# Space Tentacles - Integrating Multimodal Input into a VR Adventure Game

Chris Zimmerer[*] Universität Würzburg
Martin Fischbach[†] Universität Würzburg
Marc Erich Latoschik[‡] Universität Würzburg

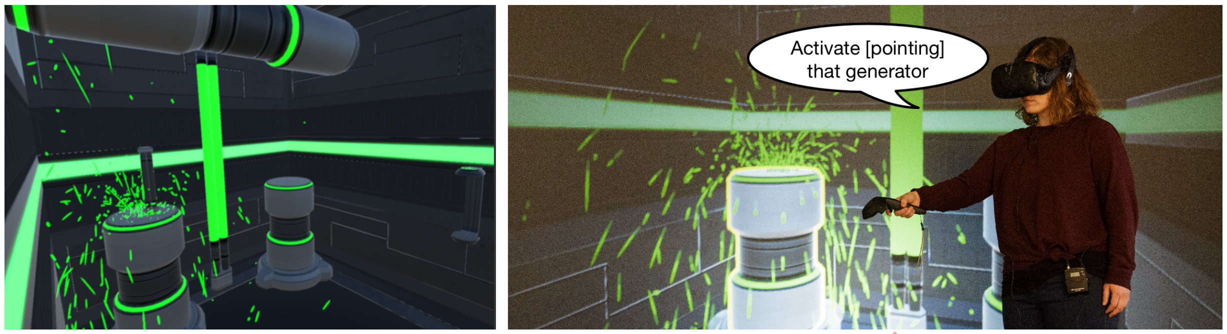

Figure 1: The user is immersed in a futuristic environment (left) where she has to solve puzzles to progress the storyline of an adventure game. She has the choice to interact with the environment either by direct manipulation or by multimodal, combined speech and gesture input, e.g., to ask for guidance or to tell the system to perform certain tasks, e.g., to activate a generator (right).

## Abstract

Multimodal interfaces for Virtual Reality (VR), e.g., based on speech and gesture input/output (I/O), often exhibit complex system architectures. Tight couplings between the required I/O processing stages and the underlying scene representation and the simulator system's flow-of-control tend to result in high development and maintainability costs. This paper presents a maintainable solution for realizing such interfaces by means of a *cherry-picking* approach. A reusable multimodal I/O processing platform is combined with the simulation and rendering capabilities of the *Unity* game engine, allowing to exploit the game engine's superior API usability and tool support. The approach is illustrated based on the development of a multimodal VR adventure game called *Space Tentacles*.

**Index Terms:** Human-centered computing—Interaction paradigms—; Software and its engineering—Software architectures—; Computing methodologies—Virtual reality—

## 1 Introduction

Multimodal Interfaces (MMI) and their potential benefits [8] appear promising for VR [3]. However, their VR-related pros and cons are rarely investigated so far. Today, several tools support implementation of the various stages of multimodal interfaces. Still, the integration of all required I/O sub-systems into a complete multimodal VR interface typically results in complex system architectures. Tight couplings between the I/O processing stages and the underlying scene representation and the simulator system's flow-of-control tend to result in high development and maintainability costs [1].

VR platforms that support the realization of MMIs have to fulfill a large variety of functional requirements, ranging from stereoscopic rendering and physics simulation to the combined analysis of complex multimodal input, like speech and gestures. As a consequence, platforms that support all this functionality at once are rare [1].

[*]e-mail: chris.zimmerer@uni-wuerzburg.de
[†]e-mail: martin.fischbach@uni-wuerzburg.de
[‡]e-mail: marc.latoschik@uni-wuerzburg.de

Commercial platforms, like *Unity* [1] or *Unreal* [2], are widely used to develop VR applications. They excel in API usability and specific VR- or games-related functionality but do not support sub-systems for MMIs. Typical ad-hoc implementations that aim to integrate MMIs, i.e., platforms that focus on the analysis of multimodal input, into VR-platforms, however, suffer from severe maintainability pitfalls [1].

**Contribution:** We showcase a maintainable approach for realizing multimodal VR interfaces using commercial game engines by the example of the *Space Tentacles* adventure game (see Fig. 1 for a description of the game). This approach applies *cherry-picking* [9], a software integration concept for combining desired functionality from multiple platforms based on a semantic description layer. This layer is implemented by means of the software techniques *Code from Semantics* and *Semantic Grounding* [1]. We combine the functionality of the *Simulator X* platform [4] for analyzing speech and gesture input with the simulation and rendering capabilities of the *Unity* game engine. Both software techniques allow to foster modifiability as well as reusability. They improve the maintainability of the overall solution, i.e., of the platform-compound. The improvement is further supplemented by a special consideration of the platform-compound's usability for application developers.

## 2 Related Work

Benefits of multimodal interaction in other domains, like Augmented Reality [5], have been shown. This is expected for VR as well [8] despite formal evaluations being sparse. Both *Virtuelle Werkstatt* [3] and *Iconic* [2] are examples for complex VR applications with multimodal interfaces. They have been developed with research VR platforms supporting the realization of MMIs. But both platforms are not available anymore [1].

Employing game engines in VR research has been advocated for some time [6]. Ad-hoc implementations integrate different SDKs for speech and gesture recognition in commercial platforms to develop multimodal VR interfaces.Yet, this approach is not always straight forward and prone to errors [7].

## 3 System Design

Space Tentacle is implemented by cherry-picking functionality from two platforms as illustrated in Fig. 2: Game mechanics (logic) as

[1]http://www.unity3d.com
[2]https://www.unrealengine.com

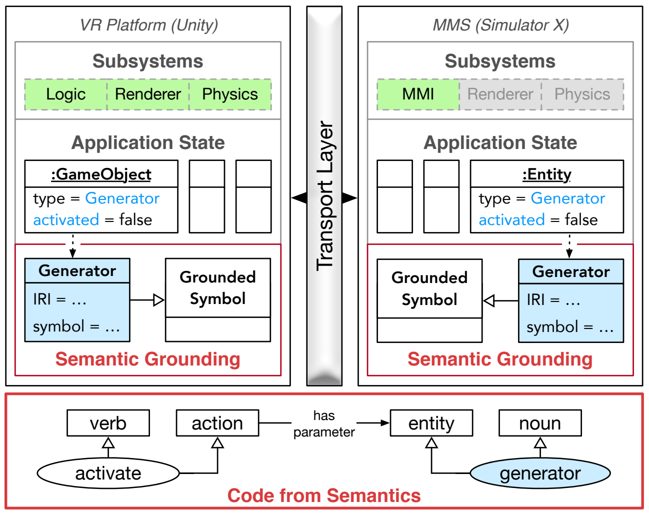

Figure 2: Space Tentacle's architecture is a platform-compound, consisting of Unity and Simulator X. Extensions to the cherry-picking approach from [9] are marked red. Examples of externally defined concepts that are relevant for representations and APIs as well as their integration into both platforms is colored blue. See text for details.

well as the virtual environment (rendering, physics) are realized in Unity. The combined analysis of speech and gesture input is implemented with the open source platform Simulator X. Context information, like which objects are near to each other or at which object the user is pointing at, is calculated in Unity using colliders. Application state synchronization is implemented by means of a TCP transport layer as proposed by [9]. The necessary matching of application state elements (entities), however, is refined beyond the utilization of entity and component IDs. We base this matching on semantic descriptions that are externally defined using the *Code from Semantics* technique (see lower red box). These definitions signify concepts that are relevant for APIs, like properties and behavior of (virtual) environment elements as well as of the system itself. They provide a common ground for communication and are automatically transformed into first-class citizens of the target programming languages in order to realize the *Semantic Grounding* technique in each platform (see upper red boxes). Two exemplary definitions of relevant concepts for Space Tentacles are the noun `generator`, which denotes an entity, and the verb (to) `activate`, which denotes a behavioral aspect of the application, named *action*. The externalization furthermore permits to define relations between these behavioral aspects and state elements, i.e., the fact that a generator can be activated (not illustrated).

The transformation of these definitions for Unity are realized using dedicated data structure, i.e., *GroundedSymbols*. They comprise references, i.e., *IRI*s, that facilitate lookups and allow to exploit available reasoning algorithms running on the external definition at runtime. *GroundedSymbols* are used twofold within Space Tentacles: As values of *GameObject* properties, like `Generator`, and as identifiers for properties themselves, like `activated`.

Simulator X already adopts the required software techniques. Moreover, it utilizes additional semantics based software techniques [1], e.g., to uniformly represent application behavior. This uniformity is highly beneficial for multimodal input processing. User input has to be analyzed with respect to the current context, i.e., the application state, in order to derive a command, i.e., an action the application can execute. The verb of the utterance "*Activate [pointing] that generator*", for example, is mapped to the action `activate`. Subsequent processing accesses the application state to perform a semantic consistency check, i.e., to check if the user really points at a generator that is not already activated. These informations are computed within Unity and synchronized to Simulator X's application state facilitated by the common grounding.

## 4 Results

The presented system design refines the *cherry-picking* approach by applying the software techniques *Code from Semantics* and *Semantic Grounding*. This fosters reusability due to the platform independent external definition of identifiers, used for representing and accessing application state and -behavior. As a consequence, the error prone use of manual IDs or entity names is omitted. The also external definition of relations between state- and behavioral aspects is beneficial for the combined analysis of multimodal input and facilitates modifications. Taken together, these aspects improve maintainability. They are supplemented by a dedicated Unity editor script that graphically realizes the specification of what to bidirectionally synchronize.

## 5 Conclusion

This paper presents a maintainable solution for realizing multimodal VR interfaces using a commercial game engine by means of the *cherry-picking* approach. The Space Tentacles adventure game serves as complex application example. This game as well as the *Unity-Simulator X* platform-compound will be the testbed for future work: (1) We aim to further reduce the development and maintainability effort of multimodal VR applications by refining the *Code from Semantics* technique and by improving its tool support. (2) We will make use of this reduced effort to efficiently research the initial suitability prediction of MMI for VR, which is still unanswered.

## References

[1] M. Fischbach, D. Wiebusch, and M. E. Latoschik. Semantic Entity-Component State Management Techniques to Enhance Software Quality for Multimodal VR-Systems. *IEEE Transactions on Visualization and Computer Graphics*, 23(4):1342–1351, 2017.

[2] D. B. Koons and C. J. Sparrell. Iconic: Speech and Depictive Gestures at the Human-machine Interface. In *Conference Companion on Human Factors in Computing Systems*, CHI '94, pp. 453–454. ACM, New York, NY, USA, 1994. doi: 10.1145/259963.260487

[3] M. E. Latoschik. A User Interface Framework for Multimodal VR Interactions. In *Proceedings of the 7th International Conference on Multimodal Interfaces*, ICMI '05, pp. 76–83. ACM, New York, NY, USA, 2005. doi: 10.1145/1088463.1088479

[4] M. E. Latoschik and H. Tramberend. Simulator X: A scalable and concurrent architecture for intelligent realtime interactive systems. In *2011 IEEE Virtual Reality Conference*, pp. 171–174, March 2011. doi: 10.1109/VR.2011.5759457

[5] M. Lee, M. Billinghurst, W. Baek, R. Green, and W. Woo. A Usability Study of Multimodal Input in an Augmented Reality Environment. *Virtual Real.*, 17(4):293–305, Nov. 2013. doi: 10.1007/s10055-013-0230-0

[6] M. Lewis and J. Jacobson. Game Engines in Scientific Research. vol. 45, pp. 27–31. ACM, 2002. doi: 10.1145/502269.502288

[7] M. Müller, T. Günther, D. Kammer, J. Wojdziak, S. Lorenz, and R. Groh. Smart Prototyping - Improving the Evaluation of Design Concepts Using Virtual Reality. In S. Lackey and R. Shumaker, eds., *Virtual, Augmented and Mixed Reality*. Springer International Publishing, Cham, 2016.

[8] S. Oviatt, B. Schuller, P. R. Cohen, D. Sonntag, G. Potamianos, and A. Krüger, eds. *The Handbook of Multimodal-Multisensor Interfaces: Foundations, User Modeling, and Common Modality Combinations - Volume 1*. Association for Computing Machinery and Morgan &#38; Claypool, New York, NY, USA, 2017.

[9] D. Wiebusch, C. Zimmerer, and M. E. Latoschik. Cherry-Picking RIS Functionality – Integration of Game and VR Engine Sub-Systems based on Entities and Events. In *10th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. IEEE Computer Society, 2017.