2024 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct) | 979-8-3315-0691-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISMAR-Adjunct64951.2024.00189

An XR GUI for Visualizing Messages in ECS Architectures

Ben Yang* Xichen He† Jace Li‡ Carmine Elvezio§ Steven K. Feiner¶

Department of Computer Science
Columbia University

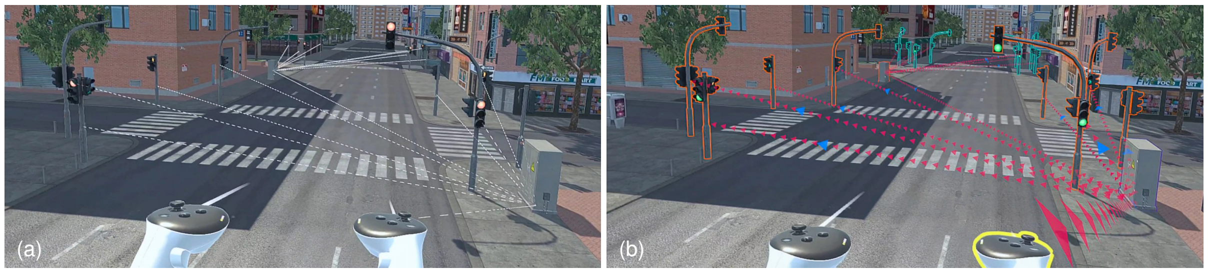

Figure 1: A VR scene in our XR GUI. (a) Thin dashed white lines link entities (e.g., VR controllers, traffic controller boxes, and traffic lights), illustrating their relationships in the Mercury network built on top of Unity. (b) Additional information is presented: entities are outlined in different colors based on their level of message propagation (yellow for the right VR controller, purple for the traffic controller box on the right, orange for one set of lights and the controller box back toward the left, and bluish green for the next set of lights) and linked by red arrowheads. Messages are shown as animated blue arrowheads traveling along links.

Abstract

Entity–Component–System (ECS) architectures are fundamental to many systems for developing extended reality (XR) applications. These applications often contain complex scenes and require intricately connected application logic to connect components together, making debugging and analysis difficult. Graph-based tools have been created to show actions in ECS-based scene hierarchies, but few address interactions that go beyond traditional hierarchical communication. To address this, we present an XR GUI for Mercury (a toolkit to handle cross-component ECS communication) that allows developers to view and edit relationships and interactions between scene entities in Mercury.

Index Terms: Visualization–Visualization systems and tools–Visualization toolkits; Human–computer interaction (HCI)–Interaction paradigms–Virtual reality.

1 Introduction

Game engines such as Unity and Unreal are commonly used to build extended reality (XR) applications with hierarchically arranged entities and components. Unity has experimented with 2D graph views to facilitate analysis and understanding of entities and relationships in these applications [16]. There are several open-source projects that help with code analysis (e.g., [10, 15]), but focus specifically on understanding event calls or animation flow, and not visualizing the scene content and its hierarchy directly.

With modern Entity–Component–System (ECS) architectures [2, 9], it is easy for events to be propagated vertically up and down the scene hierarchy by transmitting events through the same parent–child relationships defined by the scene hierarchy. However, it requires more work to configure horizontal communication between components (e.g., a message to one or more entities that doesn't come from a descendant or ancestor), which is common in XR scenes that have many modular UI components. Communication of this sort also requires more mental effort to visualize abstractly. To address these concerns, we are creating an XR GUI that allows developers to visualize these connections and see messages being sent between components in real time, both in XR and in a companion 2D editor view.

*e-mail: by2297@columbia.edu
†e-mail: xh2623@columbia.edu
‡e-mail: yl4862@columbia.edu
§e-mail: ce2236@columbia.edu
¶e-mail: feiner@cs.columbia.edu

2 Related Work

As much development of real-time interactive systems occurs in ECS-enabled environments, the hierarchical relationships between entities are clearly represented in the scene graph. However, alternative visualization methods [3] have been developed to facilitate understanding of the complex relationships that can form between components attached to entities, which are often non-hierarchical.

Message communication paradigms are well explored in the software engineering literature (e.g., [5]) and there has been much recent work in the visualization of inter-component communication as game engines have become more popular (e.g., [13]). While data-flow visualization has long been of interest to the software-engineering community [1], these visualizations often focus on highlighting how a particular variable or function is invoked across entities and components. However, this can be difficult when the relationships between entities and components become more complex and extend beyond a small set of variables and function invocations.

Shahin et al. [14] examine a variety of techniques for visualizing software architecture and find that graph-based tools are the most popular. But 3D applications often have their own considerations that make traditional 2D graph-based visualization approaches insufficient. For example, Merino et al. [12] visualize software in an XR headset as 3D urban scenes and found through a user study that doing so increases codebase comprehension. CodeHouse [8] represents the source code of a project as a house in an XR headset, with individual rooms depicting modules of the underlying code. The Reality Editor [7] is a graph-based XR app that allows end users to connect Internet-of-Things devices together without writing code and see the connections.

With the emergence of more complex communication paradigms for ECS-based systems [4], there is a need for new mechanisms to visualize the more abstract relationships between entities and components, specifically in the way messages pass between abstract relay constructs. In particular, XR real-time systems often have complex scenes whose relationships include ones that are independent of a traditional scene hierarchy, leaving developers to visualize much of the communication network offline. Consider a city scene in which a user can control traffic lights on a street, as shown in Figure 1: The relative positions of the user and the lights in the scene hierarchy are not relevant to controlling the lights. In that case, how should we display these connections to the developer? Building on related work, our demo addresses this by using a separate node-based visualization of entities and their relationships, and the messages passed between entities.

3 Implementation

We developed our system in Unity using Mercury [4, 11], a bidirectional messaging framework that standardizes horizontal communication between components that are not related as an ancestor and descendant. Our demo runs on a Meta Quest 3 headset using Meta Quest Link, driven by a computer with an Intel® Core™ i9-11900K and Nvidia GeForce RTX 4090. Our implementation includes a 3D XR visualization that shows messages propagating in an interactive scene and a 2D editor built with the NewGraph [6] package.

3.1 XR Message Visualization

Running a scene in Unity play mode that includes our 2D editor window will automatically invoke our system, allowing developers to see connections between Mercury entities in the scene. For example, Figure 1(a) visualizes the connections as thin dashed white lines. Additional information can also be shown. For example, Figure 1(b) shows red arrowheads on the spatial paths between entities with Mercury network connections and those entities are outlined in different colors based on their level of message propagation.

Further, developers can see messages being sent in real-time between Mercury entities, shown by animated blue arrowheads that move along the Mercury network paths in Figure 1(b). Mercury allows developers to filter these messages, potentially causing the messages to skip relay nodes with specific tags. Remembering the types of tags assigned to Mercury entities can be challenging. With our XR GUI, developers can better comprehend Mercury tag filtration of messages.

Changes to the Mercury network while in Unity play mode are saved to a log file. After the developer leaves Unity play mode, they can press an editor button to apply the changes made to the Mercury network in play mode, allowing the developer to debug these changes in real-time before considering whether or not to apply them afterward.

3.2 2D Graph-Based Editor Window

The 2D graph-based editor window visualizes the network of Mercury message relay nodes in a scene as a graph. In Mercury, communication between nodes is achieved by using a routing table of input or output nodes. We treat this table as an adjacency list to visualize relationships between entities. Each relay node is represented as a vertex in the graph, and its immediate connections to other Mercury entities are represented as edges. The output of a node is always connected to the input port of another node. The representation of a node is highly customizable, since any serializable variable can be shown in a Mercury graph node.

Updates to the routing table in the inspector can be displayed in the GUI upon rebuilding the editor graph, resulting in the appropriate connections being drawn. This is triggered automatically by listening to the built-in events of the Unity editor, which are fired when the scene hierarchy is changed or any code assemblies are recompiled. Advanced developers can improve performance by disabling the automatic rebuild feature, using the “Refresh” button to do so manually only when necessary, or customizing this functionality by calling `RenderGraph()` themselves in other scripts.

4 Conclusions and Future Work

We presented an XR UI for visualizing and editing bidirectional networks separate from the scene hierarchy. Moving forward, we will expand our system to allow for more comprehensive editing of relay networks in XR, which will eventually support full scene and message composition. We intend to release it as an open-source package.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants IIS-1514429 and CMMI-2037101.

References

[1] G. Abram and L. Treinish. An extended data-flow architecture for data analysis and visualization. *ACM SIGGRAPH Computer Graphics*, 29(2):17–21, 1995. doi: 10.1145/204362.204366 1

[2] S. Bilas. A Data-Driven Game Object System. https://www.gamedevs.org/uploads/data-driven-game-object-system.pdf, 2002. [Accessed 05-08-2024]. 1

[3] E. Chu and L. Zaman. Exploring alternatives with Unreal engine's blueprints visual scripting system. *Entertainment Comp.*, 36:100388, 2021. doi: 10.1016/j.entcom.2020.100388 1

[4] C. Elvezio, M. Sukan, and S. Feiner. Mercury: A messaging framework for modular UI components. In *Proc. CHI 2018*, pp. 1–12. ACM, New York, NY, USA, 2018. doi: doi.org/10.1145/3173574.3174162 1, 2

[5] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G. Booch. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Reading, Mass, 1st ed., Nov. 1994. 1

[6] Gentlymad Studios. NewGraph. https://github.com/Gentlymad-Studios/NewGraph. [Accessed 30-07-2024]. 2

[7] V. Heun, S. Kasahara, and P. Maes. Smarter objects: Using AR technology to program physical objects and their interactions. In *Proc. CHI '13 Ext. Abstracts*, p. 961–966. ACM, New York, NY, USA, 2013. doi: 10.1145/2468356.2468528 1

[8] A. Hori, M. Kawakami, and M. Ichii. CodeHouse: VR code visualization tool. In *2019 Working Conf. on Sfw. Vis (VISSOFT).*, pp. 83–87, 2019. doi: 10.1109/VISSOFT.2019.00018 1

[9] A. Martin. Entity Systems are the future of MMOG development. https://new.t-machine.org/index.php/category/entity-systems/, 2007. [Accessed 05-08-2024]. 1

[10] L. Mefisto. UnityEventVisualizer. https://github.com/MephestoKhaan/UnityEventVisualizer. [Accessed 24-07-2024]. 1

[11] MercuryMessaging. https://github.com/ColumbiaCGUI/MercuryMessaging. [Accessed 26-08-2024]. 2

[12] L. Merino, A. Bergel, and O. Nierstrasz. Overcoming issues of 3D software visualization through immersive AR. In *2018 IEEE Working Conf. on Sfw. Vis. (VISSOFT)*, pp. 54–64, 2018. doi: 10.1109/VISSOFT.2018.00014 1

[13] B. Sewell. *Blueprints visual scripting for Unreal engine*. Packt Publishing Ltd, 2015. doi: 10.1007/978-1-4842-6396-9 1

[14] M. Shahin, P. Liang, and M. A. Babar. A systematic review of software architecture visualization techniques. *J. of Sys. and Sfw.*, 94:161–185, Aug. 2014. doi: 10.1016/j.jss.2014.03.071 1

[15] SolarianZ. UnityPlayableGraphMonitorTool. https://github.com/SolarianZ/UnityPlayableGraphMonitorTool. [Accessed 24-07-2024]. 1

[16] Unity. Graph View Reference. https://docs.unity3d.com/ScriptReference/Experimental.GraphView.GraphView.html. [Accessed 24-07-2024]. 1