Extending the Open Source Social Virtual Reality Ecosystem to the Browser in Ubiq

Sebastian Friston
Ben J. Congdon
sebastian.friston@ucl.ac.uk
ben.congdon.11@ucl.ac.uk
University College London
London, UK

Nels Numan
Klara Brandstätter
Lisa Izzouzi
nels.numan.20@ucl.ac.uk
k.brandstatter@ucl.ac.uk
l.izzouzi@ucl.ac.uk
University College London
London, UK

Felix J. Thiel
Jingyi Zhang
Daniele Giunchi
felix.thiel.18@ucl.ac.uk
jy.zhang@ucl.ac.uk
d.giunchi@ucl.ac.uk
University College London
London, UK

David Swapp
Anthony Steed
d.swapp@ucl.ac.uk
a.steed@ucl.ac.uk
University College London
London, UK

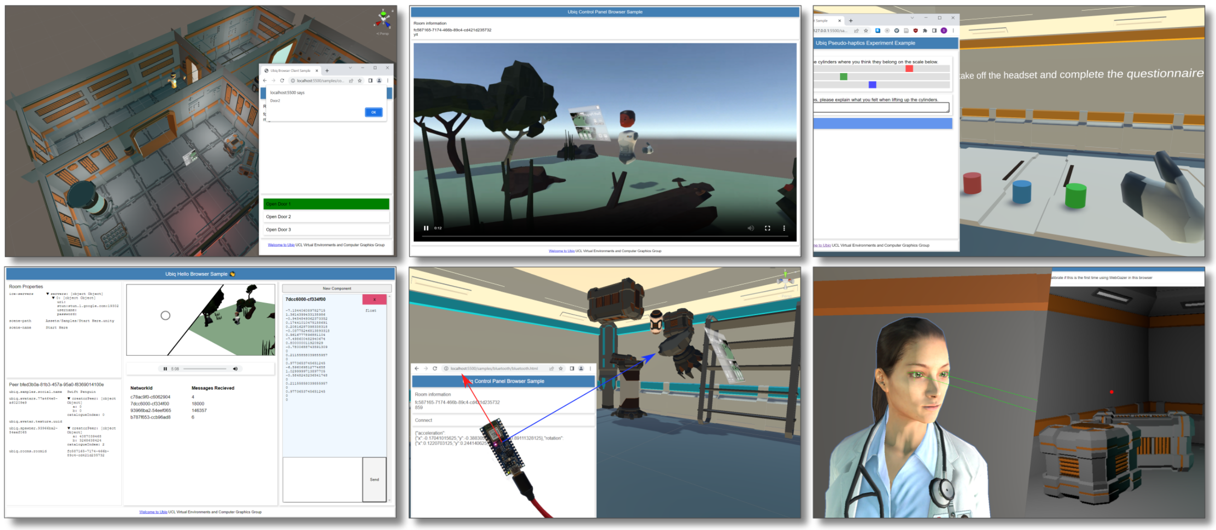

Figure 1: Six multi-modal proof of concepts of extending a social virtual reality system with web interoperability (from left to right, top row first): remote environment control, remote video streaming from a user view, directly triggering a desktop questionnaire for a user, online diagnostics and instrumentation, novel device support and eye tracking to control a character. Additional images are available in the supplementary materials.

ABSTRACT

Social VR (SVR) systems are VR systems with a common subset of features facilitating unstructured social interaction. In the real world, social situations have many purposes, each with a different set of requirements, and roles its participants take - creator, moderator, performer, visitor, etc. Yet, common SVR systems typically offer only a single client to users. Even if there are versions for different platforms, there is a one-size-fits-all approach to the user experience. Consequently users need to employ workarounds or build their own functionality to support specific roles, where this is possible at all. We argue that platforms need to develop more open frameworks that support different processes and user interactions. One way to do this is through using appropriate web standards and an open messaging system in order to allow distributed clients that can leverage the strongest features of heterogeneous computing platforms. Supporting asymmetrical capabilities greatly increases the scope of supported virtual social interactions and potential use cases of SVR. We take a qualitative experimental approach to exploring cross platform support in this way, from a designers perspective. We use the open-source SDK Ubiq, and create a library that allows building Ubiq Peers using web standards and thus clients that can operate solely in a web browser or certain Javascript environments. We validate our approach by demonstrating six proof of concept demonstrators that would be difficult or impossible to achieve in most other SVR systems, and report on what we encountered for the benefit of other SVR designers.

Web3D '23, October 09–11, 2023, San Sebastian, Spain
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0324-9/23/10.
https://doi.org/10.1145/3611314.3615903

CCS CONCEPTS

• **Human-centered computing** → **Interaction paradigms**; • **Networks** → *Application layer protocols*; • **General and reference** → **Cross-computing tools and techniques**.

KEYWORDS

social-vr, interoperability, toolkits

ACM Reference Format:
Sebastian Friston, Ben J. Congdon, Nels Numan, Klara Brandstätter, Lisa Izzouzi, Felix J. Thiel, Jingyi Zhang, Daniele Giunchi, David Swapp, and Anthony Steed. 2023. Extending the Open Source Social Virtual Reality Ecosystem to the Browser in Ubiq. In *The 28th International ACM Conference on 3D Web Technology (Web3D '23), October 09–11, 2023, San Sebastian, Spain.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3611314.3615903

1 INTRODUCTION

Social VR (SVR) systems are VR systems with a common subset of features facilitating unstructured social interaction. SVR has been imagined in fiction for decades, and the first SVR systems were constructed only shortly after the first VR systems [Singhal and Zyda 1999]. This is in part because VR system requirements such as the ability to capture and embody a person in a digital space lend themselves well to computer-mediated communication - once a person is embodied digitally, it is a short step to transmit this representation to others.

Very few of the many contemporary SVR platforms [Schulz 2020] currently support cross-platform play other that they may support a similar bespoke client on multiple platforms (e.g. across variants on Android for Meta's devices, Pico's devices, etc. or Windows variants). Thus while they may support both desktop-based and mobile-based headsets, they use the same programming environment, and almost certainly the same engine. Even Meta, whose stated aim is to create an omnipresent metaverse, which implies cross-platform interop [Havele et al. 2022], supports only the latest headsets.

This is not surprising, as cross-platform play has many challenges. Apart from low-level considerations such as protocol support, serialisation and endianness, some concepts such as scene graphs or dynamic typing might exist on one platform without an analogue on another. Cross-platform support is essential to increase user bases and take advantage of network effects. Platforms such as Roblox now support cross-browser and VR play [Sammy 2022].

There is yet another level though. Re-creating the same experience through different mediums supports an important use case. However, where asymmetry is supported, it opens up a whole new set of possibilities. The strengths of each platform can be leveraged to maximise the functionality of the system-as-a-whole, while minimising development time by making the best tool for the job available. The value of this becomes apparent when considering different types of *users*. In SVR, different participants take on different roles: attendee, content-creator, moderator, performer, maintainer, and many more. Each role has specific interface requirements, but few SVR systems consider this.

We suggest that SVR systems need to have many more modalities than simply embodied interaction to succeed in the general use cases they are targeting. In this paper, we discuss the role of asymmetric cross-platform support, specifically interaction with the web. By exposing events and streams that underpin the SVR over web standards, an SVR that is primarily experienced through a native client can interact with standard web technologies.

Such systems are very rare however. Mozilla Hubs [Mozilla Corporation 2021], for example, is the only SVR based entirely on open standards, but even its client is still symmetrical.

To explore this topic, we describe how we extended the open-source SVR system Ubiq to support such inter-operation. Ubiq is a toolkit for SVR research, teaching and development. It was primarily written as a Unity package that allows developers to quickly develop complex immersive SVR applications. We developed a set of components in order to create Javascript-based Ubiq Peers, with a specific focus on peers that would work in web browsers.

We discuss Ubiq's use case and operation, the development of this new functionality, and how we designed a set of demonstrators to validate its implementation and show what can be done with it. Our aim is both to argue for the benefits of building SVR systems from heterogeneous clients this way, and to distil what we have learnt for other practitioners who wish to do so.

2 PREVIOUS WORK

2.1 Commercial Systems & Use Cases

While many commercial SVR systems are nominally symmetrical, the need for different modalities shows up in a number of ways. *The Under Presents*, an immersive VR theatre experience, provides actors with a virtual dressing room containing non-diegetic 2D menus in 3D. These allow actors to give themselves abilities, such as teleporting attendees, to aid their performances [Coulombe 2020]. Producers of *Out Of This World* [Greenhalgh et al. 1999], a VR-based game show, combined dedicated virtual camera controls with real-world analogue mixers to live-broadcast television from inside an SVR. *Desert Rain* [Koleva et al. 2001] employed a variety of physical control systems, in addition to functional props such as swipe cards, to orchestrate a mixed reality theatre experience where the mixing of the virtual and real was used for dramatic effect. In the VR game *Keep Talking and Nobody Explodes*, asymmetry is embraced as a mechanic. Some players use real-world resources to help an immersed user complete a task [Steel Crate Games 2023]. Saffo et al [Saffo et al. 2021] replicated lab-based VR studies in VRChat. They noted how VRChat's graphical programming language allowed them to re-create experiment logic, but a major limitation was the inability to communicate data outside of the client. Vitillo used VRChat to stage a virtual concert [Vitillo 2021]. They remarked on its strong abilities in virtual staging, but severe limitations in crowd sizes, and in controls to enable synchronisation or handle disruptive users. Slater et al [Slater et al. 2000] created a bespoke facial expression controller based on mouse-gestures for virtual rehearsal in SVR. In RecRoom [Brown 2021], users can re-skin objects with pre-existing functionality. The client implements a number of games, such as Charades, in which users have different roles with specific abilities. In other situations, users can unlock abilities through 'room keys'. Virbella adopts a traditional user-admin-owner model where users are assigned a fixed role [Virbela 2022]. Data exchange remains a consideration; for example, organisers of a virtual innovation event were unable to marry-up Virbela attendees with ticket holders managed on another platform. The authors also noted concerns about privacy - not because of the way the VR world operated, but simply due to it being a digital medium with opaque data flows [Jauhiainen 2021]. It is interesting to contrast RecRoom's metaphor, where authority is granted by being in a place or holding an object, with *The Under Presents* use of a dedicated non-diegetic interface, and Virbela's, of signing in with different accounts. Browser support is rare, though not non-existent ([Sammy 2022]) for games, but more common in productivity tools. For example, Spatial [Spatial 2023] and Glue [Glue 2023] have browser clients. Other systems (e.g. Engage [Engage 2023]) attempt as broad cross-platform support as possible, but do not include the web browser.

2.2 Toolkits

VR systems, and especially SVR systems, share a consistent subset of features, leading to a number of toolkits at different levels. These began with APIs aimed at supporting complete distributed graphics applications, such as Blue-c [Naef et al. 2003] and Wolverine [Chardavoine et al. 2005], on which VR functionality could be added. Systems such as SCIVE [Latoschik et al. 2006] and FlowVR [Allard et al. 2004] aimed to allow building distributed VR systems as an arrangement of heterogenous nodes, and defined protocols through which new components could be added. These contrast with MASSIVE [Greenhalgh and Benford 1995] and DIVE [Carlsson and Hagsand 1993], which aimed to provide an already functional SVR architecture on which users could build specific applications. Nowadays, most VR applications are built using engines such as Unity or Unreal. Unreal has a built-in networking model. For Unity there are a number of networking SDKs (some of which are also available to Unreal developers who need specific features) including Photon [Exit Games 2021], DarkRift [Dark Rift 2021], MLAPI [Unity Technologies 2021] and Mirror [Mirror 2021]. There are also many general purpose VR SDKs, such as MRTK [Microsoft 2022], which we leave to various application specific surveys (e.g. [Wolfel et al. 2021]).

One of the most interesting tools to consider is Ready Player Me (RPM) [Altundas and Karaarslan 2023]. Users design avatars in browsers which can then be loaded into a variety of SVR platforms. As far as we are aware RPM is the only example of the Feature-as-a-Service model applied to SVR. This model could play a significant role in the future of a federated metaverse.

2.3 Standards

For the web, the array of standards is now deep enough to support a federation of complete 3D XR worlds running entirely in the browser. These include XR device interaction (WebXR) as well as interactive 3D scene descriptions (X3DOM) and interchange of video (MP4), images (JPEG) and 3D models (glTF) [Havele et al. 2022]. Mozilla Hubs is a proof-of-concept SVR application built on these technologies [Mozilla Corporation 2021].

For distributed computing, the Virtual Reality Peripheral Network (VRPN) provides a device and network independent standard for combining VR peripherals [Ii et al. 2001]. A more recent example is the Robot Operating System (ROS) [Stanford Artificial Intelligence Laboratory et al. 2018], a set of standards and tools for building graphs of message passing components. ROS is designed for robotics but could build any distributed computing system.

On the web, the W3C maintains standards such as Simple Object Access Protocol (SOAP) for arbitrary object passing [W3C 2023a]. WebSockets and HTTP, for transport, are also standardised. The existence of non-standard, if open, solutions such as Socket.io suggests there may still be a standards gap at the messaging level however [SocketIO 2023]. The web also presents security standards [W3C 2023b]. It is particularly interesting to consider this, because the approach - based on sandboxing untrusted code - is different to that of many VR operating systems, which use signing to execute trusted code.

2.4 Summary

There is a broad set of mature open standards at the many layers involved in building SVR. There is also a long history of research into system architecture. However, almost all contemporary social VR applications offer only closed clients. These clients, if available for more than one platform, aim for parity between platforms, rather than embrace the differences between them. Even those built using open standards are designed around symmetrical experiences.

As we have seen however, this closed nature and enforced symmetry can be an impediment to a variety of real and desirable use cases. Users have to work around the design of the client, where this is even possible.

3 MOTIVATION & METHODOLOGY

We believe the potential of SVR systems can be greatly enhanced by opening them up to platforms other than embodied VR. Just as in the real-world, building a functioning social space involves many tasks, and not all are equally suited to being embodied in a contemporary HMD. This is clearly imagined by anyone who has tried to input text using a VR keyboard. As we have seen, when given the choice, producers use a mix of clients, and usually a mix of virtual and real control systems. Where teams are limited to a single client, they often have to build their own interfaces - but depending on the client these can be cumbersome to build and/or use. Teams need the ability to see what is going on at multiple levels, and to quickly manipulate the environment or other users. These interfaces are best built from a combination of virtual and real components, with the virtual ones using whatever framework is most productive. Extracting data from systems is also important. This is especially so for researchers, but we also see commercial examples, such as the need to integrate ticketing systems.

Simply using open standards exclusively is not enough though. If clients assume symmetrical capabilities, the burden of supporting a specific platform becomes greater with each feature, and some will be excluded entirely. Systems must be open at the messaging layer. Requiring only a subset of functionality to be shared allows more clients to contribute to the state of the SVR, even if each can only represent and communicate with a part of it.

Symmetrical cross-platform interoperability is already challenging however. Examples of symmetrical SVR systems are few, and asymmetrical examples fewer still. To explore this topic, we take a qualitative experimental approach, attempting to solve problems in an open system. Our aim is to demonstrate the validity of our

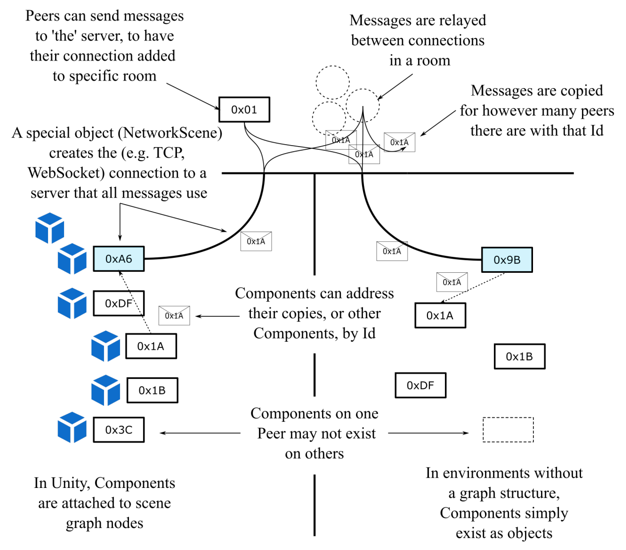
Peers can send messages to 'the' server, to have their connection added to specific room
Messages are relayed between connections in a room
Messages are copied for however many peers there are with that Id
A special object (NetworkScene) creates the (e.g. TCP, WebSocket) connection to a server that all messages use
0x01
0xA6
0x9B
0xDF
0x1A
0x1B
0x3C
Components can address their copies, or other Components, by Id
Components on one Peer may not exist on others
In Unity, Components are attached to scene graph nodes
In environments without a graph structure, Components simply exist as objects

Figure 2: Diagram showing two Ubiq Peers, each with 3-4 Components, exchanging a message via the server.

argument, and also to distil lessons from our attempts for other SVR designers. We implement heterogenous cross-platform support for the open-source SDK Ubiq, extending the social fabric to Javascript environments, especially the browser.

The modern web browser is one of the most capable computing platforms. Browsers are ubiquitous and robust. They excel at constructing rich, interactive 2D interfaces. Though less commonly used, they also support 3D, including immersive 3D. They have considerable device capabilities, including for USB and Bluetooth. They have the ability to do multimedia processing, including node-based computation, such as with WebAudio. Further, this is all made available in a secure sandbox that can run anywhere. By supporting the browser, SVR designers can leverage these abilities to build new modalities. Primarily this is oriented around building 2D interfaces for multiple types of user, however we also consider the browser as a data source, and a co-processor.

4 UBIQ

Ubiq is an open-source SVR toolkit for research and teaching [Friston et al. 2021]. Ubiq's aim is to address features that traditional commercial platforms do not. These include the ability for users to run their own server, to have full source code access, to extract metrics and data, and to run arbitrary code on users' machines. In this section, we review some concepts that are important for understanding or relevant to cross-platform interop. Figure 2 illustrates message passing between two Ubiq Peers, and is explained in detail at the end of this section.

4.1 Messaging

A Ubiq session consists of a number of Peers. Each Peer has one or more networked Components - objects that can send or receive messages. Components are responsible for deciding when to transmit, performing serialisation and deserialization, and acting upon the contents of the message to drive the larger application. The networked behaviour is distributed among various Components, following a composition-based programming model. Messages are binary blobs, but usually contain JSON. Some messages, such as avatar transforms, use binary encoding to avoid serialisation overheads. This is up to the developer.

4.2 Addressing

Components send messages to each other across Ubiq's messaging layer. Components have Ids, and Messages are addressed to an Id. Ubiq uses application layer multicasting, so all Components with the same Id receive messages. Ids are 64-bit. Some Ids are reserved, but otherwise are generated by each Peer as required. All Peers have a 'root' *NetworkScene* Component with an Id unique for that Peer, for example. In Figure 2, the two NetworkScene Components have generated Ids of 0xA6 and 0x9B, while the server has a reserved Id of 0x01. A common design pattern is for Peers to compute Ids deterministically in order for Components to communicate without having to synchronise Ids explicitly. This is often done by using an already synchronised Id as a namespace. For example, an Id reserved for a service can be hashed with a Peer's NetworkScene's Id, to uniquely identify the instance of that service on that Peer.

4.3 Server and Rooms

A Peer Group is a set of Peers that exchange messages. Peers can be connected in a literal peer-to-peer architecture or via Ubiq's rendezvous and relay server. (The messaging layer and application layer multicast work identically in any case.) The server is written in NodeJs and allows Peers to join a Peer Group (a 'room') through the use of join codes or UUIDs. Join codes and UUIDs are shared out-of-band. Rooms with a given UUID are created on demand, allowing UUIDs to be baked into applications so they will automatically join the same room on start-up, or computed deterministically based on some other criteria.

4.4 Cross Platform Use

Peers are simply any process that can send and receive Ubiq messages. There are no requirements that a given Peer is embodied, for example. This allows very minimal Peers to be created, and functionality to be federated between Peers dedicated to particular services. Ubiq-Genie [Numan et al. 2023] exploited this feature, creating Peers to provide an interface to different GANs or LLMs for synthesising voice, text and images in response to user statements. Ubiq Peers have been implemented in C# & Javascript (Js), in multiple platforms that support these languages. In theory, Ubiq could be supported in any language that supports its underlying network protocols.

A diagram of Ubiq message passing between two platforms is shown in Figure 2. In this diagram, two Peers are connected via a server. Each Component is shown with its address. The addresses shown in the diagram are valid, but would be more complex in real world use. The Peers have already negotiated membership to the same room. They do this by sending messages to the Server Component (at address 0x01) to start and stop forwarding between sets of connections. Component 0x1A is sending a message to its counterparts on other Peers in the room. We see how the message is passed from the Component to the NetworkScene (highlighted in

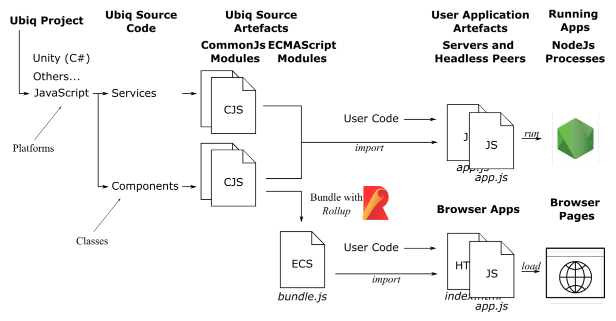

Figure 3: Diagram showing the project structure and the dependencies and processing steps for the code and artefacts.

blue; 0xA6 for the Unity Peer), which then passes it via TCP to the server, which forwards it to all other members of the room (multicasting). In the browser Peer, the NetworkScene (0x9B) receives the message (via a WebSocket) and forwards it to the Component with a matching Id.

5 BROWSER PEER

In this section, we design a Js library that allows browser pages to join Peer Groups, making the strengths of the browser and other Js contexts available to Ubiq users.

5.1 Architecture

The library is provided as an Ecmascript (ES) module that can be imported into another module (e.g. in NodeJs) or loaded into a page. The development practice is to write all code as CommonJS modules that can be utilised by NodeJs, so any class could be used in a page's main thread, a WebWorker, or in a headless Node process. Rollup[1] is used to package the classes into a single ES library, insert polyfills to hide differences between Node and the browser. This is illustrated in Figure 3.

No platform-specific conditionals are required in the code, though sometimes the use of an API must be carefully considered so that it has the same behaviour on Node and in the browser. For example, the 'binaryType' member of 'WebSocket' must be explicitly set to "arraybuffer", which has no effect in Node, but is necessary to get the right type in the browser. 'Buffer' must be explicitly imported, even though it is implicitly available in Node.

5.2 Protocol

The first thing a Peer does is make a connection to a Peer Group. Android and PC clients use TCP. The browser library uses WebSockets. WebSocket Server support was added to the server. As messages are received, their binary payload is interpreted as a Ubiq message and inserted into the Peer Group via the same method calls the TCP connections use. In Js, incoming messages are wrapped in an object. Their address is read through binary operations on the payload ArrayBuffer. The wrapper provides helper methods to interpret the Ubiq payload as strings or JSON objects, mirroring the C# API. For binary messages, Js users perform deserialisation themselves in their Components (e.g. using DataViews).

[1]https://rollupjs.org/

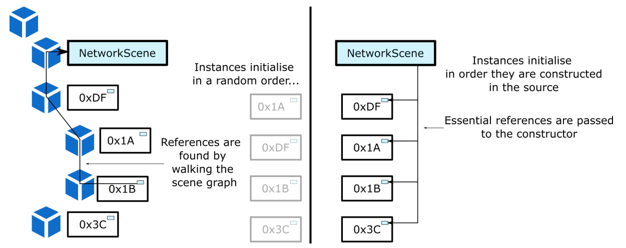
NetworkScene
0xDF
0x1A
0x1B
0x3C
Instances initialise in a random order...
References are found by walking the scene graph
0x1A
0xDF
0x1B
0x3C
NetworkScene
Instances initialise in order they are constructed in the source
Essential references are passed to the constructor
0xDF
0x1A
0x1B
0x3C

Figure 4: Diagram showing the project structure and the dependencies and processing steps for the code and artefacts.

5.3 Object Graph

In Unity, all Components are attached to scene graph nodes. Components use this graph to infer relationships based on their positions relative to other Components. Components could find the (geodetically) closest instance of a specific class to generate deterministic Ids, for example. Ubiq uses this mechanism to have Components associate with their closest NetworkScene, without having to maintain explicit references, aiding modularity in Unity.

Js does not have an inherent scene graph, and we do not assume users wish to create 3D applications. As such as Js Components take in their NetworkScene as a constructor parameter, ensuring that it is unambiguous, and never null. Components can still find other components registered to the NetworkScene by class name, by enumerating all registered objects by the constructor name. There is still an object graph, but it is typically much flatter than in Unity.

These different approaches make the most of each platform: in Unity, the execution order is hidden and cannot be controlled, so relationships are defined by locations in the object graph. In Js, initialisation is sequential and relationships are closer to the explicit ordering (Figure 4).

5.4 Ids & Hashing

Ubiq encourages generating Ids deterministically to reduce explicit synchronisation and make development simpler. This is done by hashing one or more variables such as strings, or other Ids, together into a 64-bit result that is used as the address. One challenge is that string hashing implementations can differ by platform, so all string hashing in Ubiq is explicitly declared to use MD5 hashes of UTF-8 encodings. MD5 is simple enough that it can be re-implemented relatively easily. It is also reasonably fast, as hashes do not have to be cryptographically secure. A challenge of hashing in Js is that many functions take advantage of integer overflow behaviour to maximise diffusion, however such bitwise overflow does not occur when using many of the Js math operators [Darpinian 2022]. We therefore design hash functions around the Math.imul function, which has the same overflow behaviour as the C operation. (Another possibility would be to perform other operations with bitwise operators, or to use WebAssembly.) Similarly, while Js has the concept of BigInt, it does not represent a 64-bit number in the same way as a long in C/C++/C#. We therefore store NetworkIds as two Js Numbers, which can be considered 32-bit integers for the purpose of hashing.

The 64-bit Ids do not have to be globally unique, but do need to be well distributed enough to avoid collisions within the Peer Group [Jesus et al. 2006]. Browsers try to minimise the amount of reliable system information available in order to reduce the potential for fingerprinting [Laperdrix et al. 2020]. Therefore, we rely on a function that is a combination of the performance counter and Math.random to generate 'unique' Ids. A similar approach is used to create v4 GUIDs, though we may soon be able to transition to using the crypto namespace [contributors 2023] as all connections are secure by default in the latest implementation of Ubiq for the browser.

5.5 Design-Time Ids

A feature of Ubiq's server is that any Peer can create a room with a specific GUID on-demand, allowing GUIDs to be used as pre-shared secrets for automated rendezvous. This presents a challenge for documentation and sample production however. Ideally this would be demonstrated by having all samples connect to the same room automatically, but if a GUID were baked into the source all users would end up in each other's samples. Consequently, Ubiq is designed so that on a new checkout the 'sample' GUID is initialised on first-use, and used from then on.

This is a major challenge for the browser because it does not have access to the host's filesystem, so is unable to generate and initialise a GUID itself. At best it is able to detect if the GUID is uninitialised, and report this to the user or join a different room. This is not ideal and breaks the status of the browser Peer as a first-class Peer, but we are currently unaware of a supported way to break the sandbox.

5.6 Audio Communication

Ubiq uses WebRTC (RFC 8825) [Kleinhout 2021] to facilitate peer-to-peer audio. WebRTC stacks create their own connections, ideally RTP over UDP, or through TURN servers over TCP as a fallback. To instruct two WebRTC stacks to establish a connection, the host configures the media stream(s) (such as an audio from a microphone) and pass signalling messages between them. Despite being a standard, WebRTC implementations on different platforms advance at different rates, and divergence and bugs can result in failure to establish connections. In Ubiq, a dedicated Component manages signalling for a single WebRTC PeerConnection. Each pair in a Group has a dedicated PeerConnection, and so pair of Components with a shared Id used to exchange its signalling messages. Ubiq supports four WebRTC stacks: a native .NET implementation, a Unity package based on Chromium, a Microsoft package also based on Chromium, and the browser's implementation. The Ubiq Component has its own protocol which can identify which back-ends in use and make adjustments to the offer and answer exchange ordering to ensure the stacks inter-operate correctly, based on what we have identified works empirically[2].

6 USE CASES AND EXAMPLES

In this section, we demonstrate various things that can be done with browser Peers. The demonstrations are based on some of the challenging use cases discussed in Section 2. Specifically, the need for performers and moderators to have abstract views and dedicated controls over environment (e.g. *The Under Presents*, Vitillo). For producers to bind novel hardware (e.g. Desert Rain). For developers to inject or extract data (e.g Jauhiainen), and build asymmetric interfaces (e.g. *Keep Talking and Nobody Explodes*). Browser peers run entirely locally from static resources (HTML, CSS & Js), and simply require loading the web page.

6.1 Wizard-of-Oz Control Panels

In this example (Figure 1, Top-Left), an immersed participant navigates through a set of doors. At each they interact with a button, notifying an experimenter via a browser-based control panel. The experimenter can use the panel to open doors when they are ready for the participant to proceed. Traditionally this would be achieved by having a keyboard or other device attached to same device as the participant. Not only is this becoming more difficult on standalone devices such as the Quest, but a web page is easier to write and extend. In a symmetric client, developers would have to build in-game control systems, which is not only tedious, but possibly also challenging to do while also preventing 'regular' participants gaining access to it also.

6.2 Streaming Immersive Views

Spectating can be an important aspect of VR. An experimenter may wish to see a view of the world, or a view might be generated for an audience. VR devices such as the Quest can stream to companion apps, and SteamVR supports showing a VR view on a monitor on the HMD PC. These however are only accessible in one place, and have limited configurability. Using WebRTC, Ubiq can stream arbitrary views from inside an embodied Peer to the browser. In this sample (Figure 1, Top-Middle), the browser Peer creates a virtual camera in a specific Peer. This view is platform agnostic so works on any device and can be streamed anywhere. The user can configure it, including creating entirely new viewpoints, at runtime. Multiple streams could be combined in one page to build a control room type view.

6.3 User Interfaces

VR experiments commonly ask participants to complete questionnaires. However, both building and answering questionnaires in 3D can be tiresome. While separate solutions such as Google Forms are available, they require records to be married up post-hoc, a potential source of error, and the forms cannot react in real-time. This example (Figure 1, Top-Right) re-creates a simple pseudo-haptics experiment. A web browser attaches to the same session as a VR headset. When appropriate, the experiment state machine prompts the user to remove the headset and answer questions displayed in the web page. The VR process and browser communicate so that users must proceed in the right order. Combined with Ubiq's distributed logging, records from both sources end up in the same place automatically.

6.4 Diagnostics and Projections

This sample (Figure 1 Bottom-Left) demonstrates instrumentation of an SVR session. The browser can create widgets to listen on arbitrary addresses and decode binary messages from user-supplied

[2]Interested readers may refer to the comments in the WebRTC.jslib plugin at https://github.com/UCL-VR/ubiq/blob/master/Unity/Assets/Runtime/Voip/Implementations/Web/Plugins/WebRTC.jslib

formats, all alterable at runtime. The widgets can also send messages. These controls can be used, for example, to detect if user Components are transmitting incorrect data, or not transmitting at all. Users can also use different approaches to visualise the virtual world. In one way, streams of avatar updates used to drive a 2D 'projection' of the world into a WebGL canvas. In another, FBXs are loaded into a Three.js[3] scene to show a locally rendered birds-eye view in 3D. As a demonstration of flexibility, opening two pages and giving a new Network Id to two widgets allows users to create a simple chat tool, without any functionality in Ubiq or the sample dedicated to this.

6.5 Bluetooth Interaction

A recent addition to browsers is support for pairing Bluetooth devices. In this example (Figure 1, Bottom-Middle), we program a BLE (Bluetooth Low Energy) enabled Arduino[4] to send measurements from its Inertial Measurement Unit (IMU). This is paired in the browser, and the IMU data is forwarded to a virtual world, where it is used to control the gravity vector of the virtual environment's physics simulation.

6.6 Eye Tracking

The browser also presents possibilities for computation. In this example (Figure 1, Bottom-Right), we use the WebGazer [Papoutsaki et al. 2016] library to perform eye tracking using a webcam, while a browser user interacts by using a 2D video-game style interface. The browser user puppeteers a RocketBox [Gonzalez-Franco et al. 2020] avatar with movable eyes. Eye direction is controlled by projecting the user's gaze into the camera frustum as if it were placed at the avatar's head. This allows immersed users to perceive the browser user's gaze. The 3D world in the browser is rendered using Three.js.

The vast majority of SVR systems have no device support outside of the VR interaction devices inherently supported by the VR SDKs used to build them. Thus getting device data into such clients would usually involve emulating other interaction devices such as mice or keyboards.

7 DISCUSSION

Many SVR use cases have requirements that cannot be achieved by a single client or single set of embodied metaphors. For example, producers may want a disembodied or birds-eye view of the experience, and the ability to make quick changes. Researchers may want to extract data or observe their participants. When building control systems, flexibility in interface design is key to maximising efficiency. In many cases there are standard designs (e.g. lighting desks and analogue mixers) which are known and preferred for some tasks. Trying to build interfaces in a one-size-fits-all client may be possible - for some situations - but rarely will they be optimal.

We argue that modalities are not simple independent features. Instead, we consider capabilities (real or digital), social roles, and social situations to be highly inter-dependant. Different capabilities and interfaces have affordances for different roles, and social situations are defined by how roles mix. Further, how a capability is gained - e.g. by having it associated with account, by being in a place, or by having possession of a physical device - itself has information in how the social situation is structured and expected to play out. The ability to compose capabilities and platforms in different ways is important then for exploring the full range of potential virtual social experiences.

Additionally, apart from interfaces, opening up systems allows interaction with autonomous processes. This gives designers much more flexibility in how to achieve functionality in mixed-reality systems. Examples include enabling interaction with physical props, performing tasks such as eye-tracking with devices dedicated to this purpose, or leveraging non-portable stacks such as LLMs.

Across a number of demonstrators we show how the browser can be leveraged to quickly build efficient and maintainable user interfaces for different roles. We also show new possibilities for visualisation through streaming local renders, and integration of hardware devices. Our demonstrations aim to show a wide breadth, however the interop required of the developer is relatively simple - consisting primarily of sending serialised JSON. This though can be deceptive. It doesn't mean that interop is restricted only to common transport & serialisation. Many transport and serialisation protocols have extensive cross-platform support (e.g. WebSockets & MsgPack). While this support is necessary, a working system must also have a common addressing and identification system, security model, and conventions. It is these that we consider in this paper. We see for example the challenge of designing APIs that can work with multiple metaphors and graph structures, and the risk of relying on functions and types that may not be available on all platforms. It is achievable, but requires care and is not something that can be added in at a later date without engagement from the SVR maintainers. While we have experimented in Ubiq, SVR systems are defined as such by a common subset of functionality. The challenges and solutions described we would expect to apply to many similar systems as they will undoubtedly share the same goals and metaphors, and perhaps even many of the same software components and protocols.

Another consideration is portable representations vs open messaging. Standards such as X3DOM offer portable representations of functional worlds. Unity, supporting cross-compilation, arguably does something similar. Approaching multi-modal use by using portable representations and open messaging are quite different. Standardised representations require all platforms to support the full range of functionality, whereas with an open messaging layer, only subsets of each system need to have a mutual understanding. This is a design consideration rather than a technical one; an X3D scene could use any number of technologies to communicate with say, a Node process, for example. However, the ability to build a system from clients that only partially overlap is an important part of leveraging platform heterogeneity, and therefore an important consideration.

One of the arguments for closing a system is security. Ubiq operates in a high-trust environment not available to game developers, for example, due to concerns such as cheating. However here the web is especially amenable because its security model is built around running arbitrary code in sandboxes associated with a trusted origin. It could be imagined that this security model could be leveraged to create an origin associated within a particular group

[3]https://threejs.org/
[4]An Arduino Nano 33 BLE: https://store.arduino.cc/products/arduino-nano-33-ble

concept on a given platform, and this could allow JavaScript in the browser to be safely executed, isolated from other domains.

8 CONCLUSION

Asymmetric cross-platform support that can leverage the heterogeneous parts of different computing platforms is rare in contemporary SVR. We argue, however, that this is a mistake, and that the full potential of SVR systems will only be reached when systems can be composed of different modalities. This is so they can support esoteric functions, but also the different roles that underpin different social situations.

We have experimented with an approach to cross-platform interoperability in the browser with the open-source SDK Ubiq. We implemented support for Ubiq in a Js library for both the browser and NodeJs. We validated our implementation with a number of proof-of-concepts based on real-world use cases, that would be challenging or impossible to support in most contemporary SVR systems. We described the different levels at which we had to support inter-operation, where they could diverge and where they had to remain the same, and the technical challenges involved in doing so. We find that cross-platform support like this is not straightforward. Usually only system creators have deep enough access to do it, and likely it will require changes to core functionality.

While there are improvements to be made, our implementation is successful. We show a variety of proof of concept modalities to validate both our implementation and demonstrate the value that can be achieved by leveraging heterogeneous computing platforms. Specifically, the modern browser. Our hope is that others working in the social VR space recognise the importance of asymmetric cross-platform support, and can learn from our technical work, especially the need to consider how cross-platform support must be baked-in from the start. All the source code for the Js library and the samples demonstrated above, is maintained on GitHub at https://github.com/UCL-VR/ubiq/tree/samples-browserextended.

ACKNOWLEDGMENTS

This work was partly funded by United Kingdom EPSRC project Graphics Pipelines for Next Generation Mixed Reality Systems (grant reference EP/T01346X/1), EU Horizon 2020 project RISE (grant number 739578), EU Horizon 2020 project CLIPE (grant number 860768) and EU Horizon 2020 project PRIME-VR2 (grant number 856998).

REFERENCES

Jérémie Allard, Valérie Gouranton, Loïck Lecointre, Sébastien Limet, Emmanuel Melin, Bruno Raffin, and Sophie Robert. 2004. FlowVR: A Middleware for Large Scale Virtual Reality Applications. In *Parallel Processing. 10th International Euro-Par Conference*. Springer Berlin Heidelberg, 497–505. https://doi.org/10.1007/978-3-540-27866-5_65 ISSN: 03029743.

Sercan Altundas and Enis Karaarslan. 2023. Cross-platform and Personalized Avatars in the Metaverse: Ready Player Me Case. In *Digital Twin Driven Intelligent Systems and Emerging Metaverse*, Enis Karaarslan, Ömer Aydin, Ümit Cali, and Moharram Challenger (Eds.). Springer Nature Singapore, Singapore, 317–330. https://doi.org/10.1007/978-981-99-0252-1_16

Cameron Brown. 2021. RecRoom Developer Blog - The Circuits Handbook. https://blog.recroom.com/

C. Carlsson and O. Hagsand. 1993. DIVE A multi-user virtual reality system. In *Proceedings of IEEE Virtual Reality Annual International Symposium - VRAIS '93*. 394–400. https://doi.org/10.1109/VRAIS.1993.380753

François Chardavoine, Sylvain Ageneau, and Benoît Ozell. 2005. Wolverine: A Distributed Scene-Graph Library. *Presence: Teleoperators and Virtual Environments* 14, 1 (2005), 20–30. https://doi.org/10.1162/1054746053890297

MDN contributors. 2023. Crypto: randomUUID() method. https://developer.mozilla.org/en-US/docs/Web/API/Crypto/randomUUID

Alex Coulombe. 2020. Let's Dive Into The Under Presents: Tempest. https://medium.com/alive-in-plasticland/lets-dive-into-the-under-presents-tempest-pt-1-2d1ef2168c5f

Dark Rift. 2021. *DarkRift Networking*. https://www.darkriftnetworking.com/

James Darpinian. 2022. Integer math in JavaScript. https://james.darpinian.com/blog/integer-math-in-javascript

Engage. 2023. Engage. https://engagevr.io/

Exit Games. 2021. *Photon*. https://www.photonengine.com/

Sebastian Friston, Ben Congdon, David Swapp, Lisa Izzouzi, Klara Brandstätter, Daniel Archer, Otto Olkkonen, Felix J. Thiel, and Anthony Steed. 2021. UBIQ: A system to build flexible social virtual reality experiences. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology, VRST*. Association for Computing Machinery. https://doi.org/10.1145/3489849.3489871

Glue. 2023. Glue. https://www.glue.work/

Mar Gonzalez-Franco, Eyal Ofek, Ye Pan, Angus Antley, Anthony Steed, Bernhard Spanlang, Antonella Maselli, Domna Banakou, Nuria Pelechano, Sergio Orts-Escolano, Veronica Orvalho, Laura Trutoiu, Markus Wojcik, Maria V. Sanchez-Vives, Jeremy Bailenson, Mel Slater, and Jaron Lanier. 2020. The Rocketbox Library and the Utility of Freely Available Rigged Avatars. *Frontiers in Virtual Reality* 1 (Nov. 2020), 561558. https://doi.org/10.3389/frvir.2020.561558

C. Greenhalgh and S. Benford. 1995. MASSIVE: a distributed virtual reality system incorporating spatial trading. *Proceedings of 15th International Conference on Distributed Computing Systems*, 27–34. https://doi.org/10.1109/ICDCS.1995.499999 Publisher: IEEE Comput. Soc. Press ISBN: 0-8186-7025-8.

Chris Greenhalgh, John Bowers, Graham Walker, John Wyver, Steve Benford, and Ian Taylor. 1999. Creating a live broadcast from a virtual environment. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*. ACM Press, 375–384. https://doi.org/10.1145/311535.311591

Anita Havele, Nicholas Polys, William Benman, and Donald Brutzman. 2022. The Keys to an Open, Interoperable Metaverse. In *The 27th International Conference on 3D Web Technology*. ACM, Evry-Courcouronnes France, 1–7. https://doi.org/10.1145/3564533.3564575

Russell M Taylor Ii, Thomas C Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T Helser. 2001. VRPN: A Device-Independent, Network-Transparent VR Peripheral System. In *Proceedings of the ACM symposium on Virtual reality software and technology*. https://doi.org/10.1145/505008.505019

Jussi S. Jauhiainen. 2021. Entrepreneurship and Innovation Events during the COVID-19 Pandemic: The User Preferences of VirBELA Virtual 3D Platform at the SHIFT Event Organized in Finland. *Sustainability* 13, 7 (March 2021), 3802. https://doi.org/10.3390/su13073802

Paulo Jesus, Carlos Baquero, and Paulo Almeida. 2006. ID Generation in Mobile Environments. *HASLab - Artigos em atas de conferências internacionais* (2006), 1–4. http://hdl.handle.net/1822/36065

Huib Kleinhout. 2021. WebRTC is now a W3C and IETF standard. https://web.dev/webrtc-standard-announcement/

Boriana Koleva, Ian Taylor, Steve Benford, Mike Fraser, Chris Greenhalgh, Holger Schnädelbach, Dirk Vom Lehn, Christian Heath, Ju Row-Farr, and Matt Adams. 2001. Orchestrating a mixed reality performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Seattle Washington USA, 38–45. https://doi.org/10.1145/365024.365033

Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2020. Browser Fingerprinting: A Survey. *ACM Transactions on the Web* 14, 2 (May 2020), 1–33. https://doi.org/10.1145/3386040

Marc Erich Latoschik, Christian Fröhlich, and Alexander Wendler. 2006. Scene Synchronization in Close Coupled World Representations using SCIVE. *The International Journal of Virtual Reality* 5, 3 (2006), 47–52.

Microsoft. 2022. Architecture overview — MRTK2. https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/architecture/overview

Mirror 2021. *Mirror Networking*. https://mirror-networking.com/

Mozilla Corporation. 2021. *Mozilla Hubs*. https://hubs.mozilla.com

Martin Naef, Edouard Lamboray, Oliver Staadt, and Markus Gross. 2003. The blue-c distributed scene graph. In *Proceedings of the 2003 IEEE Virtual Reality Conference*. IEEE Comput. Soc, 275–276. https://doi.org/10.1109/VR.2003.1191157 ISSN: 1087-8270.

Nels Numan, Daniele Giunchi, Benjamin Congdon, and Anthony Steed. 2023. Ubiq-Genie: Leveraging External Frameworks for Enhanced Social VR Experiences. In *2023 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. IEEE, Shanghai, China, 497–501. https://doi.org/10.1109/VRW58643.2023.00108

Alexandra Papoutsaki, Patsorn Sangkloy, James Laskey, Nediyana Daskalova, Jeff Huang, and James Hays. 2016. WebGazer: Scalable Webcam Eye Tracking Using User Interactions. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI, 3839–3845.

David Saffo, Sara Di Bartolomeo, Caglar Yildirim, and Cody Dunne. 2021. Remote and Collaborative Virtual Reality Experiments via Social VR Platforms. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama Japan, 1–15. https://doi.org/10.1145/3411764.3445426

Kimanthi Sammy. 2022. Play Roblox on Web Browser. https://alvarotrigo.com/blog/roblox-web-browser

Ryan Schulz. 2020. *Comprehensive List of Social VR Platforms and Virtual Worlds*. https://ryanschultz.com/list-of-social-vr-virtual-worlds/

Sandeep Singhal and Michael Zyda. 1999. *Networked virtual environments: design and implementation*. Addison-Wesley, Reading, MA.

M. Slater, J. Howell, A. Steed, D-P. Pertaub, and M. Garau. 2000. Acting in virtual reality. In *Proceedings of the third international conference on Collaborative virtual environments*. ACM, San Francisco California USA, 103–110. https://doi.org/10.1145/351006.351020

SocketIO. 2023. SocketIO. https://socket.io/docs/v4/how-it-works/

Spatial. 2023. Spatial.io. https://spatial.io/

Stanford Artificial Intelligence Laboratory et al. 2018. *Robotic Operating System*. https://www.ros.org

Steel Crate Games. 2023. The Co-Op Bomb Defusing Party Game. https://keeptalkinggame.com/

Unity Technologies. 2021. *MLAPI - Unity Multiplayer Networking*. https://docs-multiplayer.unity3d.com/

Virbela. 2022. Campus Roles. https://support.virbela.com/s/article/Campus-Roles

Antony Vitillo. 2021. Behind the scenes of the VR concert Welcome To The Other Side. https://skarredghost.com/2021/01/05/jean-michel-jarre-vr-concert-postmortem/

W3C. 2023a. Simple Object Access Protocol. https://www.w3.org/TR/soap/

W3C. 2023b. W3C Security Activity. https://www.w3.org/Security/

Matthias Wolfel, Daniel Hepperle, Christian Felix Purps, Jonas Deuchler, and Wladimir Hettmann. 2021. Entering a new Dimension in Virtual Reality Research: An Overview of Existing Toolkits, their Features and Challenges. In *2021 International Conference on Cyberworlds (CW)*. IEEE, Caen, France, 180–187. https://doi.org/10.1109/CW52790.2021.00038