causes for completing an action. The first is the `executeAction` method in the `RuleEngine`, which occurs after triggering a rule as a consequence of the completion of another action. The entry point in the rule execution loop is the second cause, which is an interaction between the user and the environment or between two game objects. The responsible for raising such notifications is the implementation of the taxonomy types. It must define a mapping between the internal state of the VR objects in the engine and its high-level representation for the EUDev and vice-versa, and publish messages on the *EventBus* whenever the values change.

Considering the scenario in Table 5.1, the `RuleEngine` contains a single *Rule* object, including an *Action* instance (picking the shoes) in the event field and another instance (showing the text panel) in the actions list. The rule triggers as a consequence of the interaction between the user and the environment. Thus, the *Interactable* implementation is responsible for publishing a `pick` message on the `EventBus`. The `RuleEngine` receives the notification and enters the rule execution loop, it performs the action corresponding to showing the information panel, checks whether it triggers other rules and, supposing that there are no further rules to execute, it ends the loop.

5.2 Proof-of-Concept Implementation

In this section, we discuss the main properties of the implementation we provided for the solution depicted in Section 5.1. We selected Unity3D[3] as VR engine and C# as de-

[3] `https://unity.com`

velopment language. The discussion has two main objectives: the first is demonstrating that the approach is feasible, while the second is showing concrete opportunities for customizing the approach for different needs through extensions. The plugin relies on the standard Unity XR Interaction Toolkit for supporting VR interactions, minimizing the requirements for external libraries. We developed the other functionalities leveraging objects directly provided by the game engine. The code is available on GitHub[4], together with a short demonstration video [5].

5.2.1 The ECALibrary

The `ECALibrary` contains the implementation of the classes included in the VR taxonomy introduced in Section 5.1.1. A TB can assign a given `GameObject` (the generic VR object in Unity) to at most one category. It is not mandatory to assign each `GameObject` to a category. The TB assigns only those that the EUDev can further configure.

The root of the simplified object hierarchy we created for representing VR environments to EUDevs is the `ECAObject` class. It contains the variables and actions common to all categories. The same applies to the `ECABehaviour` component and its sub-classes, which model the interactive behaviours. They are basically a C# implementation of the taxonomy described in Section 5.1.1.

Both the `ECAObject` and the `ECABehaviour` are components in the Entity-Component-System architecture supported by Unity. They are sub-classes of `MonoBehaviour`, the abstract component in Unity. We do not use inheritance for expressing the category hierarchy, which would break the composition in the ECS architecture. However, we specify that the implementation of a component requires another one through the `RequireComponent` class-level attribute. We implemented the inheritance in the taxonomy through such an attribute.

An interesting point to discuss here is the effort we put into implementing the rule execution support that does not assume any object categorization, keeping its definition agnostic of the types in the `ECALibrary`. The only distinction we assume in the categorization is between objects and behaviours. The other pieces of information about the category or behaviour type name, the list of state variables, and the exposed actions are included as metadata in the component class implementation, declared as C# custom attributes[6]. Given such a structure in the implementation, the following is both the description of how we defined the taxonomy in Section 5.1.1 and of how third-parties can enhance the set of available categories.

The first step for including a component among those managed by the rule engine is specifying a type name that a EUDev can understand. For doing this, we use the `ECAType` class-level attribute, which marks it as a managed component and provides the runtime (see Section 5.2.2) with the information on the type name.

[4] `https://github.com/cg3hci/ECARules4All`

[5] This video can be found in the supplementary material of the original paper submission

[6] `https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/attributes/creating-custom-attributes`

```
[ECAType("human")]
[RequireComponent(typeof(Animal))]
public class Human : MonoBehaviour { ... }
```

Listing 5.4: Annotating a Unity Component for managing it at runtime through ECA rules

Then, the rules run-time needs information about the variables that maintain the state and the methods implementing the actions. For defining the state variables, we provide another custom annotation (`ECAStateVariable`) marking public instance variables or properties of a C# class. A state variable has of the following base types: `Boolean`, `Color`, `Float`, `Integer`, `Position`, `Path`, `Identifier`, `Rotation`, `Text`, `Time`, having a straightforward meaning. Listing 5.5 shows two sample annotations defining two state variables in the `Character` object. The first annotation marks a flag for identifying the character controlled by the final user. The annotation defines the name of the object variable as presented to the EUDev and its type in the rule system. The second annotation defines the variable containing the life points for a character, stored as a float.

```
[ECAStateVariable("playing", ECABaseType.Boolean)]
public ECABoolean playing;

[ECAStateVariable("life", ECABaseType.Float)]
public float life;
```

Listing 5.5: Annotating the instance variables of a Unity Component for modifying them through rules

Finally, a third custom attribute (`ECAActionAttribute`) marks the actions that a given object or behaviour provides to EUDevs for defining the environment behaviour. It has a constructor for each action type we identified in Listing 5.2, defining the rule syntax. It specifies the allowed taxonomy type for each part of the action. Listing 5.6 shows three examples of recurrent action definition patterns. The first defines the *swims to* action for an aquatic animal, which moves it to the position specified in the parameter. The second allows any *ECAObject* to look at (i.e., turn to) another object. Finally, the last sample is a passive action. Differently from the previous ones, the action belongs to *Character* type in the taxonomy in the EUDev syntax, but the method is actually defined in the *ECAFood* class, which is the implementation of the *Food* type.

```
[ECAAction(typeof(ECAAquaticAnimal), "swims to", typeof(ECAPosition
    ))]
public void Swims(ECAPosition p) { ... }

[ECAAction(typeof(ECAObject), "looks at", typeof(ECAObject))]
public void Looks(ECAObject o) { ... }

// the following method belongs to the class ECAFood
[ECAAction(typeof(ECACharacter), "eats", typeof(ECAFood))]
public void EatenBy(ECACharacter c) { ... }
```

Listing 5.6: Annotating a Unity Component instance variable for modifying it through rules

In order to implement the sample configuration scenario in Table 5.1, the player in the Unity scene is associated to the `ECACharacter` component, the virtual object representing the shoes contains both the `ECAClothing`, the `ECAPlaceholder` and the `ECAInteractable` components, while the virtual information panel contains the `ECAText` component. They define the actions and the status variables required for implementing the logic in Unity (e.g., the visibility boolean, the text in the information panel etc.).

5.2.2 The Rule Engine

The `ECARuleEngine` is the class responsible for executing the rules defined by the EUDev. When a template configuration starts in the VR experience mode, it loads the rules from a text file and builds their object-oriented description. The implementation of the engine does not assume any structure on the ECAObject types, but it dynamically obtains it reading the custom attributes. Thus, no change to its implementation is required if a TB extends the set of available objects or behaviours. This means that extending the taxonomy has no impact on the engine, and the new rules are ready to be used by EUDevs.

An `ECARule` is the C# representation of a rule. An `ECACondition` is an object describing either simple or composite predicates on managed objects. The action and the condition description contain references to `GameObjects` in the VR environment. For instance, if we consider the sample action "`the human visitor looks at the art painting`" (*[subject] [verb] [object]*), the object-oriented description will contain a `GameObject` reference having id *visitor* and containing the component `ECAHuman`, a reference to its `LooksAt` instance method and a reference to a `GameObject` whose identifier is *painting*.

Supposing that all these pieces of information are available, the problem of writing a method in the `ECARuleEngine` class for executing a generic action is simple: we receive an `ECAAction` instance as a parameter, and we invoke the referenced method on the subject via reflection. We pass the direct object as a parameter and, in the more general case, all the other values or references requested by the six action types we identified (see Listing 5.2).

We assume that rules and Unity share the same virtual object identifiers and that the `GameObjects` contain the components from the `ECALibrary`, specified by TBs. In such a case, identifying the components and the VR object references is trivial. The last information we need is related to the methods that implement the actions and the instance variables that represent the state in a VR object. Having the custom attributes identifying both state variables and actions we discussed in Section 5.2.1, we can inspect a `GameObject` instance using C# reflection as shown in Listing 5.7. We loop over all the components associated with the object, and we look for those marked with the `ECAType` annotation, indicating a managed type. Then, we go down looking for action methods or, symmetrically, for state variables. In this second step we search for the `ECAActionAttribute` at the method level. For state variables we look for the

ECAStateVariable on instance variables (or properties). Once we found one, we add the action or state variable to the ones supported by the current VR object instance.

```
foreach(Component c in obj.GetComponents())
{
    Type cType = c.GetType();
    if(Attribute.IsDefined(cType, typeof(ECAType)))
    {
        // we found a managed type
        foreach (MethodInfo m in cType.GetMethods())
        {
            ECAActionAttribute[] actions = (ActionAttribute[])
                m.GetCustomAttributes(typeof(ActionAttribute), true
                    );
            foreach (ECAActionAttribute a in actions)
            {
                actionMethodList.Add(a, m);
            }
        }
    }
}
```

Listing 5.7: Finding all the actions associated with a GameObject instance. We follow a symmetric procedure state variables.

The object-oriented representation of the rule in Listing 5.3 in the proof-of-concept implementation is the one described in Section 5.1.3. The reflection-based implementation of the engine requires including into ECAAction references to game objects for the subject and the direct object, while the verb field corresponds to a method reference (i.e., an instance of the MethodInfo class). The event references the player as subject, the pick method as verb and the virtual shoes object as the direct object. The action references the text panel as subject and the shows method as verb. Such structure allows theRuleEngine to find the rule to execute by comparing the ECAAction instance received in the notification with the event field of each ECARule in the list. In addition, it allows executing the actions by invoking the verb method through reflection.

5.2.3 Adding Configuration Points to a Scene

This section briefly details the operations that a TB should perform to make a generic Unity Scene configurable by a EUDev, transforming it into a template.

The first step is loading the plugin into the Unity project. The plugin consists of a simple Unity prefab, called ECAKit that a TB can simply drag among their assets for empowering a scene with the ECALibrary solution. Prefabs in Unity are game objects persisted with all their components and configurations that may be loaded in different scenes.

The second step the TB needs to perform is assigning one of the Unity Components described in Section 5.2.1 to all VR objects he would like to open for EUDev configurations. Such assignment may also include the library extension with ad-hoc objects or behaviours.

Finally, the TB creates an executable version of the template, performing a standard build of the Unity project. TBs will share the result with the EUDevs for the configuration. No further build is required after creating the template.

5.2.4 Configuring a Template

This section briefly describes a proof-of-concept interface we implemented to provide EUDevs with support for configuring a template. After a TB creates the template, the changes specified by a EUDev are stored as external assets of the template. They are dynamically loaded at the environment startup or immediately after they are defined. This allows distributing a template as a simple Unity-based application without any build step for the configured environment. The resulting template executable application runs in two modes:

- In the **VR experience** mode, the template loads the EUDev configuration and supports experiencing the resulting VR environment, executing the specified set of rules in response to the user's interactions. The rules and the other external assets defined may differ according to the different configurations, changing the resulting experience. A given template configuration is a bundle of both the template executable and the configuration assets and rules.

- In the **authoring** mode, the EUDev exploits an immersive authoring interface for defining a new configuration or modifying an existing one. Such mode requires a "secret" action sequence for being activated, including a password, which TBs may generate for each template customer. The EUDev creates the rules in such a mode and associates the external assets with placeholder objects.

Launching a template application starts by default in the VR experience mode. To activate the immersive rule authoring interface, the EUDev clicks a button hidden under the wrist of the non-dominant hand. A numeric keypad asks for an access code to avoid unauthorized and unwanted activation of the authoring interface (Figure 5.3-A). Upon successful authentication of the EUDev, the template starts the authoring mode, and the application shows the interface in Figure 5.3-B. It contains four buttons: the first allows creating a new rule (Add Rule), the second put the template in the VR experience mode for testing the current rules set (Test Rules), the third shows a panel containing all the rules in the environment (Show All Rules), while the fourth allows selecting custom 3D models for placeholders.

The rule list is a simple panel containing the natural language representation of each rule in the environment. The EUDev can pick one of them for changing it, or s/he can create new ones. In both cases, the editing UI shows the rule detail visualization (Figure 5.3-C). Each line contains one of the natural language sentences belonging to the rule, according to the language specified in Section 2.2. The initial adverbs identify the

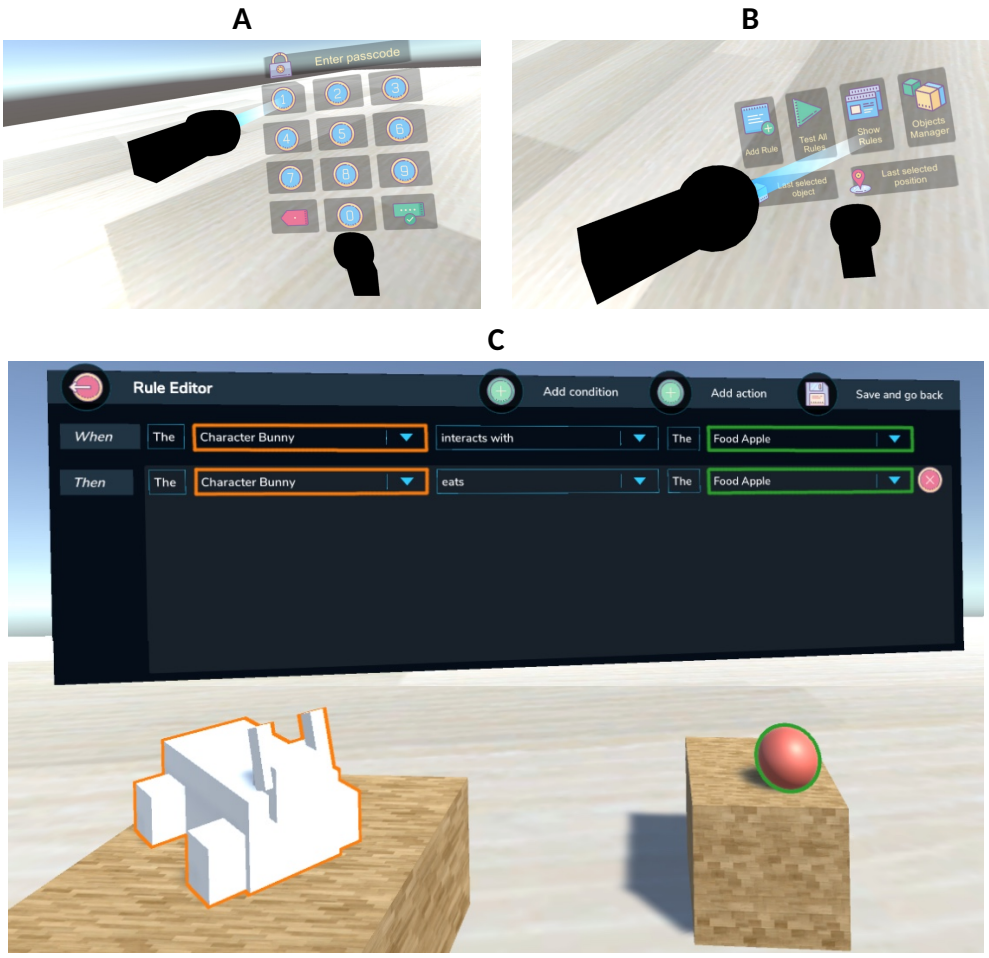


Figure 5.3: Rule-editing interface for EUDevs. Part A shows the activation of the authoring mode, part B shows the non-dominant hand menu for activating the different authoring sections, while part C shows the rule editing panel and the highlighting of the involved VR objects.

rule parts, while the fields in each line allow editing the sub-parts of an action or a condition. We designed the panel trying to minimize possible syntax errors while preserving the readability of the rule.

When starting from an empty rule, the EUDev needs to insert at least an action for the *when* part and an action in the *then* part. So, it includes both the *when* and the *then* keywords with two fields inviting the EUDev to select the subjects of the two actions. Pointing to one of the highlighted fields, the EUDev can choose the subject of the action or the condition simply by pointing it in the virtual environment. Once selected, the object's type and name will appear inside the field. The editor assigns a highlight colour to the VR object, drawing a border both around the object and around the field in the panel (see Figure 5.3-C). The EUDev can cancel the selection by pressing the X button inside the text field. Once the EUDev has selected the subject, the editor calculates the set of verbs it supports. Then it shows this set as a standard list menu and, depending on the selected verb, the editor will support picking further objects or entering values through pointing or standard input techniques (e.g., virtual keyboards). It will not allow completing an action with objects or values that do not respect the language specification.

Since the possible values for each action or condition part depend on its predecessor, the editor enforces such dependencies when the EUDev changes one of the values s/he entered at a previous step by cleaning all the values that depend on it. For instance, if the user changes the subject after selecting a verb, the editor clears the verb too. The same applies to the object or values, which depend on both the subject and the verb.

A EUDev can also insert additional actions in the *then* part by pressing the add action button in the main menu, creating a new slot below the *then* clause, and starting from the subject selection. Finally, s/he can insert one or more optional conditions by pressing the add condition button in the menu. The first time the EUDev presses this button for a rule, it will add the *if* part between *when* and *then*, and the condition editing will start from picking the subject. The second time will add a list menu for selecting the boolean operator (and/or).

We included a dedicated panel for managing `Placeholders`, which correspond to a virtual object whose appearance cannot be determined while creating the template. For setting the actual content, which means replacing the mesh of an exiting `GameObject`, the EUDev uses the interface in Figure 5.4. On the left-hand part, the UI shows the list of the existing placeholder in the virtual environment. On the right-hand part, we have a list of meshes, which contains the EUDev specific 3D models. Currently, they must be loaded into a specific sub-folder at the same level of the template executable, but we plan to support online repositories in future versions of the plugin. Assigning a replacement to a placeholder implicitly defines a rule containing an action that we execute at the environment startup without passing from the rule editor. This is a compromise for keeping low the barrier since startup events are hard to understand for non-programmers.

Finally, the UI allows saving a rule once its definition is correct from a syntax point of view. Rules are persisted in a text file, and they are added to those available for the `ECARuleEngine` (or replaced it if we are modifying existing ones), to support the runtime execution.

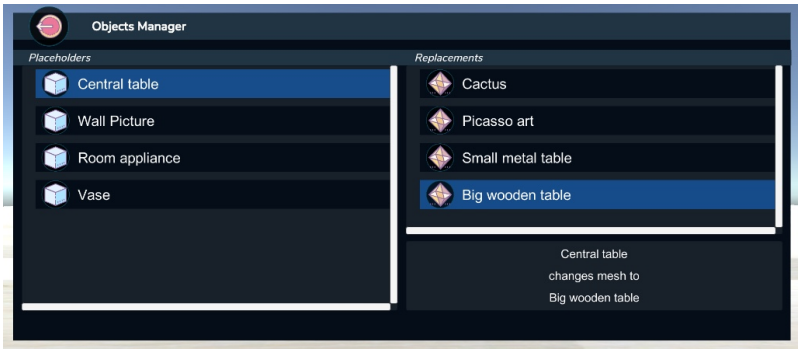


Figure 5.4: The UI panel for managing placeholders.

5.3 Development Case Studies

We demonstrate the support provided by the ECALibrary plugin reporting on the development of two sample templates: a virtual showcase and a virtual museum. We show their flexibility in accommodating two different usage scenarios for each template. We created the configuration ourselves for showcasing the plugin implementation.

5.3.1 Case Study 1: Virtual Shop

Template

The template consists of a large square room, including tables, clothes hangers and shelves where EUDevs can show their products. Figure 5.5 depicts how the template appears before an EUDev configuration. We placed four tables at the same distance from the corners of the room, with shelves positioned in front of them and mannequins in between. In addition, there are some additional elements supporting the visualization of multimedia content, represented as a TV screen. We included a small screen on each table and a bigger one near the wall opposite to the initial position of the visitor. Nearby the bigger mannequins, we placed some buttons for supporting the EUDevs in defining some interaction or animation on their content.

More in detail, the template contains many `ECAObjects` the user can configure:

- All the furniture and light elements in the environment are available for EUDev configurations respectively through the `ECAFurniture` and `ECALight` categories.
- All the light sources in the environment are available for EUDev configurations through the `ECALight` object.
- The TV has a `ECAVideo` interaction.
- Each mannequin is a `ECAMannequin` object.

Figure 5.5: The virtual shop template.

- There is an audio source in the shape of a radio, which is associated with a `ECASound` behaviour.
- All the areas of interest in the shop (for instance around a table, nearby the mannequins, etc.) have an `ECAInteractable` object for triggering proximity-based interaction.
- On top of each table there is a `ECAText` object that may be used for including additional information on the available items.
- All items on top of the tables, on the shelves, etc. are `ECAPlaceholders`.
- A `ECATimer` for time-based interaction.

Clothing Store

The first sample configuration we deliver for this virtual shop template represents a clothing store. In such a configuration, the placeholders throughout the room have been replaced with a custom clothes model. The large TV on the wall is hidden together with the placeholders on the shelves. The overall idea is to support the inspection of different clothes on top of the mannequins and show advertisements on the production process and quality. In the following, we summarise the supported interactions.

The shop includes a customized advertisement video that plays when the user approaches one of the tables and interacts with it (by pressing a button on the remotes). The environment sets an ambient blue light focused on the table and plays an advertisement on the small TV placed on top of it (see Figure 5.6-A).

```
when    the character visitor interacts with the furniture table1
then    the light mainLight turns off
        the light tableLight1 turns on
        the video tableAdvertisement1 plays
```

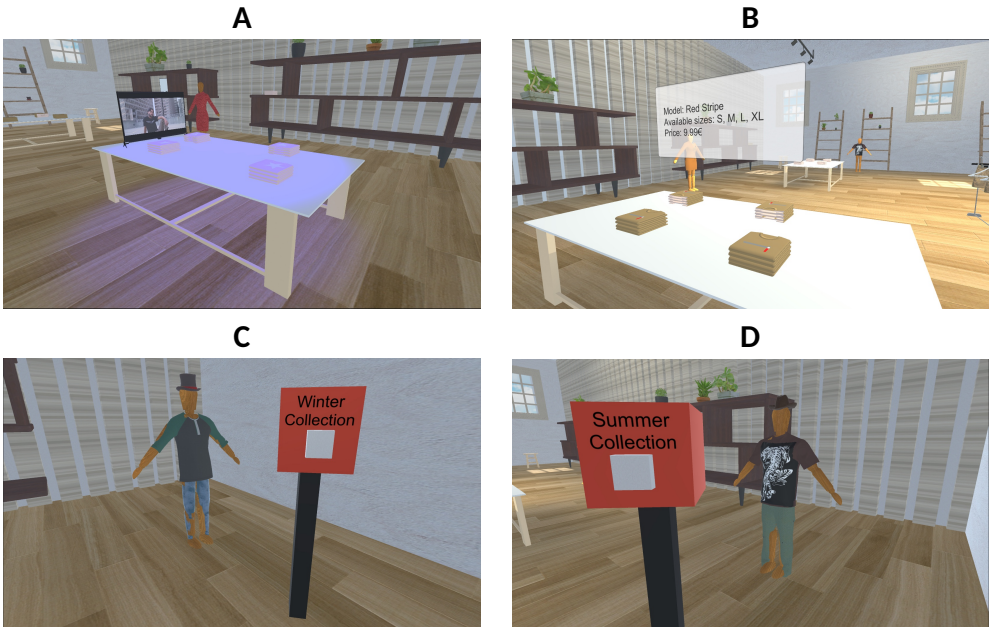


Figure 5.6: Sample interactions in the clothes store configuration. A) When the visitor approaches the table, a blue light highlights the table, and the TV monitor plays an advertisement. B) The visitor approaches another table, and the environment shows some price tags. C and D) The mannequin wears a different outfit according to the button pressed.

The buttons near the bigger mannequin support the change between the winter and summer collection of clothes, changing the mannequin's dress. The interaction is depicted in Figure 5.6-C and D. The rule for setting the winter collection is the following:

```
when    the character visitor presses the button bigButton1
then    the mannequin bigMannequin wears the clothing hat
        the mannequin bigMannequin wears the clothing greentshirt
        the mannequin bigMannequin wears the clothing bluejeans
```

In addition, when the visitor approaches one of the clothes in the shop, a price tag appears on top of the nearby items. The resulting interaction is displayed in Figure 5.6-B. The tag is hidden after 5 seconds.

```
when    the visitor interacts with the table2
then    the text infoPanel2 shows
        the timer timeActions changes duration to 5
        the timer timeActions starts

when    the timer timeActions reaches 0
        then the text infoPanel2 hides
        ... // the other infoPanels hide too...
```

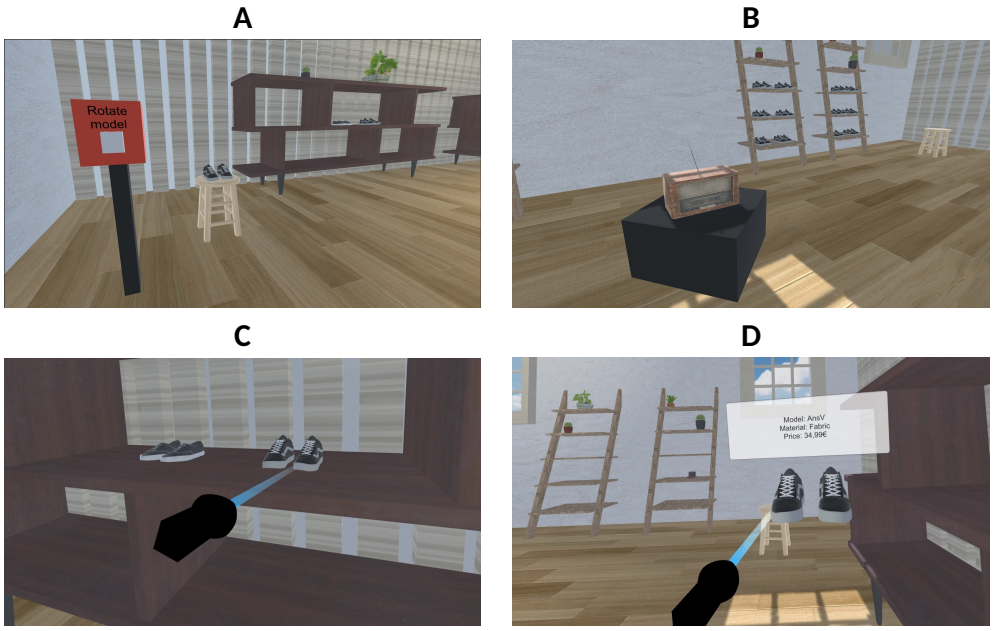


Figure 5.7: Sample interactions in the shoe store configuration. A) Pressing the button, the stool will rotate by 30°. B) Interacting with the radio will turn on or off the background music. C and D) When the user manipulates the shoes, above it will appear an informative panel. It disappears as soon as the user releases them.

Shoes Store

We created a virtual shoe shop on top of the virtual shop template in the second configuration. The pieces of furniture specifically useful for clothes are hidden, e.g., the racks and the clothes hangers, the mannequins, etc. Shoe models replace the placeholders on the shelves and tables position. The small TVs on top of each table and the large TV on the wall are hidden since this configuration does not use multimedia promotional content. We replaced the placeholder on top of the stool with a custom 3D model of shoes. When the visitor presses the nearby button, the stool rotates together with the shoe model for facilitating the visitor inspection (see Figure 5.7-A). The associated rule is the following:

```
when    the character visitor presses the button bigButton1
then    the furniture stool rotates by 30 degrees around Y
```

We show the information about the shoe price when the user points them. The environment shows a panel including such pieces of information. The panel closes as soon as the user releases the shoes (Figure 5.7-C and D). We used the rule defining the first interaction as a sample for describing our approach in Sections 5.1 and 5.2.1 (see Listing 5.3). The rule for the second one is the similar, replacing the *picks* action with *releases* and *shows* with *hides*.

Finally, the visitor can decide whether or not to listen to some background music while visiting the shop. For stopping or playing the music, it is sufficient to approach the radio

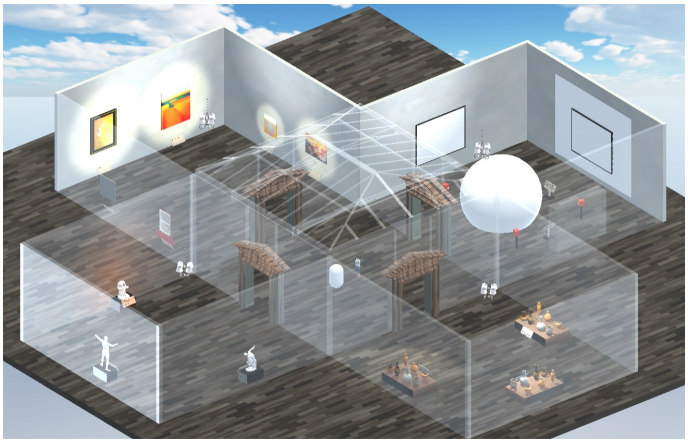


Figure 5.8: The virtual museum template (isometric view - walls and ceilings are transparent for clarity).

in the environment and interact with it (see Figure 5.7-B). Such interaction allows us to show a rule having a condition statement. A similar rule stops the music if it was playing.

```
when    the visitor interacts with the audio radio
if      the audio radio playing is true
then    the audio radio stops
```

5.3.2 Case Study 2: Virtual Museum

Template

The template scene consists of four square rooms connected by the main entrance having a large skylight that gives natural light to the room. The main room is connected with the other four through open doors. We included a dark parquet floor and white concrete walls creating a simple yet elegant environment for placing artworks, which are placeholder elements in the rooms besides the ceiling chandelier. The four rooms contain some sample artworks for providing EUDevs with easy starting points for different museum types: one contains paintings, one pottery, one a collection of sculptures, while the last contains a multimedia room. Figure 5.8 shows an overview of the template.

The `ECAObjects` included in the template are the following:

- All the rooms in the environment, associated to the `ECABuilding` category, for implementing interactions when the user enters or leaves the room.
- All the pieces of art are `ECAArtworks` in the environment. They are also associated to a `ECAPleaceholder` behaviour for changing their 3D model
- Environmental lights (`ECALight` objects) controlling the illumination settings
- A directional light (`ECALight`) for each artwork dedicated to its highlighting.

- A non-playable human character, playing the role of a virtual museum guide (a `ECAHuman`).
- A set of videos in the multimedia room (`ECAVideo` objects)
- A set of `ECAButtons` for selecting up to 4 different videos in the multimedia room.
- An `ECAAudio` for reproducing music or environmental sounds.
- A `ECAText` and an `ECAImage` for putting information about each artwork.
- A 360^o video for showing immersive video contents.

As in the previous case study, we show that different results can be achieved using the same template providing two sample configurations.

Virtual Art Gallery

For the first scenario, we suppose having to promote an art show in the real world. People interested in learning more and visiting the real show can first experience a preview in the virtual environment. When the visitor approaches an artwork, a virtual museum guide will briefly introduce it by voice, an ambient light will fade out while projectors aimed at the artwork will light up.

```
when    the character visitor interacts with the building
   statueRoom
then    the human guide starts-animation "artwork1Presentation"
        the light globalLight changes intensity to 0
        the light spotLight1 changes intensity to 1
```

The visitor can touch the artwork to show a metal plate with the artwork's name, and by touching the plate, the environment shows a descriptive content represented as an image (see Figure 5.9).

```
when    the character visitor interacts with the art Artwork_1
then    the furniture Artwork_1_plate shows

when    the character visitor interacts with the furniture
   Artwork_1_plate
then    the image information changes hides
```

Virtual Geography Museum

In this configuration, instead of paintings, the museum shows big pictures of real places, like Niagara Falls or the Grand Canyon, creating a virtual experience for education purposes. When the visitor enters the room with the pictures (the paintings in the template), the environment plays an ambient sound from the place portrayed in the pictures.

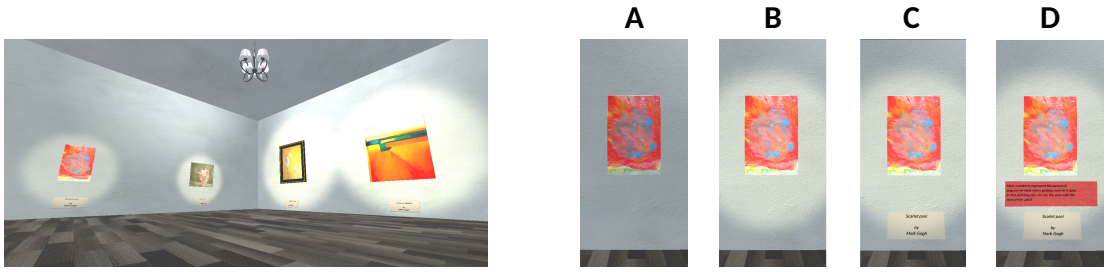


Figure 5.9: Left: Each room in the museum has a specific soundtrack that plays when the player enters it, the music stops when the player leaves the room. Right: When the user approaches any artwork, the corresponding plate with information will appear.

Figure 5.10: When the user interacts with the plate, a 360° video starts playing around him. As the video ends, it disappears (the second image shows the outside of the video-globe, the user will be inside it, as shown in the image on the right).

```
when    the character visitor interacts with the building
   paintingRoom
then    the audio backSound plays
```

The visitor can touch the plate with the place's name to trigger a fade out of the lights and play a 360° video. When the video ends, the lights go back to normal and the visitor can continue exploring the museum (see figure 5.10).

```
when    the character visitor interacts with the furniture
   plate1
then    the light globalLighting changes intensity to 0
        the audio backSound stops
        the video video360 plays

when    the video video360 ends
then    the light globalLighting increases intensity
        the audio backSound plays
```

In addition, the user can select the 2D video playback in the multimedia room through two buttons. For instance, the button on the left triggers the following rule:

```
when    the character visitor pushes the button videoButton1
then    the video video1 changes source to "sea.mp4"
        the video video1 plays
```

5.4 Developer Evaluation

Besides the validation through the development of the case-studies, we ran a lab study with developers for getting their feedback on the perceived *usefulness* of the overall approach and the *usability* of the Unity plugin we developed.

Procedure The study consists of five different parts. In the first one, the participants read the study description statement and expressed their informed consent for participating in the study. After that, they filled out an anonymous demographic questionnaire, including questions on their experience level in VR development.

In the second part, we asked them to read a brief description of the roles in the envisioned EUD workflow (see Section 5.1), how to create a template using the Unity plugin, and how to extend the library. Afterwards, they filled out a questionnaire evaluating the perceived usefulness of the approach and the expected usage difficulty through a set of Likert items. This allowed us to measure the expectations of the developers *before* using the plugin.

In the third part, they executed three development tasks, starting from the scene of the Virtual Shop template. The tasks are the following:

T1 The participant has to provide the EUDevs with the support for defining the environment's reaction when a push-button in the scene is pressed. This task represents the case in which it is sufficient to add an existing component to define the configuration point of the behaviour.

T2 The participant has to find the 3D model of a stool into the environment and allow EUDevs to replace it with their own 3D model (e.g., of a chair). This task represents the case in which the TB knows that the EUDevs would like to put their own content instead of the model included in the scene (e.g., their products in a showcase).

T3 The participant has to extend the set of objects provided by the plugin adding a *key* that can unlock (i.e., make visible) another object. The EUDev must define the object to unlock through a rule. This task represents the case in which the object set is not enough for the TB purposes and requires an extension.

At the end of each task, the participant answered the Subjective Mental Effort Questionnaire (SMEQ) [SD09] for measuring the effort, while we collected the completion time.

The fourth part consists of filling a post-test questionnaire, including the same Likert items from the second part and assessing differences in the perception *after* the usage. In addition, we asked developers to rate the plugin's *efficiency* and *effectiveness* together with their general *satisfaction*.

The fifth part consists of a debriefing phase, where we discussed with each participant the problems they encountered and the strong points they noticed in the approach.

Participants Participation in the study was completely voluntary. We contacted former students who graduated from our University and work or study in game and/or VR development. We invited those who responded to the call in our lab for individual sessions. To thank them for their time, we offered a coffee break at the end of the session.

Nine people participated in the study, eight males and one female. Their age ranged from 22 to 37 years old (M=26.7, Md=22, SD=4.95). They had a good education level: 3 a Bachelor Degree, 5 a Master Degree and 1 a PhD. The development experience was also good: 5 participants had 4 to 6 years experience, 3 from 7 to 10 years and one more than 10 years. They had less experience in game and VR development, but it was adequate for the purpose of the study: 8 from 1 to 3 years and one more than 10 years. All participants were proficient in Unity, but they used also 3D and VR-related tools: 3 used Unreal Engine, 3 A-Frame, 2 Blender, 2 Maya, one 3D Studio Max one Steam VR. We asked them to self-assess their skill in different tasks related to VR development on a 1 to 7 Likert scale. They rated high their ability to define the environment behaviour (M=5.3, Md=6, SD=1.94) and the interaction (M=4.9, Md=5, SD=1.83). Instead they gave an average rating to their 3D object modelling abilities (M=2.9, Md=2, SD=1.83), their expertise in modelling environments (M=3.0, Md=3, SD=1.58) and their skills in composing them (M=3.8, Md=4, SD=1.71).

Results All participants concluded all tasks, even if one of them required suggestions for properly defining the behaviour of the new object in T3. Figure 5.11 shows the differences in the completion time and perceived effort (measured through the SMEQ [SD09] questionnaire) between the tasks. We expected T3 to be the most difficult among the tasks since it requires the development of a dedicated class in C#, and this is clearly confirmed by the results. T1 and T2 took a comparable amount of time, while T3 required longer.

The effort reported by participants through the SMEQ [SD09] follows a similar trend. T1 and T2 received similar evaluations, and T3 required a higher effort as expected (see Figure 5.11, right part). According to the labelling of the difficult levels in [SD09], the median for both T1 and T2 sets this tasks between “Not very hard to do” and “A bit hard to do”, closer to the lower level. T3 is instead near the “A bit hard to do” level. The high standard deviation reflects some difficulties encountered in this task by some participants. We clarify better this point through the debriefing comments.

Such results show that exploiting existing components in the plugin for inserting configuration points requires little additional time and effort. Writing an extension is more difficult but less frequent. Nevertheless, the time and effort required seem reasonable for underlying task complexity.

Following the guidelines for evaluating HCI toolkit research [LHV+18], we included questions assessing the perceived usefulness of the approach and its usability in terms of templates development. We collected the participants’ opinions on different statements related to the utility and the difficulty of the approach before and after completing the tasks for highlighting differences between the expectation and the actual experience. Fig-

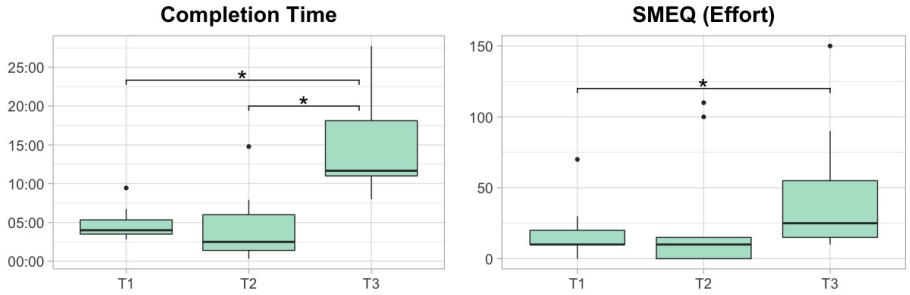


Figure 5.11: Task completion time and perceived effort [SD09]. Highlighted pairs have a significant difference (Wilcoxon signed-rank test, $p < .05$. We used the Bonferroni p-valued adjustement for repeated comparisons).

ure 5.12 (top part) lists the statements we included in the questionnaire and the rating pre (blue) and post-task (red). The two sets of answers are pretty similar. We did not register any significant difference in both the usefulness and the difficulty questions. This means that, overall, the plugin met the participants' expectations. However, we notice that the post-task assessment of the statement "I will use the plugin for creating templates" follows a decreasing trend, while we notice an increased perception of the overall utility after the usage. This means that our participants appreciated the idea and its implementation, but some of them reconsidered a possible direct involvement in the envisioned meta-design process. Among the questions related to the expected and perceived difficulty, we notice a decreasing trend for the statement about the plugin extension. This is related to the difficulties encountered in T3 that we already mentioned, and we will discuss when reporting on the debriefing.

Finally, the items about the overall usability of the approach (see Figure 5.12, bottom part) received a very good assessment, showing an overall appreciation of the plugin support. They included statements about the ease of use, effectiveness and satisfaction.

Debriefing We collected the participants' feedback at the end of each session, which is helpful for understanding strengths and weaknesses in our approach and for improving it in the next iterations.

We received both positive and negative comments about the project from all participants and suggestions for improving it. Generally speaking, the project received positive feedback concerning its ease of use, internal structure and taxonomy, usefulness, extensibility and reusability. One participant stated that the idea could be very useful for avoiding the redefinition of standard behaviours in an end-user development setting and even in VR environments created by experts. A second participant, who was already familiar with ECA rules, underlined the ease of use of the component-based approach for defining configuration points. In addition, he considers successful the representation of the rule in natural language for involving users without programming experience. Another participant highlighted that the TB requires skills spread over different roles in a typical game

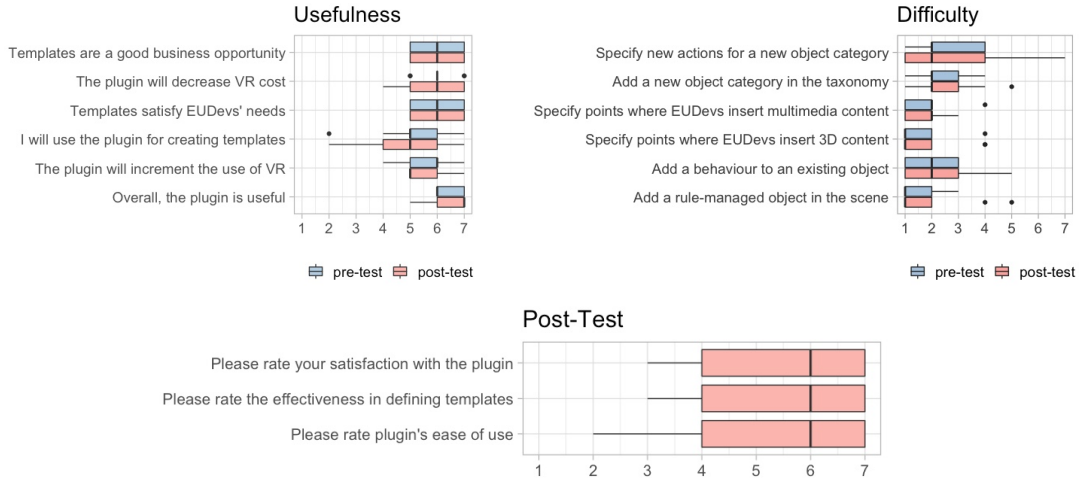


Figure 5.12: Participants assessment of the usefulness, difficulty and overall usability of the approach. They assessed the usefulness and the difficulty before and after completing the task. Values are in a 1 to 7 Likert scale.

development organization, including game developers, level designers and 3D artists. We agree in general with this comment, and we would need further research for understanding whether the plugin is effective across these three roles, especially considering people acquainted only with 3D modelling.

Negative feedback comments expressed some confusion in the taxonomy structure and a perceivable initial learning curve to get used with the object categorization. Indeed, some taxonomy elements have names recalling similar concepts, and the difference between them needs some learning effort. For instance, a participant spent some time in grasping the difference between the *Behaviour* (base class for all behaviours identifiable across the different categories of objects), *Interactable* (a *Behaviour* associated with something a character can interact with) and *Button* (a perceivable object that a character can push), since they looked similar to him (e.g., a button should be interactable and it must have a behaviour). In general, assigning names is not easy. But it is also true that we define similar configurations in different ways in many toolkits. In ours, something that logically corresponds to a button may be defined through the object in the taxonomy, but also assigning an *Interactable* behaviour to objects that belong to a different category (e.g., a chair). The difference is, as explained in Section 5.1.1, the physical appearance. If it resembles something that we can push, we consider it a *Button*, otherwise it belongs to a different object category, and we associate an *Interactable* behaviour. Both ways are correct depending on the situation.

Task 3 also highlighted difficulties in understanding the extension process. In particular, the information included in the code documentation and the descriptive material we created for the test was not enough for one participant. He expected some startup code, such as the skeleton of a dummy extension, to fill up with custom code. He found the material we provided particularly unhelpful, and he stated that reconstructing the

process from such descriptions required too much effort. We agree that including additional information and startup code would ease the task for TBs, and we will complete the documentation in further versions of the plugin. However, the issue seems more related to the documentation provided than the EUD approach itself. Therefore, even if the difficulty is relevant, we do not consider it a barrier to using the plugin.

Some participants suggested to improve the current documentation with more examples for our future developments, even in the natural language rules part. This could help the TB to understand how a EUDev specifies rules and how s/he can use the plugin. One of them proposed creating a debug mode to test the rule set in play mode, reporting the TB when an action is triggered using a console in the game tab of Unity. We agree that this is useful, but it is currently beyond the scope of the proof-of-concept implementation. Another participant requested more help from the development environment while formalizing new actions for the EUDev to guarantee the expected syntax for the rules through an auto-completion feature. Again, it would be very nice to have it in a product, but it goes beyond our proof-of-concept.

5.5 Demonstrating Rules using ECARules4All

In a follow-up project to ECARules4All, we propose an authoring interface based on programming-by-demonstration [CH93] to define event-condition-action (ECA) rules. The workflow proposed by the tool involves three roles, applying the meta-design approach described in Section I. The first is the *Template Builder* (TB), representing users with good 3D modelling and game programming skills. They create the template environments and define high-level actions on different object categories, which allow end users to define specific environment reactions in the configuration phase. The second is the *End-User Developer* (EUDev), representing users without skills in 3D modelling and game programming wanting to create VR content for their business or leisure (e.g., touristic promotion, content advertisement, etc.). They download or buy templates and configure them by adding custom content and interactions. The latter exploits the high-level actions on virtual objects defined by TBs, using ECA rules readable as natural language sentences (e.g., *when the visitor interacts with the table, then the advertisement video plays*. Finally, the *Users* are the consumers of the VR experiences. The current implementation of the tool allows authoring the ECA rules through a dedicated panel, including drop-down lists for the selection of the different rule parts. While this modality enables EUDevs to define the rules they need, the authoring support does not fully exploit the manipulation interactions possible in VR environments.

5.5.1 Demonstrating Rules in VR

To ease the development of such rules, we are currently working on applying programming-by-demonstration techniques [CH93]. In such a setting, the EUDev *demonstrates a rule* to the environment by executing a set of actions and showing their effects by

direct manipulation of the objects. The environment interprets such actions for inferring the rule. In designing the support for the demonstration, there are two main problems to consider. The first is that the authoring environment does not separate the authoring and the test mode for rules. Therefore, after defining a rule, the environment immediately allows the EUDev to test while immersed in the virtual experience. The environment needs an explicit command from EUDevs to understand that they are currently demonstrating a new rule rather than testing an existing one. We supported such a command through a recording metaphor: the EUDev pushes a recording button, and the environment considers all the following interactions for the rule inference. The second problem is that only some of the actions performed by the EUDev during the demonstration are required for defining a new rule. EUDevs may perform actions such as moving around or selecting objects or options for easing further actions and not for the demonstration itself. For instance, they can change their current position to ease pointing at an object or grasping it, but that movement is not required in the rule. Considering that it would be difficult for the environment to tell such a difference, we mitigate this problem by including an interface for recognising the relevant actions and discarding unnecessary ones. In summary, the EUDev has to perform the following actions to demonstrate a rule: 1) recording a set of interactions, 2) recognising the actions relevant to the rule definition, and 3) selecting the triggers and the related consequences in the action set.

Interaction Recording. To demonstrate a rule, the EUDev starts by recording a set of interactions. The environment provides a record button on a non-dominant hand menu, starting the recording when toggled and stopping it when released. During the recording, the environment collects a screenshot every 0.5s, trying to capture the interactions one moment before they happen to obtain a representative frame. We identify the interactions relying on the events defined by the underlying MRTK toolkit [Mic22b] we used for implementing the tool and inspecting the target game object state after the event handling (e.g., the position after a move event). Once an interaction event is detected, the current screenshot is scaled to a 720p resolution and persisted as a thumbnail representing the interaction. Besides the screenshots, we also save a description of the interaction in a subject-and-verb form such as *the player interacts with Cube* or *the player jumps to the position (-1.5, -0.4, 1.3)*. The tool exploits these two pieces of information for presenting each recorded interaction to the EUDev in the next phase, as depicted in Figure 5.13.

Action Recognition. After the recording phase, the EUDev must tell which recorded interactions are useful for the rule definition. The tool presents the recorded interactions using a video-clip metaphor (see Figure 5.14). The EUDev toggles (Figure 5.14-A) or untoggles (Figure 5.14-B) the interactions that were meant for the rule definition.

Identifying the rule parts. The last step in the process is defining the trigger and the consequences of the rule (or rules) to define. By default, the tool assumes that the first action is the trigger (the *when* part of a rule) and all the following actions are consequences (the *then* part). EUDevs can change such a simple default proposal by using two features. The first one is the trigger selection, which supports changing the action that triggers the execution of a rule (see Figure 5.14-A and B). To do this, the EUDev activates the tool and points at the clip representing the trigger, confirming the selection with a

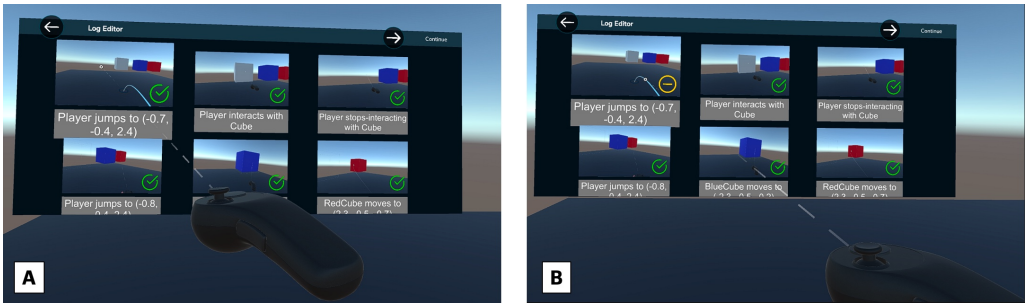


Figure 5.13: Interface for the action recognition. The panel shows each interaction and action performed during the recording phase, using a video clip metaphor. Each clip has a screenshot of the EUDev's view immediately before the interaction and a small textual description as the title. By selecting (part A) or deselecting (B) a clip with the remote, the EUDev confirms or discards the clips useful for defining the desired rules.

button on the remote controller. The second feature allows the separation of clips belonging to different rules. The EUDev selects the scissor tool, points at the links between the action clips, and splits the action sequence into a new rule (see Figure 5.14-C and D). After that, the EUDev can define another trigger for the new rule. It is possible to iterate this process to create a list of rules from a recording. A rule must include at least a trigger and an action (i.e., a minimum of two clips). When this process is finished, the tool recaps the rule(s) that are going to be added to the environment as natural language sentences. The EUDev can immediately test the new rules by interacting with the VR environment.

5.5.2 Preliminary Validation

Even though the support is at a preliminary stage, we performed an initial validation for assessing the number of actions we request to a EUDev for defining the same set of rules through the form-based authoring panel described in Chapter 5 and the one we propose in this section. We selected the subset of the rules defined for the *Virtual Shop* and *Virtual Museum* sample environment in Section 5.3, consisting of 5 rules from each environment. The detail about the rule set is available as supplemental material for this section. We counted the number of interactions in the authoring interface as a proxy for estimating the effort required in both authorings. The demonstration approach consistently required fewer interactions for defining the rules of the*Virtual Shop* (demonstration: M = 16.6, SD=3.6; form-based M = 19.2, SD=5.5) and the *Virtual Museum* (demonstration: M = 16.6, SD=2.8; form-based M = 19.4, SD=4.8). The result is encouraging, and we will maintain the goal of providing an interface requiring less operating effort concerning the form-based baseline.

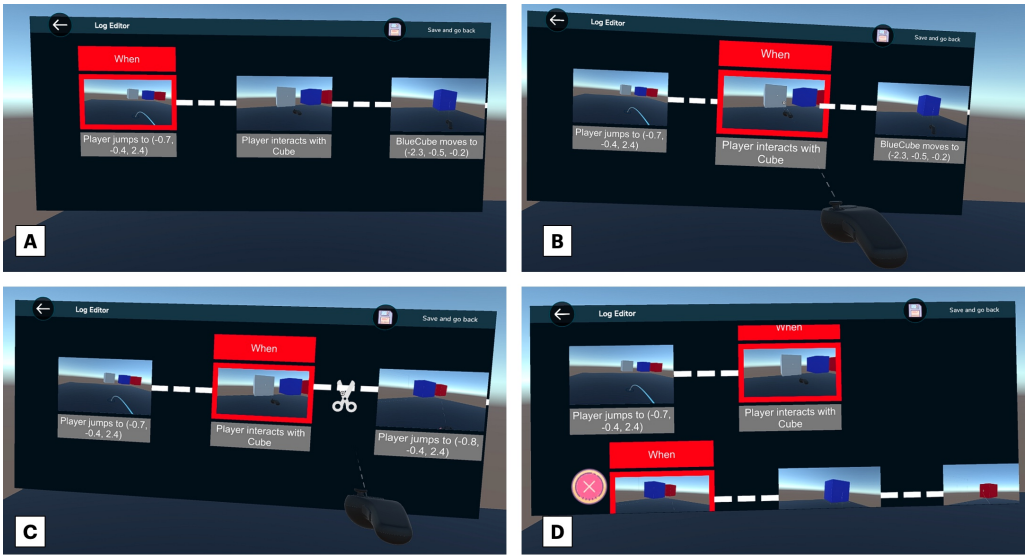


Figure 5.14: Interface for trigger and actions identification. After the recognition step, the tool assumes the first action as a trigger for a single rule. The tool allows selecting another one by pointing and selecting another among the clips (part B). In addition, the EUDev can split the trigger-action chain using the scissor tool. This allows to create more than one rule using the same recording session (parts C and D).

5.6 Discussion

In this section, we summarise the contributions of this chapter, the lesson learned and the limitations of the work. We introduced an end-user development approach for supporting the configuration of a VR environment by users without skills in programming and/or 3D modelling. The approach leverages on meta-design [FGY+04, CFMP05, ABC+15], introducing the hybrid role of the EUDev, that is, both a designer and a user. The EUDev configures the templates created by TB, providing customised VR experiences for users who consume the final VR content. Having defined the roles, we presented an engineering solution for supporting the EUD approach, based on natural language Event-Condition Action rules, defined by EUDevs. We describe the high-level actions on top of a taxonomy defining categories of virtual objects. We also provide a solution for executing rules at runtime without rebuilding the VR environment after the configuration by EUDevs.

Besides the overall approach, we discussed a proof-of-concept implementation in Unity3D. Its peculiarity relies on extensibility, which mitigates possible shortcomings in the taxonomy of 3D object modelling. By including some custom annotations, TBs can provide further actions and object categories to EUDevs.

To demonstrate our claims, we provided a validation of the approach. The plugin discussed in the chapter is a toolkit enabling TB to create templates. Properly validating a toolkit in HCI research is a debated topic in the community. An excellent summary of the documented techniques in the literature is available in [LHV+18]. Following their categorization, we validated through demonstration and a user study. The demonstration, discussed in Section 5.3, shows that the current proof-of-concept implementation of the

approach can support the definition of templates for two different use cases, providing support for creating substantially different experiences through rules. The user study shows that potential TBs consider the plugin (and the overall approach) useful and usable, requiring a reasonable effort for defining configuration points. From the discussion with participants, we identified points where they require improvement, especially in the documentation of the extension mechanism.

This work has, of course, limitations. First, we focused on the approach definition and the support provided to TBs. In the current state of our research, we cannot claim the usability of the rule language, the effort required by EUDevs for configuring a VR environment or the effectiveness of the immersive rule-authoring interface. While we are positive that the rule-based approach may have comparable effectiveness documented in other domains (e.g., for IoT, see Section 2.2), further research is required. In particular, we need to investigate the differences in rule authoring through desktop or immersive interfaces. In an immersive mode, EUDevs would avoid authoring and test cycles, requiring to put on and take off the HMD. In contrast, creating many rules in the immersive mode may be tedious, so there is space for hybrid solutions. However, no configurations are possible without the VR environment support, and this work is the first step towards that goal.

A second point to discuss, which is somehow connected with the previous, is the limited scope of the object taxonomy. Our goal was not to provide complete modelling of all the possible categories of VR objects but only include commonly used definitions that may be reused across different templates, using our experience in creating VR environments and looking at existing categorizations. This allows TBs to start with a reasonable base. Otherwise, they would need to redefine the whole rule language. However, we expect different domains to have different needs in expressing rules. So, we included an extension mechanism that allows such adaptation. We are persuaded that searching for a complete taxonomy in all templates is not a realistic goal. We may find arguments in favour of this having a deeper look at the literature references discussed in Section 2.2, where it is clear that we have no shared definition even of the rule format, which are adapted in different variants for different domains.

Finally, the last limitation that requires discussion is the small number of participants in the user study. This is mainly due to the difficulty in recruiting people corresponding to the TB profile, who are experts busy in their own development tasks. In addition, for keeping short the time required for each evaluation session, we included very specific tasks and not, e.g., creating an entire template or engaging in open-ended tasks where the participants could be encouraged to explore the tools, again to make the recruitment feasible. We are aware that more in-depth evaluations would be required, even outside the lab environment, but, as discussed in [LHV+18], this is seldom possible in evaluating research toolkit prototypes.

In section 5.5 we described a work-in-progress implementation of an interface for defining behavior rules by demonstration in a Virtual Reality environment. The support consists of a recording phase, where the environment collects all the interaction performed by the end-user developer and a two-step recognition phase, where the end user

first recognises the actions that were meant to create the rule in the recording and then distinguish between triggers and consequences for creating the final rule set. The support requires a general way of defining how to demonstrate specific actions and support for conditional rules. In the future, we will focus on overcoming two limitations of the system to provide a support equivalent to the current rule editors in the literature. The first one is the set of actions the EUDev can demonstrate, which are limited to attributes related to virtual objects' visibility (visible or hidden) and transform changes (position, rotation scale). We do not currently support the high-level actions defined by Template Builders for EUDevs. The limitation is not related to the detection support during the recording phase: we are already able to detect the execution of such actions when triggered by existing rules. What is currently missing is an interface for enabling the EUDev to demonstrate actions going beyond simple manipulations, which should be defined *ad-hoc* considering the high-level action semantics. Such an interface should be defined or tuned by TBs. For instance, changing the colour of an object requires a specific UI for selecting the colour or a metaphor for "painting" the object during the demonstration. The TB should define such a demonstration sequence while creating the high-level action for changing the colour. Another relevant example is demonstrating a change in a character's clothing (e.g., wearing a t-shirt). A possible demonstration may be grabbing the t-shirt and releasing it on the character, which requires tracking for specific manipulations (e.g., the collision with the character) and interpreting them as a specific high-level action. Providing a general way to support such demonstrations is still an open problem.

The second limitation is related to the definition of conditions in the ECA rule format. While the events (triggers) and the actions fit well, the recording metaphor for the demonstration defining conditions on the objects' state is less straightforward. It is easy to record the current state of a given virtual object and to derive conditions testing that one or more attributes have the same value stored during the recording phase. It is more complex to demonstrate different comparison operators, e.g., greater or less than for numbers or sub-string containment. However, the problem is less important than the first one considering that end-user developers seldom define rules, including conditions [UPYHB+16, ABFS23].

Chapter 6

XRSpotlight

In this chapter, we present XRSpotlight, a system that aims to empower novice XR developers to find and adapt existing examples to their own XR designs by identifying and representing interactions in natural language. The proposed approach is based on a model abstraction that unifies existing XR toolkit implementations and provides an interface designed to enable novices to understand interactions in a toolkit-agnostic way. XRSpotlight lists all the interactions defined in a Unity scene and presents them as rules in constrained natural language. This enables novices to identify incomplete specifications of interactions (e.g., by highlighting possible missing components such as colliders and rigidbodies), find similar interactions as examples to adapt, and copy-paste interactions from examples using the same or different toolkits. These features allow a simpler understanding of the XR interactions, linking its simplified view with the more complex definition in Unity and guiding the transition of the XRSpotlight support.

The main contributions of our work are: *i)* the design of our abstraction model behind XRSpotlight, which enables example-driven development of XR interactions across different toolkit implementations; *ii)* the demonstration that our abstraction model correctly captures existing toolkit implementations and matches professional VR developers' mental model of how XR interactions are implemented with different XR toolkits; *iii)* our study with novice XR developers showing that XRSpotlight better supports them in inspecting example XR interactions compared to traditional means (using only the Unity Inspector) and enables them to build new XR experiences. The results obtained in the study with novices suggest that XRSpotlight could effectively support them in the transition from our interaction model representation towards programming interactions without having to rely on our tool support.

6.1 XRSpotlight Walkthrough

Before illustrating how XRSpotlight works, we discuss the support it offers to novice developers through concrete usage examples. The first three steps in this walkthrough are a subset of the tasks we used for the user evaluations in Section 6.6. James is a novice XR

Figure 6.1: An overview of the scenes and interaction used in the walkthrough. A: the coffee cup in the MRTK sample scene has a manipulation interaction. B: the kitchen scene with the egg ingredient shows feedback on manipulation. C: the same egg shows the feedback on pointing after changing the rule trigger. D: the feedback replicated on the wrong ingredients. E: the pan drag interaction imported from a SteamVR sample.

developer, so he is familiar with programming languages but has little experience building XR environments or using game engines. He is working on an ongoing project which uses Unity and MRTK to define interactions in a virtual kitchen, where the user can simulate the cooking of simple recipes. The Unity scene is furnished with assets representing kitchen furniture, tools and ingredients and has MRTK imported.

Interpreting and finding examples of interactions in the scene. James' goal is to create guidance for selecting the ingredients in the current recipe. He wants to show feedback (correct or wrong) as long as the user manipulates an ingredient. It is the first time he creates such an interaction, so he needs to understand how to implement it. He starts from a working example by inspecting one of the examples shipped with the MRTK toolkit, the hand manipulation scene. James would like to find an interaction similar to the one he needs to implement, and he picks the coffee cup object since it provides manipulation affordance (Figure 6.1-A). By default, the Unity Inspector lists many components, some of them seem related to the interaction, and others seem not, but he needs to figure out the example relevance. To understand the interaction provided by the object, James opens the XRSpotlight window. The interface shows the list of interactions producing consequences in the experience, expressed as trigger-action rules, as Figure 6.2-D shows. The novice understands that the cup reproduces a sound when a user selects it and stops when released, which is a good starting point for his development task. XRSpotlight helps James identify how MRTK defines this interaction by highlighting the section in the Unity Inspector. James identifies the event related to the interaction in MRTK and the method used for playing the sound (see Figure 6.2-E).

James has found a relevant example, but he wants to see if there are *better* examples. XRSpotlight supports this task by finding a list of similar rules (see Figure 6.3). He understands that many objects in the scene behave similarly when moved, even if they have different shapes. Then James realizes that the example he selected is good enough for his purposes.

Example transition to a different scene. The next step is adapting the rules in the MRTK example scene to the kitchen environment. James uses XRSpotlight to copy the

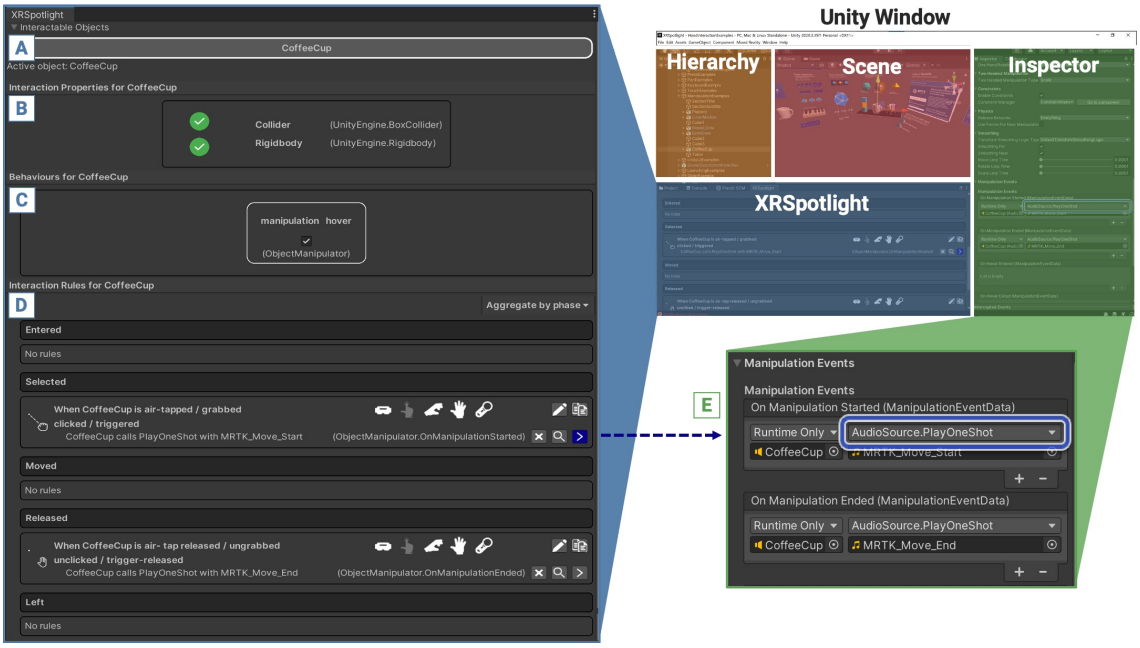


Figure 6.2: The XRSpotlight panel in the Unity editor. A: foldout menu showing all the interactable objects in the scene (partially cut for readability). B: information about the support to collision detection and physics. C: interactive behaviours provided by the components associated with the XR object. D: interaction rules, aggregated by phase (see Section 6.4.1). Each rule entry has icons representing the modalities it supports and a set of buttons allowing (starting from the upper left): to change the trigger, copy the rule, delete the action, find similar actions and show the definition on the Inspector. E: Highlighting the definition of a rule action (consequence) in the Unity Inspector.

first coffee cup rule for the movement and to paste it into one of the ingredients in the kitchen scene, the egg (Figure 6.4-A). XRSpotlight notifies James that he needs to define the interaction effect since the code defining them in the source scene is not available in the kitchen environment. He wants to show feedback (correct or wrong) as long as the user manipulates an ingredient. Therefore, James writes a script to show the visual feedback and connects it to the interaction for moving the egg, copied by XRSpotlight and highlighted in the inspector. After that, the kitchen environment contains an interaction in which the egg shows visual feedback as long as the player is moving the egg (Figure 6.1-B). James follows the same procedure with the second rule in Figure 6.2-D to hide the feedback when the user releases the egg.

Adapting and Replicating interactions in the scene. Having defined a working interaction, James tests the experience but, even if the feedback on the egg shows and hides correctly, he realises that the selected interaction is not well suited for the task. He thinks it would be better if the user sees the feedback by just pointing at it. Thus, James aims to modify the interaction accordingly. XRSpotlight allows him to change the trigger part of the rules, leaving the effects as they are (Figure 6.4-B). Changing both the rules' triggers,

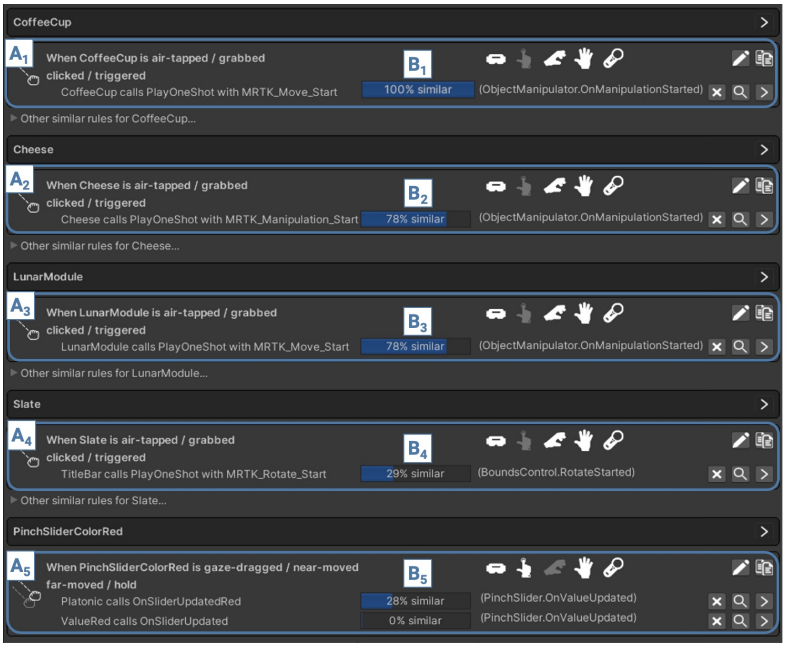


Figure 6.3: XRSpotlight shows the interactions similar to the one currently selected by showing an ordered list of rules involving other objects in the scene. For each object (A_i), it shows the most similar rules it supports, including itself (A_1). Next to each row, we show the percentage of the similarity (B_i) and rules similar to the one displayed.

the egg shows the feedback when pointed and hides it when left (Figure 6.1-C). Now, the novice developer would like to replicate the interaction for each ingredient on the kitchen table. He uses the copy-paste feature of the previous step (Figure 6.4-C) and all the ingredients support the feedback interaction (Figure 6.1-D).

Adapting examples from other toolkits. Suppose that James's next goal consists in implementing an interaction where when a pan is dragged, it makes smoke disappear. He recalls having used a similar interaction in a recent project he found on GitHub. James founds the repository again after a quick search, but *ExamplePan* uses SteamVR, not MRTK. For adapting the example, he would need to master both toolkits, but he cannot find similar projects using MRTK. XRSpotlight allows copying the interaction from the SteamVR scene and pasting it inside the MRTK scene (Figure 6.1-E). Relying on its modelling abstraction, it searches for the best correspondence between the two toolkits in modelling the considered interaction, if any. The steps are similar to the previous ones and usually require adapting the interaction consequences.

6.2 XRSpotlight overview

In summary, the support provided by XRSpotlight to the tasks depicted in Section 6.1 consists of three main features: *(1)* expressing interactions in natural language, *(2)* finding

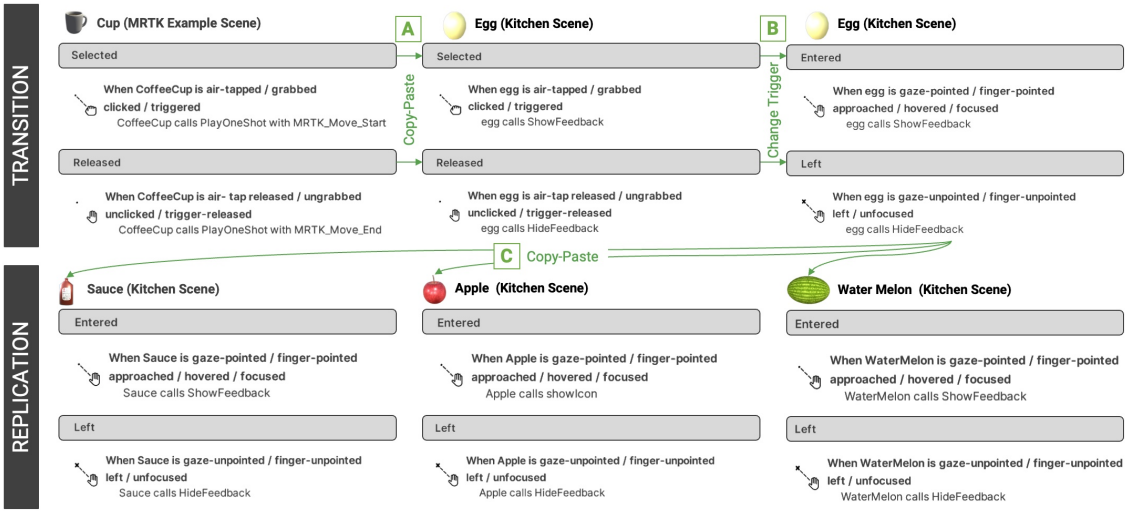


Figure 6.4: The import (transition) of an interaction from an example through XRSpotlight, and its replication on other objects. It starts with a copy of the rules Selected and Released from the coffee mug located in the MRTK example on the egg in the kitchen scene (A). Then, the rules are adapted to support a hovering interaction through the change trigger feature (B). After that, the obtained rules are replicated on three other ingredients through copy-paste (C).

examples of similar interactions in the scene, and *(3)* copy-pasting interactions in a toolkit-agnostic way. Figure 6.5 provides an overview of our XRSpotlight system.

XRSpotlight supports the first feature through a panel providing a rule-based description of the interactions involving the selected game object. In Unity, identifying the interactions and their effects currently requires inspection through different editor panels and/or reading code, presenting an overwhelming amount of information that makes it hard for novices to form the correct understanding. We use trigger-action rules described using natural language sentences to express interactions and communicate when a rule applies (e.g., when the user grabs the object) and its consequences (e.g., support dragging).

The second feature allows developers to use a rule for finding further interaction examples in the current scene, according to a similarity measure provided by XRSpotlights's underlying model to sort and filter interactions in complex scenes. This enables interaction-oriented navigation of the objects in the scene and supports finding the available variants of a relevant interaction and selecting the one that best suits the developer's needs. XRSpotlight facilitates the inspection of these variants by grouping rules by phase (i.e., when the interaction happens), modality (i.e., interaction technique) or action (rule's consequence), which is semantically richer than existing inspection tools.

The third feature supports transferring the interaction definition from an example to a target game object, mimicking the well-known copy-paste function. Behind the scenes, XRSpotlight uses its model of toolkit implementations to identify all the relevant elements

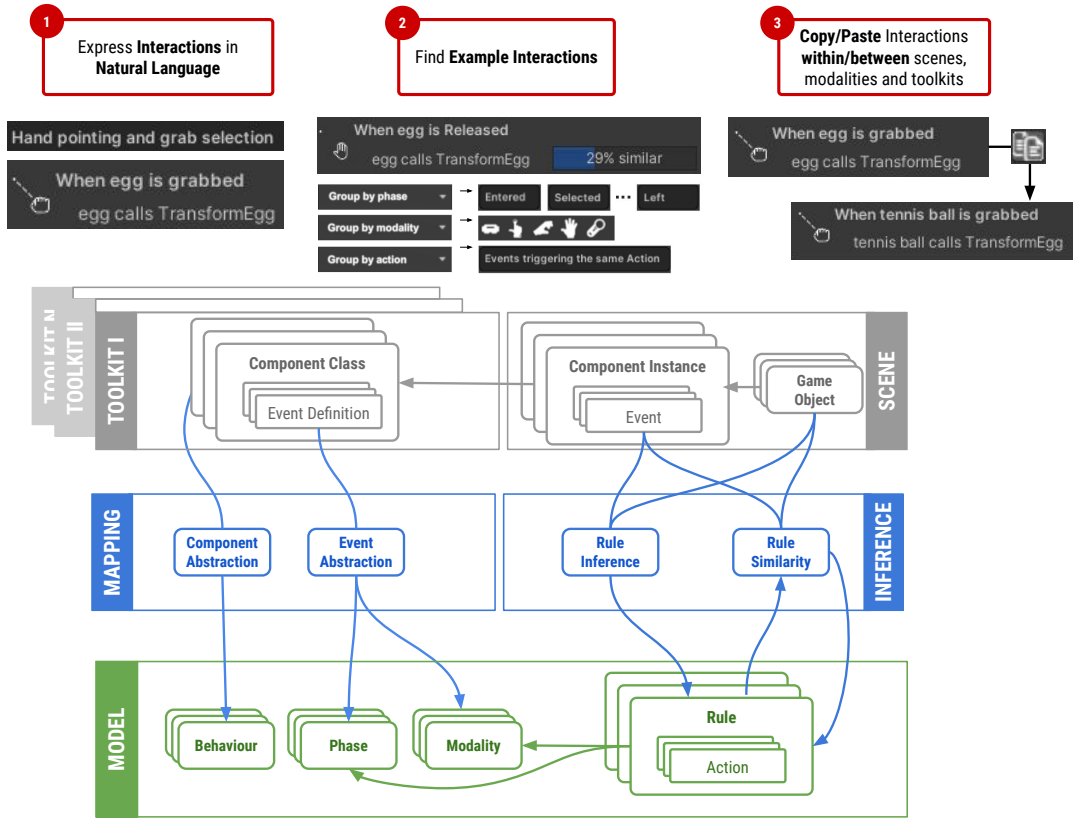


Figure 6.5: Using an abstraction of existing toolkit implementations in our model and a mapping from each toolkit to the model, XRSpotlight can express interactions in a toolkit-agnostic way using natural language, find similar interactions for a given example, and copy-paste interactions between scenes to support the use and adaptation of examples provided by different toolkits. At the technical level, toolkit capabilities are captured in terms of behaviours, phases, and modalities, while the interactions defined in a scene are represented as trigger-action rules.

(i.e., components, events, and listeners) and recreates them in the target object. The underlying model allows transferring interactive behaviors not only within the same scene, but also between two scenes in different Unity projects. This case is particularly relevant for novices who often try to replicate interactions shipped in the toolkit sample scenes. Finally, the model also supports transferring examples between different toolkits, finding the equivalent elements in the target toolkit.

6.3 Toolkit Analysis and modelling

For deriving the toolkit-agnostic model representing XR interactions, we proceeded to an analysis of the existing toolkits, which informed the three main traits of the resulting

model: *(1)* the concepts of interaction phases and modalities, *(2)* the distinction between interactions available and exploited, *(3)* the modelling of rule triggers and their effects. Considering the large number of available XR toolkits (see Table 3.1), we selected two representatives for proceeding with the in-depth analysis. We eventually chose the *Mixed Reality Toolkit* (MRTK) [Mic22b] and *SteamVR* [Sof22a]. They are among the most popular toolkits in XR development, provide high-quality documentation (including example experiences), and have an active developer community. They support basically the same interactions, but the differences in how they implement them make the toolkits relevant to our analysis.

Toolkits overview. MRTK started out as an AR toolkit on HoloLens V1 but has since grown into a versatile XR toolkit that unifies the way interactions are supported and implemented on different AR and VR devices. For example, on HoloLens V1, it limits the interaction model to gaze pointing and air-tap selection, while for V2 it supports both near (hand pointing and touch selection) and far manipulation (hand pointing and grab selection). On VR headsets, it supports both motion controllers and hand tracking, if available. Such complexity in the interaction devices and techniques translates into a large set of components, many of which are adaptable for more than one interaction modality. Some of them may be configured for obtaining different and custom interactions, while others provide support to common interactions. It contains also a set of widgets for creating UIs panels.

SteamVR is a popular toolkit focused on VR experiences supporting all the major VR headsets and the corresponding motion controllers. SteamVR offers a small set of components focused on VR interaction. It does not compose them into a widget library. It contains a generic component marking interactive objects in a scene and supports more complex interactions through the composition of additional components. For instance, for dragging and throwing an object the toolkit provides a component for the former and the latter interaction, but there are no predefined interaction sets.

Method. We analysed the main example scene provided with each considered toolkit to understand the toolkit structure and interaction definition better. Such example scenes have a reasonable complexity and aim to cover most of the toolkits' features, making them highly relevant for our analysis. For MRTK, we selected the `HandInteractionExamples` [Mic22a]. The scene contains simple 3D objects like polyhedra, mugs etc. demonstrating manipulations in different modalities, virtual objects representing keys or buttons for demonstrating press interactions and panels or widgets for demonstrating the interactions with UI elements. For SteamVR, we focused on the one showcasing interactions with different categories of objects [Val22]: UI elements, throwable objects, sliders and handles-based manipulation, and virtual remotes. We manually identified and counted the toolkit components associated with the objects in the scene, listing the supported interactions (e.g., focus, grab, hold, press etc.), the interaction technique (e.g., gaze pointing, near or far hand interaction, laser pointing etc.), and how it was defined at the source code level (e.g., events used, listener registration, coding the effects). We used such data for identifying differences and similarities between the toolkit and as ground truth for validating our modelling (see Section 6.5).

Similarities. We identified the following common traits between the two toolkits:

1. *Components and Events*. Both toolkits comply with the entity-component-system structure used in all game engines. They consist of a set of components raising events for notifying the interaction sequencing.

2. *Event Ordering*. Both toolkits define interactions through an explicit ordering among events, splitting them into an ordered sequence. For instance, the manipulation interaction consists of a manipulation start event, followed by a certain number of manipulation changes and then a manipulation end.

3. *Custom configuration*. Both toolkits include highly customizable components, able to express quite different interactions. They require in-depth analysis to establish which interaction they support on the current object.

4. *Interaction effects*. In both samples, we noticed that dedicated custom components define the interaction effects, for easing its replication across different game objects.

Differences. The following are the main points where the two toolkits differ:

1. *Interaction Modalities*. MRTK supports gaze pointing and near and far interaction with both hands or remote controllers. SteamVR instead supports only remotes for near and far interaction.

2. *Predefined interactions*. MRTK includes many components specifying pre-defined interactions, such as hover, click, push, hold, manipulation etc. In SteamVR, it is possible to support all these interactions, but the developer needs to configure the game object through different components.

3. *UI widgets*. MRTK contains components and widget objects for creating UIs in the XR experience. SteamVR supports the interaction with objects representing UI elements, but it does not provide any widgets.

4. *Complexity*. Supporting configurable and pre-defined interaction components, different interaction modalities and UI widgets makes MRTK a complex toolkit aiming at offering out-of-the-box solutions to developers. This results in components defining overlapping interactions (i.e., we have different solutions for creating the same interaction). SteamVR follows a different approach, offering lightweight components supporting simple interactions. Developers obtain more complex ones by composition.

Summary. MRTK and SteamVR follow two different engineering approaches: MRTK cover multiple modalities and provides dedicated components for specific interactions,

while SteamVR includes a small set of components but highly customizable. Both approaches rely on event ordering for correctly specifying interactions: dedicated components have dedicated events, customizable ones have polymorphic events. The behaviour of the objects is defined either through event handlers or through specific components (e.g., rigid body, colliders etc.).

6.3.1 Modelling Toolkits and Interactions in the Scene in XRSpotlight

In our model (the ***Model*** green box in Figure 6.5), we represent a ***Toolkit*** supporting XR interactions as a set of ***Components*** (similarity 1) that developers attach to interactable objects. The modelling abstraction XRSpotlight uses for representing interactions in XR experiences enables a consistent representation of interactions across different XR devices and toolkits (see Figure 6.5). We started by leveraging the toolkit similarities to build its foundation and we followed a hybrid approach for modelling different interaction aspects [PCD92]. Each component provides a set of ***Events***, notifying interactions involving the considered object. A toolkit *Component* associated with an XR object provides a set of interactive ***Behaviours*** (e.g., focus, grab, hold, press, rotate, scale, translate, manipulation, hover etc.). Usually, developers exploit a subset of these behaviours to create the interactions (difference 2). The analysis of the two reference toolkits also highlighted differences that challenge the modelling. The first is providing a representation of the event sequencing (similarity 2), which should be understandable for novices without further documentation, but consistent across all the interactions (difference 4). We propose using an abstract point-and-select sequence independent from the pointing device. Inspired by Buxton's model of graphical input [Bux90], we defined the ***Phases*** as the events related to the change of state in such an abstract interaction, which a pointed object should hypothetically raise. The following is the list of the identified *Phases*. While not all of them are used in every interaction, they can consistently cover all sequences in the interactions we analysed in both toolkits. We modelled the phases following the *superset* approach in [PCD92], by providing an abstraction that covers the entire spectrum of the analysed toolkits.

- **Enter**: The user is currently pointing at the object, without confirming the selection.
- **Select**: The user has confirmed the selection.
- **Move**: The user moves the pointer without releasing the selection (e.g., still pressing the selection button).
- **Release**: The user releases the selection.
- **Leave:** The user moves the pointer out of the object.

The second modelling challenge we encountered in creating the abstraction was specifying which type of controller the user is provided with for interacting with the environment (difference 1), without replicating the interaction definition for each device. We

introduced the *Modality* concept as a concrete technique for performing the pointing, including a pointing and a selection technique. Currently, our tool identifies the following modalities (but the set may be increased in the future):

- **Gaze-pointing and air-tap selection** (e.g., gaze-and-commit interaction in MRTK)
- **Hand pointing and touch selection** (i.e., selecting by touching with the fingertip)
- **Hand pointing and grab selection** (i.e., selecting by grabbing an object)
- **Laser pointing with hand-gesture selection** (e.g., point-and-commit in MRTK)
- **Laser pointing with remote controller button selection** (i.e., using remotes for pointing and selecting)

To map toolkit components towards the model, we rely on a *toolkit mapping* file containing: 1) the *Behaviours* supported by the components in the toolkit ; 2) the *Phase* and 3) the *Modalities* associated with each interaction event in the toolkit components (see the blue box called Mapping in Figure 6.5). Such mapping allows managing the overlapping interaction definitions in MRTK and the simpler approach of SteamVR simply and consistently (difference 4). Each toolkit supported by XRSpotlight requires the definition of a mapping file. The effort required for defining and maintaining it is similar to providing metadata required for customizing the appearance of a component in the Unity Inspector (i.e., inserting custom attributes on variables and methods), and it depends on the toolkit's complexity.

We rely on ***Rules*** (Model box in Figure 6.5) for providing a simplified representation of interactions provided by an XR object. Rules follow the Trigger-Action paradigm [UPYHB+16], including the trigger and a set of *Actions*. We selected such a paradigm given its effectiveness in no-code programming tools (ECARules4All - Chapter 5, [UPYHB+16, CDRMR19, BCF+21, FST+19]). In our model, a trigger is an instance of an *Event* defining an interaction in a toolkit component. The trigger is the event that fires a rule. We know which events define an interaction from the mapping file described in Section 6.3.1. From a modelling point of view, a trigger is the concrete implementation of a *Phase* (i.e., an abstract event) in a given *Modality*. Therefore, we may identify a trigger through a pair $\{p, m\} \quad p \in Phase, m \in Modality$ (similarity 3). The arrows going from the *Rule* element to the *Phase* and *Modality* represent the trigger in a rule. In the rule visualization panel, we assign specific names to represent each trigger since the name of a phase is quite abstract. For instance, we call *drag* the move phase in the modality hand pointing and touch selection. An ***Action*** corresponds to invoking a method provided by a component on a game object as a reaction to the rule trigger (Similarity 4), which is the lowest common denominator in defining reactions to toolkits events [PCD92]. When the interface supports different interactions to achieve the same effect, the tool produces a different rule for each trigger, with the same action as a consequence. A specific grouping technique allows the identification of multiple triggers resulting in the same action. We

decided to avoid introducing UI widgets in our model since it already grasps their interactive part, while the specific appearance and behaviour are not in the scope of this work (difference 3).

6.4 XRSpotlight Implementation

To demonstrate the effectiveness and validity of the modelling for supporting the features we designed, we implemented them using Unity. After we have seen them in action in the walkthrough (Section 6.1), we show in this section how they work in the general case. This section follows the ordering of the features in the overview (Section 6.2). The current implementation consists of a Unity Editor Panel extension in C# at a research prototype stage, aiming at demonstrating the technical feasibility and the advantages of XRSpotlight. The source code is publicly available on GitHub[1].

6.4.1 Rule Inference Algorithm

The rule inference algorithm generates the rule-based description using a constrained natural language representation of the interactions on an XR scene. It requires as input a Unity scene and a toolkit mapping file, containing a priori knowledge for mapping the considered toolkit towards our abstract model (currently available for MRTK and SteamVR). It starts by identifying the objects that support interactions by listing those currently contained in the scene. For each of them, it exploits the Unity API for listing its components, searching for instances of those included in the toolkit mapping. If it contains a mapping for at least one of these components, the algorithm marks the object as interactable. For each interactable object, the algorithm lists the *Behaviours* according to the knowledge about the component in the toolkit mapping (e.g., the object is throwable, grabbable etc.). As we already explained in Section 6.3.1, these are the available interactions. Some of them may be exploited and have associated rules, others may not. For creating the *Rules*, the algorithm loops on the events defined by the interactable components. When it finds one event having one or more attached listeners, it retrieves its *Phase* and *Modalities* from the toolkit mapping for creating the rule trigger (or triggers if the event supports multiple modalities). The action part consists of the listeners attached to the event. Each listener contains the information for invoking a method on a Unity object, including an optional parameter. We generate natural language sentences using a constrained grammar, by composing predefined descriptions for triggers and actions into the rule structure.

For showing an example of the inference procedure, we consider the rule *When the egg is hovered, then the egg calls showIcon*, listed among those in Figure 6.6. The algorithm recognises the *egg* as an interactable object since it has the `ObjectManipulator` among its components, which is included in the MRTK [Mic22b] toolkit mapping file as shown in Table 6.1. Among the component's events, the file maps the `OnHoverEntered`

[1] `https://github.com/cg3hci/XRSpotlight`

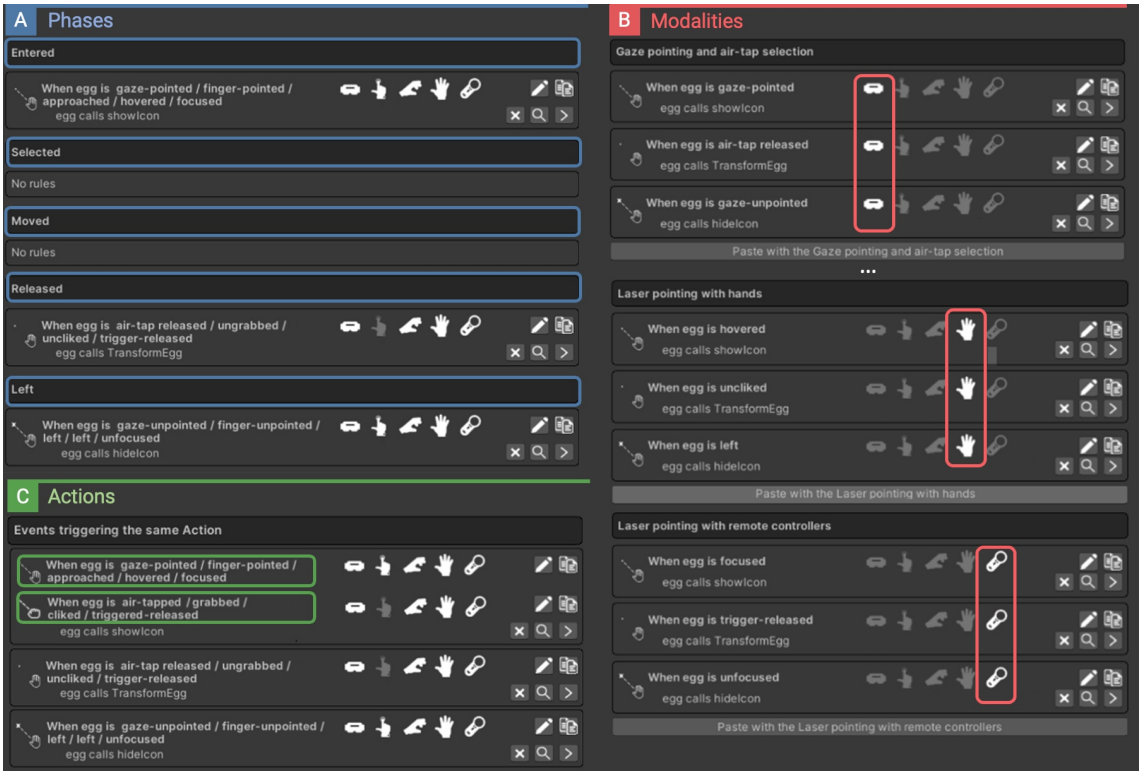


Figure 6.6: Different rule groupings for representing the same interaction (the egg hovering feedback described in Section 6.1). In part A, rules are grouped by phases (blue boxes), showing multimodal reactions to three phases: enter (pointing started), release (the selection disengaged) and left (pointing ended). In part B, rules are grouped by modality (red boxes). Rules working in multiple modalities are reported in each corresponding group, using the specific trigger name. In part C, rules are grouped by action, highlighting the events sharing the same effect (green boxes).

event to the *enter* phase for all the supported modalities. Since the *egg* object contains a handler for such an event, the algorithm infers one trigger for each modality. Our example relates to the modality of the *laser pointing and hand-gesture selection* and the *enter* phase. The corresponding action describes the Unity Action associated with the event handler, which invokes the `showIcon` method for providing feedback on the ingredient selection.

XRSpotlight provides different views on the initial rule set inferred by this simple procedure. The first is the *Phase* strategy, which creates a group for each phase, by combining their actions, as shown in Figure 6.6-A. The technique creates a compact representation by aggregating multiple triggers (i.e., phase-modality pair) in a single rule. It shows the differences in the phase handling among the modalities. The disadvantage is the difficulty in relating the changes in the Unity Inspector and the resulting multimodal rules. Indeed, the rule obtained through the grouping combines multiple toolkit events, having the same effect but raised by different interaction techniques.

Table 6.1: A JSON excerpt from the MRTK [Mic22b] toolkit configuration file. It contains a list of the components making an XR object interactable. We show the entry for the `ObjectManipulator`, which defines a Unity event in the `OnHoverEntered` property field. Such event corresponds to the *enter* phase, and it supports the five modalities (pointing and selection technique): i) gaze and air tap, ii) hand pointing and touch, iii) hand pointing and grab, iv) laser pointing and hand, v) laser pointing and button. Further events have similar entries in the `event` array, while further JSON objects in the root array describe other interaction components.

```
[  ...,
     {
          "className": "Microsoft.MixedReality.Toolkit.UI.
             ObjectManipulator",
          "isComponent": true,
          "events": [
            ...
            {
              "reference": [{
                "member": "property",
                "name": "OnHoverEntered"
              }],
              "phase": "enter",
              "modality":  ["gaze", "hand-touch", "hand-grab",
                            "laser-hand", "laser-button"]
            },
            ... ]
   },
]
```

The second strategy groups the rules by *Modality*. We insert each rule in a separate group for each modality, duplicating those having more than one (see Figure 6.6-B). This view provides a direct relationship between a rule and the correspondent definition in the Unity Inspector since each rule has exactly one trigger. However, it could create a longer list of rules applying to more than one modality. The last strategy groups the rules by *Action*, see Figure 6.6-C. If the same action appears in more than one rule, we combine the phases and the modalities in the final rule. The visualization has the purpose of supporting the developer in finding out how many events have the same listener, which is useful for isolating a given dynamic behaviour. It does not support creating a general understanding of the entire interaction with the object, but it aims at highlighting the actions having multiple triggers (i.e., different interactions having the same consequences).

6.4.2 Rule Similarity Algorithm

Mapping the interactions towards the abstract modelling elements allows for defining a similarity relationship, considering which ones are different and establishing a distance

function among them. Since a rule consists of a trigger and a set of actions, we define the similarity function considering both elements. If we compare the triggers of two rules r and s, we can distinguish whether they are defined on the same XR object or not ($T^o_{r,s} \in \{0,1\}$), whether they have the same *Phase* or not ($T^p_{r,s} \in \{0,1\}$), the number of modalities which are not in common between the two triggers ($0 \leqslant T^m_{r,s} \leqslant 5$). Instead, if we compare two actions α, β, we can check whether they are defined on the same XR object or not ($A^o_{\alpha,\beta} \in \{0,1\}$), whether they invoke the same method of the same component ($A^m_{\alpha,\beta} \in \{0,1\}$), whether the method has the same parameter or not (if any) ($A^p_{\alpha,\beta} \in \{0,1\}$).

$$
\begin{aligned}
t(r,s) &= w_1 T^o_{r,s} + w_2 T^p_{r,s} + w_3 T^m_{r,s} \\
a(r,s) &= \sum_{\alpha_i \in Act_r} \min_{\beta_j \in Act_s} \left(w_4 A^o_{\alpha_i,\beta_j} + w_5 A^m_{\alpha_i,\beta_j} + w_6 A^p_{\alpha_i,\beta_j} \right) \\
d(r,s) &= w_t t(r,s) + w_a a(r,s)
\end{aligned}
\tag{6.1}
$$

We obtain the final distance value as defined in Equation 6.1 by weighting the different values we get from the trigger comparison ($t(r,s)$) and aggregating those obtained by the comparison of the actions (Act) belonging to the two rules[2]. Such a definition allows limiting the comparison on triggers or actions, which is useful for the copy-paste implementation.

6.4.3 Copy-Paste Implementation

When a rule is copied, XRSpotlight serialises its definition in a file located in a fixed path, to allow data sharing among objects in different Unity projects. Figure 6.7 shows the algorithm for pasting the copied, highlighting the different paths XRSpotlight follows when the source and the destination scene are the same (green), different scenes using the same toolkit (blue) or different scenes using different toolkits (red). For pasting the copied rule, the tool searches for the trigger having the highest similarity score (i.e., the minimum distance) in the components provided by the destination XR toolkit, by using the $t(r,s)$ function in Equation 6.1. In the case of many triggers having the maximum score, the selection process prefers components that the object already contains. Otherwise, it attaches the selected component to the destination. If the tool does not identify any trigger having a similarity higher than zero, the paste fails, and the tool shows an explanation message.

Next, for each action in the source rule, XRSpotlight tries to create an exact copy in the destination rule. This is possible only if the object involved in the source action is also defined in the destination scene, i.e., when the copy and the paste happen in the same scene.[3] Otherwise, the tool shows a list of game objects that may replace the one in the source action since they support the same method. If no object in the scene provides

[2] we use $w_{1\ldots6} = 1$ and $w_a = w_b = 0.5$

[3] In general, the game object involved in an action may be different from the game object where the

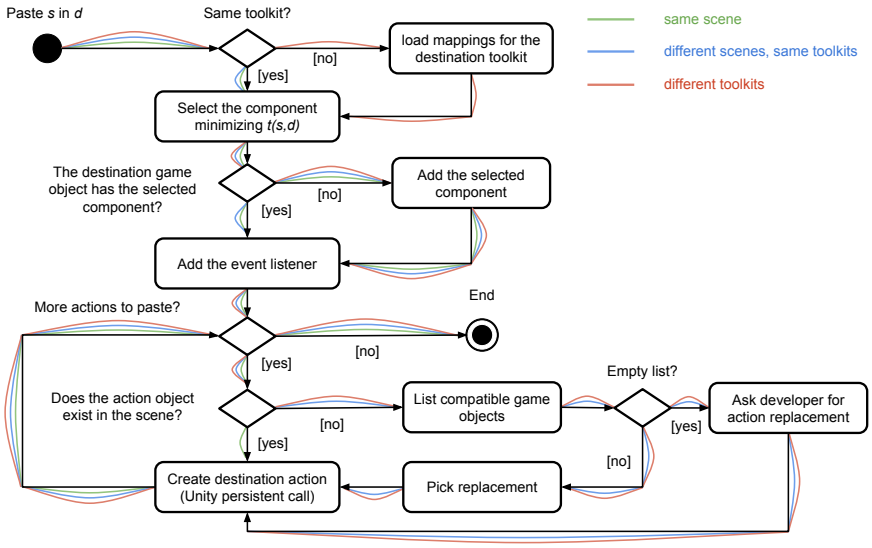


Figure 6.7: How to paste a previously copied rule s on a destination d depends on the target scene. In the same scene (green), the trigger uses the same source component and event. The actions do not need to find compatible objects since they are available in the scene. Considering two different scenes (blue) using the same toolkit differs for the action replication, requiring that the involved scripts are available in the target scene, and the developer selects among the compatible objects. Otherwise, the actions require replacement. Finally, the paste between two different toolkits (red) requires establishing a correspondence between the two toolkits through the model. The action replication logic is similar to the previous case. We provide a concrete copy-paste examples in Section 6.1.

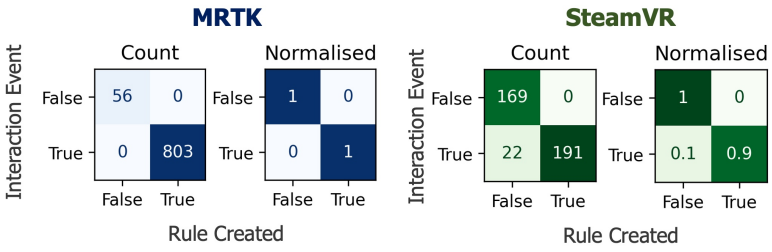
the same method defined in the source action, it is not possible to recreate the action in the destination scene. In this case, the list would be empty and the tool suggests the developer where s/he can add a custom replacement (if needed).

Another version of the paste feature is available when rules are grouped by modality, to support replicating the same interaction using different techniques or devices. In this case, we limit the trigger similarity search only on the components in the destination toolkit that support the selected modality. We use the copy-paste procedure also to move the effects of a rule towards another trigger, eventually deleting the original rule. We designed the copy-paste feature to replicate the interaction structure from the source object without establishing a synchronized link. This approach allows novices to tailor interactions to their specific needs without the risk of automatic overwrites from the original. A live link could hinder customization and introduce unintended disruptions if modifications to the source rule affect dependent copies, potentially leading to unpredictable project behaviour.

---

rule is defined. For instance, in rule `when elevator_button is clicked then the elevator_door calls open`, the rule is defined on the `button_elevator`, while the action involves `elevator_door`.

Table 6.2: XRSpotlight performance in identifying interactive events in the MRTK and SteamVR sample scenes. Step 1 considers the binary classification problem of whether a given event associated with an XR object represents an interaction. True positives are interaction events correctly identified (rule created), true negatives are events not related to the interaction not generating a rule, true negatives are interaction events not identified, false positives are events not related to the interaction generating a rule. We do not consider events having no associated handler. Step 2 instead reports on the correctness of the rules generated from the identified events (i.e., true positives in Step 1), reporting the count of the correct rules and rate.



Step 1: Identifying Interaction Events	Accuracy	Precision	Recall	F-Score
MRTK	1	1	1	1
SteamVR	0.94	1	0.90	0.94

Step 2: Rule Correctness	MRTK	%	SteamVR	%
Correct Rules (correct trigger and actions)	683	85%	191	100%
Wrong Rules (wrong trigger and actions)	0	0%	0	0%
Wrong Rules (wrong trigger, correct actions)	120	15%	0	0%
Wrong Rules (correct trigger, wrong actions)	0	0%	0	0%

6.5 Toolkit Model Validation

We inspected the main example scenes of MRTK [Mic22a] and SteamVR[Val22] using XRSpotlight for validating the underlying model by assessing the accuracy and correctness of the interaction representation.

Method. We proceeded with the assessment in two steps. First, we checked whether XRSpotlight identifies the events defining interactions on XR objects. We consider it as a binary classification problem: given an XR object event, XRSpotlight establishes whether it defines an interaction or not. If the algorithm described in Section 6.4.1 generates a rule then the event is considered interactive, otherwise it is not. We obtained the ground-truth through manual inspection and labelling of the two example scenes. The second step consists in assessing the correctness of the rule identified by XRSpotlight. We consider a rule as correct if it defines the correct trigger and the correct actions, while it is wrong if we have an error in either part. We again used the data we produced through manual inspection as ground truth.

Results. As shown in Table 6.2, the identification was fully correct with MRTK, while

there were 22 interactions not identified in SteamVR. XRSpotlight missed their definitions because it cannot find interactions defined through scripts in code, as this information is not directly exposed in the Unity Inspector. They are explicitly designed to show how developers can customize the Interactable component beyond its standard support, going down into its abstraction. This can be regarded a technical limitation. It has less of a measure of success for our modelling approach since the two ways of defining interactions are equivalent. We would require a code analysis module for finding the ones missed with SteamVR.

On the other hand, the rules identified in SteamVR are all correct. We have lower performance for MRTK, where rules are correct in 85% of cases. The errors are all related to the handling of two components. The first is the `BoundsControl`, which attaches sub-objects representing manipulation handles for resizing and rotating that are not represented in the rule trigger. The second is the `InteractableOnPressReceiver` that has a `InteractionFilter` field which allows filtering the modality between hand pointing with touch selection and laser pointing with gesture selection or both. The generated rules are correct only in case both modalities are allowed.

6.6 User Evaluation

We conducted two user studies, one with expert developers and one with novices with two main goals: *(1)* verifying that the toolkit abstraction aligns with experts' mental models of how interactions should be implemented (calibration), and *(2)* assessing the benefits and shortcomings for novice developers in inspecting and creating XR interactions using our system (validation). After assessing the correctness of the representation with experts, we ensure that its level of complexity is adequate for novices.

Both studies were conducted remotely using Zoom video conferencing. XRSpotlight was presented from the laptop of the moderator, passing screen control to participants during tasks. The two studies lasted one hour, and each participant was compensated with an Amazon gift card for their time ($30 USD for experts and $25 USD for novices).

6.6.1 Calibration with XR Development Experts

In the calibration with XR experts, we wanted to observe how experienced Unity developers approach the implementation of interactions using a familiar XR toolkit with the goal of establishing their mental model (i.e., the steps they envision for implementing the interaction). Then, we used this mental model to identify opportunities for improvement, both in terms of how we conceptualized interactions in our model and how we facilitated implementation through our XRSpotlight system.

We defined expert developers to have at least two years of experience in professional XR development. We recruited six participants in our collaboration network via email, three with expertise in MRTK and three with expertise in SteamVR. All participants were

male, and their age ranged between 26 and 33 years old. They had at least four years of expertise in XR development.

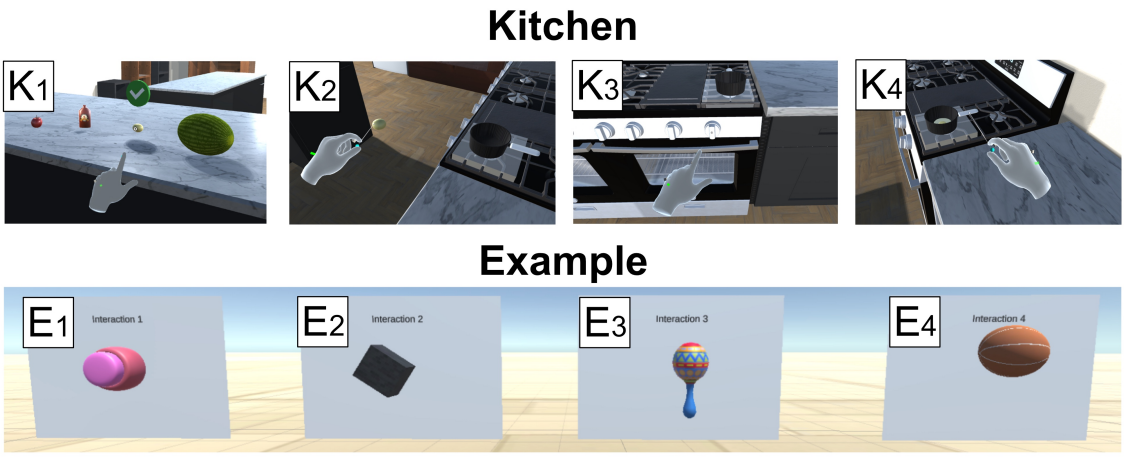


Figure 6.8: Overview of both experts' and novices' studies. The tasks inside the kitchen scene are represented in the first row (K1-K4). The second row shows the Example scene containing four example interactions: A button that pops up red particles when clicked (E1), a cube that rotates when focused (E2), a maraca that plays a sound when manipulated (E3), a basketball affected by gravity when grabbed (E4).

Table 6.3: Summary of the two studies organization. The tasks refer to Figure 6.8

Study	Solution Type	Tool	Task 1	Task 2	Task 3	Task 4
Expert	Mental Model	None	K1	K2	K3	K4
	Inspection	XRSpotlight, Inspector	K1	K2	K3	K4
			System walkthrough			
Novice	Inspection	XRSpotlight, Inspector	E1	E3	E2	E4
			Comparison			
	Implementation	XRSpotlight Inspector	K1	K3		
			Comparison			
	Mental Model	None	K4			

Method

The evaluation was structured into three parts (see Table 6.3).

Mental Model. In the first part, participants were asked to articulate their mental model about completing four VR implementation tasks conforming to four steps in a recipe to cook fried eggs. As a starting point, we provided a Unity scene called Kitchen,

furnished with kitchen-style 3D assets and with their preferred toolkit (MRTK or SteamVR) already imported. Each of the four tasks consisted of implementing an XR interaction (see Figure 6.8, kitchen row): *(1)* pointing at the egg shows a 3D icon to indicate it is the correct ingredient, *(2)* moving the egg closer to the pan cracks the egg, *(3)* switching the stove button produces the sound of a frying egg, and *(4)* dragging the pan reduces the smoke. Participants were asked to think aloud and edit the Kitchen scene, providing some stub implementation for facilitating the articulation of their mental model. They were also given access to the toolkit documentation and Google. At the end of each task, we asked them about their confidence in their solution on a 7-point Likert scale. Since we recruited experts, the overall goal was not to judge the mental models of the experts. Rather, we wanted to see to what extent the mental models matched the modelling abstraction used in XRSpotlight and to identify commonalities that may later help novices form a correct solution to these tasks.

Inspection. In the second part, participants inspected the scene we prepared with our working implementation as solutions to the previous tasks, using the Unity Inspector window or XRSpotlight (in both cases the provided solution is the same). For each task, we asked them to inspect the scene and to explain 1) how they think our solution works and 2) the difference with respect to their solution. We used the first explanation for ensuring the correct understanding of the solution, in particular, while using XRSpotlight. Instead, listing the differences was useful for establishing the conceptual gaps between a solution supported by XRSpotlight and the one they proposed. Half of the participants inspected the first two interactions using only the Unity Inspector and the remaining two using XRSpotlight. The other half used XRSpotlight first and then the Unity Inspector, to reduce order effects. We kept the order of the tasks. We applied a direct observation strategy [LHV+18] since we were interested in collecting qualitative feedback on how they would approach the inspection task. At the end of the inspection session, we asked them to list two advantages and two limitations of solving the tasks with the Unity Inspector alone as compared to using XRSpotlight complementing it.

Walkthrough. The last part consisted of a walkthrough demonstration [LHV+18] of XRSpotlight's main features to critique them from the experts' perspective. We pre-recorded videos demonstrating three key features: "finding similar interactions", "copy-and-paste within the same toolkit" and "copy-and-paste across different toolkits." For each feature, we asked them if and how they would include it in their professional workflow and why.

Feedback

The toolkit abstraction aligns with the expert's mental model. All experts described correctly the solutions we proposed for the tasks in Figure 6.8 (kitchen row), with both the Inspector and XRSpotlight. We can summarise the differences the experts identified between their mental model and the proposed solutions into three categories: 1) use of the collider and the rigid body, 2) selection of a different event for triggering the interaction and 3) adding further checks in the triggered method for fine-tuning the effect. All the

identified differences would result in a solution XRSpotlight can represent, and the expert did not highlight any gap in the abstraction concepts. Instead, they considered the complexity of the toolkit modelling to be at the right level of abstraction for novices (P2: "*the categorization about move, select, enter [...] are more meaningful to me, but also for novices*").

XRSpotlight provides a useful interaction overview. The experts noted the main advantage of XRSpotlight over the Unity Inspector is its centralized representation of interactions, which allows designers to have a useful overview of the interactive capabilities in the current XR experience. P1 appreciated that "*everything about interactions [is] gathered in one place*", while P5 added that it "*shows how the objects interact in the scene*". In addition, participants also appreciated the balanced display of information, which they considered sufficient for understanding the interaction but not overwhelming. P2 commented that "*in the Inspector I have a lot of Unity properties that I don't care about. I care more about triggers and interactions. In [XRSpotlight] it's very clear to me.*"

Need for fine-grained interaction descriptions for more advanced developers. While the experts expressed that XRSpotlight provides an ideal level of abstraction for novice developers, they identified a need for more fine-grained controls to achieve more complex or robust interactions. For example, many experts (P2, P3, P4, P6) wanted to implement constraints on interactions, (e.g., by using colliders to ensure the pan is dragged to a minimum distance away from the stove), which the current XRSpotlight implementation cannot identify, since requires a code analysis in addition to scene inspection, but they can be expressed in event-condition-action rules (See Chapter 5).

Finding similar interactions for debugging purposes. Regarding the feedback we received in the walkthrough demonstration, the feature for finding similar interactions (see Figure 6.3) was considered useful since Unity scenes are usually full of objects and interactions, and there is a lack of targeted locating technique. P2 and P4 suggested that they would use this feature for debugging purposes, seeing if they have forgotten something or checking the implementation. This was a very interesting point since we designed the feature for supporting exploration and not debugging.

Copy-paste increases the development efficiency. All the experts agreed that the copy-and-paste feature inside the same toolkit could be useful in their workflow. Some interesting observation came from P2: "*I can for instance copy from an object that has the interaction that I want without recreating it completely and I can be sure that the interaction weights are the same.*", and P3: "*I feel like this is a nice little kind of quality of life feature. I'm sure that Unity doesn't have a way to do that by default.*" The participants recognized the copy and paste between toolkit feature as helpful in two contexts: migrating to another toolkit and for novice users that want to understand the mechanism inside the scene. However, P6 thinks that "*this is really cool from a technical standpoint, I appreciate it, but I'm not sure about the use cases in a real workflow because if I'm working on a project that has SteamVR and then I have used MRTK before, I feel that there are so many differences in what I'm doing [... and it] would probably require more work than redoing it. I can see some cases where it could be useful but not in my real workflow.*". We agree with this comment since we designed the feature for novice developers,

who may struggle in identifying the needed elements or even which toolkit they should use [ABM+20], and not for experts that may be fluent in more than one toolkit.

6.6.2 Study with Novice XR Developers

After the calibration with the experts, we evaluated XRSpotlight with novice XR developers, i.e., people having familiarity with programming languages but little experience in building XR environments and using game engines. The study has two different goals: i) understanding the benefits and limitations of using XRSpotlight compared to traditional means (using only the Unity Inspector) and ii) seeing to what extent novice developers shape a correct mental model so that they could correctly approach the development of XR interactions. The study group comprised eight participants, seven of them had a bachelor's degree, and one had a master's degree. Participants' ages ranged from 22 to 30; six were male, and two were female. All participants had at least one year of experience in XR programming, but none self-reported being experts. We recruited them by contacting students of XR development courses in our Universities.

Method

We first explained to the participants the goal of the study, the information we were going to collect and we asked them to sign the informed consent. After that, the study consisted of three parts: the inspection, the implementation and the mental model assessment in a specific scenario (we provide a summary in Table 6.3).

Inspection. In the Inspection tasks, they were asked to inspect the Unity scene called "Example Scene" represented in Figure 6.8-Example. They had to inspect and describe (think-aloud protocol) the user's interactions and effects for 1) interactions E1 and E3 and 2) interactions E2 and E4. Half of the participants started using Unity Inspector, and the other half used XRSpotlight. We introduced our tool through a short video walkthrough of the system. After each task, we asked the participants to rate the difficulty level on a 7-point Likert scale. At the end of all the inspections, we asked for feedback on comparing the task completion under both conditions. Before moving on to the second part of the study, participants watched a video showing the interactions and the effects for each example in Figure 6.8-Example since they would need them as starting points for the next tasks.

Implementation. The second part consisted of implementing K1 and K3 (see Figure 6.8), the same as the experts' study. Participants could use the example scene to transfer the needed interactions, but unlike the experts, they were asked to provide a working implementation in Unity. Again, half of the participants used Unity Inspector first and XRSpotlight after, with a reversed order for the other half. We asked for the difficulty level on a 7-point Likert scale after each task, and we asked them to reflect on the advantages and limitations of the two conditions at the end of the implementation part.

Mental Model Assessment. In the final part, they had to describe how they would obtain K4 (see Figure 6.8), the last interaction in the experts' study (dragging the pan), by

Table 6.4: Correctness criteria per task type.

Solution Type	Correct	Partially Correct	Incorrect
Mental Model	Answer contains a strategy for adding a correct interaction component *and* triggering the right event and effect.	Answer contains either a wrong interaction component or trigger.	Answer contains both a wrong interaction component and trigger.
Inspection	Answer contains the correct type of interaction and effect.	Answer is wrongly describes an interaction or effect.	Answer is missing an interactions or effect.
Implementation	Answer contains all the correct elements (i.e., collider, script, interaction component and event)	Answer is missing the collider required for detecting the interaction.	Answer is missing a required interaction component or the event is not exact.

using the think-aloud method and with a particular focus on MRTK. Ultimately, we asked the participants how confident they were about their solution on a 7-point Likert scale.

We tracked the time on task and counted the number of accesses to elements in Unity, providing irrelevant information for the current task to assess possible efficiency differences between the two conditions. In addition, we evaluated the correctness level of the tasks applying the criteria in Table 6.4, depending on the type (inspection, implementation and mental model assessment).

Results

Inspection. Regarding the inspection tasks (E1, E3, E2, E4), we registered a slightly higher completion rate for XRSpotlight (13 to 12 correct, 1 to 1 partially correct and 2 to 3 incorrect). Figure 6.9 (top) shows that XRSpotlight performed better on E3, while the Inspector has an advantage on E2. We registered the same trend in the time on task. This slightly higher completion rate reflects on reporting the perceived difficulty since XRSpotlight has been found slightly easier to use (mean, median and mode for the two conditions are the following: $M_{xrs} = 1.25$, $Md_{xrs} = Mo_{xrs} = 1$ vs $M_{ins} = Md_{ins} = Mo_{ins} = 2$). A clear advantage provided by XRSpotlight in this type of task is finding interaction-relevant information mostly at the first attempt. The Inspector instead requested one more attempt on average, especially in the first two tasks (E1 and E3).

Figure 6.9 (bottom) provides an interesting perspective on the same data. It splits participants by starting condition, i.e., grouping those who started with the Inspector first and then used XRSpotlight (I→X) and those who followed the inverse order (X→I). In that case, the results show that participants starting from XRSpotlight completed all inspection tasks correctly and spent less time in the inspection. The perceived difficulty was lower when using XRSpotlight (E1 and E3), while it was higher for tasks completed through the Inspector (E2 and E4). We can provide a consistent explanation for these trends, which is confirmed by participants' feedback and observation (see Section 6.6.2):

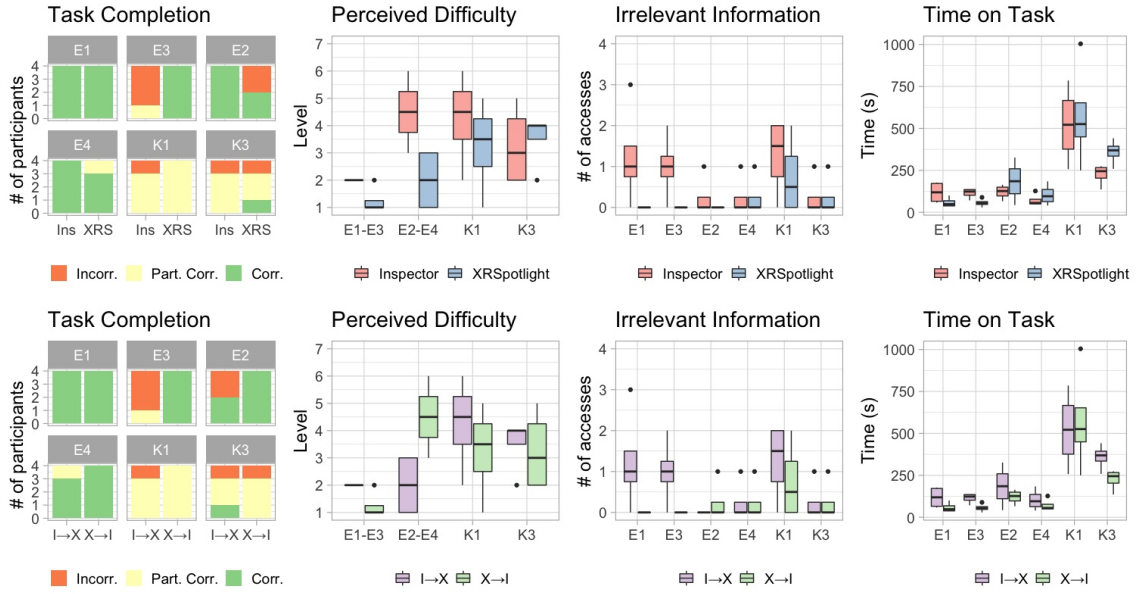


Figure 6.9: Summary of the data collected during the novice study, including the correctness of the proposed solution to the task (see Table 6.4), the perceived difficulty, the number of accesses to irrelevant information during the task, and the time on task. We report the values per experimental condition (Inspector and XRSpotlight, top row) and per group defined by the condition ordering (bottom row): Inspector - XRSpotlight (I→X) or XRSpotlight - Inspector (X→I).

XRSpotlight favours the transition from a guided to an autonomous inspection. Indeed, after solving the first two inspection tasks through our tool, participants acquired the ability to identify the relevant elements in the Inspector for describing the interaction, and they made no further mistakes with the Inspector, completing the tasks in less time.

Implementation. Regarding task completion, using XRSpotlight, one participant completed the task correctly, one incorrectly (wrong event choice), and the rest partially correctly (forgetting about the collider). Using the Unity Inspector, two participants did the task incorrectly (wrong event choice), and six partially correctly (forgetting the collider). The perceived difficulty for the implementation tasks is comparable in both conditions. For XRSpotlight, part of it was caused by switching between two different views instead of using the same one from the beginning, which also explains the higher time on task for K3.

Mental Model Assessment. In the third part, almost all of the participants (7 out of 8) discussed a correct mental model for completing the presented scenario. The confidence level on the solution was high ($M = 5.125, Md = 5, Mo = 5$). It is encouraging to notice that they could work on the same task as the experts and be almost successful as them (the mental model assessment task was the same across the two studies).

Feedback

Simplicity. The inspection tasks highlighted that participants appreciated the simplicity and the reduction of the viscosity [Ols07] supported by XRSpotlight. Four participants out of eight affirmed that it was easy to understand the rules with XRSpotlight, even for non-experts (P2, P4, P6, P8). In addition, P7 said that "*you can easily understand what to do in a Unity scene*". They motivated this explicitly by referring to the list of the interactable objects and that "*the user can better focus on the relationship between the type of interaction and then the effect it's clear on the screen.*" (P3).

Transition out of the tool. The completion rate trend discussed in Section 6.6.2 shows that XRSpotlight helps transition from the rule-based simplified abstraction to a proficient understanding of the example organization in the Unity Inspector. P8 relied on the field identified through the XRSpotlight highlighting feature for inspecting the next examples using the Inspector. P5 mentioned that "*[...] it's easier to manage the rules with the XRSpotlight window and then set more parameters in the Inspector*". Commenting on the implementation tasks, three participants justified their confidence rating by saying that they have just understood how to do this interaction recalling the events and components to use in MRTK but depicting the solution in terms of the proposed rule-based description. This suggests that applying such representation for learning purposes may be a relevant future direction of the project, which we will cover in future works.. For instance, P7 stated that "*I should create a rule and . . .*", while P3 "*I would [select the correct game object] and add the selection interaction as the effect of shaking the pan*".

Copy-paste is the most useful feature. In the second part of the study (implementation tasks), seven participants out of eight reported the copy-and-paste feature as the main advantage of the system. P7 said that "*it's easier to copy rules to have them immediately, using the Inspector you have to create them*". All the participants used this feature while doing the task with XRSpotlight, even if two participants said that they would have preferred to create a rule from scratch through XRSpotlight. Most of the participants have chosen the source rule having the most similar type of interaction. Other participants preferred the effect similarity: they started from a rule having a similar effect compared to the one they wanted to create. In this case, some of them used the feature for changing the trigger to adapt it.

Novices want to edit rules in XRSpotlight. To obtain the correct rule when using XRSpotlight every participant had to use the Unity Inspector window to adapt the pasted rule according to their needs. As we expected, six participants stated that they would have liked to edit the effect of a rule directly from XRSpotlight, without changing their focus to the Inspector. This is clearly a fair limitation, considering that we took the explicit design decision of not making XRSpotlight a replacement for the Unity Inspector, avoiding duplicating its features.

Usability and missing features. We could distinguish the identified limitations into two categories: i) interface usability issues and ii) missing features. In the first category we have the vocabulary used for the triggers that were confusing for some participants, the phases without rules were considered unclear and P1 did not grasp the relationship

between the action label and the method's name. We may address all these problems including further information (e.g., help) in the XRSpotlight panel. Among the missing features category, two participants requested to have more settings related to colliders and interaction components. Indeed, in E2, two participants misunderstood the effect of the cube interaction using XRSpotlight.

6.7 Discussion

Simplifying the example inspection. In the complex landscape of existing toolkits for creating XR interactions [NS18], we introduced a tool empowering novice developers in example-driven development targeting the difficulties in selecting the right toolkit and example resources [ABM+20]. We consider as novices people with basic programming knowledge but without experience in XR or game engines. We showed that the underlying abstraction captures the peculiarity of two different and widely-used toolkits, such as MRTK [Mic22b] and Steam VR [Sof22a]. Besides the technical validity of the modelling, we also demonstrate the abstraction matches the experts' mental model when planning the implementation of XR interactions (see the study with experts in Section 6.6.1). This has particular relevance for novices, whose target is acquiring and mastering the experts' mental model. Such abstraction eases the reasoning about how to create interactions provided by concepts explicitly designed for modelling such interactions without getting lost in the hurdles of the XR environment representation [IK15, NS18], as shown in the novice study.

Transitioning out of XRSpotlight. The ceiling is one of the main problems affecting simplified XR development tools (see Section 3.1): when novices require going beyond the specific predefined tasks or platforms, they must face the complexity of professional game engines such as Unity [ABM+20]. XRSpotlight is the first tool proposing a simplified abstraction of XR interactions *inside* a professional game engine, aiming at guiding novices through its complexity rather than replacing it. In our view, linking the simplified view (XRSpotlight's rules) with the detailed definition in the Inspector allows for a guided transition *out* of its support. We found evidence of the desired effect in the higher completion rate of the group that started the novice evaluation using XRSpotlight.

Creating interactions through copy & paste. It is also relevant to analyse the differences in the opinions on the copy-paste feature between novices and experts. Such a feature provides semi-automatic support for transferring an example of a relevant interaction into the environment under development. When transferring from a toolkit documentation sample, the effect of the interaction usually needs adaptation by the developer, so the transferred information is the toolkit component and event the sample uses (i.e., the trigger). Such information is relevant for novices and less useful for experts, who can select a trigger without guidance. However, they found the feature useful in the case of repetitive interaction definitions in the same scene.

Limitations. There are some limitations in this work, which would require further research in the future. Avoiding the replication of the Unity Inspector editing features in-

side XRSpotlight was a design choice we carefully considered. The evaluation with novices highlighted that they would like to edit rules in the panel since switching between XRSpotlight and the Inspector requires effort. We think that such effort should not be completely removed, otherwise there will be no transition out of our abstraction and the novices will be locked into XRSpotlight. Further research is required to find the right balance between the editing effort and the transition support. As mentioned in Section 6.3 we focused our study on SteamVR and MRTK due to their popularity in the field. Given constraints in resources and development time, we did not extend our investigation beyond these frameworks, aiming to maintain efficiency and expediency in our research process. The tool is implemented as a research prototype, and the evaluation highlighted some usability issues requiring some development work. Nevertheless, the provided features demonstrated that it eases novices in inspecting and the understardanding XR examples. The feedback and the performance in the novice study suggests that XRSpotlight may be useful for learning how to correctly implement XR interactions. However, more research is required for assessing its effectiveness and we plan to cover this aspect in future work, like inviting more participants, and proposing a more open-ended evaluation of the toolkit. The rule representation may be improved by providing a summary of the actions' effects. This would require code analysis, but it would increase the novice's understanding. Through code analysis, we can also identify the interactions not listed in the Unity Inspector panel. Since the number of interactions defined only through code in real-world projects are relevant, this may result in an incomplete overview of the environment interaction capabilities. Finally, the tool requires extensions to further modalities, such as vocal or gestural interaction, and for supporting the identification of multimodal interactions.

Chapter 7

ViRgilites

In this chapter, we present ViRgilites, a system aiming to guide users in performing multimodal interactions in a VR environment. ViRgilites is a meta-level user interface [Cou06], meant to support the user to explore, discover and perform interactions with virtual objects. Missing visibility of possible interactions, the lack of standardised, widely accepted interactions with 3D objects and the uncertainty of the outcome of executed interactions in VR, MR and AR environments make these very challenging environments to support efficient interactions and avoid erroneous user actions. Our approach informs users about the possibilities to execute tasks in goal-driven virtual environments, improves the discoverability of tasks supported by the virtual environment and provides feedforward on what follows when a selected interaction has been performed. We included an approach to automatically recommend and prioritize interaction modalities depending on the application domain, the context of use and the predefined preferences of the designer. We validated ViRgilites in a formative study using two real-world use cases, focusing on gathering user feedback on the design and anatomy of the meta-user interface.

7.1 Multi-Level Interaction Guidance for Virtual Reality

We distinguish between tasks, actions and interactions. *Tasks* are sequences of actions that a user has to complete for reaching a desired goal[1] An *action* represents a step towards the completion of a task. Considering the different modalities usually available in XR environments, the user can select among different *interactions* to perform the same action. Therefore, for each action, we have a set of interactions that are equivalent in terms of effects.

The interaction level feedforward, as also shown in Figure 7.1, presents the various options to perform an interaction to the users, ViRgilites provides designers and developers with the freedom to provide one or more interaction techniques within a virtual

[1]+In our examples, we consider simplified scenarios where a training procedure requires completing a single case. In the general case, a procedure may require the completion of more than one task.

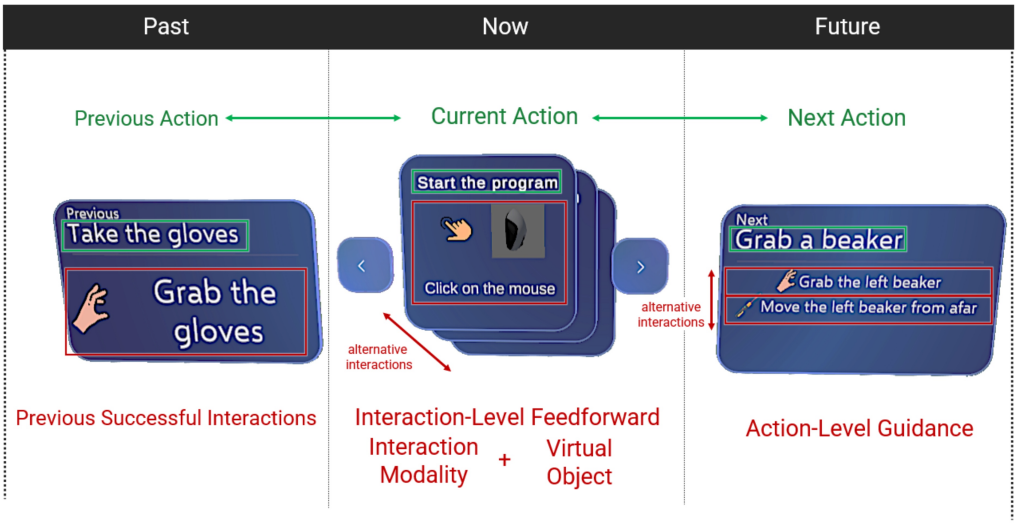


Figure 7.1: The anatomy of ViRgilites: from feedback of past actions and interactions, over the current action and interactions at hand, to guidance toward the next action and interactions. In ViRgilites, the virtual object depicted alongside the interaction, provides a shortcut to navigate to the object in the virtual environment.

environment, while minimising the cognitive effort for users to explore, discover and use possible interactions.

Multiple sensory affordances convey information about the available interactions on a given object to trigger the same system function (i.e., complete the same action). A simple yet informative example is designing an interface suggesting how to grab an object to move it elsewhere. If we are designing for VR devices supporting hand tracking, we may want to support such operation at design time through a grabbing action. In addition, we also want to support the selection from a distance, using the laser pointing technique for reducing the environment navigation. In such a setting, we need to reveal the same functional affordance (the position change) using two sensory affordances, one for communicating the grab action and one for the laser-pointing. The physical affordance (i.e., the object shape) should communicate that we may grab the object, but it does not communicate how to laser-point it, since usually the shape does not change according to the interaction modality.

The action level guidance is included in ViRgilites to ensure users are aware how their actions contribute to completing a task and reaching a goal. In this sense, a series of actions is like an operation procedure that users have to perform. For ViRgilites to be useful, such operation procedures tasks that consists of multiple actions need to be defined for a virtual environment, as happens in [LLZM+21, WRF+22]. This ensures actions have a context and purpose, and are not isolated with respect to the other actions performed with the virtual objects. It also helps the user to become aware and knowledgeable on the various tasks that are supported in the virtual environment.

Such multi-level interaction support allows users to understand which interactions are

available in the simulation and to anticipate their effects. Suppose we model a VR task's solution through the classic human problem-solving theory by Newell and Simon [NS72]. In that case, ViRgilites allows users to understand 1) which operators are available and 2) which is the next state in the problem space reachable by applying each operator. This provides information on the possible changes at an *action* level. However, users also require information on the outcome at a *task* level, i.e., whether or not the new state gets closer to the desired world configuration or which one among the possible new states is the closest to the user's goal (or sub-goal). Considering a training scenario, the feedforward support should communicate how a given interaction contributes to the task's progress, helping the trainee towards the goal of the exercise.

Another relevant aspect of our work is the supporting a mixed initiative process for selecting the interaction modality. While there are different options for triggering a function, some are more effective than others in different contexts. For instance, a vocal modality should be avoided in a noisy environment, while hand-based interactions should not be available if users have their hands full. On the one hand, XR applications have different means for sensing the context, thus the UI may react to changes and adapt the interaction by proposing the most suitable modality given the sensed data. We discuss a generic prioritisation method for controlling this aspect during the UI development. On the other hand, users must have the final word on the selection, so the UI must contain means for overriding the system choice if needed.

7.2 The Design of ViRgilites

The method we have followed for designing the interface of ViRgilites is the following: first, we reviewed the literature related to existing solutions for these problems [MMH23, VLvdHC13, WDO04, BEDF15, Nor99, Nor08]; then we organized a design workshop with 7 experts where various design alternatives were proposed. These alternatives were discussed and resulted in an initial prototype design for ViRgilites. A functional implementation of the design was realised, and further iterations were made on the design to obtain a workable prototype. For these iterations, we always took into account the basic design principles: visibility, affordances, mappings, metaphors, consistency, feedback, and constraints [Nor02]. In the following paragraphs, we will summarize how the design workshop results influenced our design choices.

Interaction Alternatives During the design workshop, one of the aspects that challenged the participants was how to represent alternative ways of reaching the same goal. In addition, XR interaction environments support different modalities for performing the same functional action (e.g., grab and laser-pointing selection). All participants agreed that the guidance interface must represent such alternatives. Some distinguished between those that originated by the procedure and those generated by the interaction modality. We eventually used the same representation for both sources, but we made

such a decision passing through different iterations that separated the modalities from the alternative steps in the procedure.

Alternative Selection This design choice concerns who is responsible for selecting alternatives for completing the current task. Some participants would opt for a completely adaptive solution that assigns this concern to the system, others for an adaptable interface that gives complete control to the user. The discussion on this point led to a mixed solution: the system selects the alternative it considers the best, but the user can override the selection at any time. In ViRgilites, the system proposes an alternative according to a prioritisation function, and users can change this proposal whenever they like.

Feedforward Levels When discussing which information to include in the interface, the proposals covered both feedforward information on the outcome of the current interaction (*interaction-level* feedforward) and how it contributes to reaching the current goal (*action guidance* level). The guidance is peculiar of goal-oriented interactions, where the system can establish in advance the user's goal (e.g., a correct final configuration in a training environment).

Direct vs Indirect affordances There are many alternatives for designing both the functional (interaction outcome) and physical affordances (how to act) in XR environments, as summarised in [MMH23]. The most relevant aspect of our work is distinguishing between a *direct* or *indirect* representation. The former relies on a preview simulating the targets, the actions and the outcomes in the XR environment. The latter relies on abstractions, such as explanatory text or icons. During the workshop, participants highlighted that it is possible to provide a generic direct representation of the action for performing an interaction (e.g., grabbing, pointing, looking at an object, etc.). However, the same does not apply to the action outcome, whose direct representation requires a customised design for each step. This limits the design of a generic guidance interface, which was the focus of the workshop. Therefore, they opt for indirect approaches which allow the generation of the affordances using structured information. We eventually selected a text-based format, but it is possible to select other media. In addition, we selected to use the same representation (indirect) for both affordances to concentrate the information on the guidance panel.

Feedback Besides the notification of the correct execution of an interaction, participants agreed on including a feedback element in the guidance interface, to show the history of the previously executed actions in the procedure. An interesting discussion among the participants highlighted the trade-off between the number of previous steps maintained in the UI and the space required for displaying the information. On the one hand, increasing the number of previous steps helps users remember the previous actions leading to the current state. In a training context, this may affect the learning outcome. On the other hand, the visualisation of many steps may visually interfere with the

surrounding environment, occluding the user's view. Participants concluded that keeping going beyond the previous action would cost too much UI space.

Future Steps Including information about the next steps in the guidance UI poses a design problem similar to maintaining the information about the previous interactions. How many steps should be included in the visualisation? However, while past steps represent a linear path, the action in the future may include alternatives. In this case, the design choice concerns i) how many next steps should be included, ii) how to represent the alternatives and iii) how to select and/or filter them for the presentation. We tried different solutions during the iterations, representing the alternatives for the next steps in a more compact visualisation than the current interaction and using the same prioritisation function for sorting them.

7.3 Anatomy of ViRgilites

The design of ViRgilites consists of three parts, as outlined in Figure 7.2:

the left panel (Figure 7.2-Pa) contains information on the previously completed action and the interaction selected for completing it,

the middle and main panel (Figure 7.2-C) displays how to execute the current action and presents alternative interactions that have the same result, and

the right panel (Figure 7.2-Na) shows information about the next action that will follow when the current one is completed, and the corresponding supported interactions.

ViRgilites presents the current context of a task by showing the latest previous interaction thus informing the user what was done to get to the current state, and by presenting what the next actions will be when the current one is completed successfully. Notice the next action (right panel) can change while exploring the options for current action (middle panel), thus providing feedforward on what can or should be done when completing the current action in a particular way. For example, consider the task of moving a box from one shelf to another. If the user is considering a grab action the box from the first shelf using the virtual hand modality, the next action of the same task is navigating the VR environment until reaching the second shelf, which will appear in the right panel (Figure 7.2-Na). An equivalent action in a modality supporting selection from afar (e.g., laser pointing) does not need an intermediate navigation, the user can directly drop the box from afar, and the action in the right panel will be updated accordingly. Section 7.4.2 describes our approach to calculating the order in which the suggestions are displayed. The middle panel represents the alternative interactions to reach the same result as a stack of cards (Figure 7.2-C). The user can navigate through this stack and explore and select different ways to complete the current actions (Figure 7.2-P and N). The panel on the left (Figure 7.2-Pa) contains feedback informing the user on the last completed action.

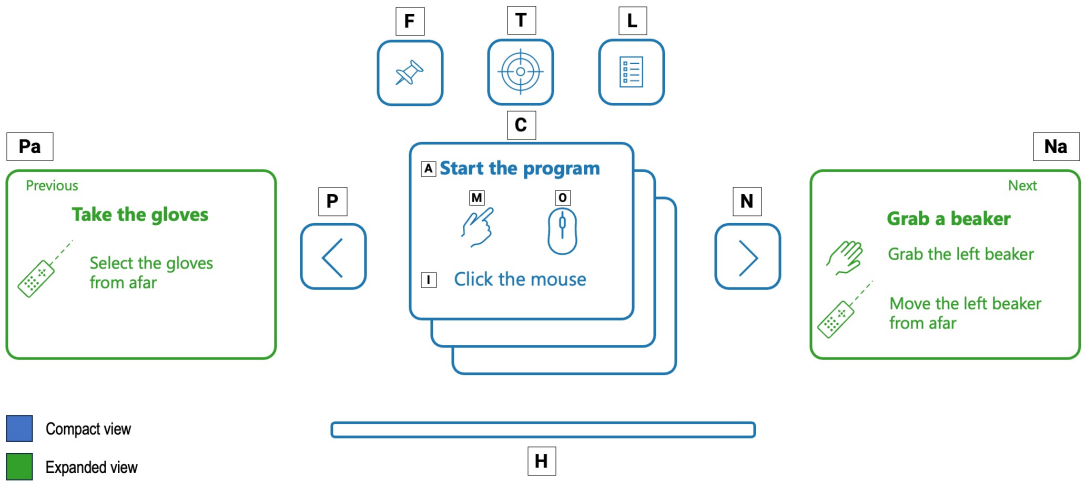


Figure 7.2: Organisation of the ViRgilites guidance interface. It contains three panels: the middle (C) containing information on how to execute the current action, the left (Pa) on the previous successful action (and interaction), while the right on the next possible action and the associated interactions (Na). Users browse the available interactions (if more than one) using two dedicated buttons (P and N). The top-left push button activates or deactivates the pinned mode (F), the top-center (T) activates the target object highlighting, while the top-right (L) activates the compact (blue part) or expanded (blue plus green) view mode. The bottom handle (H) enables the reposition of the user interface in the virtual space.

ViRgilites uses it both as a task completion notification, moving a card from the central to the left to represent the progression and as a support for helping users recall past actions. We limited the number of past actions to one. The card representation is mostly the same as used in the central part, with the differences of the removal of the interacted object and the rearrangement of the information of the card.

The panel on the right (Figure 7.2-Na) contains further feedforward information related to the overall procedure goal. It shows a preview of the next steps in the procedure, provided that the user completes the current action using the modality selected in the card deck. Therefore, the content in the right panel *depends* on the selection in the card deck. The representation in the next action panel is slightly different from the others. First, it groups equivalent alternatives for executing the same action in a single group. The equivalent effect is the title of a list of modalities and operating suggestions. Second, it lists the various alternative modalities in a list representing branches in the procedure. We use the prioritisation function to order the available interactions in the card (see Section 7.4.2). In summary, the next action panel offers insight into the next step considering the current modality choice. In this way, the user can decide how to continue, considering the current action and the next step.

7.3.1 Exploring and Selecting the Current Interaction

The cards are sorted using the adaptive prioritisation function we discuss in Section 7.4.2, to help the user quickly find the most appropriate way to complete the current action. While the system controls the ordering of the alternatives at the beginning of each action, users are in charge of selecting the appropriate modality: they can keep the modality suggested by ViRgilites, or they can browse the card stack (Figure 7.2-C) until they find the most appropriate for them. Each card contains four pieces of information. The first is the goal of the current action (Figure 7.2-A), which anticipates the consequence of the interaction in the procedure completion context. The second is an icon representing the modality associated with the current card (Figure 7.2-M), together with another icon that represents the target object (Figure 7.2-O), which is a snapshot of the 3D object in the environment. The fourth is a label describing how to operate the interaction using the considered modality (Figure 7.2-I). The cards in the central part of the interface list the possible actions for continuing the procedure. They may be equivalent ways for activating the same function depicted as cards having the same title and target object but different modalities and operating suggestions. They may also represent branches in the procedure, i.e., entry points for different paths (sequences of tasks) leading to the same overall goal.

7.3.2 ViRgilites Placement

The pin push button in the middle part (Figure 7.2-F) allows users to control the position of the guidance interface in the 3D environment. By default, the pin button is not active,

and when activated, the panel "follows" the users changing its position and orientation that guarantees a comfortable interaction in the user's field of view[2]. ViRgilites freezes its current position and orientation in the 3D environment when the pin button is pushed once again. Regardless the state of pin button, the user can move the interface around using the handlebar (Figure 7.2-H) at the bottom. Users can then interact with the XR environment without the panel occlusion. Instead, the list button (Figure 7.2-L) hides or shows the left and right panels, switching between a compact and an extended version of the guidance interface.

7.3.3 Navigation aids

Navigating smoothly through a virtual environment is crucial for a user-friendly experience, especially when the user must interact with dispersed virtual objects. Our system addresses this by incorporating specific navigation features geared toward user-object spatial interactions. Interacting with the icon of the virtual objects in the top card in the central panel (Figure7.2-T), prompts the system to guide the user directly to a virtual object's location. This guidance can manifest in various forms, tailored to the most probable way the user will interact with the object. If the object is within reach and the user is expected to simulate real-world interaction, an arrow and line will appear, directing them toward the object until the action is completed. Conversely, if close proximity to the object is required –like touching or grabbing it– and it's too far away, the system will automatically teleport the user closer. These added features augment the existing navigation options provided by the standard VR setup.

7.4 Meta-level Interactions: Triggers, Progress and Adaptation

In the previous section we introduced the concept of multi-level feedforward for Extended Reality environments. ViRgilites can provide users with full flexibility to decide how they want to interact. They can navigate through the different options and pick the one that best suit their needs or preferences for completing tasks.

7.4.1 Triggering the interaction guidance

The effectiveness of a guidance system hinges not just on the information it provides, but also how and when that information appears [MMH23]. An essential aspect of its design is figuring out the right moments to activate and show the guidance. We call this a 'meta-interaction' because it shapes the way users interact with particular parts of the interface that guide the user on how to interact with the system [Cou06].

[2] please see https://learn.microsoft.com/en-us/windows/mixed-reality/design/billboarding-and-tag-along

One of such triggers to start ViRgilites is a change in *context* [DAS01]. A change in context could imply that some ways of interacting in the virtual environment might become more or less effective, thus should be taken into account by the guidance system. In our work, we have exploited the ideas of sensing and modelling context for reacting to its changes, since it is a widely investigated topic, and there are different dimensions relevant for adaptive behaviours such as user characteristics [PZCG13], physical elements [YLS+14] or social context [LPF11]. In some models, the task itself is part of the context [WV09]. In our work, changes in the context result in a different prioritisation of the steps in the training procedure described in Section 7.4.2. As a result, the guidance system may suggest another modality for completing the same task or an alternative path for achieving the same goal. For instance, consider a scenario where a user has the option to either utter a voice command or employ tactile interaction with an object. The optimal interaction modality depends on the ambient noise level. In a high-noise context, tactile interaction would be prioritised as the most efficacious mode of interaction. Conversely, in a low-noise environment where the user must navigate between distant objects, voice commands would be promoted as the most efficacious mode.

7.4.2 Adaptation and Prioritisation of the Steps

The middle pane of ViRgilites presents a stack of cards, containing all alternative interactions that can be performed at the current time, and that all have the same result. The suggested interaction is selected according to a priority function. We rank these alternatives using a prioritisation function that takes into account two classes of parameters: the requirements from the application domain and the context in which the interactions take place. Then, the best interaction is suggested to the user, through the user interface.

We represent a procedure as a set of interactions I, and each interaction $i \in I$ is associated with an interaction modality $m(i)$. Therefore, prioritising an interaction also prioritises the associated modality. Each function defines a set of parameters, and we represent them with the vector $\mathbf{v}_i$. We consider another vector $\mathbf{w}_{m(i)}$ of the same length for defining the weight (i.e., the importance) of each parameter for the given interaction modality. Both vectors contain values between zero and one, and the weights sum to one. Once the parameters have been identified, the priority of an action is simply the dot product $p_i = \mathbf{w}_{m(i)} \cdot \mathbf{v}_i$. We do not calculate the priority of all possible actions at each step, but we use the definition of the procedure for obtaining, given an interaction i, the set of its possible subsequent interactions $nxt(i)$ and the set of its alternatives $opt(i)$. ($nxt : i \rightarrow B, i \in I, B \subseteq I$), and the set of its alternatives ($opt : i \rightarrow B, i \in I, B \subseteq I$)

A similar approach is used for the visualisation of the list of the interactions that the user will do on the next step. In this case, the list is sorted following the same algorithm used for the suggestion of the best possible interaction, but uses an heuristic variant to evaluate the rank of each interaction.

We use both *static* and *dynamic* weights in our weights vectors. Static weights represent upfront, predefined values, such as the importance of using a specific modality

for the application domain. They encode the designer's knowledge of different aspects of the interaction and domain. Dynamic weights, on the other hand, represent values that can change over time in response to interaction or context-related events, such as the distance between the user and the target object. This can lead to different recommended interaction modalities for the same set of available interactions for the same action, depending on the situation.

To easily add or change the way prioritisation is done, we use a Strategy pattern [GHJV95]. This allows using ViRgilites in a wide variety of situations and application domains, and even change the approach for different types of users. The following parameters are considered when recommending interaction modalities:

- **Interaction fidelity** (static, designer-defined). The degree of correspondence between the current action and its real-world counterpart. For instance, a direct manipulation of an oven temperature knob is more faithful than a remote selection.

- **Target Distance** (dynamic, derived from the XR environment). The inverse of the physical distance between the user and the target object of the current action. The designer relies on the modality for establishing the weight (e.g., higher for direct manipulation).

- **Modality Change** (static, designer-defined). It estimates the cost for the user of a possible modality change for executing the next step. Denoting the current interaction as i and having a matrix C containing pre-defined costs for the change of each possible pair of modalities, the parameter value is $1 - \min_{n \in nxt(i)} C_{m(i),m(n)}$.

- **Context Information** (dynamic, derived from the context model). It expresses the feasibility of the current modality and/or action considering the current context model. The definition of this parameter is tightly coupled with the training procedure, and it may include environment sensing (e.g., avoiding voice commands in noisy environments), tracing previous executions of the procedure by the same trainee for identifying preferred interaction modes or prioritising unexplored procedure branches.

For a better understanding of how the algorithm selects an interaction with respect to another, we provide the following example in which we explain how our particular algorithm chooses between different types of interactions with objects. Imagine we have three interactions for selecting the same target object (e.g., collecting a screwdriver from a toolbox).

Grab - a 'grab' interaction

Far - a 'far' interaction (interaction from a distance)

Touch - a 'touch' interaction

Table 7.1: Example of Modality Costs table. The table shows the effort needed from current modality (row) to next modality (column). We evaluated every cell using the same criteria (hence the discrepancy in an inverse reading of the table). Higher values are better (0 → Very hard to switch between modality, 1 → no effort on changing).

	Gaze and Commit	**Touch**	**Grab**	**Voice**	**Far Interaction**	**None**
Gaze and Commit	1	0	0.5	0.5	0.5	0
Touch	0.5	1	0.5	0.5	0	0
Grab	0.5	0.5	1	0.5	0	0
Voice	0.5	0.5	0.5	1	0	0.5
Far Interaction	0.5	0	0.5	0.5	1	0.5
None	0.5	0	0.5	0	0.5	1

Table 7.2: Final Values for score evaluation. The chosen modality is underlined in bold. Context evaluation has been omitted since they scored all 1 and its contribution does not influence the final choice.

	Action Fidelity	**Distance from Target**	**Modality Change Cost**	**Final Value**
Grab	1	0.9	0.5	0.875
Far Interaction	0	1	0	0.5
Touch	**1**	**0.9**	**1**	**1**

In our scenario, the user's next task is to touch another object. For our example's purpose, let's assume:

- *Touch* and *Grab* have high 'interaction fidelity' (quality or realism of the interaction).
- *Far* has low interaction fidelity.

The system will assign 1 as interaction fidelity to *Touch* and *Grab* and 0 to *Far*. *Touch* and *Grab* involve a greater distance to the target than *Far*. The algorithm also considers the 'modality' of the interaction (the way the interaction is performed). Since a far interaction (like *Far*) can be done from anywhere in the environment, it scores higher than *Touch* and *Grab* in this aspect. In this case we can assume that *Far* gets 1 where *Touch* and *Grab* get less than 1 (we can say that, in a 10x10m square, the user is 1m away from the target object, so the final value will be 0.9).

The algorithm uses a predefined matrix to calculate the change in modality (the shift from one type of interaction to another, see Table 7.1). In our case:

- *Touch* scores 1, as we have defined that changing from one touch interaction to another is effortless.
- *Grab* scores 0.5, since from a grab interaction to a touch interaction the effort is minimum.

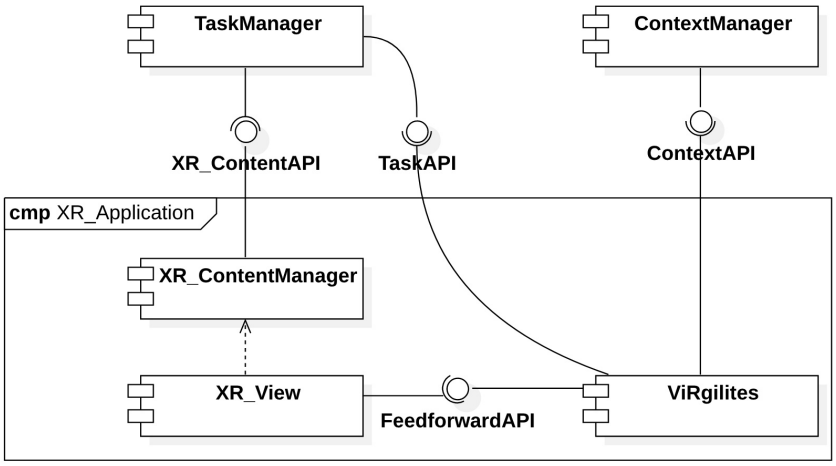


Figure 7.3: UML component diagram of the ViRgilites architecture. The *TaskManager* and the *ContextManager* provide the information to the feedforward component (ViRgilites). The *XR_ContentManager* synchronises the state of virtual and real objects that blend in the *XR_View*.

- *Far* scores 0 because for the user to touch the object he also needs to move and approach the object.

However, since there are no 'context changes' (factors related to the specific situation or environment), all interactions initially receive 1.

Now, assume the designer prioritized interaction fidelity and modality change equally in the algorithm ($w_{fidelity} = w_{modality} = 0.25$ but did not consider target distance ($w_{distance} = 0$) and placed a high emphasis on context information ($w_{context} = 0.5$). Given these settings, the algorithm will ultimately recommend *Touch* as the best choice (*Grab* = 0.875, *Far* = 0.5, *Touch* = 1 - see Table 7.2).

7.5 Architecture of the system

This section discusses the architecture of ViRgilites. Unity and the Microsoft MRTK 3 [Mic22b] toolkits are used for creating the virtual environment and managing the interactions. ViRgilites to supports the execution of a series of complex interactions in a VR environment without prior experience or knowledge of how the environment works. We validated this by using ViRgilites in two scenarios: one for training users in chemistry lab tasks using nuclear materials, and the other for training users in performing food preparation tasks in the kitchen. Both scenarios require users to execute a series of tasks that can be done in various ways; users can perform tasks like they would do in real life and train their motor skills too, or users can make use of other, more convenient, interaction modalities so they can focus on the high-level steps that need to be executed.

Figure 7.3 shows the components included in the prototype. The XR application consists of three main components. The *XR_View* visualises the XR content and interfaces, blending the virtual and real elements and managing the interaction. Such visualisation depends on the current state of the objects in the XR environment (both real and virtual).

An important requirement in a training setting is that the XR environment supports different training procedures, which may be modified or changed according to the varying learning needs. Therefore, hard-coding the procedure management in the XR environment limits its reuse. To overcome this problem, we designed the prototype applying the solution proposed in Chapter 5. This solution proposes the reproduction of virtual environments that can support more than one procedure, and by doing so not being hard-coded and locked to a single experience. These environments, called *templates*, then expose the control of managed (e.g. prepared beforehand in a way that can be retrieved by procedure actions) XR objects through the *XR_ContentAPI*. The advantage of such an organisation is twofold. On the one hand, it allows the modelling of the environment characteristics relevant for goal-oriented applications in XR. On the other hand, it allows the creation of goal-oriented procedures without requiring further builds of the XR application, thus enabling domain experts to author procedures without involving developers.

The *TaskManager* contains the logic for executing a training procedure (i.e. a task). In our implementation, its definition consists of a text file containing the name of the task and a list of *Action* elements containing the following information: i) the position of the action in the procedure; ii) the action description. In turn, each action contains at least one or more *Interactions*, representing alternative ways for completing the same step. An interaction contains the following information: i) two values defining the subject and the target of the action to be performed (e.g. if the user have to do something to an object called “Pan”, then the subject would be “Player” and the target would be “Pan”, if it is a collision between two objects, “egg” and “Pan”, subject will be “egg” instead); ii) the modality required for performing the interaction; iii) a text describing how to perform the interaction; vi) the array of values described in section 7.4.2. The *TaskManager* exploits the *XR_ContentAPI* for receiving notifications about the current interactions and updating the runtime state of the current task. Given the structure of this component, the prototype can execute different procedures for the same environment by simply loading other definition files.

Besides controlling the procedure progress, the *TaskManager* exposes the procedure state through the *TaskAPI*, which notifies the subscribing components about the completion of an interaction, the history of the previously executed actions and allows forward navigation of the possible options for continuing the procedure execution. The *TaskManager* supports a Unity event publisher that can be triggered for signalling the change of the task triggered by the correct execution of an action. Depending on the events that are triggered and the status of the application, The *TaskManager* can set active tasks, it can check whether tasks were completed, it can check whether a whole procedure was completed, it can provide the list of actions for a given step, and manages the progress of the procedure. ViRgilites exploits the APIs for controlling the information provided

in the *XR_View*. As described in Section 7.2, such presentation depends on the context. Considering the variety of entities and sensing devices contributing to building a context representation, we isolated its management into a dedicated component, again external to the XR Application. This solution allows having a context representation that fits the considered environment. Indeed, some require only simple information on the state of the world, while others may need to model even complex social relationships. The *ContextManager* is responsible for providing an API accessing the context information, hiding the modelling and sensing complexity to its subscribers. In particular, the APIs provide an event publisher that notifies the system of context changes (this event is triggered by the sensors in the user environment - managed by the *ContextManager* - and subscribed by the *TaskManager* in the proposed implementation). The component responsible for feedforward presentation instantiates the user interface depicted in Figure 7.2, utilizing MRTK 3 as the foundational framework [Mic22b]. It employs the prioritization algorithm described in Section 7.4.2 and interfaces with the *Context API* to query the context of use. The system constantly updates the guide based on both task conditions and the overall context. Finally, it allows the *XR_View* component to control the guidance system through the XR devices. ViRgilites updates the ranking of the "next" panel list using the associated triggered function, and it signals the change itself (mainly used for notifying the user of the update through alerts in the *XR_View*). ViRgilites implements functions for listening to context changes (published by the *ContextManager* and used by prioritisation algorithms that require assessing the context as a parameter) and a function that provides the final suggestion to the middle panel list. The system also takes from MRTK the default player prefab, which includes hand management support[3]. The hands, when used with supported XR devices (e.g. Meta Quest 2, 3 and Pro) enable the interaction with other objects in the scene using poke, grab, far ray, and gaze pinch interactors. To enable the communication between an MRTK-supported game object (i.e. a Unity GameObject including an MRTK-provided component such as PressableButton, ObjectManipulator or StatefulInteractable Components) and ViRgilites, it is required to insert a script that forwards the MRTK-interaction event towards the ViRgilites component for updating the guidance system.

7.5.1 Example Scenarios

To demonstrate the system's validity, we developed two simple scenarios, one depicting a waste disposal procedure for nuclear material in a chemistry lab and the other showing how to cook an Italian focaccia in a kitchen environment.

7.5.2 Chemistry Lab Scenario

This scenario is situated in a chemistry lab. To mitigate the risk of contaminating the surroundings or propagating hazardous substances, users are tasked with disposing of a

[3] https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-input/packages/input/hand-tracking#hand-controller-prefabs

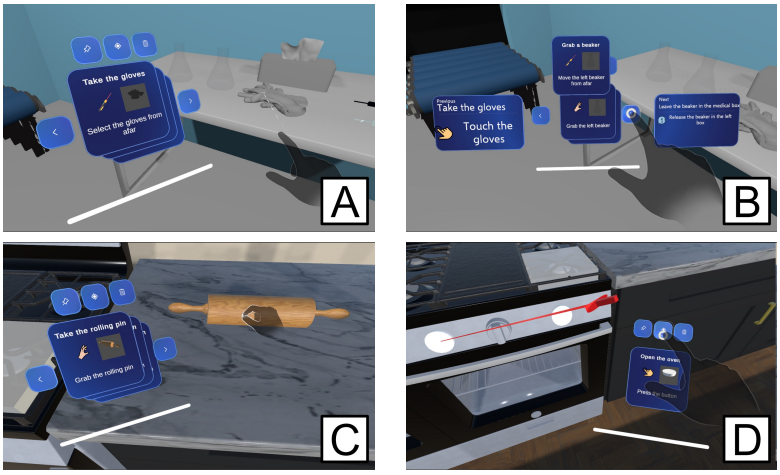


Figure 7.4: ViRgilites interactions in the chemistry lab (A, B) and kitchen (C, D) scenarios. Figure A shows the user picking up the gloves from afar. Figure B shows the user changing the interaction with another one. Figure C shows the user grabbing the rolling pin. Figure D shows the user trying to identify which object he should interact with.

beaker in a designated waste basket. This task requires precision and close adherence to the protocol. The sequence of actions to execute is as follows:

1. Start the computer program for registering the activities in the lab using the computer mouse in the scene;
2. Pickup a pair of gloves from a table and put them on;
3. Request a beaker from outside;
4. Grab the hazardous waste from the table;
5. Insert the hazardous waste in the beaker;
6. Grab the beaker from the table;
7. Position the beaker into the waste basket

At the beginning of the procedure, the interface initially displays itself in compact mode, directing the user to start by clicking the computer mouse. Initially, there are no alternative interactions. The user then uses standard MRTK gestures (as described in section 7.5) to teleport close to the computer desk and clicks the mouse. Once the mouse has been clicked, the interface updates to show the next step, which is grabbing some gloves. For this action, the system suggests using a near interaction for picking up the gloves, but after the trainee changes position, the prioritisation algorithm re-evaluates the parameters and the context of the environment, and determines that a far interaction is now more suitable for this step. The interface promptly switches to the correct

recommendation when this happens. Once the gloves has been acquired (as seen in Figure 7.4 - A), the interface updates again, instructing the user to request a beaker from another room to put the hazardous materials inside of it. This time, the user has the choice to either use a voice command ("Give me the beaker") or touch a button to get it manually. Since the user environment is quiet, there are no context changes related to it being noisy, and because of that, the system suggests using the voice command. The user tells the keyword, and the beaker appears in the scene. Now, they have to put the hazardous waste inside the beaker, which is positioned on the table near the waste baskets, and they can do it either by grabbing, ray selecting or gaze committing to the object. At this point, the user wants to know the next action to decide on the appropriate modality and switches to the extended view of the interface. In the next task panel, the user sees that they need to move the materials in the beaker. Since the user is far from the materials, they select the "gaze and commit" modality, look at the object and make a pinch gesture to start the movement phase; the user then inserts the materials inside the beaker by moving them near the beaker. The next task is to pick up the beaker. Here, the user is presented with the choice of grabbing the beaker, selecting it from a distance or gaze-committing from a distance as well, and the system recommends the far interaction, since according to the prioritisation formula, it is the most suitable interaction modality. Once again, having a look at the extended view of the interface, the panels show that they need to move the beaker into the waste basket. This time, they swap the card and select the grab interaction (Figure 7.4 - B). Finally, the user's task is to take the beaker to the waste basket, as indicated by the extended view of the interface. When this task is completed, the procedure finishes, displaying the "Procedure complete!" message in the middle panel.

7.5.3 Kitchen Scenario

A similar process can be illustrated for the kitchen scenario, where the objective of the user is to prepare a focaccia. To achieve this, the user needs to adhere to the subsequent steps:

- Preheat the oven;
- Take the rolling pin;
- Flatten the dough with the rolling pin;
- Take the oil;
- Put the oil on the dough;
- Open the oven door;
- Bring the dough into the oven;

- Close the oven door (for cooking the focaccia);
- Open the oven door (after the focaccia has been cooked);
- Serve the focaccia on a plate;

First, when the user enters the scene, the interface displays the compact view, similar to the previous scenario. The user is tasked to touch the oven button to preheat the oven. There are no alternatives so the user proceeds to click the button. The second step tasks the user to pick up a rolling pin. The recommended modality is to grab the rolling pin remotely. Still, in this case, the user would like a realistic approach and opts for the arrow controls of the interface to select the grab interaction instead. Once the user has done that (Figure 7.4 - C), the interface updates, illustrating the next action: flattening the dough on the table with the rolling pin, using any available means. In this case, the user has to bring the rolling pin using any possible interaction technique available, but since he already took the rolling pin with a grab interaction he proceeds to complete the action simply by moving the rolling pin to the dough. Now, since the user is distant from the bottle of oil in the scene, when the task for picking up the oil is presented, the system promptly changes the suggested interaction from grab to far interaction. The user this time is fine with the choice of the system and picks the oil bottle from afar. The subsequent interaction is similar to the flattening of the rolling pin so the user brings the oil bottle to the dough. Now, the system tells the user to open the oven door using a button. Unfortunately, the button icon is similar to the oven on/off button, so when the user presses the original button nothing happens. In order to disambiguate the image, he presses the locator button to visualise the arrow with the line pointing to the correct object. At the same time, since the selected modality is a touch interaction, the user is teleported near the target object to facilitate interaction with it (Figure 7.4 - D). Once the user clicks on the button, the interface asks the user to bring the focaccia to the oven grate. The user picks the focaccia from afar and places it into the oven. The last actions of the users are to close the oven by touching the correct button, opening it again (for the purpose of the scenario the focaccia is instantly cooked once the oven door closes) and bringing the focaccia into a plate for serving it. Once again, the interface will display the “Procedure Complete” message, for communicating the end of the procedure.

7.6 Formative and User Study

We conducted a combined formative and user study with the goal of explore potential issues to make ViRgilites more usable and increase the acceptance rate. The user study part focused on the usability of the current project, while the formative part focused on collecting feedback for future works. We invited 10 participants (1 female, 9 male, age range: 23 - 36 years old, Avg.: 26,8, Md: 27, SD: 3.823). In particular, we invited 5 participants with self-declared proficiency in VR environments, and 5 participants with

self-declared little to no experience. Participation was voluntary and no compensation of any kind has been handed out.

7.6.1 Method

The evaluation, after collecting the consent of the participants and their demographic data, was structured in four parts.

Tutorial: The participants were introduced to the interface, explaining its goal and its various parts, using a VR test scenario. The participants had to complete a simple task, while using ViRgilites and being guided by the evaluator on how to interpret the language of the interface. The task consisted in selecting three cubes in order (red first, green second, blue as last). Moreover, in a separate area of the same scenario, a set of cubes were available to be freely manipulated, and a nearby panel (with a different layout than the one used by ViRgilites) would show the same icons of the interface of ViRgilites and a text explanation of the performed modality.

Waste Management Scenario. In the second part, the participants performed 4 tasks in a virtual chemistry lab environment, while having the possibility to provide feedback after each task. After each task, the participants were asked about their level of comprehension of the user interface and, if multiple interactions were available for the current action, the understanding of the interaction suggestion based on the context. All ratings used a 5-point Likert scale. The four tasks performed by the participants were the following:

- Task 1 (T1): Perform an action with a single interaction available. The participants needed to touch a mouse on a nearby desk;
- Task 2 (T2): Perform an action with multiple possible interactions. The suggested interaction was the one to perform. The participants needed to perform a far interaction selection of a pair of gloves on a desk;
- Task 3 (T3): Perform an action with multiple possible interactions. The interaction to perform was not the suggested one. The participants needed to perform a grab interaction on a beaker in the same desk of the gloves.
- Task 4 (T4): Perform an action with a single interaction available. The participants needed to bring the beaker taken from the previous task to a nearby medical box. The difference between this task and the first one is that in this task there is no recommended interaction, and the task would have been completed as long as the beaker reached the medical box.

Kitchen Scenario. In the third part, the participants were required to complete a full task from start to finish without interruptions between the actions. The task (T5) to be completed is the same depicted in Section 7.5.3. Participants were free to complete the actions in the task as they seen fit, it was up to them whether they wanted to change the

interactions or keep using the ones suggested by the system. At the end of the task the participants were asked the questions in the previous part once more.

Semi-structured interview. In the last part we collected qualitative comments on the features of the system, any positive and negative comments about it, whether if they would have had ideas for improving the interface, and whether if they felt the level of feedforward implemented in the system would help an user in understanding how to do the task that the system is requiring him to do, and additional comments on the ranking system for the next action.

7.6.2 Results

Feedforward Comprehension

Over the five tasks, the participants were asked to answer the following question on a Likert scale from 1 to 5: "Did you understand how to do the interaction based on the information in the interface?", where 1 was "Not at all" and 5 was "Completely". This question aimed to gather information on the level of helpfulness the feedforward system ViRgilites currently implements.

As shown in Table 7.3, T1 scored the lowest score above all (Avg.: 3.8, Md: 3.5, SD: 0.92, Mo: 3, Min: 3, Max: 5), and the same behaviour is reflected if we consider the scores of the participants proficient in VR environments (Avg.: 3.8, Md: 4, SD: 0.84, Mo: 4, Min: 3, Max: 5) and the novice participants (Avg.: 3.8, Md: 3, SD: 1.09, Mo: 3, Min: 3, Max: 5). This can be expected since, even after the training session, we can consider T1 as a period of accustoming to the visual language of the system. The other tasks scored average values more or equal than 4.5 (T2: 4.9, T3: 4.7, T4: 4.5, T5: 4.7), and it is interesting to notice that the average scores for each single task tends to be generally higher with the novice participants with respect to the scores of the proficient ones, with the exception of T5, that shows the opposite trend (T2: 5 vs 4.8, T3: 5 vs 4.4, T4: 4.8 vs 4.2, T5: 4.6 vs 4.8).

Automatic Interaction Suggestion

For inferring whether the participants understood how the Interaction Suggestion system suggested a particular interaction with respect to the other ones we asked them, when the test task had an action with more than one interaction (T2, T3 and T5) the following question: "Based on the hypothetical scenario we gave you, did you understand why the system recommended you that interaction technique?", using the same Likert scale and extremes as described in Section 7.6.2.

Table 7.4 shows the scores for this question. In this case, they were lower with respect to the feedforward comprehension (T2: 3.9, T3: 3, T5: 3.9) showing nonetheless that there was a general understanding of the feature. In particular, it is interesting to notice that novice participants had a worse understanding of the feature with respect to the proficient ones (T2: 3.6 vs 4.2, T3: 2.4 vs 3.6, T5: 3.8 vs 4) meaning that experienced

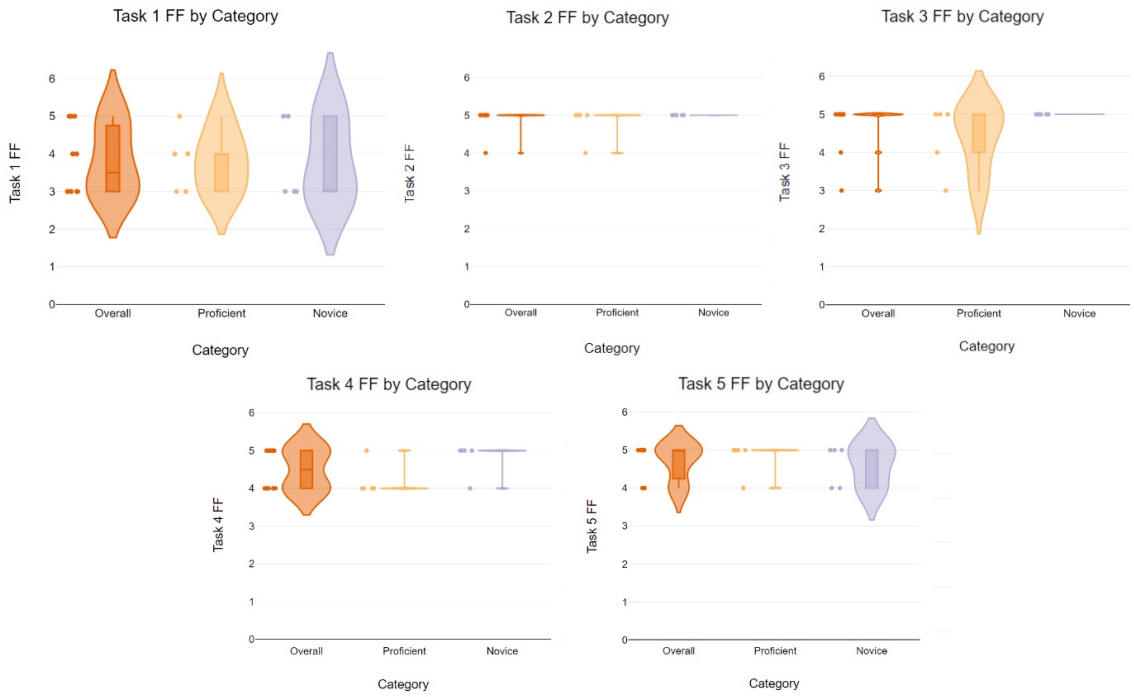


Figure 7.5: Violin box plots of the Feedforward comprehension results

Table 7.3: Evaluation Results for Feedforward comprehension from the participants

		Average	Median	STD	Mode	Min	Max
Task 1	**Overall**	3,8	3,5	0,92	3	3	5
	Only Proficient	3,8	4	0,84	4	3	5
	Only Novices	3,8	3	1,09	3	3	5
Task 2	**Overall**	4,9	5	0,32	5	4	5
	Only Proficient	4,8	5	0,45	5	4	5
	Only Novices	5	5	0	5	5	5
Task 3	**Overall**	4,7	5	0,67	5	3	5
	Only Proficient	4,4	5	0,89	5	3	5
	Only Novices	5	5	0	5	5	5
Task 4	**Overall**	4,5	4,5	0,53	4	4	5
	Only Proficient	4,2	4	0,45	4	4	5
	Only Novices	4,8	5	0,45	5	4	5
Task 5	**Overall**	4,7	5	0,48	5	4	5
	Only Proficient	4,8	5	0,45	5	4	5
	Only Novices	4,6	5	0,55	5	4	5

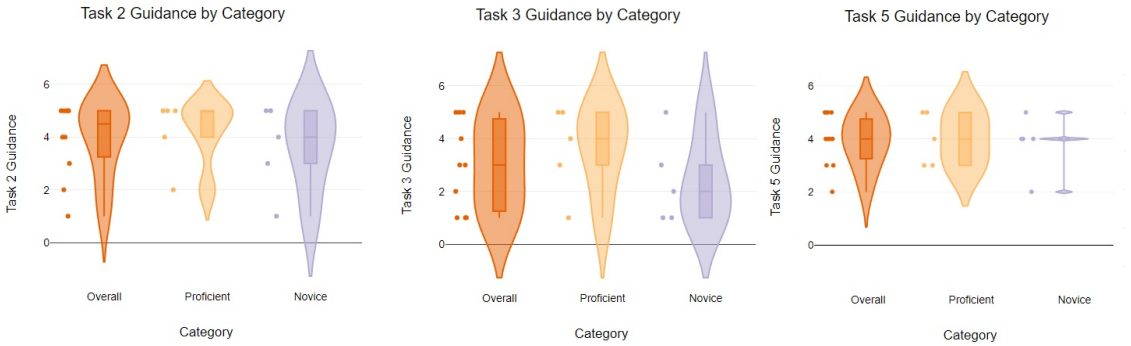


Figure 7.6: Violin box plots of the Automatic Interaction Suggestion results

Table 7.4: Evaluation results of the Automatic Interaction Suggestion from the participants

		Average	Median	STD	Mode	Min	Max
Task 2	**Overall**	3,9	4,5	1,45	5	1	5
	Only Proficient	4,2	5	1,30	5	2	5
	Only Novices	3,6	4	1,67	5	1	5
Task 3	**Overall**	3	3	1,70	1	1	5
	Only Proficient	3,6	4	1,67	5	1	5
	Only Novices	2,4	2	1,67	1	1	5
Task 5	**Overall**	3,9	4	0,99	4	2	5
	Only Proficient	4	4	1	5	3	5
	Only Novices	3,8	4	1,09	4	2	5

participants could grasp more the reasoning behind the selection of an interaction over the other available ones.

7.6.3 Discussion on Feedback

After gathering the experience of the participants we can report on the feedback provided by them.

Helpfulness of the feedforward interface. All participants agreed on the utility of the interface and its feedforward features, when explicitly asked to give an opinion on it. During the test tasks, the participants has been asked on which elements of the UI helped them understand how to do the action. Over the course of the five tasks, the most helpful element was the text (29 times out of 50, 58%), followed by the interaction technique icon (27 times out of 50, 54%) and the object image (19 times out of 50, 38%).

Utility of the automatic Interaction Suggestion. P6 and P8 appreciated the automatic selection of the interaction, P6 stated that because it removed the need to select the most reasonable one for the action to perform.

Object Locator is useful but improvable. The object locator, in the same questions as above, was less voted as useful (10 times out of 50, 20%), but in the open questions the participants agreed on its usefulness (P1, P2, P6, P9). P6 and P7 denoted that it may have been unnecessary to teleport the user when the locator button is pressed and the interaction technique to perform is either a touch or a grab one, because of the unexpected positioning of the teleportation (P6). P2 and P7 noticed the teleportation being too near to the target object, and P7 suggested to make the teleportation to a nearby position of the target object, instead of the object itself.

Utility of the Next panel A particular pain point that most of the participants (P1 - P5, P7, P9) stated, when asked about positive or negative aspect about the "Next" panel was that they did not feel the need to use it or they just forgot to have it. One participant (P6) appreciated its value and features ("It is very useful to already see the ranking of the modalities beforehand to know what is expected of you next and how it is done 'best'"). We can infer that the feature, despite not being prominent in the user interface, has potential to be useful in certain situations, but most of the times becomes an accessory information, not vital to the task execution. P5 and P8 suggested, as an improvement for the system, to keep the extended information on by default, and P8 also suggested to make the visualisation of the interface semitransparent, in a way that does not disturb the view of the user, while retaining its functionality.

Improvements to the system. Many participants took the opportunity, when asked, to express ideas for improving the overall experience with the system. In this section we are going to report the suggestions strictly inherent to the user experience with the interface, improvements of the toolkit itself will be omitted, because out of scope for this research. P2, P4 and P8 suggested a different way to identifying the object, by highlighting it instead of pointing an arrow at it. P4 also suggested to indicate an object being out of sight by using a lateral arrow to the field of view, like it was used by Lallai et al. [LLZM+21]. P1

suggested to put a sound trigger when the action was successfully performed, instead of relying solely on the user interface. P8 suggested to show only two cards when two interactions were available, because it felt confusing to see the discrepancy between number of cards and number of alternatives. Finally, P7 suggested to keep the position of the interface relative to the position of the user, instead of being absolute with respect to the position in the environment.

Chapter 8

BeatriXR

This chapter presents BeatriXR, a toolkit for integrating feedforward information in any VR environment. This system builds its features upon the work presented in chapter 7 by integrating a system for generating and showing in real-time how the user should perform an interaction, either by displaying it in the same environment of the user, or using a system of multiple windows that compare multiple interaction methods with respect to the same action. This toolkit only requires information about the target object and the interaction modality and, once triggered, it performs the animation to show to the user. This system provides a way to create more natural and intuitive VR experiences, using the information feedforward can provide. Moreover, showing different interaction techniques on a single object can provide a way for the user to tailor the experience to his needs, by choosing which interaction can be the best one for him (either for personal choice or user accessibility preferences).

The contributions of this work are two: an usable reference model that showcases in practice the features described by Muresan et al. [MMH23], and a methodology where the model, in conjunction with Large Language Models (LLM) allow to create environments where a procedure designer without experience in feedforward techniques gets alternative designs. We have validated the suggestions provided by different AI components with experts, to understand how this approach can be beneficial for potential users. We report on our design guidelines, developed around BeatriXR, as follows: first, we present the implementation of BeatriXR as a proof-of-concept system, that showcases how can feedforward be implemented into a virtual environment; then we have conducted a user study to assess whether the techniques implemented in the system are effective at a practical level.

8.1 Direct Feedforward implementation for VR environments

In this section, we will provide a summary of the VR feedforward concepts introduced by Muresan et al., as mentioned in section 4 the foundational design of BeatriXR is based on

their proposed framework. The user can interact with the feedforward by invoking it in various ways, depending on the chosen settings, which may be the default ones, or some options specifically changed either by the user or the decisional support of the system (see section 8.2.4).

8.1.1 Feedforward Guidelines for Virtual Reality

The framework outlined by Muresan et al. [MMH23] showcases the various ways in which feedforward can be implemented in a virtual environment. The authors categorise these possibilities into three distinct stages: Triggering, Previewing, and Exiting.

Triggering

In the Triggering stage the authors identified two elements: *trigger types* and *signifiers*. The *trigger types* can be of multiple types and may require an 'and/or' relationship between conditions. The trigger types outlined include the following:

- **Gaze Triggers**: Activated by a gaze of the user on an object or location, useful for signalling intent to interact.
- **Location Triggers**: Activated by user proximity or specific locations, showing potential actions when close to an object.
- **Action Triggers**: Activated after performing certain actions, often in step-based instructions, guiding users to the next step.
- **Event Triggers**: Activated by events or states in the virtual environment independent of direct user actions, often including temporal conditions.
- **Persistent Triggers**: Pre-existing feedforward that requires no additional activation.

Multi-triggers combine different types, allowing additional conditions that do not directly depend on the user (e.g. combining event and location conditions for a richer feedforward experience).

Signifiers instead are cues that help identify the presence of feedforward, and can make also the triggering of the feedforward explicit. These signifiers can be implemented as text, images or labels, with the possibility to show additional data, if needed.

Previewing

Once the feedforward has been triggered it will be shown to the user. In this section, we will follow the same approach taken by Muresan et al.: all the discussions that refer to actions (the interactions that the user needs to perform) can be also applied to outcomes (what the result of his action will be) as well, since the design spaces of these two elements overlap.

- **Level of detail**: This refers to the order, grouping, and granularity of the actions being previewed in a single instance of feedforward. The level of detail can vary depending on the complexity of the interaction being previewed. For example, a feedforward may show the complete process of starting a car, including pressing the ignition button, putting the car in gear, and steering the wheel. Alternatively, it may show only the key steps, such as pressing the ignition button.

- **Target**: This refers to what element is being previewed. This could be the action, the outcome or the avatar performing the action. Multiple targets can be previewed simultaneously.

- **Avatar Type**: The avatar type parameter specifies which avatar carries out the previews, with options including full avatar (the avatar of the user, in our case the digitally rendered, tracked hands of the user), partial avatar (a selection of the avatar), and real user (a humanoid representation of the user that performs the movements required for the action).

- **Duplication**: This parameter determines whether the feedforward uses copies of objects or the original objects to preview interactions. Copies can be ghosted or retain their original material.

- **Perspective**: This parameter determines the viewpoint from which the preview is shown. It can be either first-person or third-person. In a first-person perspective, the virtual hands of the user perform the actions and his position changes during the preview. In a third-person perspective, the user observes a ghosted copy of their avatar performing the actions.

- **Transform modifier**: The transform modifier parameter reflects where the target representation is positioned, whether in the original position or in a different (e.g. slightly dislocated) one.

- **Representation**: The preview can directly represent the actions and outcomes involved or use indirect representations such as visual or audio cues. For example, the iPhone unlock screen uses the text label “slide to unlock”, which is an indirect representation of the action and outcome.

- **Rendering**: This parameter determines the appearance of the previewed targets. Targets can be ghosted (highlighted, transparent, coloured, etc.), or rendered in their original form.

- **Playback**: This parameter determines how the preview is played. Previews can be played at different speeds, repeated, paused, or stopped.

Exiting

Following the presentation of the feedforward representation, various dismissal methods can be employed. To distinguish these mechanisms from those used to initiate feedforward display, the authors introduced the concept of *Untriggers*. Notably, gaze, action, and position exhibit analogous behaviour: gazing at a target can be converted into gazing away from it, approaching the target becomes moving away, and executing an action switches from invoking feedforward to dismissing it. Moreover, users have the option to terminate previewing through playback functionality.

Upon detection of an untrigger event, the feedforward presentation can be terminated. At this stage, the system can initiate an *exit transition* in one of three ways:

- **Environment Rewinding**: the animation is reversed and played back to its initial state prior to the start of the feedforward, restoring all elements to their pre-feedforward configuration.
- **Instantaneous Return**: the system directly transitions the environment to its initial state without animating this change.
- **Visual Effects** (like occlusion): employable to temporarily block or obscure the view of the user, effectively signifying the end of the feedforward presentation.

8.1.2 Example Scenario

To showcase the feedforward capabilities offered by BeatriXR in this section we will provide a scenario that shows the support that it gives during the execution of a VR experience. In this scenario, the trainee is going to learn how to handle dangerous materials in a chemistry setting. The steps to complete this task are the following:

- Operate the computer for initiating the procedure, by using the nearby mouse in the scene;
- Acquire a pair of gloves from the table;
- Insert a vial of hazardous waste in a beaker;
- Grab the hazardous beaker from the table;
- Position the beaker into the biohazard waste box;

The procedure utilizes various feedforward representations, and as the trainee lacks knowledge of the optimal choice, they unconditionally accept all system-suggested changes at each stage (refer to Section 8.2.4). The initial step sees the trainee positioned at the centre of the room, visually scanning for the mouse. Upon locating it, an animation initiates, featuring a transparent avatar moving away from the trainee towards the mouse. The avatar touches the mouse with its finger before vanishing. The trainee then

emulates this action, completing the step. Then, the user interface directs the trainee to acquire a pair of gloves. Recognizing the 'grab' gesture as the required interaction, the trainee approaches the gloves. Once in proximity, another animation commences, showcasing a pair of opaque hands executing a grasping motion while manipulating a ghosted pair of gloves distinct from the originals. The trainee is then instructed to insert a vial of hazardous material into a beaker. When prompted to adjust settings, the trainee discovers the necessity of manual animation activation. Upon activation through the user interface, the trainee is guided by a pair of ghosted hands within their perspective, demonstrating the gesture (in this case, a repositioning via raycast interaction). Returning to their original position with a fade in to fade out visual effect, the trainee moves towards the vial, aims it at the beaker using the ray, and inserts the vial accordingly. For the final two steps, the system settings are configured to display them concurrently. As the trainee approaches the beaker, a ghosted avatar grabs a ghosted version of the beaker and transfers it to the waste box. The trainee successfully completes the task by replicating these two actions (as also indicated by the user interface of the system).

In general BeatriXR shows, once invoked, a representation of what the user should do in the system in order to proceed in the steps of a task, using multiple sensory affordances that convey information to the user. For example, consider the design of showing how to move an object from a position to another. First of all, it is important to understand the type of the XR device the user is currently using, whether it supports controllers or hand tracking as a control setup. This support can be implemented at design time, by the developer. Once done that, the system needs to show how the functional affordance (the act of moving the object) can be explained using multiple sensory affordances, one for each possible gesture (e.g. a grab gesture or a raycast interaction). If there was no explanation, the physical affordance alone (the object representation) would not be sufficient for highlighting every single possible interaction (the raycast one, for example, would be harder to teach, with respect to the grab gesture that comes naturally). In this work, we aimed to provide a way for the users to understand how to perform correctly the interactions required by the system, for eventually let them complete successfully a task inside a virtual environment. Moreover, using an user interface that shows in an indirect fashion what to do, as also shown in chapter 7 offers multiple avenues for effectively instructing users in their tasks.

8.2 System Design

The design of this approach consists in three elements:

- The **User Interface** guides the user in the execution of the scene, containing information about the procedure steps, actions and possible interactions for each action, using indirect feedforward techniques. It also shows information on the progress and the next action he will perform on successfully completing the current one;

- The **Avatar Representations** are enabled on user request, and perform a demonstration of the interaction the user will have to do, using direct feedforward techniques;

- The **AI module**, an external component that can be queried either at run-time with its dedicated API (for suggesting on-the-fly changes through analysis of the context) or at design time (for suggesting alternative configurations to the procedure builder).

8.2.1 The User Interface

BeatriXR presents a series of panels that help the user understand how to proceed towards task completion. We can distinguish two areas:

- The **Main interface** allows the user to understand the progress inside the task execution;

- The **Feedforward Panels** let the user tweak the direct feedforward experience by modifying various related settings.

The Main Interface

The main interface is subdivided into three main panels: the central part shows what is the current action to do, and with which interaction can be performed. The side parts shows what the user successfully performed as the last action (on the left panel) and what the system expects the user to do for the next action (on the right panel) The central panel presents the current set of interactions that can be performed to reach the same result. The panel is presented as a stack of cards that show, for each one, instructions on what is the current action, what is the required interaction for performing it, which object is the target for the interaction, and a textual description of the interaction gesture. The user can choose to swap the topmost card with another one (if any) using the arrow buttons at the sides of the panel. When an interaction has been successfully performed the current interaction will be shown in the left panel, and the information about the next action will be brought to the middle panel, visually hinting a progress towards the completion of the whole task. The left panel shows a similar information with respect to the middle panel. Indeed, it presents the most recent successful interaction for the task. The amount of information is almost the same as the current panel, with the exception of the target object, which is omitted. The right panel shows what the user will have to perform once the current action has been completed. This panel shows a list of possible interactions under a single action, and this list is ordered using the algorithm presented in ViRgilites [ALRS24]: the context of the environment is taken in account for prioritising the list of interactions of this list and the suggested interaction for the next step, when the interactions are moved into the middle panel card stack.

The Feedforward Panels

When the user looks at his left hand palm, the system shows a menu with button that show a series of panels, that allows the user to customise the feedforward presentation and control it. The *Settings* button shows a panel that presents a series of controls that describe how the avatar appears, how it shows the information to the user, and how it leaves the scene. Each option represents the feedforward guidelines we have implemented (see section 8.4.1) with a user-friendly description of each category and their options (e.g. "Trigger Type" becomes "How do you want to enable the interaction preview?", "Gaze" becomes "Looking at the object", etc.). Some of the options cannot coexist, since they would contradict themselves (e.g. the type of the avatar cannot be full and partial at the same time), and to compensate that situation the interface has been programmed to deselect the options that are incompatible, when an option gets selected.

The *Feedforward Control* button shows to the user a series of media player-like options that allow the user to have control over the avatar animation. The user can indeed pause and resume the animation, rewind it or fast forward it, enable the loop repeat mode or stop it before its end. Moreover, the user can initiate the feedforward animation of the avatars using two other buttons: the single avatar representation (described in section 8.2.2) and the multiple avatars representation (described in section 8.2.2).

8.2.2 The Avatar Representations

BeatriXR allows the user, while interacting with the user interface, to get a visual representation of an interaction through the use of virtual avatars. These avatars demonstrate how to complete the current action showing the gesture only when needed. In BeatriXR there are two possible representations to show these interactions: a single avatar that shows the current selected interaction, and a panel that compares two possible interactions at the same time.

Single Avatar Representation

The single avatar representation, once summoned with the dedicated button of the user interface, begins its interaction by disembodying from the user itself, suggesting that the interaction must be eventually performed by the user. Depending on the interaction method the avatar may relocate itself to be at reach of the target object (e.g. if the interaction is "Far Interaction" then the avatar does not need to move, if the interaction is "Grab" instead, it will move near the object instead). The avatar will then perform the correct interaction gesture animation while pointing at the target object. The target object is copied, and the avatar shows the result of the interaction on the newly spawned representation. Depending on the transform modifier setting (see Section 8.1), the copied object can either appear in the same position as the original (overlapping it or replacing it, depending on the duplication setting) or be positioned at a set distance that differentiates it from the original (in our implementation, we empirically established a distance for

the copied object that was appropriate for distinguishing it from the original representation). Once the animation is completed, by default, both the avatar and the copy of the target object will disappear. It is important to underline the fact that the avatar does not have a priori knowledge of the environment, since it uses Inverse Kinematic [1] techniques to adapt the animation and point itself towards the target object. If the user decides to choose another interaction while the animation is running then the current animation will stop, so the correct interaction animation can start automatically on its place.

Multiple Avatars Representation

The multiple avatars representation of BeatriXR takes advantage of the previous one, inheriting most of the features previously described while having a different interface: the user, to enable this view, must select two different interactions (if applicable), and then he will have to press the "Compare" button to enable the animations. In this case the avatars will not disembody from the user, but two panels will show the avatars in their own space, performing their animations separately. The animation method is the same as the single avatar one, and the animation is seen through a shoulder-mounted virtual camera.

8.2.3 BeatriXR Capabilities

Both the settings and the avatar representations implement the design space of Muresan et al. [MMH23] and makes it possible to customise it on-the-fly. Figures 8.1, 8.2 and 8.3 show the various feedforward capabilities of BeatriXR.

Figure 8.1 shows the triggering capabilities for the system. For the *Trigger* category, we have implemented the following: *Gaze* (Panel A), activated when the user looks at the target object; *Position* (Panel B), activated when the user moves nearby the target object; *Timer* (Panel C) that periodically triggers the animation after a set amount of time; *Action* (Panel D), that in BeatriXR is triggered through a button press in the playback interface; *Persistent* (Panel E) that continuously starts the animation, also immediately after its end. For the *Signifiers* category instead (Panel F), that are hints that suggest the presence of feedforward for the target object, we have them implemented as an arrow that points to the target object of the representation.

Figure 8.2 instead shows the previewing capabilities for BeatriXR. *Level of Detail* (Panel A) describes how much complex the animation should be, either showing the full animation (first panel) or only parts of it (second panel, only the current step is visible); *Targets* (Panel B) define which parts of the preview should be shown: 1) Action (implemented as an arrow and a panel with the interaction modality attached to it), 2) Outcome, 3) Avatar; *Avatar Type* (Panel C) defines what kind of avatar, if visible, will be shown: full avatar (first panel), parts of avatar of the user (second panel), real user (third panel); *Perspective* (Panel D) defines the point of view of the user, either first person (first panel)

[1] https://en.wikipedia.org/wiki/Inverse_kinematics

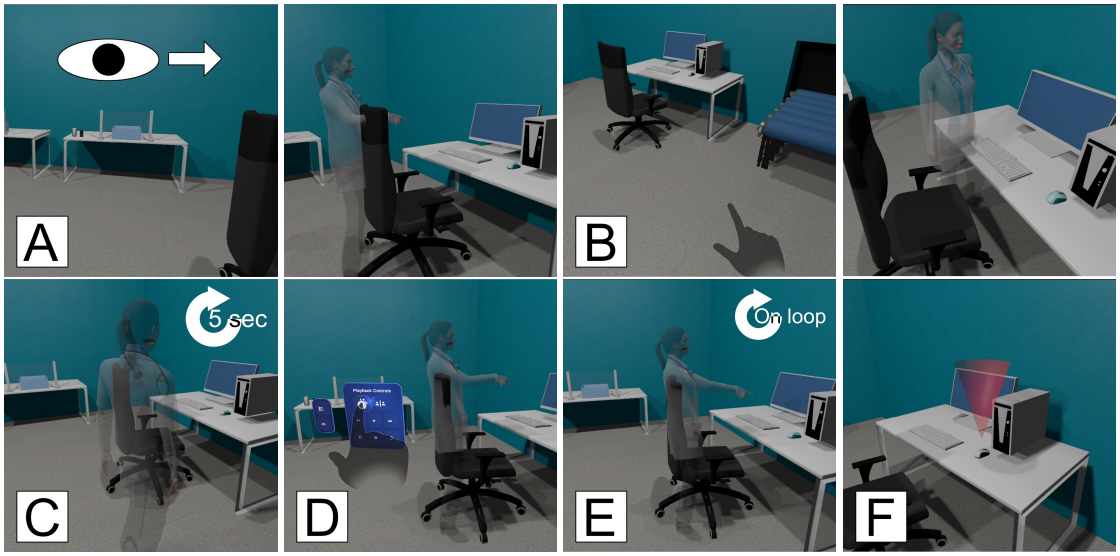


Figure 8.1: Triggering capabilities for BeatriXR. The supported settings are: Gaze (Panel A), Position (Panel B), Timer (Panel C), Action (Panel D), Persistent (Panel E). Signifiers (Panel F) show the presence of a representation for the target object.

or third person (second panel); *Transform Modifier* (Panel E) defines if the target object representation during the preview should be in a slightly offset position (left panel) or in the original one (right panel); *Rendering* (Panel F) defines if the avatar or the outcome, or both, should be shown as transparent (first panel) or fully opaque (right panel); *Duplication* (Panel G) defines if the avatar or the outcome, or both, should be shown as duplicated (first panel) or in a single instance (right panel, for the avatar side the user will not see his own).

Finally, Figure 8.3 shows the exiting capabilities for this work. For the *Untrigger* category we have implemented the following: *Gazing Away* from the target object (Panel A), *Moving Away* from the target object area (Panel B), *Automatically* as the animation naturally ends (Panel C). For the *Exit Animation* category instead we have: *Rewinding* the preview from the end to the start (Panel D), *Instantaneously returning* to the original state (Panel E) and a *Fade in to fade out effect* at the end of the preview (Panel F).

8.2.4 Artificial Intelligence Dynamic Settings Change

BeatriXR includes a decision support system that leverages the capabilities of Large Language Models (LLMs) to recommend override feedforward options created by the procedure developer. These pre-defined options are evaluated and assessed by the LLM to determine their suitability for presentation to the final user, who can then decide whether to apply them or not, using the in-environment user interface. The LLM-based system analyses the current action, the current feedforward settings, and the pre-made override options created by the procedure creator, taking into account the current context of

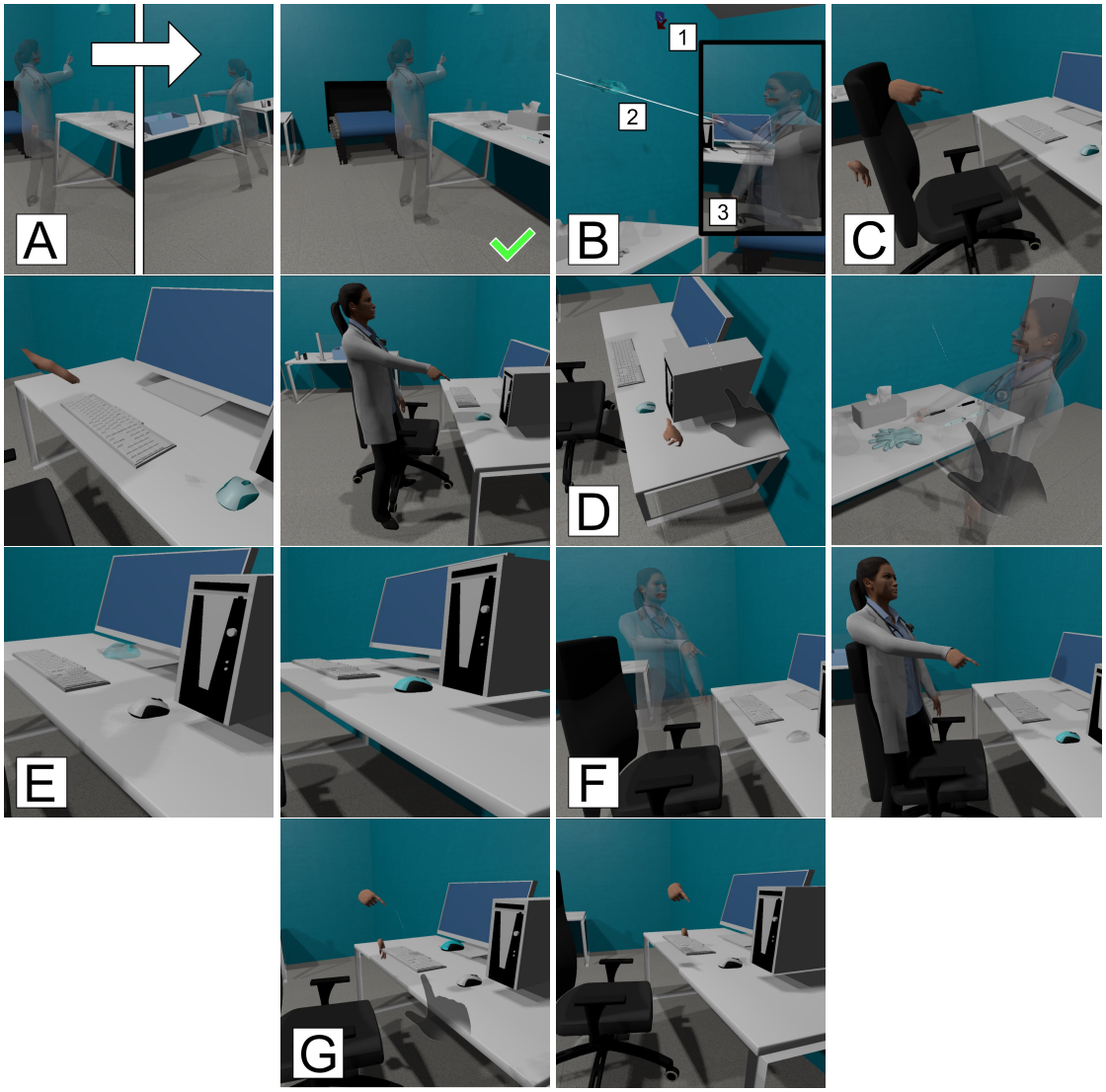


Figure 8.2: Previewing capabilities for BeatriXR. The supported settings are: Level of Detail (Panel A), visibility of the Targets (Panel B), Avatar Type (Panel C), Perspective (Panel D), Transform Modifiers (Panel E), Rendering (Panel F), Duplication (Panel G).

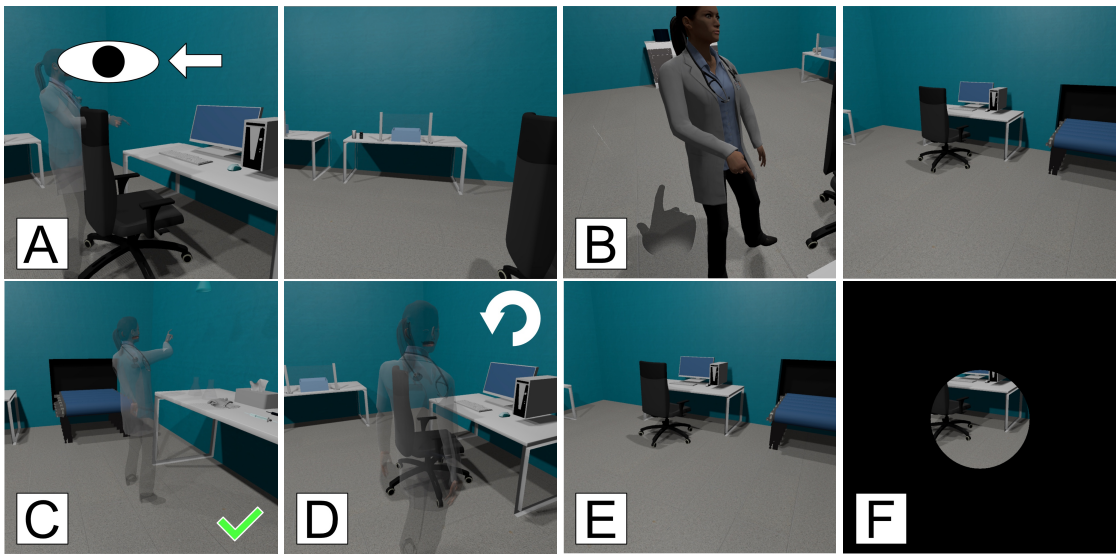


Figure 8.3: Exiting Capabilities for BeatriXR. Untriggering settings support: Gazing Away (Panel A), Moving Away (Panel B), Automatic dismissal (Panel C). Exit Animation settings support: Rewinding of the animation (Panel D), Instantaneous return to the original state (Panel E), Fade in/Fade out effect (Panel F).

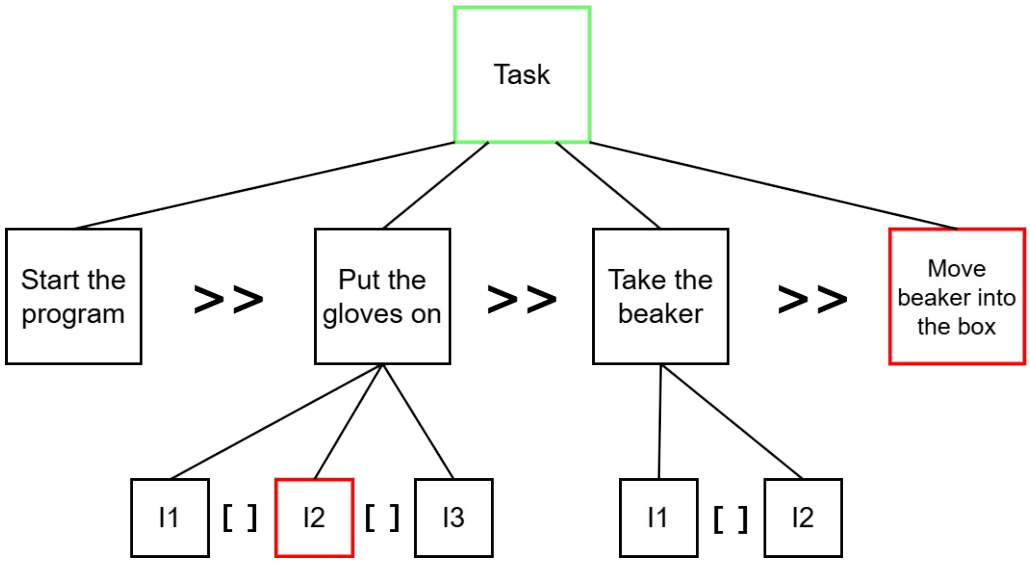


Figure 8.4: Following a simplified version of the "Chemical Lab" scenario in section 8.1.2, the system has, right from the start, default values for the feedforward (green box), if a particular step may require a change of settings, the procedure creator can also set some override options (red boxes) that can be suggested and enabled, if the AI algorithm evaluates it is a good idea to do so. The figure uses the notation from ConcurTaskTrees [PMM97] for showing the flow of the procedure, where a step enables the following one once the current one is completed ($\rangle\rangle$) and the interactions can be chosen by the user for deciding how to complete the step ([ ])

the user environment, including user preferences and user experience (see Figure 8.4). It is important to note that the override options presented to the user are applied over pre-defined defaults established by the procedure developer during the design phase or the user itself using the user interface, serving as a starting point for LLM-driven suggestions tailored to the current user context. The integration of the LLM in this system effectively eliminates the need for manual rule creation, as the intelligent model assumes responsibility for evaluating and suggesting override options based on its training data and contextual analysis.

Artificial Intelligence for the Procedure Creator

The LLM has been also implemented as an independent module, allowing it to function separately from the operational environment of the primary system. This design enables the LLM to be queried and utilized without initiating or depending on the execution of the main environment, and overall it offers a multi-layered assistance throughout the procedure creation and execution. When operating in this stand-alone mode, the LLM assumes the role of a specialized assistant to the domain expert, streamlining the creation and refinement workflow of tasks creation. the LLM is capable of receiving essential information, such as the default feedforward settings and the detailed sequence of action steps, and processing these inputs to generate targeted and personalised recommendations. Specifically, the LLM analyses these inputs to identify optimal or alternative configurations that may enhance the feedforward representation. These suggested adjustments, or “overrides”, are presented to the domain expert as options that can be adopted directly or further customized to meet specific operational requirements or preferences. By dynamically generating context-specific suggestions for feedforward switching, the LLM removes the need to establish rigid, hard-coded rules, allowing for a more adaptive and responsive approach based on the context where the end user will eventually take part in the task execution.

8.3 Implementation

This section discusses the architecture of BeatriXR. The system uses the Unity Game Engine and the MRTK 3 toolkit [Mic22b] for managing the environment and the user interactions.

Using MRTK enables BeatriXR to support a series of complex interactions in VR with minimal setup beforehand (namely: the attribution of MRTK-defined Components to the elements in the scene, so they can be interacted from the toolkit). We have validated this approach by creating a scenario that resembles a laboratory room. In this scenario the user can complete actions on interactive objects using either real-like (e.g. physical grab of an object) or virtual actions (e.g. far selection and manipulation of an object).

BeatriXR consists of 7 modules that are interconnected through the use of APIs to provide the proposed result (see Figure 8.5). This approach allows a developer to swap parts

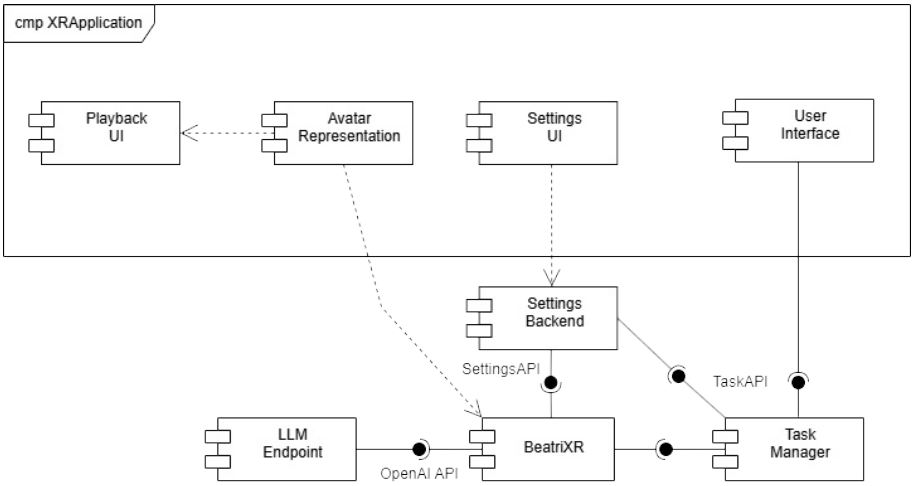


Figure 8.5: UML component diagram for BeatriXR

of the system with different implementations while achieving similar or better results, as long as the API calls are correctly satisfied.

Task Manager The *Task Manager* module contains the logic for managing and exposing the task actions information in the system. This is the module that keeps track of the progress of the user, and signals to the system the change of the action, after a successful execution of the previous one. In our proposed system, its implementation consists of a JSON-formatted text file that contains the name of the procedure and a list of actions, ordered by execution order. Each action contains its description and a list of interactions, which are alternative ways to achieve the same goal. Each interaction contains i) their respective textual description, ii) the modality required to perform the interaction, iii) a reference to the target object and who needs to perform the interaction. All this information is eventually exploited through its API by the *User Interface* module, that shows the information provided by it. The previous modules have been implemented by applying the solution proposed in chapter 7. Indeed, from ViRgilites we have inherited the structure of the system, the management of the tasks, the adaptation of the preferred task depending on the context and the base user interface that allows the user to select and perform the necessary interaction for completing the steps of the task.

BeatriXR The *BeatriXR* module allows the user and the procedure manager to tailor the default feedforward user experience by selecting the options inside the system in a separate set of interfaces, other than showing the result using the avatar included in the project, while being able to manipulate it with playback controls. In our implementation we have provided a series of modules that work in conjunction to deliver the proposed experience. These modules are: i) the *avatar representation*, ii) the *settings backend*, iii) the *settings UI*, iv) the *playback UI* and v) the *LLM endpoint*.

Avatar Representation The *Avatar Representation* uses a model taken from the Rocketbox library [2] and we have created a set of animations that represents each interaction type (Gaze and commit, Grab, Far Interaction, Touch, Repositioning) using the built-in Unity model animator. As mentioned in section 8.2.2 the animations are also supported by Inverse Kinematics techniques, so they can be personalised on the fly, without an additional developer intervention. Depending on the selected settings the avatar receives variants of its default mesh, in order to get displayed either in an opaque or transparent appearance, partially or fully.

Settings Backend and UI To accurately represent the intended animation, the *Settings Backend* module gathers essential data from the TaskAPI of the *Task Manager*, specifically the interaction type and the target object. Subsequently, the animation is initiated through Unity function calls, selecting animations that correspond to the specified interaction. Depending on the configuration settings available either from the start, or subsequently modified with the *Settings UI* module, the animation is then displayed in a manner tailored to these parameters. For the multiple avatar representation, in particular, the same procedure is done twice (one for each selected representation) but the avatars are not displayed directly into the scene. Indeed, they are hidden from the view of the user using Unity layers, and shown into two renderers that display two separate cameras that show those specific layers.

Playback UI The *Playback UI* allows the control of the animation while the avatar is shown and playing the animation. This is done by connecting UI elements to Unity functions that modify the animation speed of the Animator of the avatar.

LLM Endpoint The *LLM endpoint* is used as a gateway for the system to interface with an external chatbot. In our implementation we have used the OpenAI-Unity package [3] to achieve this goal. In particular, we have modified the endpoint and the API key for OpenAI API into a local server managed by LM Studio (version 0.3.5) [4]. The server has been instructed with a system prompt (See Appendix B) that described the task at hand and the possible options that the feedforward system could handle (also including the description of possible pros and cons for each decision, and a fictional weight that the chatbot would have taken in consideration for deciding the priority for each option). For each call to the server the *LLM endpoint* would also add the current settings for the avatar representation, the context of the environment (that included any possible issue that may require a change in the representation - namely, noisy environments, controllers unavailability, VR experience of the use, requested difficulty from the user, realism of the representation) and the set of proposed changes (as described in section 8.2.4. Once sent, the LLM, as

[2] https://github.com/microsoft/Microsoft-Rocketbox
[3] https://github.com/srcnalt/OpenAI-Unity/tree/master
[4] https://lmstudio.ai/

per instructions, only returned a single “Yes” or “No”, defining its final decision. A positive decision would trigger the *Settings Backend* for asking the user for settings changes, through the *Settings UI*. As part of the project, the chatbot can also be queried separately (using LM Studio directly) by the procedure creator, using a different system prompt, for letting the chatbot help the user provide good override choices, if necessary. In order to get consistent answers from the LLM model we have set the following options in the inference settings: Top p sampling (the minimum cumulative probability to the next token) to 0, Top K sampling (limitation of the next token to the most probable one) to 1, Temperature (the randomness of the answer) to 0. These settings are designed to ensure the LLM functions in a fully deterministic manner, which is essential for the reliability of its suggestions. By maintaining this configuration, the model consistently generates the same output for a given input, providing a stable and predictable result.

8.4 Validation

In this section we demonstrate the validity of the work, by comparing the toolkit against literature, and showing feasibility of using of LLMs to automatically suggest alternatives to procedure designers.

8.4.1 Validation of BeatriXR

In section 8.2.3 we have shown how BeatriXR reproduces the vast majority of the guidelines of Muresan et al. In table 8.1 we can see which elements we have adopted and which have been left out. While ViRgilites focused on indirect feedforward techniques and guidance specifically, BeatriXR improved its baseline by introducing the vast majority of the feedforward capabilities described by Muresan et al.

In the implementation of BeatriXR we have left out three elements: Real Device as avatar type and Audio and Haptic Modifiers. These options has been left out because the primary focus of the system is on visual interaction and feedforward. We did not test any audio features in BeatriXR because we centralised our study around visual representations. The Real Device option is redundant because it overlaps with the “Full Avatar” option, since the system has been designed around hand tracking. By extension, no haptic feedback is available through the exclusive use of hands, and the headset cannot vibrate.

8.4.2 Validation of LLM Answers

To validate the idea of using LLMs as feedforward creation assistants, we have invited 6 experts (4 male, 2 female, age range 25-41) in the field of Extended Reality to provide an opinion on the quality of the answers of a set of LLM models.

Table 8.1: Comparison between the guidelines of Muresan et al. [MMH23], ViRgilites (in Chapter 7) and BeatriXR. Notes: a) The feature is possible because we implemented a similar interface to ViRgilites; b) the playback of the feedforward can be interrupted, but without an explicit button, switching to another interaction starts a new one while interrupting the old one.

		Muresan et al.	ViRgilites	BeatriXR
Triggering	**Trigger Type**	**Persistent Triggers**		✓
		Location		✓
		Events		✓
		Gaze		✓
		Action		✓
	Signifiers	**Signifiers**		✓
Previewing	**Level of Detail**	**All Actions**		✓
		A subset of actions	✓	✓
	Targets	**Avatar**		✓
		Actions	✓	✓
		Outcomes		✓
	Avatar Type	**Full Avatar**		✓
		Partial Avatar		✓
		Real Device		
		Real User		✓
		None	✓	✓[a]
	Perspective	**Third-Person**		✓
		First-Person	✓	✓[a]
	Modifier	**Audio**		
		Haptic		
		Transform		✓
		None	✓	✓[a]
	Rendering	**Ghosted**		✓
		Original	✓	✓[a]
	Representation	**Direct**		✓
		Indirect	✓	✓[a]
	Duplication	**Yes**		✓
		No	✓	✓[a]
	Playback	**Interruption**		✓[b]
		Repetition		✓
		Stop		✓
		Pause		✓
		Rewind		✓
		Speed-up		✓
Exiting	**Untrigger Type**	**Gaze**		✓
		Action		✓
		Playback		✓
		Location		✓
	Exit Transition	**Return**		✓
		Rewind		✓
		Visual Effects		✓

LMM model selection

The quality of the suggestions provided by the AI component obviously depends on the quality of the underlying LLM. Therefore, before creating the expert validation test, we needed to identify representative models to generate the suggestions. The candidates are the following 11 models from Hugging Face, from different companies, parameters size and quantisation (all the models have been fine-tuned with an "Instruct" dataset[5]):

- Google Gemma 2
  - 2B parameters
    - IQ3_M Quantisation
    - Q8_0 Quantisation
  - 9B parameters
    - Q3_K_L Quantisation
    - Q4_K_M Quantisation
    - Q8 Quantisation
- Meta LLaMa 3.2
  - 1B parameters
    - Q8_0 Quantisation
  - 3B Parameters
    - Q8_0 Quantisation
- Microsoft Phi 3.1 Mini
  - 128k parameters
    - IQ3_M Quantisation
    - Q4_K_M Quantisation
    - Q6_K Quantisation
- Mistral Small
  - 22B parameters
    - Q4_K_M Quantisation

All the models have been run locally using LM Studio 0.3.5, with the following settings for the model environment: i) temperature set to 0, ii) top-k sampling set to 1, iii) top-p sampling set to 0. These settings have been chosen for avoiding randomness in the variety of the answers and creating determinism in the tests[6]. In the next section we describe

[5] Reinforcement Learning from Human Feedback (RLHF) - Wikipedia, last accessed: 20/01/2025

[6] IBM explanation on Temperature, Top-P Sampling and Top-K Sampling, last accessed: 30/12/2024

the selection progress through a technical evaluation on the set. The best performing models will be used as part of the expert evaluation (section 8.4.2).

Test Environment All the LLMs have been prepared to accept the following system prompt (in this section we report the important parts, see Appendix B for the complete text) and user prompt, and queried them twice for confirming the similarity in the answer:

System Prompt

You are tasked with suggesting zero or more alternative new options for an animation representation in a VR environment. You should only analyze these options as a controller for the animation, as the user is not influenced by the change of any of these options, the user will only see the animation set up with the settings in this prompt.

Instructions:

1. **Read** the descriptions of the environment settings below.

2. **Note** the weight of each item in parentheses (1 = negligible importance, 5 = extremely important).

3. **Evaluate** the significance of each setting based on its weight and description.

4. **Read** the prompt of the user, which will contain a **description of the task** and the **set of default options** for the representation.

5. **Decide** on possible new settings that could be better - you can also suggest no changes for any or all options, if you evaluate it is better to do so. Please give your decision in the same line as the option, so it's clear which option you are referring to.

6. **DO NOT** talk about values outside this list, since it's not necessary and out of scope for your task.

—

Assumptions

- When talking about the "Target" the system is referring to the animation representation of the real target object. So for example, if the target is transparent then it will be transparent over the real representation of the object, if the settings say that the target is duplicated. If the target is not duplicated then the animation representation will act as the real object (even though the real object is just hidden)

- You are helping a developer in setting options for modify an animation that will show how the user have to perform its interaction (like a tutorial). The action you are going to analyze has nothing to do with the action the user has to perform to complete the steps, it is a separate function. So for example, "Trigger Mode" to "Button Triggered" means that the user pushes the

“Activate Animation” button to play the animation, that helps the user learn the action he needs to do. None of these settings affect how the user acts to accomplish the action, only the appearance and behaviour of the animation itself.

- **The developer has no control over implementation of features different than the ones described here**, the only choice he has is to set different options for the animation representation.

LLM Model Analysis Request

Please analyze all the options in the system prompt and return the analysis for each one. The analysis **MUST** include all of the following options:

1. **Trigger Mode**
2. **Signifiers**
3. **Level of Detail**
4. **Visible Elements**
5. **Avatar Type**
6. **Perspective**
7. **Target Position**
8. **Avatar Rendering**
9. **Target Rendering**
10. **Avatar Duplication**
11. **Target Duplication**
12. **Dismissal Mode**
13. **Exit Transition**

Available Settings

...

User Prompt

The task you’ll evaluate is “Start a program” with the interaction “Click the mouse” as action and “Touch” as interaction

The settings are:

Trigger Mode: Button-triggered

Signifiers: Disabled

Level of Detail: Full-Animation

Visible Elements : Target Object, Avatar

Avatar Type: Puppet Representing the User

Perspective: Third-person Perspective

Target Position: Correct Position

Avatar Rendering: Transparent

Target Rendering: Transparent

Avatar Duplication: Duplicated

Target Duplication : Duplicated

Dismissal Mode: Automatically

Exit Transition: Instantaneous

The resulting answers from each LLM have been analysed following three criteria:

- *Answer Completeness*: Whether the LLM model returns an answer on each parameter;
- *Answer Quality*: How the suggestions together may match in an actual implementation;
- *Answer Plausibility*: taken the LLM answer as valid, how coherent is the justification of the answer.

If the model returned no answer on one or more parameters, then the model was considered excluded from subsequent scoring calculations. The scoring of the remaining models was determined by averaging the scores of *Answer Quality* (rated on a scale from 1 to 5) and *Answer Plausibility* (also rated on a scale from 1 to 5) across two answers for each queried model. Table 8.2 shows the detailed scores for each model, averaged from the two answers of each model. In case of same total value, the quantisation has been used as discriminant for selecting the final choice (see Table 8.2, Google Gemma 2 9B parameters models). Ultimately we have selected 4 final models (Google Gemma 2 2B Q8_0, Google Gemma 2 9B Q8, Meta LLaMa 3.2 3B Q8_0, Mistral Small 22B Q4_K_M) for the expert evaluation phase.

Testing Phase

The validation phase of the LLM models has been made unsupervised, with the possibility for the expert to contact us in case of doubts. The validation, after collecting the consent of the experts and their demographics, was structured in three parts.

Tutorial: The experts were introduced to the topic of feedforward, with a pre-recorded video that showcases the various capabilities, as described in section 8.1.1. The video also denoted the task the experts were going to do.

Table 8.2: Summary table of the models preselection phase. In case of same total value, higher quantisation was used as tie-breaker.

Model	Parameters	Quantisation	Answer Completeness	Answer Quality (avg.)	Answer Plausibility (avg.)	Total (avg. out of 10)	Selected?
Google Gemma 2	2B	IQ3_M	✗	-	-	-	
		Q8_0	✓	5	3.5	8.5	✓
	9B	Q3_K_L	✓	4	4	8	
		Q4_K_M	✓	4	3.5	7.5	
		Q8	✓	4	4	8	✓
Meta LLaMa 3.2	1B	Q8_0	✗	-	-	-	
	3B	Q8_0	✓	5	4	9	✓
Microsoft Phi 3.1 Mini	128k	IQ3_M	✗	-	-	-	
		Q4_K_M	✓	4.5	2.5	7	
		Q6_K	✓	3	4	7	
Mistral Small	22B	Q4_K_M	✓	4	5	9	✓

User Opinion: In this section, the experts were asked to provide an opinion on how they would configure an hypothetical scenario on the following feedforward representation tasks:

- Starting the program on a computer, using a touch interaction for clicking the virtual mouse in the VR scene;
- Grabbing a beaker, using a laser-pointing technique, i.e., a ray starting from the hand or the VR controller allowing pointing from afar;
- Turning on a sink, using a grab technique, i.e., the user joins the index and the thumb fingers in proximity of an object to start manipulating it.

LLM Evaluation: In this section the experts read a set of 4 LLM answers (in randomised order to prevent recognition of a particular model), where they tried to provide a possible better alternative to the default set of feedforward options. The experts have then been asked to provide a rating to the LLM answer in a 1-7 Likert Scale (1 was labelled "Very Bad", 7 was labelled "Very Good") and also provide a comment on why they provided a particular rating. The latter two sections were repeated 3 times, one for each task.

Results

LLM performance Across the three tasks, the question posed to the experts was: "How much would you rate [the LLM] answer?". This question aimed to gather information on the quality of the answers of the LLMs, with a focus on their potential utility as an AI assistant capable of proposing better alternatives to the default values. Figure 8.6 shows the results of the ratings. Overall, LLaMa model performed better overall (Avg.: 4.94, Google Gemma 2 2B: 4, Google Gemma 2 9B: 4.06, Mistral Small: 3.89) with better scores on two tasks out of three (T1, T3). This score shows that using this model would generally return better answers than the others. It is also necessary to report the average

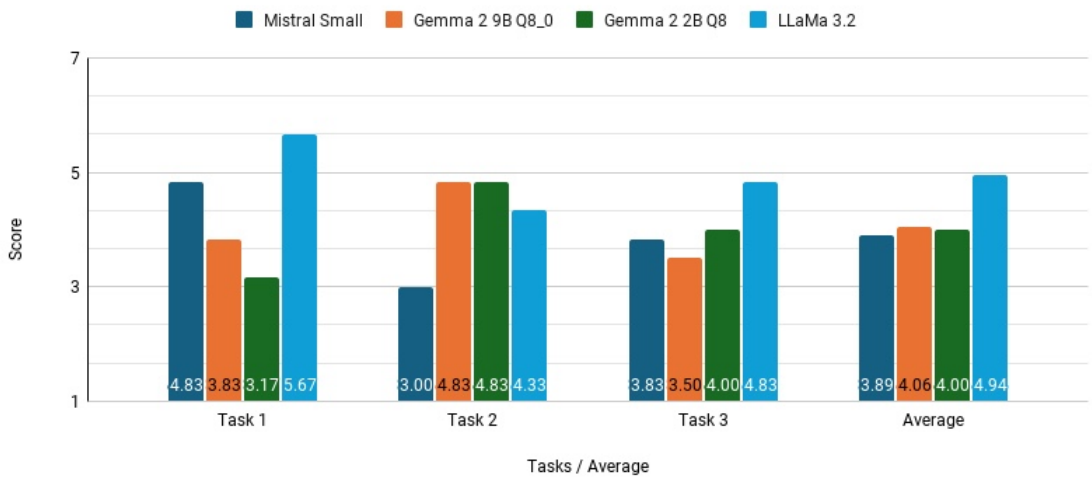


Figure 8.6: Average scores for the LLM models, after the expert validation phase.

performance of this model on T2, where Gemma 2 models performed better (LLaMa Avg.: 4.33, Gemma 2 9B Avg.: 4.83, Gemma 2 2B Avg.: 4.83). Despite this, we can report that LLaMa model never averaged below 4, which suggests good overall reliability on its use as assistant. This observation can also be confirmed when all the data is taken in account (see Table 8.3): LLaMa 3.2 stands out overall, with consistently high averages (T1: 4.33, T2: 4.33, T3: 4.83), medians (T1: 6, T2: 6, T3: 6) and modes (T1: 6, T2: 6, T3: 6), and low variability (with the exception of T2 which scored 2.07- T1: 0.52, T3: 1.6).

Comparing Expert Decisions and LLM Recommendations in Feedforward Setup

During the three tasks the experts provided an opinion on how they would configure the tasks described in section 8.4.2, and in this section we will show data on how the LLM models were agreeing with the experts about their proposed settings. For calculating the similarity of the answers we will use Fleiss' kappa [FC73]. In Table 8.4 we report the results of the comparisons. As a baseline we have also reported the Fleiss' kappa value for each task using only the opinions of the experts. LLMs values are calculated by adding their option to the original expert opinions, and recalculating the new kappa value.

Fleiss' kappa values range between 0.23 and 0.34 across tasks and groups, indicating fair agreement overall. Task 1 shows the highest kappa values (ranging from 0.31 to 0.34), suggesting slightly better inter-rater agreement for this task compared to the others. Task 3 has the lowest kappa values (0.23 to 0.28), indicating lower agreement across groups for this task. The Experts Only group generally shows values comparable to the models, with no significant advantage in agreement over the automated models. Of all the models, Gemma 2 9B consistently achieves the highest or among the highest kappa scores, suggesting better alignment with inter-rater consensus across tasks.

Table 8.3: Detailed scores for the LLM models validation for each task

		Average	Median	STD	Mode	Min	Max
Task 1	**Mistral Small**	4.83	5	1.17	5	3	6
	Gemma 2 9B Q8_0	3.83	3.5	1.47	3	2	6
	Gemma 2 2B Q8	3.17	3	2.23	3	1	7
	LLaMa 3.2	5.67	6	0.52	6	5	6
Task 2	**Mistral Small**	3	2.5	1.26	2	2	5
	Gemma 2 9B Q8_0	4.83	5	1.47	5	2	6
	Gemma 2 2B Q8	4.83	5	1.17	5	3	6
	LLaMa 3.2	4.33	6	2.07	6	1	6
Task 3	**Mistral Small**	3.83	3.5	1.94	2	2	7
	Gemma 2 9B Q8_0	3.5	3	1.76	2	2	6
	Gemma 2 2B Q8	4	4	1.1	4	2	5
	LLaMa 3.2	4.83	6	1.6	6	1	6

Table 8.4: Similarity scores of experts opinions with and without LLMs using Fleiss' kappa

	Experts Only	**LLaMa 3.2**	**Mistral Small**	**Gemma 2 2B**	**Gemma 2 9B**
Task 1	0.31	0.33	0.32	0.33	0.34
Task 2	0.28	0.27	0.27	0.27	0.31
Task 3	0.23	0.24	0.28	0.26	0.26

Discussion on Experts Feedback Along with the rating scores and the experts opinions on the setup of the various tasks, the experts gave some insights on the various models. All of the models provided suggestions for improving the design, either by agreeing with the default values or suggesting one or more changes. However, all the experts noted a general lack of detailed explanations, especially when deviating from the initial settings. This behaviour made difficult for them to evaluate the quality of the answers. The experts were also reporting a general inconsistency across all the answers. For example, Gemma 2 2B suggested duplication to ensure user understanding but then opted for transparent items (which could give them less importance) and one expert on Task 3, while reviewing the answer from Mistral Small felt the answers provided by the LLM were random. Moreover, most of the models shared a set of patterns that got picked up by the experts: three models out of four (Gemma 2 2B, Mistral Small and LLaMa) were systematically omitting the use of signifiers, which were seen by the experts as important, and the omission of them was not well received; two models (Gemma 2 9B and Mistral Small) were using gaze as the only untriggering method, and it was seen as unreliable and potentially frustrating due to the risk of unintentional triggering; three models out of four (Gemma 2 9B, Mistral Small and LLaMa) suggested the use of rewind and fade exit transitions, which experts deemed unnecessary, time-consuming and potentially disorienting in VR environments; two models out of four (LLaMa and Gemma 2 2B) presented solutions that promoted visual clutter in some capacity.

The LLaMa model showed promising potential in supporting experts, especially by offering reasonable options. While some inconsistency remained as general issue, the model proved useful even when suggesting features different from the default values. Reviewers also appreciated the effort the LLM made to explain its answers when explanations were provided.

In addition to the previous general comments, Gemma 2 2B model after the evaluation came out as too conservative, and mostly justifying the default choices, rather than improving the design. Moreover, when the model tried to propose new settings values, the proposed alternatives were seen as weak, or not matching with the other options. One reviewer noted that the model seems to blindly follow the starting settings, leading them to doubt the reasoning skills of the model itself.

Gemma 2 9B, despite scoring high Fleiss' kappa values in the previous section, received mixed opinions instead, where some expert answers expressed coherence and well-arguing of the justifications, where in other answers the experts noted a tendency of ever changing the default answer to something else, leading to concerns on its reliability on this model as well.

Mistral Small scored mixed opinions as Gemma 2 9B, where the comments split between the model being good in its suggestions, and other comments highlighting its questionable choices, accompanied with poor explanations. In particular, several experts noted that Mistral Small tended to find pros for the choices provided as prompt, without considering downsides, even if the settings were not optimal.

Post-Analysis Insights The validation of BeatriXR provided valuable insights into how the system integrates the feedforward model presented in Section 8.2.3 with LLMs to support inexperienced procedure designers in exploring alternative design options. The selection phase of the test highlighted that, for models with the same number of parameters, those with better quantisation produced higher-quality results, while for models with the same quantisation, those with a higher number of parameters performed better. Our results indicate that LLMs can generate generally agreeable suggestions that align with expert choices, demonstrating their feasibility for this task. Finally, among the models tested, Gemma 2 9B emerged as the statistically superior option for providing automated suggestions, while LLaMa 3.2 received favourable qualitative feedback. These findings suggest that either model could contribute to an efficient real-world implementation of BeatriXR.

However, our evaluation has an inherent limitation: the results are based on a predefined set of models available at the time of testing. As more advanced models become available, they may yield improved performance and superior results compared to those reported here. We position BeatriXR as an initial step in this research area, providing a foundation for future advancements in the integration of LLMs for procedural design exploration.

Part IV

Conclusions and Future Works

Chapter 9

Conclusions

This thesis explored and introduced a set of approaches to empower users in the process of creating XR content, through novel systems and methodologies. This work began from an investigation of the state of the art, moving towards the development of ECARules4All, XRSpotlight, ViRgilites and BeatriXR, to finally arrive here, where we will draw conclusions and and relate our findings back to the research questions of this thesis.

The first research question was about making XR development available to those who lack programming skills, assessing whether End-User Development (EUD) methodologies and tools can bridge gap between the required skills and those available to non-programmers to design and construct fully functional virtual environments (RQ1). This question presents an important objective, as the complexity of traditional XR development often presents a significant barrier to entry for non-programmers. To address this, in chapter 5 we introduced ECARules4All as an EUD approach for fostering configurations of VR environments by end users. Drawing inspiration from web builders and content management systems, this approach leverages natural language rules and utilizes templates derived from general-purpose environments crafted by expert developers. Additionally, it incorporates a pre-established and extensible taxonomy of virtual objects. These elements collectively enable users lacking programming skills to design personalized VR environments, reducing the need of traditional programming and proving to be a viable solution to RQ1. During the development of ECARules4All we have understood that the rule-based description of the XR environment's dynamic behaviour was effective tonon-programmers, and we leveraged this knowledge for the next research, describing in a simple way the interactions that are usually hidden during the definition of an environment.

The second question was about exploring whether EUD techniques are effective in describing existing XR environments to further groups of users, i.e., to novice developers, for helping them in learning how to create their own XR projects (RQ2). This question is important for the literature because, analysing how novices approach their work, it highlighted the importance to provide effective explanations of existing examples that they can easily understand and adapt to their needs. For that, XRSpotlight (chapter 6) achieves this by facilitating novice developers in discovering, navigating, and compre-

hending pertinent interaction examples tailored to their development tasks. Indeed, they can integrate these examples into their XR projects, facilitating a smoother transition into more complex XR environments, using an user interface that allows them to read, find and copy-paste behaviours (written in natural language) from one scene or project to another. The foundational modelling and vocabulary are designed to support these tasks in a toolkit-agnostic way, thereby streamlining the planning phase for interaction creation. An understanding of the functionality and constraints of copy-and-paste mechanisms lays the groundwork for potential enhancements and future developments. XRSpotlight directly tackles RQ2 by using its user interface to transparently show XR interactions, and making them readily accessible to novice developers through a simplified rule-based representation. Moreover, building XRSpotlight made us aware that clear and concise representations of interactions are crucial for facilitating learning and comprehension. The absence of familiar affordances for users of XR environments made us work towards the use of EUD for allowing domain experts to build guidance systems that work for their procedures.

The final question asks whether if domain experts can configure existing XR environments to offer proper guidance to novices learning a new task (RQ3). This question stems from the understanding that it is important to find a middle ground between the expert developers, which have programming skills but no expertise in a given field, and end users, which need to learn a new skill through XR environments, and empowering him with In chapter 7, we introduced ViRgilites, a system designed to help users navigate and interact in a Virtual Reality (VR) setting. Discovering and using interactions in VR applications is often cumbersome, because of the wide variety of possible interactions that have the same outcome in combination with a diversity of implementations of interaction techniques. ViRgilites offers a meta-user interface that supports users to *discover, select and perform* various multimodal interactions. In addition, ViRgilites helps users understand what tasks they can complete, shows how to do them, and presents the user what comes next. It automatically suggests the best way to interact based on the context and designer preferences. Finally, we implemented a flexible prioritisation approach that recommends, steers or even constrains users to use the most effective interaction modalities to perform an interaction. The definition of the task procedure is made by allowing the domain expert to define the modalities and the procedures using a specific file. The same file also allows the domain expert to automatically define the guidance elements that will be shown in the user interface, showing the potential for satisfying RQ3. We tested ViRgilites in a preliminary study that looked at two real-world scenarios, mainly focusing on user opinions about the design and layout of this new interface. The feedback we gathered demonstrated the usage of ViRgilites for goal-driven VR applications, and this enabled us to further improve the design and behaviour of ViRgilites.

Indeed, in chapter 8 we have presented BeatriXR, a toolkit for integrating feedforward information in any VR environment. Leveraging the results from ViRgilites and expanding them with the guidelines in literature and the latest Artificial Intelligence technologies, this allowed us to create a system able to offer better VR experiences, guiding the trainee in the learning of a new task, with both direct and indirect feedforward techniques. Start-

ing from ViRgilites as a base, BeatriXR offers a framework that is able to understand the virtual environment and create personalised and customisable animations for the objects in the scene, while also providing a set of guidelines for artificial intelligence models for suggesting alternative options to the procedure creator. BeatriXR retains the ability from ViRgilites for the domain expert to create new tasks without additional help from a developer. BeatriXR adds a new element to RQ3, where in addition to the automatic suggestion and modification through Artificial Intelligence mechanisms, the appearance of the feedforward animations can be also customised manually during the execution of the experience in order to tailor it to the needs of the user, without extra modifications to the environment to adapt them every time. We validated BeatriXR against the guidelines provided by Muresan et al., while also evaluated the efficacy of Large Language Models in this context with the knowledge of experts in the field of Extended Reality. The results from the evaluation showed that integrating AI and customizable, multi-level feedforward techniques into VR training environments can enhance user guidance and improve the effectiveness of VR training, despite already presenting room for improvement from AI recognition.

9.1 Future Works

The state of the art let us understand the shortcomings and the challenges that future research needed to tackle, and that eventually led us to the realisation of our works. For future works, we still need to survey the future literature to ensure that the next works will be on the backed up by the latest findings in the field of XR development.

Future work in End-User Development for Extended Reality could focus on defining effective interfaces supporting End-User Developers in creating behaviours, exploiting conversational interaction through the use of Artificial Intelligence and more usable graphical representations. In addition, it could better assess the rule-language definition through further studies that apply our methodologies in different environments and test variables.

In the future, a system like XRSpotlight could be created for describing more complex scenes, which contain interactions beyond examples. Since their definition focuses more on the resulting experience than on writing clean code for replication, future work could find more complex structures. An interesting path could be to shift the focus towards educational purposes, i.e., supporting people in learning how to create XR environments by gaining a simplified yet correct understanding of how XR interaction works. To this end, it could be interesting to include a non-computer science audience.

ViRgilites proved to be a good starting point to build upon, and spawn additional ideas as BeatriXR. Future work in this field could envision more studies on Artificial Intelligence integration (e.g. a virtual assistant that can further explain particular steps), or implement complex scenarios that allow the creation of different valid paths that allow a deeper exploration of a given task. Moreover, further work may encompass the implementation of XR tasks that, for example, may integrate virtual guidance in the real world, following

the trainee in the step of learning the task in a real environment.

End-User Development is a concept that is able to fit into various fields, as also seen in literature. XR enhances it by giving power to the users to shape virtual worlds with ease. Future ideas may investigate how to personalise live environments, where people could entertain social interactions in both public and private virtual spaces. Moreover, these customisations could span from simple personalisation to the creation of whole new experiences, allowing anyone to expand these worlds with their ideas, and be able to express themselves in new ways, that could not be possible in the real world.

Bibliography

[AA24] Gabriel Alex and Hassan Alaa. Design and development of an authoring tool for xr-based engineering training. In *2024 IEEE International Conference on Engineering, Technology, and Innovation (ICE/ITMC)*, pages 1–7, 2024.

[AB22] Gleechi AB. Virtualgrasp documentation, 2022.

[ABC+15] Carmelo Ardito, Paolo Buono, Maria Francesca Costabile, Rosa Lanzilotti, Antonio Piccinno, and Li Zhu. On the transferability of a meta-design model supporting end-user development. *Universal Access in the Information Society*, 14(2):169–186, Jun 2015.

[ABFS23] Valentino Artizzu, Ivan Blečić, Vittoria Frau, and Lucio Davide Spano. How end users develop point-and-click games. In Lucio Davide Spano, Albrecht Schmidt, Carmen Santoro, and Simone Stumpf, editors, *End-User Development*, pages 220–229, Cham, 2023. Springer Nature Switzerland.

[ABM+20] Narges Ashtari, Andrea Bunt, Joanna McGrenere, Michael Nebeling, and Parmit K. Chilana. Creating augmented and virtual reality applications: Current practices, challenges, and opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery.

[ACF+22] Valentino Artizzu, Gianmarco Cherchi, Davide Fara, Vittoria Frau, Riccardo Macis, Luca Pitzalis, Alessandro Tola, Ivan Blecic, and Lucio Davide Spano. Defining configurable virtual reality templates for end users. *Proc. ACM Hum.-Comput. Interact.*, 6(EICS), June 2022.

[ALRRS24] Valentino Artizzu, Kris Luyten, Gustavo Alberto Rovelo Ruiz, and Lucio Davide Spano. Direct feedforward techniques for the virgilites system. In *Companion Proceedings of the 16th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '24 Companion, page 87–88, New York, NY, USA, 2024. Association for Computing Machinery.

[ALRS24] Valentino Artizzu, Kris Luyten, Gustavo Rovelo Ruiz, and Lucio Davide Spano. Virgilites: Multilevel feedforward for multimodal interaction in vr. *Proc. ACM Hum.-Comput. Interact.*, 8(EICS), June 2024.

[AMPS22] Raffaele Ariano, Marco Manca, Fabio Paternò, and Carmen Santoro. Smartphone-based augmented reality for end-user creation of home automations. *Behaviour & Information Technology*, 0(0):1–17, 2022.

[AMZO+24] Setareh Aghel Manesh, Tianyi Zhang, Yuki Onishi, Kotaro Hara, Scott Bateman, Jiannan Li, and Anthony Tang. How people prompt generative ai to create interactive vr scenes. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference*, DIS '24, page 2319–2340, New York, NY, USA, 2024. Association for Computing Machinery.

[APF+18] Telmo Adão, Luís Pádua, Miguel Fonseca, Luís Agrellos, Joaquim J. Sousa, Luís Magalhães, and Emanuel Peres. A rapid prototyping tool to produce 360° video-based immersive experiences enhanced with virtual/multimedia elements. *Procedia Computer Science*, 138:441–453, 2018. CENTERIS 2018 - International Conference on ENTERprise Information Systems / ProjMAN 2018 - International Conference on Project MANagement / HCist 2018 - International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN/HCist 2018.

[Art24] Valentino Artizzu. End user development for extended reality. In Michael Harrison, Célia Martinie, Nicholas Micallef, Philippe Palanque, Albrecht Schmidt, Marco Winckler, Enes Yigitbas, and Luciana Zaina, editors, *Engineering Interactive Computer Systems. EICS 2023 International Workshops and Doctoral Consortium*, pages 151–165, Cham, 2024. Springer Nature Switzerland.

[BAFS23] Nikita Bellini, Valentino Artizzu, Vittoria Frau, and Lucio Davide Spano. Poster: Programming rules by demonstration in virtual reality. In *Proceedings of the 15th Biannual Conference of the Italian SIGCHI Chapter*, CHItaly '23, New York, NY, USA, 2023. Association for Computing Machinery.

[BCF+21] Ivan Blečić, Sara Cuccu, Filippo Andrea Fanni, Vittoria Frau, Riccardo Macis, Valeria Saiu, Martina Senis, Lucio Davide Spano, and Alessandro Tola. First-person cinematographic videogames: Game model, authoring environment, and potential for creating affection for places. *J. Comput. Cult. Herit.*, 14(2), may 2021.

[BD23] Philip Brookins and Jason Matthew DeBacker. Playing Games With GPT: What Can We Learn About a Large Language Model From Canonical Strategic Games? *SSRN Electronic Journal*, 2023.

[BDR+19] Will Brackenbury, Abhimanyu Deora, Jillian Ritchey, Jason Vallee, Weijia He, Guan Wang, Michael L. Littman, and Blase Ur. How users interpret bugs in trigger-action programming. In *Proceedings of the 2019 CHI Conference*

on Human Factors in Computing Systems, CHI '19, page 1–12, New York, NY, USA, 2019. Association for Computing Machinery.

[BDWK10] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. Example-centric programming: Integrating web search into the development environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 513–522, New York, NY, USA, 2010. Association for Computing Machinery.

[BEDF15] Jeremy Boy, Louis Eveillard, Françoise Detienne, and Jean-Daniel Fekete. Suggested interactivity: Seeking perceived affordances for information visualization. *IEEE transactions on visualization and computer graphics*, 22(1):639–648, 2015.

[BFF+18] Soumya C. Barathi, Daniel J. Finnegan, Matthew Farrow, Alexander Whaley, Pippa Heath, Jude Buckley, Peter W. Dowrick, Burkhard C. Wuensche, James L. J. Bilzon, Eamonn O'Neill, and Christof Lutteroth. Interactive feedforward for improving performance and maintaining intrinsic motivation in vr exergaming. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–14, New York, NY, USA, 2018. Association for Computing Machinery.

[Bil13] Mark Billinghurst. Hands and speech in space: Multimodal interaction with augmented reality interfaces. In *Proceedings of the 15th ACM on International Conference on Multimodal Interaction*, ICMI '13, page 379–380, New York, NY, USA, 2013. Association for Computing Machinery.

[BJSM17] SK Bailey, Cheryl I Johnson, Bradford L Schroeder, and Matthew D Marraffino. Using virtual reality for training maintenance procedures. In *Proceedings of the Interservice/Industry Training, Simulation and Education Conference*, 2017.

[BM08] Olivier Bau and Wendy E. Mackay. Octopocus: A dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, UIST '08, page 37–46, New York, NY, USA, 2008. Association for Computing Machinery.

[BMH+20] Alisa Burova, John Mäkelä, Jaakko Hakulinen, Tuuli Keskinen, Hanna Heinonen, Sanni Siltanen, and Markku Turunen. Utilizing vr and gaze tracking to develop ar solutions for industrial maintenance. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery.

[BPC+10] Joel Brandt, Vignan Pattamatta, William Choi, Ben Hsieh, and Scott Klemmer. Rehearse: Helping programmers adapt examples by visualizing execution and highlighting related code. 11 2010.

[BPC19] Marc Baloup, Thomas Pietrzak, and Géry Casiez. Raycursor: A 3d pointing facilitation technique based on raycasting. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–12, New York, NY, USA, 2019. Association for Computing Machinery.

[BRP+05] Eric Burns, Sharif Razzaque, Abigail T Panter, Mary C Whitton, Matthew R McCallus, and Frederick P Brooks. The hand is slower than the eye: A quantitative exploration of visual dominance over proprioception. In *IEEE Proceedings. VR 2005. Virtual Reality, 2005.*, pages 3–10. IEEE, 2005.

[BSS+13] Ligia Batrinca, Giota Stratou, Ari Shapiro, Louis-Philippe Morency, and Stefan Scherer. Cicero - towards a multimodal virtual audience platform for public speaking training. In Ruth Aylett, Brigitte Krenn, Catherine Pelachaud, and Hiroshi Shimodaira, editors, *Intelligent Virtual Agents*, pages 116–128, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Bux90] William Buxton. A three-state model of graphical input. In Dan Diaper, David J. Gilmore, Gilbert Cockton, and Brian Shackel, editors, *Human-Computer Interaction, INTERACT '90, Proceedings of the IFIP TC13 Third Interantional Conference on Human-Computer Interaction, Cambridge, UK, 27-31 August, 1990*, pages 449–456. North-Holland, 1990.

[BV17] Barbara Rita Barricelli and Stefano Valtolina. A visual language and interactive system for end-user development of internet of things ecosystems. *Journal of Visual Languages & Computing*, 40:1–19, 2017. Semiotics, Human-Computer Interaction and End-User Development.

[CC16] Joëlle Coutaz and James L. Crowley. A first-person experience with end-user development for smart homes. *IEEE Pervasive Computing*, 15(2):26–39, 2016.

[CDM19] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. A high-level semantic approach to end-user development in the internet of things. *International Journal of Human-Computer Studies*, 125:41–54, 2019.

[CDRMR19] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. Empowering end users in debugging trigger-action rules. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[CFMP05] M.F. Costabile, D. Fogli, P. Mussio, and A. Piccinno. A meta-design approach to end-user development. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 308–310, 2005.

[CH93] Allen Cypher and Daniel Conrad Halbert. *Watch what I do: programming by demonstration*. MIT press, 1993.

[CHP23] Edwige Chauvergne, Martin Hachet, and Arnaud Prouzeau. User onboarding in virtual reality: An investigation of current practices. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery.

[CL22] Jay L. Caulfield and Felissa K. Lee. Digital Simulation: Applying Critical Thinking to the Practice of Ethical Decision Making. *Journal of Business Ethics Education*, 19:35–66, July 2022.

[CLSZ23] Yiting Chen, Tracy Xiao Liu, You Shan, and Songfa Zhong. The emergence of economic rationality of gpt. *Proceedings of the National Academy of Sciences*, 120(51):e2316205120, 2023.

[CLV+19] Sven Coppers, Kris Luyten, Davy Vanacken, David Navarre, Philippe Palanque, and Christine Gris. Fortunettes: Feedforward about the future state of gui widgets. 3(EICS), jun 2019.

[CMT18] Z. Berkay Celik, Patrick McDaniel, and Gang Tan. Soteria: Automated IoT safety and security analysis. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 147–158, Boston, MA, July 2018. USENIX Association.

[CoS22] CoSpace Edu. Cospaces: Make ar & vr in the classroom, 2022.

[Cou06] Joëlle Coutaz. Meta-user interfaces for ambient spaces. In *Proceedings of the 5th International Conference on Task Models and Diagrams for Users Interface Design*, TAMODIA'06, page 1–15, Berlin, Heidelberg, 2006. Springer-Verlag.

[CPS21] Mengyu Chen, Marko Peljhan, and Misha Sra. Entanglevr: A visual programming interface for virtual reality interactive scene generation. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*, VRST '21, New York, NY, USA, 2021. Association for Computing Machinery.

[CSQ22] Feng-Kuang Chiang, Xiaojing Shang, and Lu Qiao. Augmented reality in vocational training: A systematic review of research and applications. *Computers in Human Behavior*, 129:107125, 2022.

[CTT+24] Minze Chen, Zhenxiang Tao, Weitong Tang, Tingxin Qin, Rui Yang, and Chunli Zhu. Enhancing emergency decision-making with knowledge graphs and large language models. *International Journal of Disaster Risk Reduction*, 113:104804, 2024.

[DAM17] Giuseppe Desolda, Carmelo Ardito, and Maristella Matera. Empowering end users to customize their smart environments: Model, composition paradigms, and domain-specific tools. *ACM Trans. Comput.-Hum. Interact.*, 24(2), apr 2017.

[DAS01] Anind K Dey, Gregory D Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human–Computer Interaction*, 16(2-4):97–166, 2001.

[DLTFH+24] Fernanda De La Torre, Cathy Mengying Fang, Han Huang, Andrzej Banburski-Fahey, Judith Amores Fernandez, and Jaron Lanier. Llmr: Real-time prompting of interactive worlds using large language models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[Edw04] Jonathan Edwards. Example centric programming. *SIGPLAN Not.*, 39(12):84–91, dec 2004.

[FC73] Joseph L. Fleiss and Jacob Cohen. The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability. *Educational and Psychological Measurement*, 33(3):613–619, 1973.

[FFG+17] Andrea Ferracani, Marco Faustino, Gabriele Xavier Giannini, Lea Landucci, and Alberto Del Bimbo. Natural experiences in museums through virtual reality and voice commands. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1233–1234, 2017.

[FGY+04] G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandjiev. Meta-design: A manifesto for end-user development. *Commun. ACM*, 47(9):33–37, sep 2004.

[FHR+21] Katherine Fennedy, Jeremy Hartmann, Quentin Roy, Simon Tangi Perrault, and Daniel Vogel. Octopocus in vr: using a dynamic guide for 3d mid-air gestures in virtual reality. *IEEE Transactions on Visualization and Computer Graphics*, 27(12):4425–4438, 2021.

[FMH+10] Sue Fitzgerald, Renée McCauley, Brian Hanks, Laurie Murphy, Beth Simon, and Carol Zander. Debugging from the student perspective. *IEEE Transactions on Education*, 53(3):390–396, 2010.

[FRE24] Maximilian C. Fink, Seth A. Robinson, and Bernhard Ertl. Ai-based avatars are changing the way we learn and teach: benefits and challenges. *Frontiers in Education*, 9, 2024.

[fS12] UNESCO Institute for Statistics. International standard classification of education: Isced 2011. *Comparative Social Research*, 30, 2012.

[FSAN23] Vittoria Frau, Lucio Davide Spano, Valentino Artizzu, and Michael Nebeling. Xrspotlight: Example-based programming of xr interactions using a rule-based approach. *Proc. ACM Hum.-Comput. Interact.*, 7(EICS), June 2023.

[FST+19] Filippo Andrea Fanni, Martina Senis, Alessandro Tola, Fabio Murru, Marco Romoli, Lucio Davide Spano, Ivan Blečić, and Giuseppe Andrea Trunfio. Pacpac: End user development of immersive point and click games. In Alessio Malizia, Stefano Valtolina, Anders Morch, Alan Serrano, and Andrew Stratton, editors, *End-User Development*, pages 225–229, Cham, 2019. Springer International Publishing.

[Fur08] Borko Furht, editor. *Virtual Reality*, pages 968–968. Springer US, Boston, MA, 2008.

[Gab24] Alex Gabriel. Towards semantic web integration in authoring tools for xr educational content development. In *RealXR@AVI*, 2024.

[Gam20] Fungus Games. Fungus, 2020.

[Gam22] Epic Games. Blueprints visual scripting, 2022.

[GCVMA09] Juan Manuel González-Calleros, Jean Vanderdonckt, and Jaime Muñoz-Arteaga. A structured approach to support 3d user interface development. In *2009 Second International Conferences on Advances in Computer-Human Interactions*, pages 75–81, 2009.

[GF10] Tovi Grossman and George Fitzmaurice. Toolclips: An investigation of contextual video assistance for functionality understanding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 1515–1524, New York, NY, USA, 2010. Association for Computing Machinery.

[GGM+17] Franca Garzotto, Mirko Gelsomini, Vito Matarazzo, Nicolò Messina, and Daniele Occhiuto. Xoom: An end-user development tool for web-based wearable immersive virtual tours. In Jordi Cabot, Roberto De Virgilio, and Riccardo Torlone, editors, *Web Engineering*, pages 507–519, Cham, 2017. Springer International Publishing.

[GHJ+95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, and Design Patterns. Elements of reusable object-oriented software. *Design Patterns. massachusetts: Addison-Wesley Publishing Company*, 1995.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.

[GJSW19] Danilo Gasques, Janet G. Johnson, Tommy Sharkey, and Nadir Weibel. Pintar: Sketching spatial experiences in augmented reality. In *Companion Publication of the 2019 on Designing Interactive Systems Conference 2019 Companion*, DIS '19 Companion, page 17–20, New York, NY, USA, 2019. Association for Computing Machinery.

[GMPS17] Giuseppe Ghiani, Marco Manca, Fabio Paternò, and Carmen Santoro. Personalization of context-dependent applications through trigger-action rules. *ACM Trans. Comput.-Hum. Interact.*, 24(2), apr 2017.

[GNGS24] Daniele Giunchi, Nels Numan, Elia Gatti, and Anthony Steed. Dreamcodevr: Towards democratizing behavior design in virtual reality with speech-driven programming. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pages 579–589, 2024.

[Goo17] Google. Google poly, 2017.

[Goo22] Google. Arcore documentation, 2022.

[GP96] T.R.G. Green and M. Petre. Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of Visual Languages & Computing*, 7(2):131–174, 1996.

[GPSP16] Giuseppe Ghiani, Fabio Paternò, Lucio Davide Spano, and Giuliano Pintori. An environment for end-user development of web mashups. *International Journal of Human-Computer Studies*, 87:38–64, 2016.

[HCH19] Kai-Hsiang Hsu, Yu-Hsi Chiang, and Hsu-Chun Hsiao. Safechain: Securing trigger-action programming from attack chains. *IEEE Transactions on Information Forensics and Security*, 14(10):2607–2622, 2019.

[HDP+07] Ales Holobar, Matjaž Divjak, I. Prelog, D. Korošec, and Damjan Zazula. A distributed virtual reality-based system for neonatal decision-making training. *Computer Applications in Engineering Education*, 15:329 – 339, 11 2007.

[HGG+19] Teresa Hirzle, Jan Gugenheimer, Florian Geiselhart, Andreas Bulling, and Enrico Rukzio. A design space for gaze interaction on head-mounted displays. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–12, New York, NY, USA, 2019. Association for Computing Machinery.

[HWCK07] Björn Hartmann, Leslie Wu, Kevin Collins, and Scott R. Klemmer. Programming by a sample: Rapidly creating web applications with d.mix. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, page 241–250, New York, NY, USA, 2007. Association for Computing Machinery.

[HWSW19] Daniel Hepperle, Yannick Weiß, Andreas Siess, and Matthias Wölfel. 2d, 3d or speech? a case study on which user interface is preferable for what kind of object interaction in immersive virtual reality. *Computers & Graphics*, 82:321–331, 2019.

[IK15] Michelle Ichinco and Caitlin Kelleher. Exploring novice programmer example use. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 63–71, 2015.

[Inc22a] Apple Inc. Arkit, 2022.

[Inc22b] GitHub Inc. Github copilot, 2022.

[Inc22c] Zapier Inc. Zapier, 2022. [Online; accessed 17-February-2022].

[JGH+08] Robert J.K. Jacob, Audrey Girouard, Leanne M. Hirshfield, Michael S. Horn, Orit Shaer, Erin Treacy Solovey, and Jamie Zigelbaum. Reality-based interaction: A framework for post-wimp interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, page 201–210, New York, NY, USA, 2008. Association for Computing Machinery.

[KBOR21] Veronika Krauß, Alexander Boden, Leif Oppermann, and René Reiners. Current practices, challenges, and design implications for collaborative ar/vr application development. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[KKS+19] Tiffany C.K. Kwok, Peter Kiefer, Victor R. Schinazi, Benjamin Adams, and Martin Raubal. Gaze-guided narratives: Adapting audio guide content to gaze in virtual and real environments. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–12, New York, NY, USA, 2019. Association for Computing Machinery.

[KP05] Caitlin Kelleher and Randy Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.*, 37(2):83–137, jun 2005.

[KST+13] Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R. Klemmer, and Jerry O. Talton. Webzeitgeist: Design mining the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 3083–3092, New York, NY, USA, 2013. Association for Computing Machinery.

[LBL+16] Chieh-Jan Mike Liang, Lei Bu, Zhao Li, Junbei Zhang, Shi Han, Börje F. Karlsson, Dongmei Zhang, and Feng Zhao. Systematically debugging iot control system correctness for building automation. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, BuildSys '16, page 133–142, New York, NY, USA, 2016. Association for Computing Machinery.

[LCK+97] James C. Lester, Sharolyn A. Converse, Susan E. Kahler, S. Todd Barlow, Brian A. Stone, and Ravinder S. Bhogal. The persona effect: affective impact of animated pedagogical agents. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, CHI '97, page 359–366, New York, NY, USA, 1997. Association for Computing Machinery.

[LDA+15] Jérémy Lacoche, Thierry Duval, Bruno Arnaldi, Eric Maisel, and Jérome Royan. Plasticity for 3d user interfaces: new models for devices and interaction techniques. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '15, page 28–33, New York, NY, USA, 2015. Association for Computing Machinery.

[LDA+19] Jérémy Lacoche, Thierry Duval, Bruno Arnaldi, Eric Maisel, and Jérôme Royan. 3dplastictoolkit: Plasticity for 3d user interfaces. In Patrick Bourdot, Victoria Interrante, Luciana Nedel, Nadia Magnenat-Thalmann, and Gabriel Zachmann, editors, *Virtual Reality and Augmented Reality*, pages 62–83, Cham, 2019. Springer International Publishing.

[LGK+21] Germán Leiva, Jens Emil Grønbæk, Clemens Nylandsted Klokmose, Cuong Nguyen, Rubaiat Habib Kazi, and Paul Asente. Rapido: Prototyping interactive ar experiences through programming by demonstration. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, UIST '21, page 626–637, New York, NY, USA, 2021. Association for Computing Machinery.

[LGZ+24] Zichao Lin, Shuyan Guan, Wending Zhang, Huiyan Zhang, Yugang Li, and Huaping Zhang. Towards trustworthy llms: a review on debiasing and dehallucinating in large language models. *Artificial Intelligence Review*, 57(9):243, Aug 2024.

[LH24] Nunzio Lorè and Babak Heydari. Strategic behavior of large language models and the role of game structure versus contextual framing. *Scientific Reports*, 14(1):18490, Aug 2024.

[LHV+18] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. *Evaluation Strategies for HCI Toolkit Research*, page 1–17. Association for Computing Machinery, New York, NY, USA, 2018.

[LHZO18] Sarah Lim, Joshua Hibschman, Haoqi Zhang, and Eleanor O'Rourke. Ply: A visual web inspector for learning from professional webpages. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, page 991–1002, New York, NY, USA, 2018. Association for Computing Machinery.

[LJKM+17] Joseph J LaViola Jr, Ernst Kruijff, Ryan P McMahan, Doug Bowman, and Ivan P Poupyrev. *3D user interfaces: theory and practice*. Addison-Wesley Professional, 2017.

[LK09] Gun A. Lee and Gerard J. Kim. Immersive authoring of tangible augmented reality content: A user study. *Journal of Visual Languages & Computing*, 20(2):61–79, 2009.

[LKB05] Gun A. Lee, Gerard J. Kim, and Mark Billinghurst. Immersive authoring: What you experience is what you get (wyxiwyg). *Commun. ACM*, 48(7):76–81, jul 2005.

[LKH21] Klemen Lilija, Søren Kyllingsbæk, and Kasper Hornbæk. Correction of avatar hand movements supports learning of a motor skill. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pages 1–8, 2021.

[LLZM+21] Giorgia Lallai, Giovanni Loi Zedda, Célia Martinie, Philippe Palanque, Mauro Pisano, and Lucio Davide Spano. Engineering Task-based Augmented Reality Guidance: Application to the Training of Aircraft Flight Procedures. *Interacting with Computers*, 33(1):17–39, 03 2021.

[LNKA20] Germán Leiva, Cuong Nguyen, Rubaiat Habib Kazi, and Paul Asente. Pronto: Rapid augmented reality video prototyping using sketches and enaction. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery.

[LPF11] Paul Lukowicz, Sandy Pentland, and Alois Ferscha. From context awareness to socially aware computing. *IEEE pervasive computing*, 11(1):32–41, 2011.

[LPKW06] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. End-user development: An emerging paradigm. In Henry Lieberman, Fabio Paternò, and Volker Wulf, editors, *End User Development*, pages 1–8. Springer Netherlands, Dordrecht, 2006.

[Ltd22] Extend Reality Ltd. Virtual reality toolkit, 2022.

[Mar07] Adam Martin. Entity systems are the future of mmog development–part 1. *T= Machine*, 2007.

[MDA17] Zeno Menestrina and Antonella De Angeli. *End-User Development for Serious Games*, pages 359–383. Springer International Publishing, Cham, 2017.

[MdW14] Ntima Mabanza and Lizette de Wet. *Determining the Usability Effect of Pedagogical Interface Agents on Adult Computer Literacy Training*, pages 145–183. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[Met22] Meta. Oculus integration sdk, 2022.

[MFP+19] Marco Manca, Fabio, Paternò, Carmen Santoro, and Luca Corcella. Supporting end-user debugging of trigger-action rules for iot applications. *International Journal of Human-Computer Studies*, 123:56 – 69, 2019.

[MGM+20] Paulo R. C. Mendes, Álan L. V. Guedes, Daniel de S. Moraes, Roberto G. A. Azevedo, and Sérgio Colcher. An authoring model for interactive 360 videos. In *2020 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pages 1–6, 2020.

[MHP00] Brad Myers, Scott E. Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 7(1):3–28, mar 2000.

[Mic22a] Microsoft. Hand interaction examples — mrtk2, 2022.

[Mic22b] Microsoft. Mixed reality toolkit unity plugin, 2022. [Online; accessed 27-July-2023].

[MKK+18] Florian 'Floyd' Mueller, Tuomas Kari, Rohit Khot, Zhuying Li, Yan Wang, Yash Mehta, and Peter Arnold. Towards experiencing eating as a form of play. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*, CHI PLAY '18 Extended Abstracts, page 559–567, New York, NY, USA, 2018. Association for Computing Machinery.

[MMG+22] Daniel Martin, Sandra Malpica, Diego Gutierrez, Belen Masia, and Ana Serrano. Multimodality in vr: A survey. *ACM Comput. Surv.*, 54(10s), sep 2022.

[MMH23] Andreea Muresan, Jess McIntosh, and Kasper Hornbæk. Using feedforward to reveal interaction possibilities in virtual reality. *ACM Trans. Comput.-Hum. Interact.*, jun 2023. Just Accepted.

[MOM20] Mario Montagud, Pilar Orero, and Anna Matamala. Culture 4 all: accessibility-enabled cultural experiences through immersive vr360 content. *Personal and Ubiquitous Computing*, 24(6):887–905, Dec 2020.

[Moz22a] Mozilla. Mozilla hubs, 2022. [Online; accessed 17-February-2022].

[Moz22b] Mozilla. Mozilla spoke, 2022. [Online; accessed 17-February-2022].

[NLC+20] Michael Nebeling, Katy Lewis, Yu-Cheng Chang, Lihan Zhu, Michelle Chung, Piaoyang Wang, and Janet Nebeling. *XRDirector: A Role-Based Collaborative Immersive Authoring System*, page 1–12. Association for Computing Machinery, New York, NY, USA, 2020.

[NM19] Michael Nebeling and Katy Madier. 360proto: Making interactive virtual reality & augmented reality prototypes from paper. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[NNYR18] Michael Nebeling, Janet Nebeling, Ao Yu, and Rob Rumble. Protoar: Rapid physical-digital prototyping of mobile augmented reality applications. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–12, New York, NY, USA, 2018. Association for Computing Machinery.

[NOM19] Abdallah Namoun, Ali Owrak, and Nikolay Mehandjiev. Non-programmers composing software services: A confirmatory study of the mental models and design challenges. *Applied Sciences*, 9(24), 2019.

[Nor99] Donald A Norman. Affordance, conventions, and design. *interactions*, 6(3):38–43, 1999.

[Nor02] Donald A. Norman. *The Design of Everyday Things*. Basic Books, Inc., USA, 2002.

[Nor08] Donald A Norman. The way i see it signifiers, not affordances. *interactions*, 15(6):18–19, 2008.

[NS72] Allen Newell and Herbert Alexander Simon. *Human problem solving*. Prentice-hall Englewood Cliffs, NJ, 1972.

[NS18] Michael Nebeling and Maximilian Speicher. The trouble with augmented reality/virtual reality authoring tools. In *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 333–337, 2018.

[Ols07] Dan R. Olsen. Evaluating user interface systems research. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, page 251–258, New York, NY, USA, 2007. Association for Computing Machinery.

[Ott18] Ottifox. Ottifox, 2018.

[Pat13] Fabio Paternò. End user development: Survey of an emerging field for empowering people. *International Scholarly Research Notices*, 2013(1):532659, 2013.

[PCD92] Randy Pausch, Matthew Conway, and Robert Deline. Lessons learned from suit, the simple user interface toolkit. *ACM Trans. Inf. Syst.*, 10(4):320–344, oct 1992.

[PD12] Sandra Poeschl and Nicola Doering. Designing Virtual Audiences for Fear of Public Speaking Training – An Observation Study on Realistic Nonverbal Behavior. In *Annual Review of Cybertherapy and Telemedicine 2012*, pages 218–222. IOS Press, 2012.

[Pie18a] Krzysztof Pietroszek. *Raycasting in Virtual Reality*, pages 1–3. Springer International Publishing, Cham, 2018.

[Pie18b] Krzysztof Pietroszek. *Virtual Hand Metaphor in Virtual Reality*, pages 1–3. Springer International Publishing, Cham, 2018.

[PMM97] F. Paterno, C. Mancini, and S. Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In Steve Howard, Judy Hammond, and Gitte Lindgaard, editors, *Human-Computer Interaction INTERACT '97: IFIP TC13 International Conference on Human-Computer Interaction, 14th–18th July 1997, Sydney, Australia*, IFIP — The International Federation for Information Processing, pages 362–369. Springer US, Boston, MA, 1997.

[PS19] Fabio Paternò and Carmen Santoro. End-user development for personalizing applications, things, and robots. *International Journal of Human-Computer Studies*, 131:120–130, 2019. 50 years of the International Journal of Human-Computer Studies. Reflections on the past, present and future of human-centred technologies.

[PZCG13] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE communications surveys & tutorials*, 16(1):414–454, 2013.

[RJTNT+18] Nimesha Ranasinghe, Pravar Jain, Nguyen Thi Ngoc Tram, Koon Chuan Raymond Koh, David Tolley, Shienny Karwita, Lin Lien-Ya, Yan Liangkun, Kala Shamaiah, Chow Eason Wai Tung, Ching Chiuan Yen, and Ellen Yi-Luen Do. Season traveller: Multisensory narration for enhancing the virtual reality experience. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–13, New York, NY, USA, 2018. Association for Computing Machinery.

[RMN+05] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. Design Principles for Tools to Support Creative Thinking. 1 2005.

[RSB13] Amotz Perlman Rafael Sacks and Ronen Barak. Construction safety training using immersive virtual reality. *Construction Management and Economics*, 31(9):1005–1017, 2013.

[SAB+24] James W. A. Strachan, Dalila Albergo, Giulia Borghini, Oriana Pansardi, Eugenio Scaliti, Saurabh Gupta, Krati Saxena, Alessandro Rufo, Stefano Panzeri,

Guido Manzi, Michael S. A. Graziano, and Cristina Becchio. Testing theory of mind in large language models and humans. *Nature Human Behaviour*, 8(7):1285–1295, Jul 2024.

[SBW12] Rajinder Sodhi, Hrvoje Benko, and Andrew Wilson. Lightguide: Projected visualizations for hand movement guidance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, page 179–188, New York, NY, USA, 2012. Association for Computing Machinery.

[SD09] Jeff Sauro and Joseph S. Dumas. Comparison of three one-question, post-task usability questionnaires. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1599–1608, New York, NY, USA, 2009. ACM.

[SDL+24] Elena Sblendorio, Vincenzo Dentamaro, Alessio Lo Cascio, Francesco Germini, Michela Piredda, and Giancarlo Cicolini. Integrating human expertise & automated methods for a dynamic and multi-parametric evaluation of large language models' feasibility in clinical decision-making. *International Journal of Medical Informatics*, 188:105501, 2024.

[SKW+20] Ryo Suzuki, Rubaiat Habib Kazi, Li-Yi Wei, Stephen DiVerdi, Wilmot Li, and Daniel Leithinger. Realitysketch: Embedding responsive graphics and visualizations in ar with dynamic sketching. In *Adjunct Publication of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20 Adjunct, page 135–138, New York, NY, USA, 2020. Association for Computing Machinery.

[SLVdS20] Charles Spence, Jae Lee, and Nathan Van der Stoep. Responding to sounds from unseen locations: crossmodal attentional orienting in response to sounds presented from the rear. *European Journal of Neuroscience*, 51(5):1137–1150, 2020.

[SMZW13] Richard Stone, Elease McLaurin, Peihan Zhong, and Kristopher Watts. Full virtual reality vs. integrated virtual reality training in welding. 2013.

[Sof22a] Valve Software. Steamvr unity plugin, 2022. [Online; accessed 25-April-2023].

[Sof22b] Vive Software. Vive input utility, 2022.

[SS96] A. Steed and M. Slater. A dataflow representation for defining behaviours within virtual environments. In *Proceedings of the IEEE 1996 Virtual Reality Annual International Symposium*, pages 163–167, 1996.

[STA+21] Anthony Steed, Tuukka M. Takala, Daniel Archer, Wallace Lages, and Robert W. Lindeman. Directions for 3d user interface research from consumer vr games. *IEEE Transactions on Visualization and Computer Graphics*, 27(11):4171–4182, 2021.

[Tai]

[Tak14] Tuukka M. Takala. Ruis: A toolkit for developing virtual reality applications with spatial interaction. In *Proceedings of the 2nd ACM Symposium on Spatial User Interaction*, SUI '14, page 94–103, New York, NY, USA, 2014. Association for Computing Machinery.

[TC99] David Thevenin and Joëlle Coutaz. Plasticity of user interfaces: Framework and research agenda. In *Interact*, volume 99, pages 110–117, 1999.

[TCW+20] Andrei Torres, Christopher Carmichael, William Wang, Matthew Paraskevakos, Alvaro Uribe-Quevedo, Paul Giles, and Jamie Lee Yawney. A 360 video editor framework for interactive training. In *2020 IEEE 8th International Conference on Serious Games and Applications for Health (SeGAH)*, pages 1–7, 2020.

[Tec22a] Unity Technologies. Xr interaction toolkit, 2022.

[Tec22b] Unity Techologies. Ar foundation, 2022.

[Tec22c] Unity Techologies. Unity visual scripting, 2022.

[TM02] Michael Terry and Elizabeth D. Mynatt. Side views: Persistent, on-demand previews for open-ended tasks. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology*, UIST '02, page 71–80, New York, NY, USA, 2002. Association for Computing Machinery.

[Tvo20] Company Tvori. Tvori, 2020. [Online; accessed 17-February-2022].

[uni] Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine.

[unr] Unreal Engine | The most powerful real-time 3D creation tool.

[UPYHB+16] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L. Littman. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 3227–3231, New York, NY, USA, 2016. Association for Computing Machinery.

[Val22] Valve. Quickstart — steamvr unity plugin, 2022.

[VLvdHC13] Jo Vermeulen, Kris Luyten, Elise van den Hoven, and Karin Coninx. Crossing the bridge over norman's gulf of execution: Revealing feedforward's true identity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 1931–1940, New York, NY, USA, 2013. Association for Computing Machinery.

[Vv.] Vv.Aa. Extended Reality Market Size, Share & Trends | Growth Report [2032].

[WDO04] S. A. G. Wensveen, J. P. Djajadiningrat, and C. J. Overbeeke. Interaction frogger: A design framework to couple action and function through feedback and feedforward. In *Proceedings of the 5th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, DIS '04, page 177–184, New York, NY, USA, 2004. Association for Computing Machinery.

[WRF+22] Hui-Yin Wu, Florent Robert, Théo Fafet, Brice Graulier, Barthelemy Passin-Cauneau, Lucile Sassatelli, and Marco Winckler. Designing guided user tasks in vr embodied experiences. *Proc. ACM Hum.-Comput. Interact.*, 6(EICS), jun 2022.

[WV09] Heli Wigelius and Heli Väätäjä. Dimensions of context affecting user experience in mobile work. In *Human-Computer Interaction–INTERACT 2009: 12th IFIP TC 13 International Conference, Uppsala, Sweden, August 24-28, 2009, Proceedings, Part II 12*, pages 604–617. Springer, 2009.

[WWQ+19] Huiyue Wu, Yu Wang, Jiali Qiu, Jiayi Liu, and Xiaolong Zhang. User-defined gesture interaction for immersive vr shopping applications. *Behaviour & Information Technology*, 38(7):726–741, 2019.

[XRT22] XRTK. Mixed reality toolkit, 2022.

[YHF+19] LI Yang, Jin Huang, TIAN Feng, WANG Hong-An, and DAI Guo-Zhong. Gesture interaction in virtual reality. *Virtual Reality & Intelligent Hardware*, 1(1):84–112, 2019.

[YKGE21] Enes Yigitbas, Jonas Klauke, Sebastian Gottschalk, and Gregor Engels. Vreud - an end-user development tool to simplify the creation of interactive vr scenes. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–10, 2021.

[YLS+14] Özgür Yürür, Chi Harold Liu, Zhengguo Sheng, Victor CM Leung, Wilfrido Moreno, and Kin K Leung. Context-awareness for mobile sensing: A survey and future directions. *IEEE Communications Surveys & Tutorials*, 18(1):68–93, 2014.

[YLS+21] Difeng Yu, Xueshi Lu, Rongkai Shi, Hai-Ning Liang, Tilman Dingler, Eduardo Velloso, and Jorge Goncalves. Gaze-supported 3d object manipulation in virtual reality. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[ZDAM16] Telmo Zarraonandia, Paloma Díaz, Ignacio Aedo, and Alvaro Montero. Inmersive end user development for virtual reality. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '16, page 346–347, New York, NY, USA, 2016. Association for Computing Machinery.

[ZDMA16] Telmo Zarraonandia, Paloma Díaz, Alvaro Montero, and Ignacio Aedo. Exploring the benefits of immersive end user development for virtual reality. In Carmelo R. García, Pino Caballero-Gil, Mike Burmester, and Alexis Quesada-Arencibia, editors, *Ubiquitous Computing and Ambient Intelligence*, pages 450–462, Cham, 2016. Springer International Publishing.

[ZG18] Xiong Zhang and Philip J. Guo. Fusion: Opportunistic web prototyping with ui mashups. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, page 951–962, New York, NY, USA, 2018. Association for Computing Machinery.

[ZHM+19] Lefan Zhang, Weijia He, Jesse Martinez, Noah Brackenbury, Shan Lu, and Blase Ur. Autotap: Synthesizing and repairing trigger-action programs using ltl properties. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 281–291, 2019.

[ZO20a] Lei Zhang and Steve Oney. *FlowMatic: An Immersive Authoring Tool for Creating Interactive Scenes in Virtual Reality*, page 342–353. Association for Computing Machinery, New York, NY, USA, 2020.

[ZO20b] Lei Zhang and Steve Oney. Flowmatic: An immersive authoring tool for creating interactive scenes in virtual reality. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, page 342–353, New York, NY, USA, 2020. Association for Computing Machinery.

[ZPG+24] Lei Zhang, Jin Pan, Jacob Gettig, Steve Oney, and Anhong Guo. Vrcopilot: Authoring 3d layouts with generative ai models in vr. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, UIST '24, New York, NY, USA, 2024. Association for Computing Machinery.

Appendix A

Appendix: VR Object Taxonomy for ECARules4All

In this appendix, we describe the high-level types included in the VR object taxonomy for building environments templates. The appendix completes the general overview provided in Section 5.1.1 with a detailed description of the different categories included in the taxonomy. We describe the state of *Objects* and *Behaviours* using a simplified set of base types, summarised in Table A.1.

Objects and *Behaviours* have a set of attributes depicting the state and a set of actions representing the operations they can perform (or receive, in case of passive actions). In the following, we list the state description and the actions for each one of the elements in the taxonomy.

An *Object* can be a *Character*, a *Scene*, an *Environment*, a *Prop*, a *Vehicle* or an *Interaction*. In general, each element belonging to the *Object* category has a *position* attribute, and two flags: one for controlling its visibility and the second for setting it as active or not. In this way the EUDev controls whether the object is visible or not and if someone or something can interact with it.

We use a *Character* object to represent an animal, a humanoid, a robot or a generic creature. A character can be autonomous or controlled by the user of the VR environment (the *playing* flag).

A Character, can be further subdivided in several sub-categories, according to the *Animal* or *not-Animal* classification. Figure A.1 shows the actions and the state variables associated with the *Character* sub-categories.

Environment objects represent inanimated elements that configure the VR scene, like buildings, vegetation etc., or elements that decorate or furnish a room. Figure A.2 shows the actions and the state variables associated with the Environment sub-categories.

In the *Prop* category, we represent generic objects that can be placed in a scene and manipulated by characters. We have several possible sub-categories in this case. However, in our definition, we decided to abstract the representation keeping the groups rather abstract, while delegating a more precise indication of the object type to type aliasing or extensions. Figure A.3 depicts the categorization. We provided deeper modelling

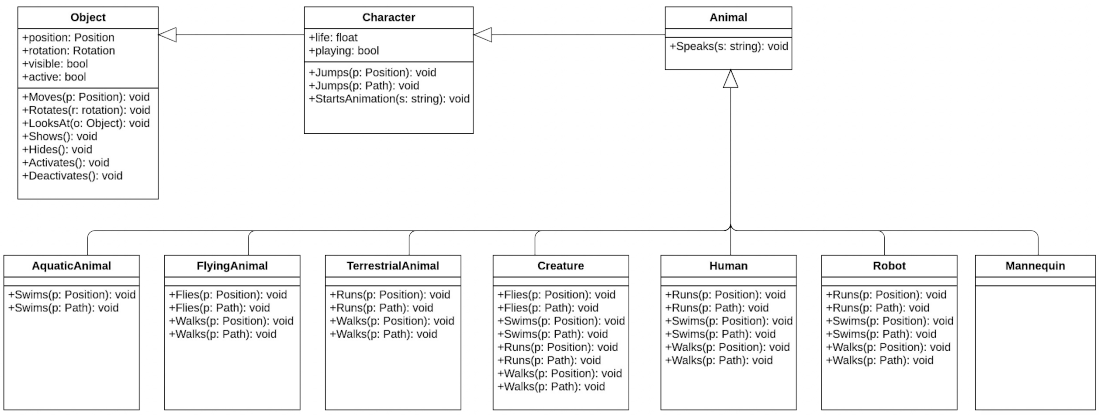


Figure A.1: Class diagram for the Character sub-category.

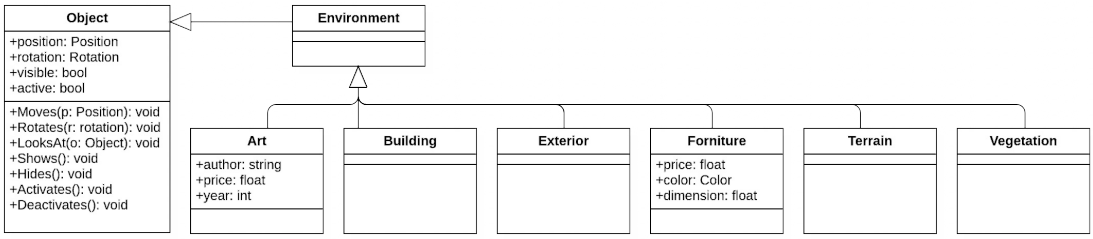


Figure A.2: Class diagram for the Environment sub-category.

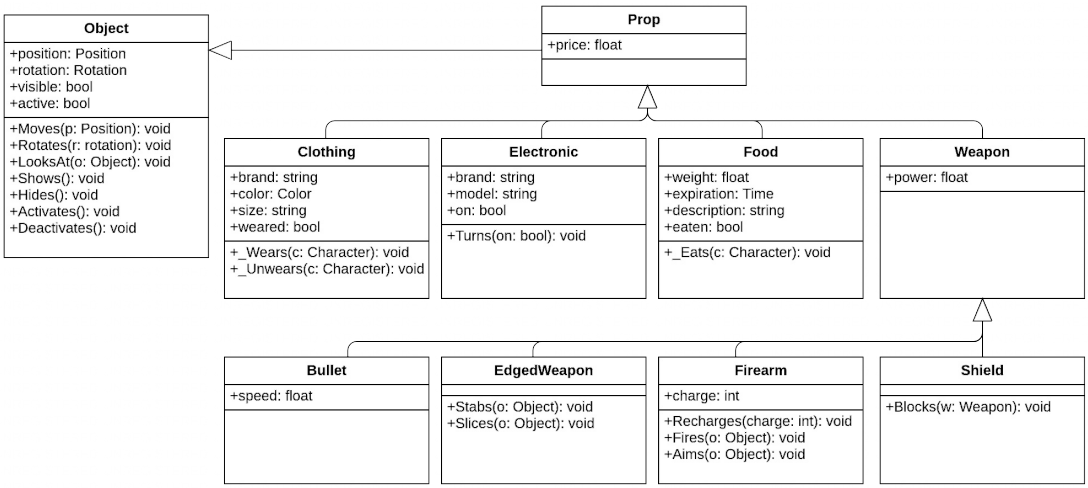


Figure A.3: Class diagram for the Prop sub-category. Actions beginning with an underscore are passive.

for the *Weapons* given their fundamental role in most video games, which we expect will also replicate in EUD VR environments.

It is worth pointing out that *Prop* objects contain actions that belong to them from a software point of view, while the EUDev would rather see them as performed by a character. We call such actions *passive*. For instance, please consider a *Clothing* object. Keeping the implementation of the *wear* action in the Clothing and not in the Character class, allows decoupling the basic actions of a character from those involving other objects. Otherwise, we would break the separation of concerns principle, and the character would become the class defining the large majority of the system. However, from the EUDev point of view, the best action description in natural language is "*the character wears the clothing*" and not "*the clothing is weared by the character*". So, passive actions in the class representation use the parameter as the subject of the sentence in natural language (see Section 5.1.2).

A *Scene* represents a setting in the VR environment a user can visit entirely in a continuous way, i.e., without teleportation. We can associate it to a videogame level: when the level changes, the scene changes too. It defines passive actions for entering and leaving the scene.

The *Vehicle* category represents, in an abstract way, all the vehicles TB can place in scene. We distinguish them according to the element they support travelling on (air, land, sea, space). Figure A.4 depicts the categorization.

The *Interactions* category contains all the elements allowing some interaction with the scene and its objects. The main point that distinguishes them from the *Behaviour* elements is that a typical user would perceive them as physical entities of their own, which would exist independently from other objects. Instead, a behaviour adds the ability of e.g., counting, opening or highlighting *other* objects. For instance, we consider two

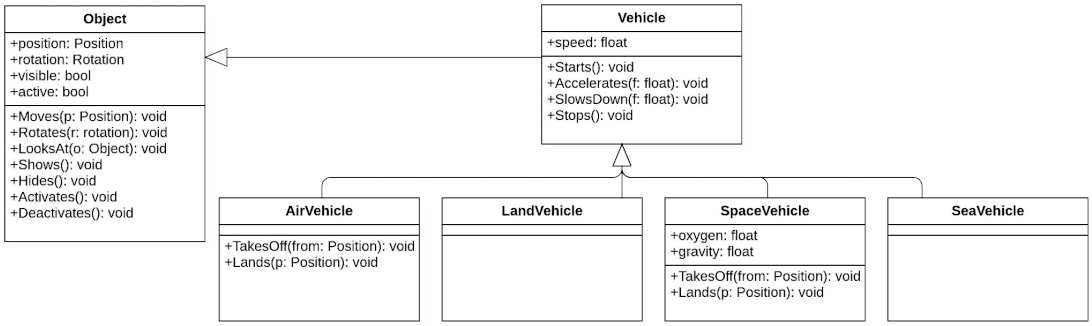


Figure A.4: Class diagram for the Vehicle sub-category.

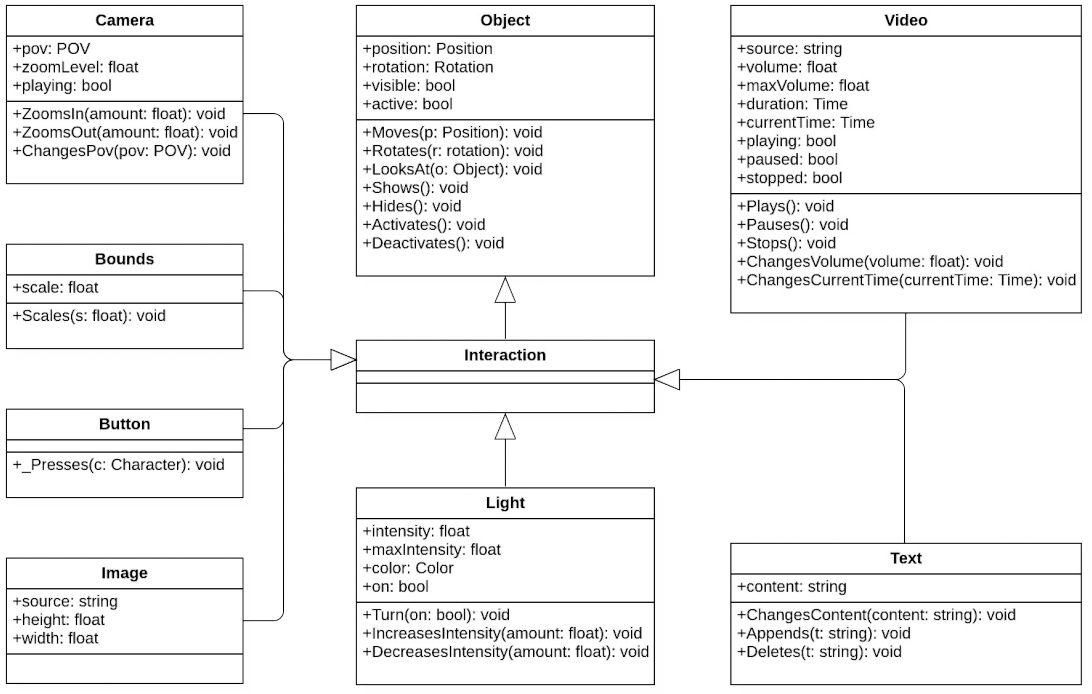


Figure A.5: Class diagram for the Interaction sub-category.

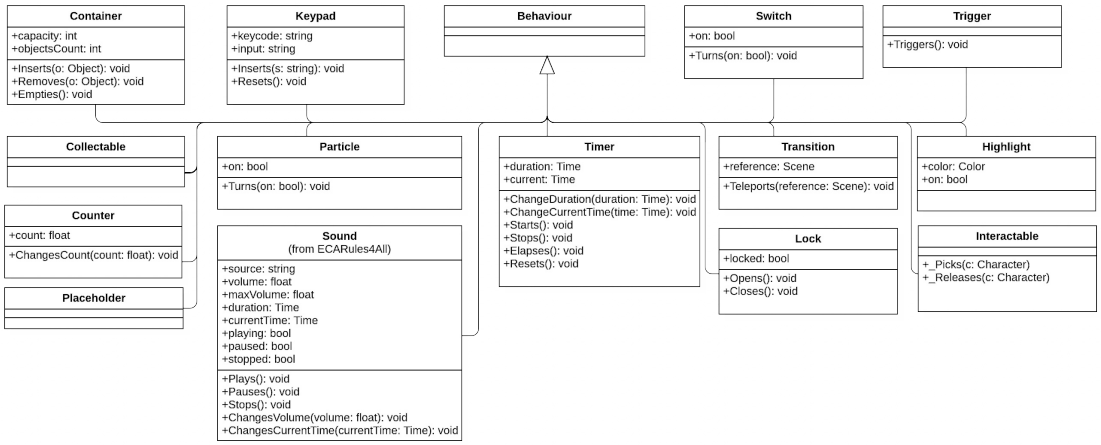


Figure A.6: Class diagram for the Behaviour category.

multimedia items such as an advertisement video and a song played by a juke-box. We model the advertisement as a *Video* interactive element (i.e., belonging to the interaction category) since the user of the VR environment perceives the 2D plane projecting the video. The song instead has no physical presence of its own, but the user associates it to the juke-box, which is its source and the object modified by the *Sound* behaviour. In this way, a TB can model objects having multiple behaviours, for instance, creating a juke-box that plays songs and highlights when the user interacts with it. Figure A.5 lists all the interaction objects and the associated actions.

Given their lack of physical presence of their own, the *Behaviour* does not extend *Object*. Figure A.6 depicts the list of available behaviours and the actions they add to the associated objects. It is worth pointing out that adding a behaviour to an object increases the number of actions that the instance supports in the description of the rules in natural language (see Section 5.1.2).

EUDev Type	**Definition**
Boolean	A boolean value. We support equivalent labels for true and false (e.g., yes/no, on/off etc.).
Color	A pair including the color name and its hex-code
Float	A number having decimal places
Integer	An integer number.
Position	Coordinates of a position in the 3D world.
Path	An ordered list of positions in the 3D world.
Identifier	A human-readable representation of a reference to an object in the scene. We will render it in natural language using the pair ¡*Type*, *Name*¿ (e.g., the Robot Daitarn-3).
Rotation	Euler angles in degrees.
Text	A string of text.
Time	A time interval (days, hours, minutes, seconds).

Table A.1: EUDev data types for the object status variables

Appendix B

Appendix: System Prompt for BeatriXR LLMs

You are tasked with suggesting zero or more alternative new options for an animation representation in a virtual reality environment. You should only analyze these options as a controller for the animation, as the user is not influenced by the change of any of these options, the user will only see the animation set up with the settings in this prompt.

Instructions:

1. **Read** the descriptions of the environment settings below. 2. **Note** the weight of each item in parentheses (1 = negligible importance, 5 = extremely important). 3. **Evaluate** the significance of each setting based on its weight and description. 4. **Read** the prompt of the user, which will contain a **description of the task** and the **set of default options** for the representation. 5. **Decide** on possible new settings that could be better - you can also suggest no changes for any or all options, if you evaluate it is better to do so. Please give your decision in the same line as the option, so it's clear which option you are referring to. 6. **DO NOT** talk about values outside this list, since it's not necessary and out of scope for your task.

—

Assumptions

- When talking about the "Target" the system is referring to the animation representation of the real target object. So for example, if the target is transparent then it will be transparent over the real representation of the object, if the settings say that the target is duplicated. If the target is not duplicated then the animation representation will act as the real object (even though the real object is just hidden) - You are helping a developer in setting options for modify an animation that will show how the user have to perform its interaction (like a tutorial). The action you are going to analyze has nothing to do with the action the user has to perform to complete the steps, it is a separate function. So for example, "Trigger Mode" to "Button Triggered" means that the user pushes the "Activate Animation" button to play the animation, that helps the user learn the action he needs to do. None of these settings affect how the user acts to accomplish the action, only the appearance and behaviour of the animation itself. - **The developer has no control over

implementation of features different than the ones described here**, the only choice he has is to set different options for the animation representation.

LLM Model Analysis Request

Please analyze all the options in the system prompt and return the analysis for each one. The analysis **MUST** include all of the following options:

1. **Trigger Mode** 2. **Signifiers** 3. **Level of Detail** 4. **Visible Elements** 5. **Avatar Type** 6. **Perspective** 7. **Target Position** 8. **Avatar Rendering** 9. **Target Rendering** 10. **Avatar Duplication** 11. **Target Duplication** 12. **Dismissal Mode** 13. **Exit Transition**

Available Settings

Trigger Mode (Weight: 4)

Determines how the user activates the animation. Options include:

- **Gaze-enabled**: Allows hands-free activation but may lead to accidental triggers. - **Position-enabled**: Allows hands-free activation but may lead to accidental triggers. - **Timer-based**: Useful for time-sensitive tasks but may be frustrating if the user is not ready. - **Button-triggered**: Provides deliberate control but requires a voluntary intention from the user. - **Automatic**: Convenient but may be disruptive if the user is not expecting it.

Signifiers (Weight: 2)

Hints indicating where the animation is available:

- **Enabled**: Aids in discovery but may clutter the interface. - **Disabled**: Creates a cleaner look but may make features less apparent.

This option may be important for “Gaze-enabled” and “Position-enabled” Trigger Modes. The other ones do not rely heavily on this option, if not for understanding where the user needs to look when the animation is triggered.

Level of Detail (Weight: 3)

Specifies the amount of detail shown in the animation:

- **Full Animation**: Provides comprehensive information but may overwhelm users. - **Necessary Interaction Only**: Simplifies understanding but may omit helpful details.

Visible Elements (Weight: 5)

The user can see combinations of:

- **Action Representation** - **Target Object** - **Avatar**

Displaying more elements provides complete context but can be distracting. Fewer elements focus user attention but might leave out important cues. **No elements** remove all visual guidance and are highly discouraged unless the user is an expert or has requested minimal guidance.

Avatar Type (Weight: 3)

If the avatar is visible, it can appear as:

- **User’s Controllers Representation** - **Selected Parts of Controllers** - **Puppet Representing the User**

Detailed avatars enhance immersion but may increase complexity. Simplified representations reduce cognitive load but might lessen engagement. **Every option** offers

an **accurate and realistic representation of the required user's actions** for completing the step. No option is fantasy-like; it always represents either the real user's controllers or a realistic human representation. The puppet representation is a realistic human representation that shows what the user should do like if another person was showing the user how to do it.

Perspective (Weight: 3)

The experience can be viewed from:

- **First-person Perspective**: Increases immersion but limits overall view. - **Third-person Perspective**: Offers broader perspective but may reduce personal connection.

Target Position (Weight: 1)

If the target is visible, it can be:

- **Slightly Dislocated**: Highlights interactivity but can confuse spatial understanding. - **Correct Position**: Maintains realism but may not draw attention.

—

Rendering Options

Avatar Rendering (Weight: 4)

- **Transparent**: Minimizes visual obstruction but may be harder to see. - **Opaque**: Clearly visible but can block the environment.

Target Rendering (Weight: 4)

- **Transparent** - **Opaque**

Similar trade-offs as avatar rendering.

—

Duplication

Avatar Duplication (Weight: 3)

If the avatar duplication is set to "Duplicated" then the avatar of the user will be shown alongside the animation representation. If the avatar duplication is set to "Singular" then the avatar of the user will be hidden and only the animation representation will be shown.

- **Duplicated**: Can demonstrate actions but may confuse the user. - **Singular**: Keeps focus on the user's own movements but may lack context.

Target Duplication (Weight: 3)

If the target rendering is set to "Transparent" then the real object will be visible and this option will show a second identic object. If the target rendering is set to "Opaque" then the real object will be hidden and this option will show a second identic object.

- **Duplicated**: Illustrates outcomes but may clutter the scene. - **Singular**: Maintains simplicity but may not provide enough information.

—

Dismissal Mode (Weight: 2)

Determines how the animation is dismissed:

- **Gazing Away**: Offers quick dismissal but may be unintentionally triggered. - **Moving Away**: Requires deliberate action but may be cumbersome. - **Automatically**: Convenient but may disrupt the user's flow.

Exit Transition (Weight: 4)

Transition back to the initial state can be:

- **Instantaneous**: Quick but abrupt. - **Rewinding from End**: Provides context but takes longer. - **Fade Out and In**: Offers smoothness but may delay return.

All the Exit Transition options are automatically triggered, the only difference is the appearance of the return of the initial state

—

Environmental Factors

Possible factors in displaying different representations of a virtual avatar when certain conditions are detected:

Noisy Environment (Weight: 1)

Background noise can interfere with understanding the instructor, making it difficult for the user.

Controller Unavailable (Weight: 2)

Without controllers, users need alternative interaction techniques like hand tracking, which may not be as precise.

Experience Level (Weight: 5)

- **Little to No Experience with VR** - **Some Experience with VR** - **A Lot of Experience with VR**

Inexperienced users may need detailed representations; experienced users may need only hints or reminders.

Difficulty Preference (Weight: 5)

- **Prefers Most Help Available** - **Moderate Amount of Help** - **Little to No Help**

Users preferring less difficulty may need the experience adjusted for lower complexity.

Realism (Weight: 3)

User preference for environment representation:

- **More Realistic**: Provides immersive experience but requires more resources. - **More Abstract**: More efficient but may not be as engaging.

List of Figures

5.1 Class diagram for the object categories in the VR templates. 26

5.2 Class diagram for the ECA rule engine. 30

5.3 Rule-editing interface for EUDevs. Part A shows the activation of the authoring mode, part B shows the non-dominant hand menu for activating the different authoring sections, while part C shows the rule editing panel and the highlighting of the involved VR objects. 37

5.4 The UI panel for managing placeholders. 39

5.5 The virtual shop template. 40

5.6 Sample interactions in the clothes store configuration. A) When the visitor approaches the table, a blue light highlights the table, and the TV monitor plays an advertisement. B) The visitor approaches another table, and the environment shows some price tags. C and D) The mannequin wears a different outfit according to the button pressed. 41

5.7 Sample interactions in the shoe store configuration. A) Pressing the button, the stool will rotate by 30°. B) Interacting with the radio will turn on or off the background music. C and D) When the user manipulates the shoes, above it will appear an informative panel. It disappears as soon as the user releases them. 42

5.8 The virtual museum template (isometric view - walls and ceilings are transparent for clarity). 43

5.9 Left: Each room in the museum has a specific soundtrack that plays when the player enters it, the music stops when the player leaves the room. Right: When the user approaches any artwork, the corresponding plate with information will appear. 45

5.10 When the user interacts with the plate, a 360° video starts playing around him. As the video ends, it disappears (the second image shows the outside of the video-globe, the user will be inside it, as shown in the image on the right). 45

5.11 Task completion time and perceived effort [SD09]. Highlighted pairs have a significant difference (Wilcoxon signed-rank test, $p < .05$. We used the Bonferroni p-valued adjustement for repeated comparisons). 48

5.12 Participants assessment of the usefulness, difficulty and overall usability of the approach. They assessed the usefulness and the difficulty before and after completing the task. Values are in a 1 to 7 Likert scale. 49

5.13 Interface for the action recognition. The panel shows each interaction and action performed during the recording phase, using a video clip metaphor. Each clip has a screenshot of the EUDev's view immediately before the interaction and a small textual description as the title. By selecting (part A) or deselecting (B) a clip with the remote, the EUDev confirms or discards the clips useful for defining the desired rules. 52

5.14 Interface for trigger and actions identification. After the recognition step, the tool assumes the first action as a trigger for a single rule. The tool allows selecting another one by pointing and selecting another among the clips (part B). In addition, the EUDev can split the trigger-action chain using the scissor tool. This allows to create more than one rule using the same recording session (parts C and D). 53

6.1 An overview of the scenes and interaction used in the walkthrough. A: the coffee cup in the MRTK sample scene has a manipulation interaction. B: the kitchen scene with the egg ingredient shows feedback on manipulation. C: the same egg shows the feedback on pointing after changing the rule trigger. D: the feedback replicated on the wrong ingredients. E: the pan drag interaction imported from a SteamVR sample. 58

6.2 The XRSpotlight panel in the Unity editor. A: foldout menu showing all the interactable objects in the scene (partially cut for readability). B: information about the support to collision detection and physics. C: interactive behaviours provided by the components associated with the XR object. D: interaction rules, aggregated by phase (see Section 6.4.1). Each rule entry has icons representing the modalities it supports and a set of buttons allowing (starting from the upper left): to change the trigger, copy the rule, delete the action, find similar actions and show the definition on the Inspector. E: Highlighting the definition of a rule action (consequence) in the Unity Inspector. 59

6.3 XRSpotlight shows the interactions similar to the one currently selected by showing an ordered list of rules involving other objects in the scene. For each object (A_i), it shows the most similar rules it supports, including itself (A_1). Next to each row, we show the percentage of the similarity (B_i) and rules similar to the one displayed. 60

6.4 The import (transition) of an interaction from an example through XRSpotlight, and its replication on other objects. It starts with a copy of the rules Selected and Released from the coffee mug located in the MRTK example on the egg in the kitchen scene (A). Then, the rules are adapted to support a hovering interaction through the change trigger feature (B). After that, the obtained rules are replicated on three other ingredients through copy-paste (C). 61

6.5 Using an abstraction of existing toolkit implementations in our model and a mapping from each toolkit to the model, XRSpotlight can express interactions in a toolkit-agnostic way using natural language, find similar interactions for a given example, and copy-paste interactions between scenes to support the use and adaptation of examples provided by different toolkits. At the technical level, toolkit capabilities are captured in terms of behaviours, phases, and modalities, while the interactions defined in a scene are represented as trigger-action rules. 62

6.6 Different rule groupings for representing the same interaction (the egg hovering feedback described in Section 6.1). In part A, rules are grouped by phases (blue boxes), showing multimodal reactions to three phases: enter (pointing started), release (the selection disengaged) and left (pointing ended). In part B, rules are grouped by modality (red boxes). Rules working in multiple modalities are reported in each corresponding group, using the specific trigger name. In part C, rules are grouped by action, highlighting the events sharing the same effect (green boxes). . . 68

6.7 How to paste a previously copied rule s on a destination d depends on the target scene. In the same scene (green), the trigger uses the same source component and event. The actions do not need to find compatible objects since they are available in the scene. Considering two different scenes (blue) using the same toolkit differs for the action replication, requiring that the involved scripts are available in the target scene, and the developer selects among the compatible objects. Otherwise, the actions require replacement. Finally, the paste between two different toolkits (red) requires establishing a correspondence between the two toolkits through the model. The action replication logic is similar to the previous case. We provide a concrete copy-paste examples in Section 6.1. 71

6.8 Overview of both experts' and novices' studies. The tasks inside the kitchen scene are represented in the first row (K1-K4). The second row shows the Example scene containing four example interactions: A button that pops up red particles when clicked (E1), a cube that rotates when focused (E2), a maraca that plays a sound when manipulated (E3), a basketball affected by gravity when grabbed (E4). 74

6.9 Summary of the data collected during the novice study, including the correctness of the proposed solution to the task (see Table 6.4), the perceived difficulty, the number of accesses to irrelevant information during the task, and the time on task. We report the values per experimental condition (Inspector and XRSpotlight, top row) and per group defined by the condition ordering (bottom row): Inspector - XRSpotlight (I→X) or XRSpotlight - Inspector (X→I). 79

7.1 The anatomy of ViRgilites: from feedback of past actions and interactions, over the current action and interactions at hand, to guidance toward the next action and interactions. In ViRgilites, the virtual object depicted alongside the interaction, provides a shortcut to navigate to the object in the virtual environment. 84
7.2 Organisation of the ViRgilites guidance interface. It contains three panels: the middle (C) containing information on how to execute the current action, the left (Pa) on the previous successful action (and interaction), while the right on the next possible action and the associated interactions (Na). Users browse the available interactions (if more than one) using two dedicated buttons (P and N). The top-left push button activates or deactivates the pinned mode (F), the top-center (T) activates the target object highlighting, while the top-right (L) activates the compact (blue part) or expanded (blue plus green) view mode. The bottom handle (H) enables the reposition of the user interface in the virtual space. 88
7.3 UML component diagram of the ViRgilites architecture. The *TaskManager* and the *ContextManager* provide the information to the feedforward component (ViRgilites). The *XR_ContentManager* synchronises the state of virtual and real objects that blend in the *XR_View*. 94
7.4 ViRgilites interactions in the chemistry lab (A, B) and kitchen (C, D) scenarios. Figure A shows the user picking up the gloves from afar. Figure B shows the user changing the interaction with another one. Figure C shows the user grabbing the rolling pin. Figure D shows the user trying to identify which object he should interact with. 97
7.5 Violin box plots of the Feedforward comprehension results 102
7.6 Violin box plots of the Automatic Interaction Suggestion results 103

8.1 Triggering capabilities for BeatriXR. The supported settings are: Gaze (Panel A), Position (Panel B), Timer (Panel C), Action (Panel D), Persistent (Panel E). Signifiers (Panel F) show the presence of a representation for the target object. 115
8.2 Previewing capabilities for BeatriXR. The supported settings are: Level of Detail (Panel A), visibility of the Targets (Panel B), Avatar Type (Panel C), Perspective (Panel D), Transform Modifiers (Panel E), Rendering (Panel F), Duplication (Panel G). 116

8.3 Exiting Capabilities for BeatriXR. Untriggering settings support: Gazing Away (Panel A), Moving Away (Panel B), Automatic dismissal (Panel C). Exit Animation settings support: Rewinding of the animation (Panel D), Instantaneous return to the original state (Panel E), Fade in/Fade out effect (Panel F). 117
8.4 Following a simplified version of the “Chemical Lab” scenario in section 8.1.2, the system has, right from the start, default values for the feedforward (green box), if a particular step may require a change of settings, the procedure creator can also set some override options (red boxes) that can be suggested and enabled, if the AI algorithm evaluates it is a good idea to do so. The figure uses the notation from ConcurTaskTrees [PMM97] for showing the flow of the procedure, where a step enables the following one once the current one is completed (⟩⟩) and the interactions can be chosen by the user for deciding how to complete the step ([ ]) 117
8.5 UML component diagram for BeatriXR 119
8.6 Average scores for the LLM models, after the expert validation phase. . . 128

A.1 Class diagram for the Character sub-category. 158
A.2 Class diagram for the Environment sub-category. 158
A.3 Class diagram for the Prop sub-category. Actions beginning with an underscore are passive. 159
A.4 Class diagram for the Vehicle sub-category. 160
A.5 Class diagram for the Interaction sub-category. 160
A.6 Class diagram for the Behaviour category. 161

La borsa di dottorato è stata cofinanziata con risorse del
Programma Operativo Nazionale Ricerca e Innovazione 2014-2020, risorse FSE REACT-EU
Azione IV.4 “Dottorati e contratti di ricerca su tematiche dell’innovazione”
e Azione IV.5 “Dottorati su tematiche Green”