An Architecture for Full Body Collaborative VR Applications

Pablo Figueroa * Timofey Grechkin † Yuanyuan Jiang ‡ Evan Suma Rosenberg §

USC Institute for Creative Technologies

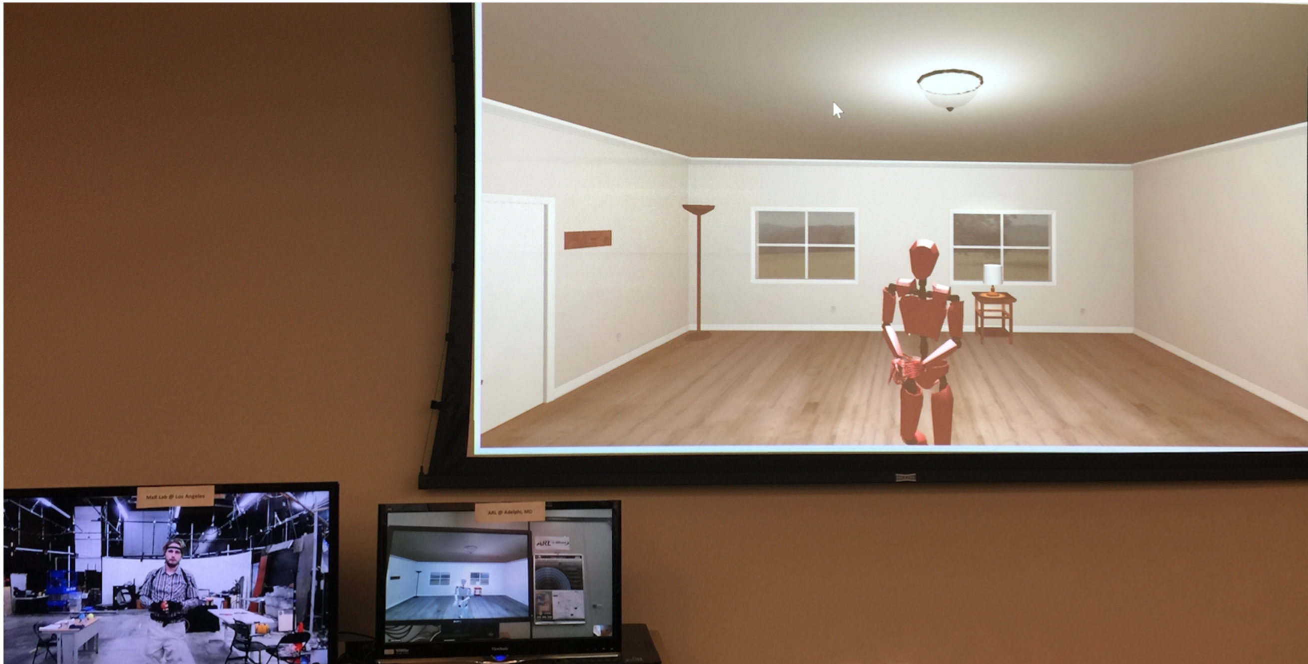

Figure 1: Demo of our system. Bottom left: Remote view of a user in the laboratory's tracking space. Top right: Local avatar of the user represented in a virtual environment. Bottom center: Video feed of our application running in real time in a remote location.

Abstract

This paper presents an open infrastructure for the development of collaborative VR applications, where full body posture and gaze information is shared among participants over secured firewalls. We describe how to leverage current technologies to handle novel requirements related to logging and replay, high refresh rates, and mobile based VR environments. Furthermore, we discuss the rationale behind the proposed solution, some limitations of this implementation, and how to compare this software architecture to similar future implementations.

Index Terms: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality

*pfiguero@uniandes.edu.co
†grechkin@ict.usc.edu
‡jyuanyuan@ict.usc.edu
§suma@ict.usc.edu

1 Introduction

Virtual Reality (VR) is becoming mainstream as a result of the commercial attention it has received since Kickstarter successfully funded Oculus —subsequently acquired by Facebook— and due to the existing quality and power of computer graphics and related technologies. Several companies are creating compelling VR experiences by means of current game engines, such as Unity and Unreal, and their ecosystems of APIs. Given that these novel technologies offer a wide variety of solutions for almost all problems in VR, we are particularly interested in how proven technologies from the VR realm could be used in these novel VR experiences. We believe this exercise could help us to understand the advantages and shortcomings of both, novel game engines and old VR technologies, as well as determine how these technologies could be integrated, and define a method to compare novel technologies to existing and proven solutions.

Specifically, we are interested in developing a Collaborative VR (CVR) infrastructure that would enable two or more users to share a virtual reality experience while sharing their full body posture. Each user is represented by a virtual avatar that shows the user's physical movements with relatively high fidelity. Such an infrastructure could be useful in a research environment where shared VR interactions can be systematically studied, where the ability to observe and control the experience is required, and where it is useful to record user movements and enable playback and review afterwards. The key requirements for this systems are, therefore, as follows:

Authorized licensed use limited to: University Of Minnesota Duluth. Downloaded on March 06,2024 at 19:32:00 UTC from IEEE Xplore. Restrictions apply.

- Use of wireless technologies for user's body tracking and display; the wireless component is particularly important to enable free locomotion for users
- Management of several input devices at their highest frequency, specifically different types of trackers.
- Record and playback of users' data.
- Remote collaboration between users, even behind tight firewall restrictions.

Figure 1 shows a usage example of our system. While a user is in the main lab wearing the mocap, a first screen shows the direct output from our desktop and a second one shows our application running in a remote lab behind a secure firewall, which is receiving the same data as our desktop.

In this paper, we describe a solution based on Unity and VRPN, which can be used in similar applications. We also report some advantages of this infrastructure in comparison to other approaches, and anticipate how this approach could be compared to other solutions in the future.

2 Related work

The work presented in this paper relates to systems for the study of human behavior in groups, to the field of CVR, and to the architectures for such systems. The following are some important results in these areas.

The interest in the study of human behavior at several levels has increased lately and has sometimes resulted in very complex systems such as [13] at one end of the spectrum. In this regard, our proposal is relatively more economical because it builds on top of low cost HMDs and their ever increasing capabilities.

CVR has been a subject of study for several years and a good reference of the related issues can be found in [14]. Our work here focuses on the issue of heterogeneity, in the sense that we want to receive information from several trackers that work at different frame rates, some of them higher than the display's frame rate. We also want to address particular issues related to managing full body tracking data from several users.

There have been previous attempts to create full body CVRs. For instance, the system in [3] uses a mocap system to capture a limited number of points of interest from two users and their surroundings. The system in [16] can follow several users, but with the line of sight limitations of a Kinect. The prototype in [12] presents a system for facial recognition and motion capture of two users where they can see virtual versions of each other through a virtual portal.

Furthermore, CVR systems for the study of human behavior have reported several architectures. VHD++ [10] used the approach of pluggable components, all at the same level, which greatly differs from more recent game engines. Moreover, even though lacking in detail regarding software integration of the hardware components, the authors in [15] use a game engine as the core of their system. Our solution uses similar components to the latter, but is not as complete in terms of sensors because it aimed at allowing more mobility to one or more users.

The VR community has developed several development environments [2, 20, 8] and specially tailored languages [1, 4, 5] with interesting features for this domain. The wealth of functionality and community support around game engines such as Unity and Unreal is overwhelming, thus making them the most commonly used technologies for current VR experiences. Our proposed solution integrates the best of two worlds: Unity and VRPN [18, 19].

We decided to use Unity, a development environment released in 2005, because of its easy learning curve and the large amount of features it provides for game development, and its full support for novel VR devices. However, during the architectural design phase, we could not easily identify mature technologies for collaborative VR application within the Unity's ecosystem. Therefore, we decided to use VRPN, a 20 year old, open source library for the integration of novel devices in VR applications, with various useful features, such as the following: Unification of data types; multipoint communication from devices; record and playback with speed control capabilities for time stamped data; use of both UDP and TCP connections for highest performance; and the possibility of using just TCP when network restrictions arise. As expected, there are several plug-ins that allow Unity users to connect peripherals through VRPN, but not all of them include the aforementioned functionalities, and all of them provide solutions that are limited by Unity´s frame rate, a fixed value of 60Hz in its mobile implementations.

Our infrastructure could be classified as an After Action Review (AAR) System, a term coined more than 20 years ago for systems that allow "collecting, analyzing and reviewing the results of a training exercise" [21]. This system provides a way to seamlessly and safely save all events generated from a short collaborative experience between two users, by using current equipment and development environments. When comparing this system to the future expectations they had at that moment, we fulfill most of their expected functionality. However, we do not provide sophisticated review capabilities since they are domain dependent, we do not require voice or video synchronization, and we do not depend upon distributed data storage such as in [7], although VRPN could handle this last feature. Previous solutions have used VRPN for AAR, such as [6]. Nonetheless, we test this solution for full body tracking. [17] presents a similar solution that integrates more input types, but they decided to limit the frame rate to 60Hz, which could be constrictive in the near future.

Several review systems have been implemented throughout the years in different domains, with particular purposes and technologies. The system in [11] proposes a closed solution in the domain of health training, so that "students could become self aware of their actions [...] and gain insight on how to improve [...]". The system in [9] presents a system that records and analyzes gaze data for multiple users, in a distributed way. Even though our system provides similar functionalities to those of previous systems, we include more data about a users´ pose. However, given our current focus, our analysis tools are still a part of the future work.

We used multithreading capabilities to extend one of the many available VRPN clients, so that it could handle in Unity two way communication of strings and refresh rates above 60 fps, which is currently a fixed value in mobile devices. In terms of hardware, we use the Perception Neuron mocap system by Noitom, which captures wirelessly up to 56 points per user, including 2 points per finger. We used Samsung's Gear VR as HMD because of its wireless capabilities, and a Phasespace system for accurate positional and directional tracking in a large space of about 20m by 10m.

3 Our Architecture

We will now describe our solution and the rationale behind it. Subsections will show the heuristic we applied to select this architecture, its main advantages, some limitations, and how we envision a comparison between this solution and future ones. Figure 2 shows an UML-like deployment diagram with the main components of the proposed solution. As can be seen in this figure, our solution follows a client-server architecture that has in its basic deployment a VRPN based server, two android based clients, and a PC client for the main control of the experience. Remote clients could be connected to the main server, even through restricted firewalls, by means of the tcp-only connections that VRPN supports.

We developed the following plug-ins within the VRPN server:

- Device support for two suits of the Noitom's Perception Neuron motion capture system. Noitom already provides a plug-in

Authorized licensed use limited to: University Of Minnesota Duluth. Downloaded on March 06,2024 at 19:32:00 UTC from IEEE Xplore. Restrictions apply.

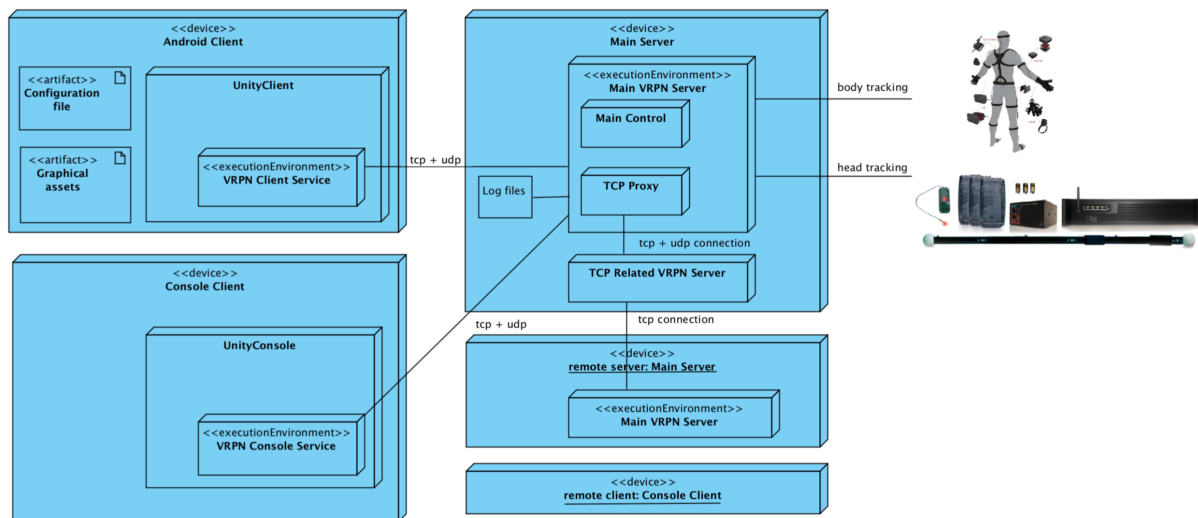

Figure 2: Overall architecture of our solution. A Main Server integrates the data from several devices and sends it to a Console Client and two Android Clients (we show one of them here). A remote Main Server receives all the information from the local server and makes it available in the remote site.

in Unity for reading such data, but the use of such a module required to run a Unity process in our server, a path we decided against given the functionality of Unity as a front end tool.

- A simulator server that generates car traffic events for some of the experiences we created. Although it is not fully described in this paper, it is worth mentioning as part of the architectural decisions we made.

- A method to seamlessly send information from a real device or from a pre-recorded log file. This allow us to simulate devices without changing anything in a client.

- A proxy device that connects to another VRPN client and broadcasts the obtained data. This functionality allowed us to encapsulate the physical architecture of our solution, whilst providing to provide one central server per site for all client connections.

- Our main server consists of a VRPN device and client that receives text messages from the console client and controls some of the previously mentioned devices and features.

Each android client preloads common geometry for rigged virtual humans and their scenario, sends and receives string commands for control purposes, and receives tracking information from our server. A special configuration file allows us to tune the specific behavior of each client, so that they can report different names to the log system, for example. Finally, the console server provides all functionality of android clients, plus a basic button based interface for sending commands to our server.

3.1 Main Advantages

There are several advantages that we envisioned from this architecture from its design, and some we found during the course of its deployment. In terms of features that VRPN supports, we took advantage of UDP connections for data with high refresh rates and TCP connections for lower rate data, text based control messages, and clients behind secured firewalls. Moreover, we made the most of its logging facilities, its multi-client support, its standard data types, the existing support for multiple devices, and the standard method to support new ones. At the beginning of our development, we also took advantage of one of the multiple VRPN plug-ins for Unity in order to support the required client functionality. We can also replace devices for others that produce the same information, by taking advantage of the standard data types that VRPN uses. However, this can be problematic in the case of quaternions, as we will describe later.

We also took advantage of the fact that VRPN is open source to develop the following functionalities:

- Multithreaded support for slow devices in the server. One example is the Perception Neuron, which is updated at 60Hz and it is slow in comparison to other devices we used such as the Phasespace traking system, which is configured in our setup to 480Hz but can run at twice this speed. This could create problems when clients cannot read data at the fastest frame rate. Even though there is a solution in VRPN — based on the *Jane_stop_this_crazy_thing()* function — that samples high frequency devices to make their event rate comparable, we decided to create a different thread that waits for data from the Neuron and does not slow down other devices. We created a shared data structure and a mutex for each tracker in the Neuron; both threads use the mutex to perform tracker operations. Given that the probability of collision between threads is very low, read operations in the main VRPN thread hardly have to wait for writing operations in the other thread.

- Multithreaded support for all devices in the client. As part of the solution for high refresh rate devices, we developed a multithreaded client for VRPN. Such a client can read events when they are generated and handle them as desired. In our case, we used the latest data available for each device at a particular frame, and we considered using some smoothing functions where necessary.

- Considering that VRPN selects a random set of ports by default, which use TCP and UDP protocols, we decided to change the code within VRPN in order to limit the communication to a specific port when firewall restrictions are mandatory.

Authorized licensed use limited to: University Of Minnesota Duluth. Downloaded on March 06,2024 at 19:32:00 UTC from IEEE Xplore. Restrictions apply.

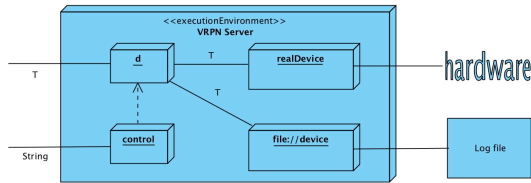

Figure 3: Seamlessly Reading from a File or a Device.

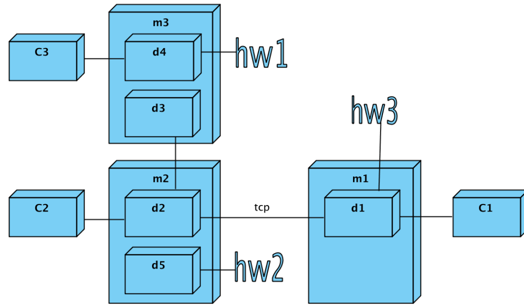

Figure 4: Proxy Mechanism.

- Our server and clients are able to receive and send string messages, which were used for control purposes. In this way, we could tell clients which scene had to be loaded at a particular time, for example.

- Figure 3 shows the mechanism we used to seamlessly send information from a file or from a real device. We used the following combination of VRPN devices: one for the real device, one for the file, one that hid the previous two and gave clients a uniform interface and device name, simply called *d* in this figure, and a control device that received string commands and told *d* which stream it should forward next. We defined a directory in the configuration file where log files could be found, and included in a string command which particular file should be loaded. Common commands such as *PLAY* or *PAUSE* were also available.

- Figure 4 shows our proxy devices. *M1*, *M2*, and *M3* are three machines that are connected to three different devices, *hw1*, *hw2*, and *hw3*. The last machine is remote and secure, connected only through TCP. We can hide this architecture from the point of view of *M3*, in particular the existence of *M1*, by creating proxy devices like *d3* and *d2*. In this figure, *d2* sends the same information as *d1* and *d3*. However, the existence of *M1* is hidden to *C2*, so that we could hide complex hardware architectures in *M2*. This configuration also makes it possible to hide TCP dependencies with proxies in VRPN. A single dropped TCP connection will hang all other devices in a VRPN server. In the same figure, *C3* will keep on getting events from *d4*, regardless of whether the connection between *d3* and *d2* is active or not, whereas *d2* will hang *d5* while the TCP connection is having problems.

Both VRPN and Unity are multi-platform; for this reason, we could target android based and windows based clients with the same codebase. However, some *ifdef* sections had to be added to the VRPN code in order to handle android's specific implementation, therefore we had to develop a specially tailored project in Visual Studio 2015 for cross compiling from Windows to android.

3.2 Some Limitations and Alternatives

We have mentioned two limitations that can be easily overcome: In order to connect more than two users, we could use several servers, each one in charge of up to two users, and use proxy servers in order to hide this architecture from clients. Our current specially tailored Visual Studio 2015 project could also be replaced by a specially tailored CMake configuration file with cross compiling features, which is the standard tool that VRPN uses.

However, the inherent complexity of supporting the integration of two technologies makes development harder. Our entire application has required approximately 8 man-months for development, and it may be argued that technology integration has taken a good part of this time. Figure 5 shows an alternative we considered, which uses a more complex, Unity based console application, and replaces VRPN with a Unity based network layer. Even though it is beyond the scope of this work to develop such an alternative and use the metrics in Section 3.3 for comparison purposes, we decided to choose the architecture that uses open source code, which we believe to be better because it gave us control during development and allowed us to reuse functionalities from the VRPN.

Finally, when compared to the implementation in Unity, we experienced certain issues with the quaternion implementation in VRPN. In Figure 5, we could used the framework provided by Noitom to animate a virtual avatar, that sends Euler angles between the Noitom's device manager and Unity. However, in Figure 2, we wanted to use quaternions between the Android Client and the Main Server, since it is part of the basic type that VRPN uses for trackers. This required the use of a EulerToQuaternion(E) function in the VRPN server, followed by a QuaternionToEuler(Q) function in the Unity client, in order to re-use Noitom's animation system. Currently, we have that $E' \neq E$ as shown in the following formula:

$$E' = QuaternionToEuler(EulerToQuaternion(E)) \quad (1)$$

Given that the relation between quaternions and euler angles is not bijective, the formula above is not strictly required for all conversions between these two representations. Consequently, Unity functions hold this equality whereas VRPN functions do not. This created problems in the integration between Unity and VRPN. Moreover, it is difficult to point out the differences between the open source versions of these functions in VRPN, and the closed versions in Unity. Therefore, our current implementation of the Noitom's driver for VRPN does not use its standard tracking type.

3.3 Evaluation

First, we used black box tests to compare the output of our system to some references. For example, we compared the output of our architecture for the Perception Neuron with the output that can be obtained from the examples provided with the equipment. We detected this way the issues related to quaternions described before.

Our implementation runs at least at 60Hz using a Gear VR. We fine tuned this feature with the aid of the Unity profiler until we achieved such frame rate. This entailed changes in the scene and in the way our VRPN thread in Android updated information to the main thread, as described in Section 3.1. Any frame rate lower than this would be very uncomfortable for the user, because the delay and related artifacts are very visible.

Traditionally, lines of code, development time, and performace metrics have been used to compare competing implementations of the same functionality. However, these do not take into account development effort in terms of time, learning curve, or bugs. We propose to complement performance metrics with the data shown in Table 1, which shows how functionality is completed over time. Values in the table range between 0 and 1, and are derived from the comments left at the source code management system in use. They can be adjusted backwards in time, for example, when a bug

Authorized licensed use limited to: University Of Minnesota Duluth. Downloaded on March 06,2024 at 19:32:00 UTC from IEEE Xplore. Restrictions apply.

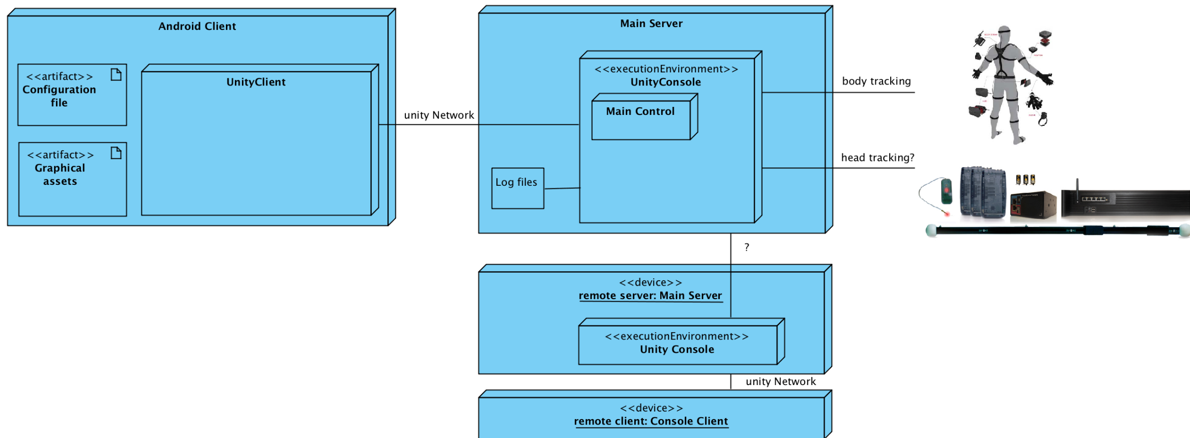

Figure 5: An Alternative Solution. The Unity Console integrates all the information from various devices and sends it to the Android clients. Remote sites connect to the console and get device information from it.

	Cumulative Development	1.3	2.9	4.8	8.4
ID	**Functionality**	**April**	**May**	**June**	**July**
1	Perception Neuron integration	0.6	0.6	0.6	0.8
2	Limiting port management	0.5	0.5	0.5	1
3	Proxy for trackers	0.2	0.5	0.5	1
4	Multithreading in the client	0	0.5	0.5	1
5	Multithreading in the server	0	0	0	1
6	Special project in Visual Studio	0	0.3	0.5	1
7	Android client for VRPN	0	0.5	1	1
8	Control commands from a console	0	0	0.6	0.8
9	Event file management	0	0	0.6	0.8

Table 1: Level of Development per Requirement over Time.

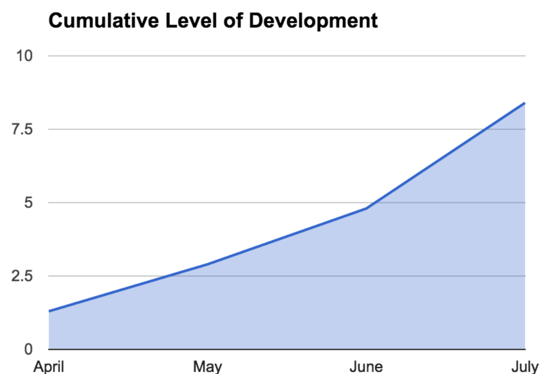

Figure 6: Cumulative Level of Development over Time.

in an existing functionality is discovered. Although these values are subjective, they can give a broad idea of the effort required to develop a particular software.

If we add the values of different criteria each month, we can produce Figure 6, which shows how development has evolved over time. Ideally, the curves with big areas are preferred, since they show that the functionality could be implemented fast, with minimal errors. If several implementations are available, time could be scaled to a uniform value and therefore the areas could be compared.

4 Conclusions and Future Work

We have presented an architecture for CVR systems that enabled us to track and record the full body of several users in a VR environment. This solution, based on Unity and VRPN, will allow us to implement all functionality of an after action review system, while it shows how existing VR technologies can be competitive in the current VR revolution. In summary, the main design decisions behind our solution were the following:

- A client should not depend on the particular architecture of a distributed solution. Proxies could help to hide such details.

- Separate the server code from all clients, so that the server could run at the highest rate possible, and does not depend on any interaction with any user. The server should run any simulation required, so its results could be distributed to all clients.

- In the server of a collaborative solution, encapsulate slow devices in a different thread, so that faster devices do not suffer from the lack of response of others.

- In mobile clients, perform device reading in a separate thread, in order to facilitate the execution of such clients at the required frame rate. This avoid issues with device drivers of slow devices that may hang a client, or fast devices which may flood such client. In the latter case, it may be necessary to discard some events.

- Logged information should be indistinguishable from the data obtained from real devices. This facilitates playback.

Our VRPN code for the support of Perception Neuron data is available in GitHub, and it is pending revision in order to be integrated to the main distribution. In the future, we plan to use this infrastructure in several user studies, and complement its current functionalities with an easier to use interface. We also plan to perform experiments with geographically-apart collaborators, to fully test the capabilities of this solution. In those cases, we plan to consider conflicting situations among several users, which are not considered in our current implementation.

References

[1] G. Bell, A. Parisi, and M. Pesce. The virtual reality modeling language. https://www.w3.org/MarkUp/VRML/, 1997.

Authorized licensed use limited to: University Of Minnesota Duluth. Downloaded on March 06,2024 at 19:32:00 UTC from IEEE Xplore. Restrictions apply.

[2] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira. Vr juggler: a virtual platform for virtual reality application development. In *Virtual Reality, 2001. Proceedings. IEEE*, pages 89–96, March 2001.

[3] S. Chagué and C. Charbonnier. Real virtuality: A multi-user immersive platform connecting real and virtual worlds.

[4] T. W. Consortium. What is x3d. https://www.web3d.org/x3d/what-x3d/, 2017.

[5] P. Figueroa, W. F. Bischof, P. Boulanger, H. J. Hoover, and R. Taylor. Intml: A dataflow oriented development system for virtual reality applications. *Presence: Teleoperators and Virtual Environments*, 5(17):492–511, 2008. An optional note.

[6] D. Friedman, A. Brogni, C. Guger, A. Antley, A. Steed, and M. Slater. Sharing and analyzing data from presence experiments. *Presence: Teleoperators & Virtual Environments*, 15(5):599 – 610, 2006.

[7] E. Kaya and F. E. Sevilgen. A fully distributed data collection method for hla based distributed simulations. In *Proceedings of the 2009 Summer Computer Simulation Conference*, SCSC '09, pages 337–347, Vista, CA, 2009. Society for Modeling & Simulation International.

[8] J. Landauer, R. Blach, M. Bues, A. Rosch, and A. Simon. Toward next generation virtual reality systems. In *Multimedia Computing and Systems '97. Proceedings., IEEE International Conference on*, pages 581–588, Jun 1997.

[9] A. Murgia, R. Wolff, W. Steptoe, P. Sharkey, D. Roberts, E. Guimaraes, A. Steed, and J. Rae. A tool for replay and analysis of gaze-enhanced multiparty sessions captured in immersive collaborative environments. In *Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium on*, pages 252–258, Oct 2008.

[10] M. Ponder, G. Papagiannakis, T. Molet, N. Magnenat-Thalmann, and D. Thalmann. Vhd++ development framework: towards extendible, component based vr/ar simulation engine featuring advanced virtual character technologies. In *Proceedings Computer Graphics International 2003*, pages 96–104, July 2003.

[11] A. B. Raij and B. C. Lok. Ipsviz: An after-action review tool for human-virtual human experiences. In *2008 IEEE Virtual Reality Conference*, pages 91–98, March 2008.

[12] D. Roth, K. Waldow, F. Stetter, G. Bente, M. E. Latoschik, and A. Fuhrmann. Siamc: A socially immersive avatar mediated communication platform. In *Proceedings of the 22Nd ACM Conference on Virtual Reality Software and Technology*, VRST '16, pages 357–358, New York, NY, USA, 2016. ACM.

[13] R. Schubert, G. Welch, S. Daher, and A. Raij. Husis: A dedicated space for studying human interactions. *IEEE Computer Graphics and Applications*, 36(6):26–36, Nov 2016.

[14] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[15] B. Spanlang, J.-M. Normand, D. Borland, K. Kilteni, E. Giannopoulos, A. Pomés, M. González-Franco, D. Perez-Marcos, J. Arroyo-Palacios, X. N. Muncunill, and M. Slater. How to build an embodiment lab: Achieving body representation illusions in virtual reality. *Frontiers in Robotics and AI*, 1:9, 2014.

[16] M. Sra and C. Schmandt. Metaspace: Full-body tracking for immersive multiperson virtual reality. In *Adjunct Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15 Adjunct, pages 47–48, New York, NY, USA, 2015. ACM.

[17] W. Steptoe and A. Steed. Multimodal data capture and analysis of interaction in immersive collaborative virtual environments. *Presence: Teleoperators & Virtual Environments*, 21(4):388 – 405, 2012.

[18] R. M. Taylor, II, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser. Vrpn: A device-independent, network-transparent vr peripheral system. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '01, pages 55–61, New York, NY, USA, 2001. ACM.

[19] R. M. Taylor II, J. Jerald, C. VanderKnyff, J. Wendt, D. Borland, W. R. Sherman, M. C. Whitton, and D. Marshburn. Lessons about virtual environment software systems from 20 years of ve building. *Presence: Teleoperators & Virtual Environments*, 19(2):162 – 178, 2010.

[20] H. Tramberend. Avocado: a distributed virtual reality framework. In *Virtual Reality, 1999. Proceedings., IEEE*, pages 14–21, Mar 1999.

[21] G. Vasend. After action review system development trends. In *Proceedings of the 27th Conference on Winter Simulation*, WSC '95, pages 1262–1266, Washington, DC, USA, 1995. IEEE Computer Society.

Authorized licensed use limited to: University Of Minnesota Duluth. Downloaded on March 06,2024 at 19:32:00 UTC from IEEE Xplore. Restrictions apply.