Unity Indie VRPN Adapter

Introduction (what is it?)

UIVA (Unity Indie VRPN Adapter) is a middle-ware between VRPN (Virtual Reality Peripheral Network) and the indie version of the popular Unity3D game engine. It enables Unity3D to talk to devices distributed at different machines via sockets, in an intuitive and relatively fast way.

Motivation (why is it necessary?)

You can ignore this part without any problem.

I was originally motivated to develop this small software by my research, as I was connecting more and more devices to the virtual environment, which is developed by **Unity3D**. I had a Wiimote, a WiiFit talking to the engine by C# socket, and two BPack accelerometers connected by serial port. All of them are connected to the same machine, by the same Bluetooth dongle. And they finally get tangled together, slowing down transmission rates and giving bad messages in an unpredictable way. In addition, initiating all devices each time before the "game" starts is a boring pain. Then I moved Wiimote and WiiFit to another host machine since they are based on sockets. Then I came to realize why not use **VRPN**, as it is so stable, fast and intuitive very well developed by the world famous VR guys (UNC Chapel Hill).

I started by searching for existing VRPN wrappers for Unity3D. And I found the **VRPNWrapper**, which is part of a bigger project called **UART**, developed by researcher in Georgia Tech. And they were so kind to share their code with me, which seems very nicely developed. However I didn't have a professional version to import native C++ DLL files (rumor said it works in indie version, I will simply get a warning message, but I never made it work). So unfortunately I had to do it myself. I then found **VrpnNet**, which is a .NET version of VRPN developed by Chris VanderKnyff. And I got an idea of building a C# middle-ware to talk to both VRPN and Unity3D. I quickly programmed the simplest prototype and tested with VRPN_Mouse, the roundtrip time of a message from the device (mouse) all the way to the Unity3D, which I worried most of this approach, seemed to be acceptable. That's when I decided to build the whole thing up.

The basic idea (how it works?)

The idea behind UIVA is very simple. As shown in Figure 1, UIVA has two parts. In the perspective of Unity3D, there is a UIVA_server and the UIVA_client. The client is a DLL file resides in the asset folder of an Unity3D game project, which exposes a couple of interfaces for any C# script that includes it to request data from certain devices. The server side has one or more VRPN_Remote objects for each device supported, and uses these to request data from the VRPN server on demand of the client side. So in essence, UIVA_server is a middle layer between VRPN and Unity3D which acts as a server for Unity3D and a client for VRPN server at the same time. When the connection is erected, Unity3D will poll the newest data from UIVA_server, which will in turn poll the newest data from VRPN.

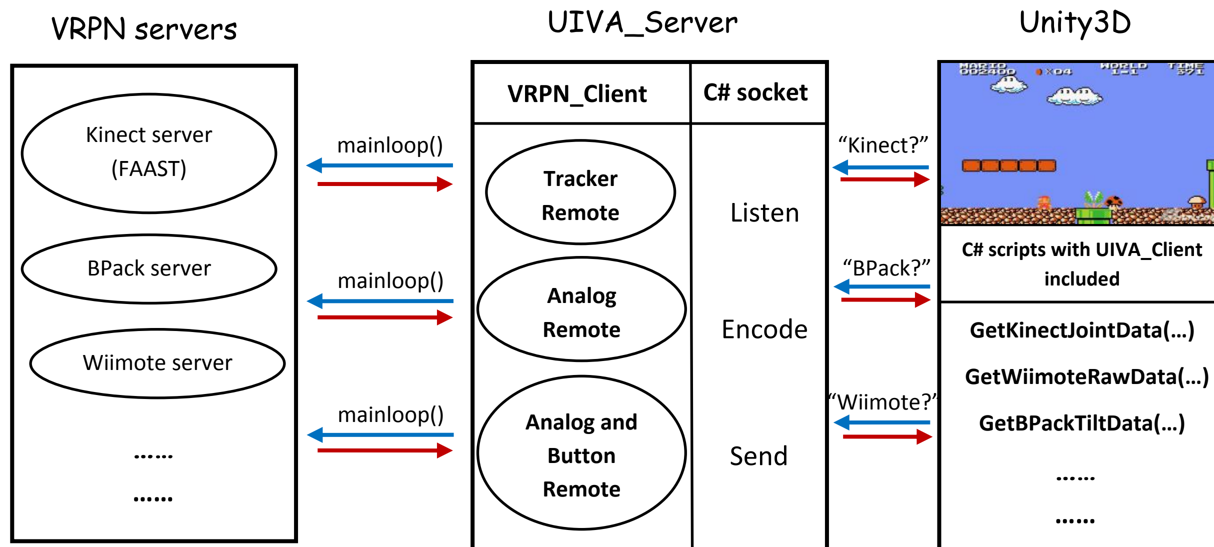

Figure 1. The UIVA framework

We will track the lifecycle of a Kinect data message as an example. When the current frame arrives in Unity3D, the C# script which includes UIVA_Client will execute its Update() function, which will in turn calls the function GetKinectJointData(int which, int jointIndex, out double[3] position, out double[3] quaterion) exposed by UIVA.dll to request the current sensor data of Kinect. In this function, UIVA_Client will send a message "Kinect?INDEX#?JOINT#?" to UIVA_Server first, and then parse the string response to get the sensor data. The UIVA_Server, on receiving this from UIVA_Client, locates the UIVA_Kinect object and calls its Update() function, which essentially calls the mainloop() function in the VRPN_Kinect_server (we are using FAAST here developed by Evan Suma from ICT, University of Southern California). The VRPN server will talk to the device to get the newest data, and then calls a callback function specified when the UIVA_Kinect object initiated a Tracker_Remote. On receiving this message, UIVA_Server encodes the data and sends it to UIVA_Client, and finally to Unity3D to update the game object. In Figure 1, the blue arrows indicate the request stream, while the red ones indicating the sensor data transmission.

Getting started (how to use it?)

UIVA is open source under the Boost Software License 1.0, same as VRPN. A SVN repository will be set up soon to allow users to add support to more devices. For now, a download link can be found at:

http://www.wpi.edu/~wangjia/UIVA.html

When you download and extract UIVA, you will find a "Release" folder and a "Source" folder, and this documentation. In both folders and their subfolders, there are README files that can warm you up really fast and easily. Basically, the sequence you should follow is:

Pair devices to machines→Start their VRPN servers→Setup UIVA_Server.cfg and run UIVA_Server.exe→ Copy UIVA.dll to your Unity game Asset folder → Write C# scripts to request data in Update() → Enjoy!

Performance analysis (how good is it?)

The main drawback of UIVA is that it added another layer between the device and the end user (Unity3D in this case). So the data transmission delay from the time the data is available in the server machine, to the time it is used in the engine, is of top concern of its performance evaluation. In addition, stableness and recoverability are also important as any server machine may go down when the system is running and the harm of this should be limited to minimum.

I am doing a lot of tests right now. And a technical report will be put on the web after I got the analysis results. Before that, it is always a beta version and I cannot guarantee that it will be able to run stably and effectively, and I won't say it will function on every computer.

There is a **special note** I want to make. As the name indicates, UIVA adapts VRPN to the indie version of the Unity3D game engine. However, it should (not tested) work in the Pro version of Unity3D as well. But personally I would suggest Unity3D Pro users to consider using the VRPNWrapper developed by Georgia Tech (part of the UART project) instead of UIVA because it wraps VRPN locally in Unity3D instead of adding an extra layer, which would generally have a higher data transmission rate. I cannot give a sure answer because I don't have the Pro version to test. And I would appreciate it very much if anyone can test it and let me know.

Support more devices (how to make it better?)

The current version of UIVA is UIVA_v1.0_beta. It supports the following devices.

Device	VRPN Server	VRPN Remote	Unity3D Interface
Microsoft Kinect	FAAST	VRPN_Tracker_Remote	GetKinectJointData(int which, int jointIndex, out double[] position, out double[] orientation);
Nintendo Wiimote controller	VRPN_Generic_Server (based on a modified version of WiiUse)	VRPN_Analog_Remote, VRPN_Button_Remote	1. GetWiimoteRawData(int which, out double accelX, out double accelY, out double accelZ, out string buttons); 2. GetWiimoteTiltData(int which, out double pitch, out double roll, out string buttons);
Nintendo Wii Fit balance board	VRPN_Generic_Server (as an extension of Wiimote)	VRPN_Analog_Remote	1. GetWiiFitRawData(int which, out double topLeft, out double topRight, out double bottomLeft, out double bottomRight); 2. GetWiiFitGravityCenterData(int which, out double gravX, out double gravY);
Wireless-T BPack accelerometer	VPRN_Generic_Server (based on VRPN_Analog_BPack.[Ch])	VRPN_Analog_Remote	1. GetBPackRawData(int which, out double accelX, out double accelY, out double accelZ); 2. GetBPackTiltData(int which, out double pitch, out double roll);

Note that to add the Wii Fit balance board support, I changed the Wiimote server class in the official VRPN release. And also I coded the BPack VRPN server by myself and it is not there in the official VRPN release either. However in the UIVA package, you can find them "Source/vrpn/vrpn". Or you can simply go to "Release/VRPN servers/" and run the servers executable files.

Adding support to other devices is super easy, since the whole UIVA thing is a simple piece of software. Please follow these steps:

1. You need to have a VRPN server for your device. This is not generally a problem, since VRPN already supports a ton of devices. And even if your device is not in the list, you can follow the guide at http://www.cs.unc.edu/Research/vrpn/Devices.html to write your own server. From my experience of writing the server for BPack, I suggest you to read an existing server of a device similar to your device and make your changes to a copy of it.
2. Open the UIVA_Server project in Visual Studio 2008, create a new class in "UIVA.cs" and name it UIVA_DeviceName. Add "double" variables for tracker and analog data, "string" variables for button data, "DateTime" variable for timestamps. Implement constructor, Update(), Encode() and several callback functions each of which bonded to a VRPN_Remote . I am not going any deeper now, because you will know how to do it 30 seconds after you read an existing UIVA _Device class, such as UIVA_Kinect or UIVA_Wiimote.
3. Open the configuration file of UIVA_Server (UIVA_Server.cfg), add a line in the format of "#DEV_DEVICENAME VRPN_Server_Name". The leading # sign is used to enable/disable the corresponding device in UIVA. For example, the line "DEV_WIIMOTE Wiimote0@localhost" tells the UIVA_Server to track a Wiimote from the local machine whose name is Wiimote0.
4. Get back to UIVA_Server, in the "Program.cs" file, do the following:
   (1) Add a List variable of your custom UIVA_DeviceName class and an integer member to keep track of how many such devices are going to be connected.
   (2) Add a case "DEV_DEVICENAME" same to what you added to the configuration file so UIVA knows what to do when it parses the file. In this case, initiate the UIVA_DeviceName object, push it to the List and increment the integer counter by 1. And then write a line to the console showing the tracking of that device is started.
   (3) Add a case "DeviceName" to the TalkToUnity() function. This switch block will be entered whenever UIVA_Server gets a request from UIVA_Client. For example, when the input string from the socket is "Kinect?1?14?", it removes the question mark and identifies that it is the first Kinect's joint 14 (right hand) data UIVA_Client is concerned about. So it checks if that Kinect is being tracked, if it is, update the VRPN_Server of it (essentially calls mainloop() in the VRPN_Server side), and forward the response message to UIVA_Client. Of course, you need to implement these codes for your custom device.
5. Rebuild the UIVA_Server solution.
6. Open the UIVA_Client project in Visual Studio 2008,
   (1) Add a GetYourDeviceData() function. Take a look at the similar functions of other supported devices and you will know what to write in your function. If your device has any accelerometer in it which gives you acceleration data on XYZ axes, there is a function called "ComputeTilt()" to help you compute the tilt value.
   (2) Uncomment Line 58 and set the project output type (right click the project and select "Properties") to "Console Application".
   (3) Rebuild the solution.
7. Test it by the following procedure:
   (1) Connect the device to a PC;
   (2) Open the "VRPN.cfg" file and enable its VRPN server if you haven't done so.
   (3) Start its VRPN server.
   (4) Open the "UIVA_Server.cfg" file and uncomment the "#" sign leading the line of your device if you haven't done so.
   (5) Start UIVA_Server.

(6) In the main function of UIVA_Client, call your GetYourDeviceData() function and print some data in the console window to see if it is working.

8. Change the output type of UIVA_Client back to "Class Library" and comment line 58. Rebuild it and copy the generated DLL file to your game asset folder in Unity3D.
9. Now you can include the UIVA_Client DLL file in any C# script in Unity3D and call the GetYourDeviceData() function to get your device data!
10. Cheers!

Acknowledge (special thanks)

Many thanks to:

1. Prof. Robert W. Lindeman, my academic advisor, for directing me, helping me and pushing me ☺ to finish this small project. He is an awesome advisor and such a nice guy!
2. The Unity team, for creating such an awesome engine, to make all game developers' and VR researchers' life so much easier.
3. The VRPN developers, for VRPN itself and specially VrpnNet (Chris VanderKnyff), of which UIVA is based on. VRPN is purely good stuff and every device fanatics should learn to master it.
4. Evan Suma at ICT of University of Southern California, for implementing the VRPN server of Kinect, and wrap it in FAAST. I was surprised of how fast it came out and how stable it is. It has been so delicious for hungry VR researchers and gamers with their Kinects rested in the corners.
5. Eric Griffith and Gerwin de Haan. Their project was so helpful when I was bumping my head against the wall knowing little of how to hack the VRPN Wiimote server. And of course, to the author of WiiUse library.
6. WiimoteLib author Brian. I put it in the Tools folder so everyone can use it to check the connections to Wii devices easily.
7. Finally, of course, to my wife Mi for the mental and physical support (great Chinese food always help) and WPI for all the facilities.

Author information

Jia Wang (wangjia@wpi.edu)
Human Interaction in Virtual Environments (HIVE) Lab
Department of Computer Science
Worcester Polytechnic Institute