Dynamic Interactivity Inside the AlloSphere

Charles Roberts, Matthew Wright, JoAnn Kuchera-Morin, Lance Putnam, Graham Wakefield
AlloSphere Research Group and Media Arts & Technology Program
University of California, Santa Barbara
{c.roberts, matt, jkm, l.putnam, wakefield}@mat.ucsb.edu

ABSTRACT

We present the Device Server, a framework and application driving interaction in the AlloSphere virtual reality environment. The motivation and development of the Device Server stems from the practical concerns of managing multi-user interactivity with a variety of physical devices for disparate performance and virtual reality environments housed in the same physical location.

The interface of the Device Server allows users to see how devices are assigned to application functionalities, alter these assignments and save them into configuration files for later use. Configurations defining how applications use devices can be changed on the fly without recompiling or relaunching applications. Multiple applications can be connected to the Device Server concurrently.

The Device Server provides several conveniences for performance environments. It can process control data efficiently using Just-In-Time compiled Lua expressions; in doing so it frees processing cycles on audio and video rendering computers. All control signals entering the Device Server can be recorded, saved, and played back allowing performances based on control data to be recreated in their entirety. The Device Server attempts to homogenize the appearance of different control signals to applications so that users can assign any interface element they choose to application functionalities and easily experiment with different control configurations.

Keywords

AlloSphere, mapping, performance, HCI, interactivity, Virtual Reality, OSC, multi-user, network

1. INTRODUCTION

1.1 Motivations

The Device Server provides tools and methodologies for manipulating control data dynamically and flexibly. It currently drives interactivity inside the UCSB AlloSphere, a three-story, spherical, immersive environment dedicated to artistic performance and scientific research [2]. Applications deployed in the AlloSphere range from algorithmic music and visualization engines to simulations of sub-atomic electron spin in hydrogen atoms. The Device Server's design stems from the practical concerns of managing interactivity in this wide variety of virtual environments; we believe it has reached a level of maturity and generality where it may be broadly useful in the NIME community.

The initial design for the Device Server addressed the following primary concerns:

- A physical space the size of the AlloSphere (approximately 715 m^3) requires multiple computers to drive active stereo visualizations and spatialized sonifications. A centralized hub for interactivity can distribute control data to these computers concurrently.
- Enable users to easily customize how they interact with applications.
- Free application developers from worrying about device drivers or any other implementation details of interactive devices, allowing them to focus on exposing interactive functionality within their applications. Instead of hard-coding any device-specific knowledge into applications, make all applications device-agnostic and relate device affordances to application functionality with an explicit mapping layer[10].
- Visual and sonic rendering machines should not be responsible for processing control data. Let the Device Server efficiently handle common control data processing tasks such as filtering and thresholding.
- Initializing visualization and sonification engines when switching from one virtual environment to another can be a complex task. A typical AlloSphere demonstration involves experiencing five or six virtual environments on different hardware and software platforms in half an hour; moving between these environments must be as smooth as possible and avoid delays and technical glitches. Use of the Device Server should automate changing interactive configurations so that no human intervention is required.
- Provide a centralized console for monitoring all control data and troubleshooting all connections.

1.2 Terminology

A *Device* is a piece of hardware or software that provides interactive affordances. A Device is often an aggregation of Controls. Example devices include Wiimotes, MIDI Keyboards and camera tracking systems.

A *Control* is a single interactive element that is part of a Device. For example, a joystick on a gamepad Device has one Control for the X direction and a second Control for the Y direction.

A *Mapping* links an application functionality to a particular Control on a particular Device, with an optional Lua

NIME2010, 15-18th June 2010 Sydney, Australia

programming language expression that is evaluated whenever data is received from the Control assigned to the Mapping.

1.3 Related Work

Frameworks for interactivity can generally be divided into those geared towards use in Virtual Reality Environments (VREs) and those geared towards use in artistic expression. The Device Server is the first framework for interactivity explicitly designed for both uses; brief descriptions of other frameworks from both fields are provided below.[1]

1.3.1 Virtual Reality Interactive Frameworks

The VR community has a longer history of creating software infrastructures for managing interactivity than that of the digital arts community. Although many interactive frameworks such as Vrui[4] and Gadgeteer[1] are part of larger VR development environments the most pervasive VR interactivity framework, the Virtual Reality Peripheral Network (VRPN)[8] is a standalone interaction library that has been in development and usage for over a decade.

In addition to handling device drivers and networked communication, VRPN and other VR frameworks commonly feature abstractions that allow one device to be easily swapped with another. In VRPN this is enabled through the use of device categories: Tracker, Button, Analog, Dial and ForceDevice; the Vrui and Gadgeteer frameworks use a similar set of categories. Any tracker device can be substituted with any other tracker; any button with any other button etc. However, these abstractions require client application functionality to be built around them. An application programmed to use a joystick for navigation cannot use a tracking device instead without changing and recompiling application source code. This is in direct contrast to abstractions found in the Device Server which decouple application functionality from device-specific constraints.

1.3.2 Interactive Frameworks for Musical and Artistic Expression

Perhaps the most significant difference between VR interactivity frameworks and those intended for artistic contexts is the use of GUIs. None of the the VR frameworks mentioned has a GUI to monitor server state or device connections; in contrast, many tools geared towards musical expression prominently feature user interfaces.

In addition to monitoring state and connections, the GUI for the Mapping Tools of the McGill Digital Orchestra Toolbox (DOT)[5] also enables users to configure and reconfigure how devices control applications. These configurations can be saved into XML documents and reloaded on demand. Unfortunately, in the current DOT externals for Max/MSP these configurations have to be loaded by hand; in the Device Server default configurations for client applications are loaded automatically so that no human interaction is required.

Other recent frameworks such as the SenseWorld DataNetwork[3] and The Bridge[6] emphasize dynamic registration of participants and devices in a musical network. The Bridge also features a decentralized network similar to what is found in VRPN; there is no central location for monitoring and manipulating control data flow. The ability to discover and utilize nodes on a network adds flexibility and dynamism, however, the current lack of recallable configurations in these two frameworks makes them less than ideal for VREs in which multiple distinct projects are frequently run.

[1]For a more thorough discussion please see http://www.charlie-roberts.com/docs/thesis.pdf

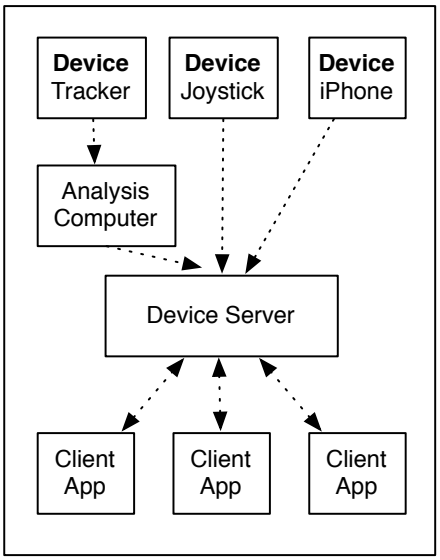

Figure 1: A Sample Device Server Network

2. THE DEVICE SERVER

2.1 Overview

The Device Server application runs under OS X. It collects data from connected interactive devices and routes them to registered client applications. All network communication is performed using the Open Sound Control protocol [9]; the ability to send and receive OSC messages is the only requirement for client applications to communicate with the server. OSC was chosen as the communication protocol for both its flexibility and its wide rate of adoption into programming platforms used in the digital arts.

Each *configuration* defines the mappings from devices' controls to application functionality, including arbitrary control data processing (described in Section 2.7), with both mappings and processing algorithms represented using the Lua programming language. Lua was chosen for its speed, dynamism, ability to perform arbitrary computation, and roots as a configuration language; more about the decision to use Lua can be read in sections 2.3 and 2.7 .

Programmers can therefore define entire configurations at the Lua source code level. Alternately, the Device Server's GUI allows easy loading, saving, and modification of configurations, automatically translating GUI-defined configurations into the appropriate Lua code. Client applications can also define configurations programmatically by indicating (via OSC messages) which devices they are interested in and how they want device output to be processed before it is sent to them.

The Device Server GUI (described in Section 2.4) also allows users to monitor device connections and the data entering and leaving the Device Server.

2.2 Device Server Network Topology

The Device Server runs on a single computer in an interactive network; Figure 1 shows a simple sample network. Client applications typically running on different computers within the network register with the server by sending a handshake message that gives the name of the application and the port and IP address where the client would like to receive control messages. Multiple client applications can be connected to the Device Server concurrently; each client possesses its own configuration files defining which devices

they will receive data from. These configurations can also define separate Lua functions for processing this data before it is sent to client. Multiple clients can thus receive data generated by the same controls on the same devices but filtered using different algorithms.

Devices can be connected to the Device Server through many different mechanisms. Direct connections via cabling, network messages sent via OSC and network messages sent via VRPN are all supported. Other types of connections can be accommodated through the use of plugins (discussed in Section 2.5.1).

2.3 Configuration

There are four types of configurations utilized by the Device Server and specified in Lua files. Alternatively, the Application Interface and Implementation configurations can be specified dynamically using OSC messages at runtime.

- Master Device List - A global configuration that assigns a unique id to every device in an interactive system.
- Device Configuration - enumerates the Controls present on a Device and the range of values each one generates.
- Application Interface - Each client application has a single Interface that defines what functionalities the application is exposing for control, what range of values each functionality expects to utilize, and the OSC address for each functionality.
- Application Implementations (aka Configurations) - Client applications can have multiple configurations that each address the application's Interface. The implementation file defines which specific controls on which specific devices are assigned to each application functionality, along with the Lua expressions and functions for processing control data.

As mentioned above, client applications have a single interface file defining the functionalities they expose and any number of implementation files defining different configurations of devices that feed control data to these functionalities. When an implementation file is loaded the Device Server creates *mappings* that define which controls on which devices feed into particular functionalities and what processing scripts are applied to the data generated by the controls. Applications can define implementations that include wired devices instead of wireless ones, multiple users instead of a single one, MIDI devices instead of Wiimotes or any other combination of controls. Users can quickly change entire interactive configurations simply by choosing another implementation through a pull-down menu in the Device Server GUI.

A very small example interface is given below for a client application that provides the affordance of moving an avatar on a two dimensional plane.

```
interface = {
 { name = "Avatar X", destination = "/nav/a1x",
       min = -1,  max = 1 },
 { name = "Avatar Y", destination = "/nav/a1y",
       min = -1,  max = 1 },
}
```

In the above interface file two functionalities, Avatar X and Avatar Y are defined. The application is expecting to receive values ranging from -1 to 1 for both. The OSC address where these values should be received is also given. This information represents constants that are always the same regardless of what implementation file is currently loaded.

Below is an example implementation for the the interface:

```
mappings = {
 { name = "Avatar X", device = "Wiimote 1",
       control = "Nunchuk X", expression = ""},
 { name = "Avatar Y", device = "Wiimote 1",
       control = "Nunchuk Y", expression = ""},
}
```

For each functionality defined in the interface (in this case Avatar X and Avatar Y) a mapping is defined that denotes the name of a control on a particular device whose values should be sent to the corresponding OSC address in the interface. This is also where Lua expressions can be defined; these will be evaluated whenever data from the corresponding control is received by the Device Server. This expression can call on predefined Lua functions and enables powerful data processing capabilities. The Device Server originally used XML as its configuration language and JavaScript for control data processing. We decided to move to Lua because it could fulfill both roles; in addition to being one of the fastest scripting languages available Lua was originally intended as a data description language. One of the unanticipated advantages to using Lua as the configuration language is that configurations can be dynamically generated. As one simple example, the for loop below would generate a configuration dictionary that would allow an application to access a device with 128 buttons.

```
controls = {}
for i=1, 128 do
  controls[i] = {
    name = "Button Grid" .. i,
    device = "ButtonDevice",
    control = "Button " .. i,
  }
end
```

Another advantage of using Lua is that the device and control names can be assigned to Lua variables making it very easy to change one device for another throughout an entire implementation.

2.4 Graphical User Interface

The GUI of the Device Server, shown in Figure 2, is one of the elements that sets it apart from other VR frameworks as well as other tools for musical interactivity. It allows users to do the following:

- Quickly see what client applications are connected and what controls are assigned to each application functionality
- Select the current client implementation file defining control mappings
- Change the devices and controls assigned to application functionalities via pulldown menus. Changes to control configurations can be saved into new implementation files for future recall.
- Monitor data flow. Users can see the raw data values generated by the devices and also see processed values that are routed to applications.
- Monitor and manage device connections. The Device Server plugin system allows each device to have its

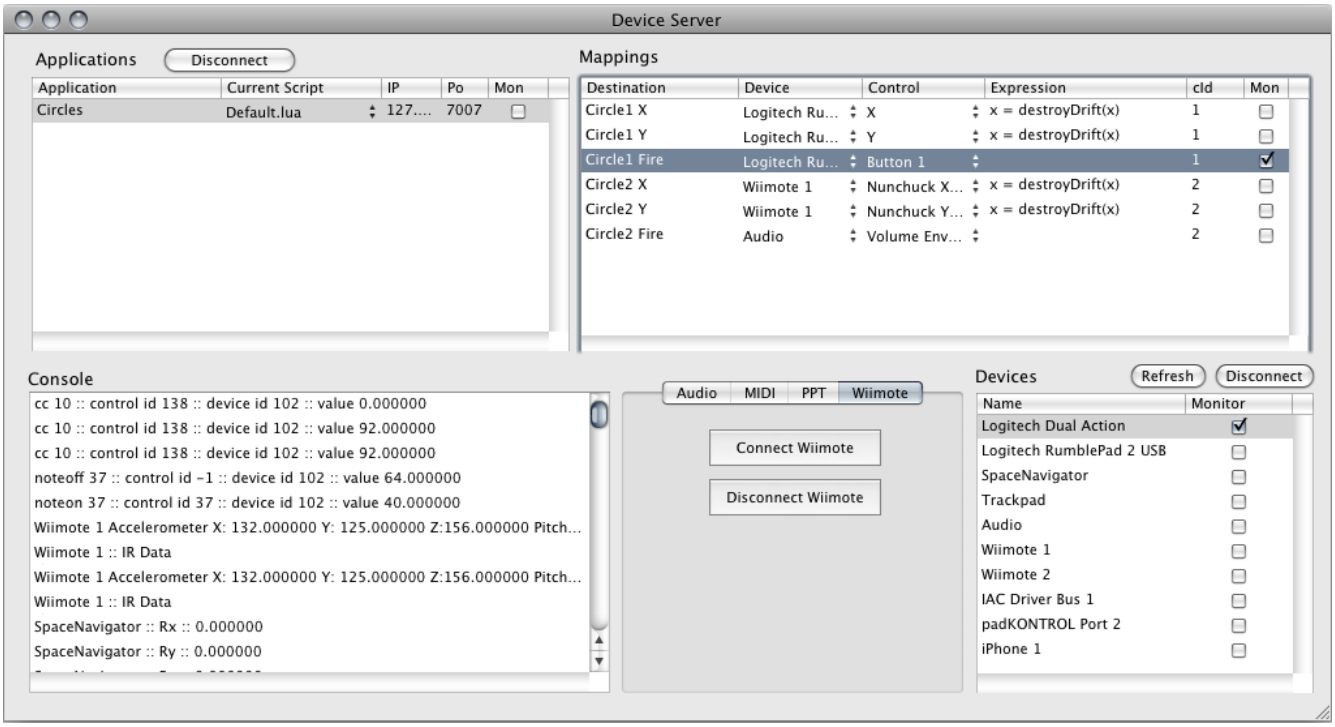

Figure 2: A screenshot of the Device Server GUI showing one client application connected to a variety of devices.

own encapsulated GUI for managing device connections. This is necessary for devices requiring Bluetooth pairing or other action sequences to establish connections.

2.5 Integrating Devices With the Device Server

To add a new Device that uses a driver or protocol the Device Server already has a plug-in for (such as HID, VRPN and MIDI) users create a Lua configuration file that enumerates the device's controls and gives the range of values each control outputs. The Device Server can be extended to support additional devices and protocols via OSC forwarding or via its plug-in architecture.

OSC forwarding of device data allows devices that are connected to other computers to be recognized by the Device Server. For example, the AlloSphere possesses a camera tracking system that gives the orientation and position of active infrared LED markers. The cameras are connected to their own dedicated computer which performs the computer vision processing needed to generate these values. This computer forwards these values to the Device Server which then routes them as required to registered applications. OSC forwarding also allows software applications such as Max/MSP to generate control data to be used by the Device Server. Thus, if Max has a driver for a device that the Device Server doesn't possess Max can be used to forward messages from that device to the Device Server. As one example of this the AlloSphere has a bridge mounted in its horizontal and vertical center lined with capacitive touch sensors. These sensors are mounted on the railings of the bridge and use a Max/MSP external to analyze their signals and forward data to the Device Server via OSC. Applications configured to receive this data can thus tell when users are touching the railings and also obtain a rough idea of user positioning on the bridge.

If a plugin has not been written for a device and there is no opportunity to feed the device data into the Device Server via OSC, a plugin can be written.

2.5.1 *Device Server Plugins*

The Device Server manages many types of device connections (MIDI, HID, VRPN) via dynamically loaded plugins which can display optional GUIs. The plugin system is designed to be open-ended and flexible; there are a wide variety of plugins currently available for the Device Server ranging in complexity from simple HID wrappers to an audio FFT analysis plugin that visualizes the bin magnitudes of incoming audio and allows applications to use this data as control sources. The infrastructure and GUI that allows the Device Server to record and play back performances is also completely encapsulated inside of a plugin.

Although the Device Server comes with many plugins they are all compiled optionally so that users don't have to load plugins they don't need. An Xcode template project is included for developers to write their own Device Server plugins. The template compiles and runs immediately without modification in the Device Server; by default it generates a stream of placeholder data that can be easily altered by developers to use data from C or C++ device-specific drivers.

2.6 Homogenization of Device Signals

A fundamental goal of the Device Server is for application functionalities to be agnostic to the devices that control them[10]. The implementor of an oscillator should not care whether the oscillator's pitch will be controlled by a MIDI keyboard or a joystick. However, the values generated by a MIDI keyboard (0-127) are different than those of most joysticks (-1.0 to 1.0 or -127 to 127). The solution is for the application to define its own units for each functionality, e.g., Hertz or MIDI note number, and leave it to the Device Server to homogenize the data from differing devices by automatically remapping them to the declared ranges of the application functionalities.

To do this the Device Server finds the application's declared range of input values for each functionality (from the

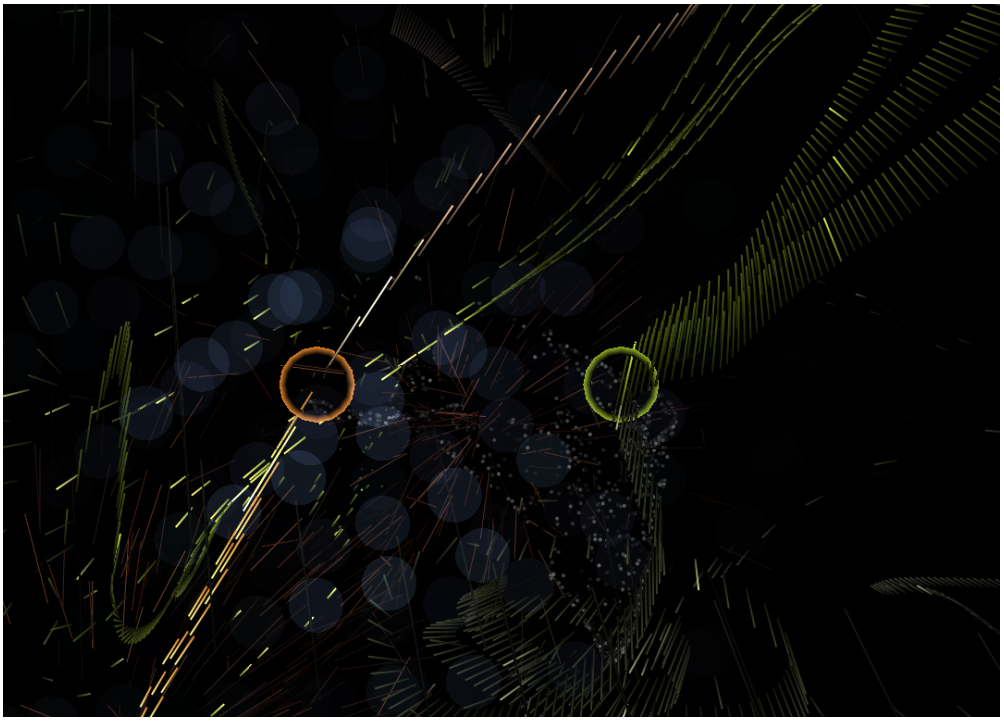

Figure 3: A Screenshot Of The Hydrogen Atom Application

interface file), compares it to the corresponding control's declared range of output values (from the device configuration file, and automatically creates an affine mapping to scale and offset values appropriately. This transformation makes it very simple to change the device/control combination feeding a functionality as the application will always receive the same range of values.

2.7 Control Data Processing With Lua

The Device Server processes control data via Lua expressions. Every mapping object may have a Lua expression assigned to it; whenever the control assigned to the mapping emits data the data is passed into the expression and the expression is then evaluated. Lua expressions may call on predefined functions to carry out common processing tasks such as integration and thresholding; these functions can be reused across multiple applications. The functions are Just-In-Time (JIT) compiled using LuaJIT 2.0[7]. LuaJIT 2.0 compilation yields performance that can be comparable to Java and C in various benchmarks. [2] [3]

A great deal of data is accessible in the scripting environment. Again, Lua scripts are triggered whenever a control assigned to a mapping generates data. But a script triggered by a mapping also has access to data values previously passed to other mappings; an expression being evaluated can even access the previous values of mappings from other client applications. This is useful in many situations; one example is when two controls manipulate the same value at different granularities such as coarse and fine pitch controls. Whenever the control assigned to fine pitch is manipulated a pitch value should be sent to the client application, but in order for that pitch value to be generated both the coarse value and the fine value need to be added together. The scripting environment is flexible enough to handle these types of situations.

[2]http://shootout.alioth.debian.org/
[3]http://dada.perl.it/shootout/

2.8 Recording And Playback Of Control Data

The Device Server allows users to record control data from performances and interactive sessions. Each control event is stored in a simple dictionary that holds an offset time from when the recording began, device and control identification numbers and the value generated by the event. The event sequence can be saved into a Lua file which can then be edited or loaded and played back in the future.

Performers routinely adjust their gestures to accommodate fixed gestural mappings to sound synthesis parameters. The ability to record and playback gestures makes the opposite practice possible: sound designers can adjust mappings and synthesis parameters to accommodate fixed gestures. Performance gestures that have been recorded can be played back repeatedly while sound designers reconfigure synthesis parameters to yield interesting sonic results. The ability to record and play back timestamped control data is also useful to researchers conducting HCI experiments.

These recording and playback features are implemented within a Device Server plugin, demonstrating that the plugin system is useful for purposes other than device drivers.

3. THE DEVICE SERVER AND INTERACTIVITY IN THE ALLOSPHERE

There are a wide variety of interactive applications deployed in the AlloSphere and all receive their control data from the Device Server. These applications include musical works, data visualizations and scientific simulations but many applications in the AlloSphere blur these lines.

The following research project within the AlloSphere shows the possibilities of interactive data mining with different user and device configurations as enabled by the Device Server.

3.1 Multimodal Representation of Quantum Mechanics: The Hydrogen Atom

This work, shown in Figure 3, interactively visualizes and

sonifies the wavefunction of an electron of a single hydrogen atom. The atomic orbitals are modeled as solutions to the time-dependent Schrödinger equation with a spherically symmetric potential given by Coulomb's law of electrostatic force. Different orbitals of the electron can be combined in superposition to observe dynamic behaviors such as photon emission and absorption.

The interactive component of the simulation allows one or more users to fly through the atom with probes that emits "stream particles" that follow along the largest changes in the probability current and gradient of the electron. The electron probability amplitude is sonified by scanning through groups of stream particles in the space. The pitch can be adjusted by the rate at which a particular set of stream particles is scanned. By assigning specific pitches to different features of the wavefunction we can generate musical results in the sonification.

The Device Server enables this simulation to be run with a variety of different user configurations and devices. The default configuration is for a single user using a wireless joystick to control both navigation and the emission of stream particles. The most common multi-user scenario is for one user to navigate through the simulation while two other users control the position of the probes and their stream particle emission.

Users can experiment with different sets of controls for navigation and data mining. More experienced computer users might prefer to navigate through the environment using a 6DOF 3Dconnexion Space Navigator while novice users use a gamepad. In the hundreds of demos we have given we have observed that wireless device communication will sometimes inexplicably fail; for this reason we also have a "all wired" configuration to fall back on in the event that we experience difficulties with the wireless devices. Switching to this fallback configuration is accomplished through a single pulldown menu in the Device Server GUI.

While developing and exploring the hydrogen atom representation, the Device Server helped when transitioning between a single-user desktop environment, where more precise control with a keyboard and mouse is often required for development, and a multi-user immersive environment, where more natural control with wireless tracking devices and game controls is desired for exploration. The separation of physical control from application logic according to the Device Server model enforces a "write-once, control anywhere" approach to application development that facilitates transitions between differing working contexts.

Key faculty and graduate student researchers associated with the Hydrogen Atom project: Professor JoAnn Kuchera-Morin, Professor Luca Peliti and Lance Putnam.

4. CONCLUSIONS AND FUTURE WORK

The Device Server provides abstractions and functionalities that have significantly eased the challenges of designing interactive applications for use in the AlloSphere. It allows interactive configurations to be easily defined and modified by both novice users and developers. It removes the burden of handling device drivers from application developers and allows them to offload control data processing away from rendering machines. The Device Server has also been used in performances and research outside the AlloSphere; we hope that it will be adopted by the greater media arts community and as such have released it as open-source software under the MIT license. It can be downloaded at http://www.allosphere.ucsb.edu/software.php.

Future work includes removing the unnecessary abstraction between devices and applications; both should be thought of simply as a collection of inputs and outputs. We would like to provide an option to use VRPN as the communication protocol in place of OSC as it is more familiar to VR researchers and computer scientists. We are interested in the possibility of a Device Server library that could be included in applications. This library would allow Lua configuration and scripting of control signals but would not require a server or network connection; device drivers would be included in the library as plugins. Finally, we would like to explore the possibilities of making the network more dynamically configurable akin to what is found in the previously mentioned SenseWorld DataNetwork and the Bridge projects.

5. ACKNOWLEDGMENTS

This worked was supported by a grant from the National Science Foundation CreativeIT program.

6. REFERENCES

[1] Gadgeteer:device driver authoring guide. 2007.

[2] X. Amatriain, J. Kuchera-Morin, T. Hollerer, and S. Pope. The AlloSphere: Immersive Multimedia for Scientific Discovery and Artistic Exploration (HTML). *IEEE MultiMedia*, 16(2).

[3] M. Baalman, H. Smoak, C. Salter, J. Malloch, and M. Wanderley. Sharing Data in Collaborative, Interactive Performances: the SenseWorld DataNetwork. In *Proceedings of the 2009 conference on New interfaces for musical expression*. Carnegie Mellon, 2009.

[4] O. Kreylos. Environment-independent VR development. In *Proceedings of the 4th International Symposium on Advances in Visual Computing*, page 912. Springer, 2008.

[5] J. Malloch, S. Sinclair, and M. Wanderley. From controller to sound: Tools for collaborative development of digital musical instruments. In *Proceedings of the 2007 International Computer Music Conference, Copenhagen, Denmark*, 2007.

[6] N. Mitani and L. Wyse. Information Sharing In Networked Music Application. In *Proceedings of the 2009 Australian Computer Music Conference*, 2009.

[7] M. Pall. The luajit project, 2008. http://luajit.org.

[8] R. Taylor, T. Hudson, A. Seeger, H. Weber, J. Juliano, and A. Helser. VRPN: a device-independent, network-transparent VR peripheral system. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 55–61. ACM New York, NY, USA, 2001.

[9] M. Wright and A. Freed. Open sound control: A new protocol for communicating with sound synthesizers. In *International Computer Music Conference*, pages 101–104. International Computer Music Association, 1997.

[10] M. Wright, A. Freed, A. Lee, T. Madden, and A. Momeni. Managing complexity with explicit mapping of gestures to sound control with osc. In *International Computer Music Conference*, pages 314–317. International Computer Music Association, 2001.