Worlds on Your Desktop

making rich virtual worlds as simple documents

What if you could open an arbitrary, visually appealing and highly interactive virtual world quickly by just clicking a file? Edit it locally, save changes, and publish it on the net to run as a shared environment where anyone can log in? Add your own custom data and functionality using familiar scripting languages? You can! Using no-strings-attached open source software.

Contents

Introduction

RealXtend is an open source project aiming to speed up the development of standards for 3d virtual worlds. The idea is to apply standards like HTTP, Collada and XMPP and libraries such as the Ogre3d graphics engine, Qt GUI toolkit and the Opensimulator world server to build a generic application platform. It is a collaboration of several small companies that utilize the base technology in different application fields, but which coordinate the development of the common core together. The work culminates in a new virtual world application called Naali, the Finnish word for the arctic fox, referring to the Finnish origins of the project and the aim to make a generic platform for virtual worlds akin to Firefox for HTML based applications.

For users with no previous experience in virtual worlds, 3d or game programming, the tool allows easy reuse of premade models and scripts from libraries on the web. Any asset reference in RealXtend can be an URL, and the Naali GUI supports simple drag&drop of 3d models from web pages to the 3d scene. You can build your world simply in the local application, like you can use a web browser to view HTML files and test the functionality without needing to setup any servers. The full scene or a selected part of it can be saved to a file, for local use later or for sharing with others.

This is unlike Second Life where all creating happens on the servers, the client application is only an interface to the server side functionality. With Opensimulator people often run a SL compatible server locally to achieve this local building [opensim-on-a-stick]_. But as Opensim is currently SQL based it can't open documents quite as straightforwardly, as it has to import everything to a database first. Also with the SL viewer and OpenSim you always need to have those two applications running, whereas Naali can run standalone as well.

For developers the key in RealXtend, in contrast with the Second Life viewer and vanilla Opensimulator, is the modularity and extensibility. Naali is a modular application where essential parts like support for different scripting languages, networking protocols, UI elements or your own arbitrary module can be enabled/disabled without recompiling. Also it is licensed under the permissive BSD-style Apache license to allow e.g. game companies to ship their own versions based on it, whereas the SL viewer used to be

GPL and was recently switched to LGPL for the newer versions. LGPL does provide interesting possiblities now as it allows library linking -- perhaps Naali and SL viewer can share parts in the future. But no matter what the underlying technology is, what we propose here as a way to achieve extensibility on virtual world platforms is a simple generic model for scenes and applications.

Naali uses a so-called Entity-Component (EC) model for the scenes. Entities are simply identitities, with no data nor typing. They aggregate components, which can be of any kind and store arbitrary data. Applications can add their own components to have the data they need for their own functionality, the code that handles the data being in preinstalled custom modules or in scripts loaded at runtime as a part of the application data. The platform provides the basic functionality for all ECs: persistence, network synchronization among all the participants via a server and a GUI for manipulating components and their attributes. A scene is defined by the entities it has -- there is nothing hardcoded about them. This differs essentially from the current Opensimulator usage when using the Second Life (SL) protocol where the model is largely assumed and hardcoded in the applications. In that model there always is a certain kind of a terrain, a sky with a sun, and each client connection gets an avatar to which the controls are mapped.

We argue that there is no need to embed assumptions about the features of the world in the base platform and protocols. There already are rich useful very different virtual worlds: The open source Celestia universe simulator obviously does not have hardcoded land and sky, when you are moving from Earth to Moon and all the way to Andromeda. Teachers of medicine do not want anything extra around when they build a RealXtend world to teach anatomy by putting the organs to right places in a human body. Games typically require custom controls, and any application benefits from being able to define the UI exactly as fit for that putpose. To demonstrate the feasibility of a generic approach, there is a growing set of application examples in the Naali example scenes directory available on GitHub [naali-scenes]_. We present two of them here to illustrate how the EC model works in practice. First there is an implementation of a Second Life (tm) style avatar, implemented using a set of ECs and Javascript code to run both on the server and the clients to implement the functionality, for example to play back the walking animation as the avatar moves. This is achieved without the base platform nor the protocol having the concept of an avatar. The other example is a simple presentation application where we use custom data to share the presentation outline for all participants, and to let the presenter control the view for the others as the presentation proceeds.

Making of Avatars

The concept of an avatar characterizes virtual worlds -- VW functionality is often described by how the user is in the world as an avatar through which she can act. It is so central that technologies like the protocol used in Second Life assume it, the concept is hardcoded in the platform. We argue that it should not exist on the base level, to allow arbitrary applications to be built. However, a generic platform must of course allow the implementation of avatar functionality on the application level. Here we describe how it is achieved using the RealXtend Entity-Component-Action model.

Avatar means two things: 1) The visual appearance and the systems built around it, to for example modify the looks and add attachments like clothing and accessories, and use of animations for communication etc. 2) The functionality that when a user connects to a world server with a client, she gets the avatar object as the point of focus and control -- for example, the default inputs from arrow keys and the mouse are mapped to move and rotate the own avatar. Here, while covering the very basics for the visual appearance, the focus is on the latter control functionality.

The server side functionality to give every new client connection a designated avatar is implemented in a simple Javascript script, avatarapplication.js . Upon a new connection, it instanciates an avatar by creating a new entity and these components to it: Mesh for visible 3d model and associated skeleton for animations, Placeable for the entity to be positioned in the 3d scene, AnimationController to change and synchronize the animation states so that e.g. the walking animation is correctly played back while the avatar moves and finally a Script component which refers to another Javascript file which implements the functionality of a single avatar. Additionally, the main application script is also executed on the client, where it only adds a function to toggle between the default free look camera and new camera which follows the avatar.

The other script for an individual avatar, simpleavatar.js, adds a few more components: AvatarAppearance for the customizable looks, RigidBody for physics (collision detection) and on the client side an InputMapper for handling user control input. So called Entity Actions are used to make the avatar move according to the user controls. These actions are commands that any code can invoke on an entity, to be executed either locally in the same client or remotely on the server, or on all the connected peers. In this case the local code for avatar control sends for example the action "Move(forward)" to be executed on the server when the up-arrow is pressed. The built-in InputMapper component provides triggering actions based on input, so the avatar code only needs to register the mappings it wants. The server maintains a velocity vector for the avatar and applies physics for it. The resulting position is in the transform attribute of the component Placeable, which is automatically synchronized with the generic mechanism so the avatar moves on all clients. The server also sets the animation state to either "Stand" or "Walk" based on whether the avatar is moving. All participants run common animation update code to play back the walk animation while moving, calculating the correct speed from the velocity data from the physics on the server.

These two parts are enough for very basic avatar functionality to work. This proof of concept implementation totals in 369 lines of fairly simple Javascript code in the two files. The visual appearance is gotten from a pre-existing c++ written Avatar component, which reads an xml description with references to the base meshes used and individual morphing values set by the user in an editor.

One thing to note is that the division of work between the clients and the server described here is by no means the only possible one. The fact that we are using the same code to run both the server and the clients makes it fairly simple to reconfigure what is executed where. This model of clients sending commands only and server doing all the movement is identical to how the Second Life protocol works. It is suitable when trust and physics are centralized on a server. The drawback is that user control responsiveness may suffer from network lag. We are planning to later utilize the physics module in client mode too to allow movement code to run locally as well.

A Simple Presentation Tool

In contrast with the avatar functionality, let's consider a different kind of an application which does not require avatars at all. There are of course many virtual worlds and games without a single character as the locus of control: Map/geographic applications like Google Earth or astronomical simulations covering the whole universe like Celestia are about efficient navigation and time control of the whole space, not about moving own presence around. Game genres like real time strategy (RTS) feature fluent selection of the multiple units which the player commands, somewhat similarly to board games like chess where the player has a group of pieces of which can choose which to move. However, to get an even more different point of view to the user controls and actions in an application, the example here is not about navigating a view of a 3d space or spatial movement of units. Instead, let's design a slideshow like presentation application using the same entity-component-action building blocks as in the avatar example.

What do we need for a presentation?

The presentation tool gives the presenter means to control the position in the prepared material, for example to select the currently visible slide in a slideshow. In a local setting where everyone is in the same physical space it is simply about choosing what to show via the overhead projector which the audience sees. In a remote distributed setting there must be some system to get a shared view over the network, and that is the use case in this example.

There are several ways how the realXtend platform could be used to make a presentation. One is to put the material in the 3d space so that by navigating the view the presenter can focus on the different topics. The material could be simply 2d text and image slides, or web pages, on plates around the space when the 3d system would be just used as a spatial organization and navigation tool for the traditional slides. Or there could be animated 3d objects in the scene. Alternatively, the viewport could stay in place while the scene is changed -- e.g. by simply changing the slide on a virtual display in world, or animating the objects in the scene so that different items come to the stage during the course of the presentation. Also, the platform and the entity-component system is not limited to the 3d view: the 2d ui is also accessible for application scripts, so they can use the network synchronized entity attribute states and actions to make

shared gui views with the 2d widgets as well. The built-in voice capability and text chat can of course be used for the talk and communicating with the others.

No matter what techniques are used to make the presentation material, to run the show the presenter typically always needs the same basic controls: the default action to advance to next point, and alternatively to reverse back to previous or jump to an arbitrary one to for example when answering a question. In this example, let's map those to the arrow keys and the default act of proceeding to the next point also to spacebar and the left mouse button. So here we diverge from e.g. the defaults in the Second Life (tm) client, where arrow keys are used to move and turn the avatar, spacebar makes the avatar jump or start flying, and the left mouse button over the 3d view triggers the possible default action on the object in the scene under the mouse cursor (the avatar touches it). SL is often used for presentations, but typically so that the default avatar controls stay and in-world buttons are used to control the slideshow. That is, pressing arrow keys just rotates or moves your avatar, and you must instead hit the right 3d object in the scene with a mouseclick to control the presentation. We argue that customizing the controls to best support the task at hand is essential, and that the application designer should be given the choice whether to include e.g. the avatar functionality or not.

Synchronizing the view, getting the presentation control actually executed when the presenter presses the keys, is now the remaining area to be designed. In the distributed system one question is where to execute what. One option is to handle the controls locally in the presenter's client: listen to key and mouse input there, manipulate the scene accordingly and get it shared via the generic scene synchronization mechanism which is there by default. For example if the slides are published on the web, the presentation script can just have a list of URLs and change the current one to be used on a webview in the scene. When running this locally in the presenter's client, no server or other participants actually need to know anything about the presentation -- there is no need for a shared presentation application among the participants. It is just a custom tool that the presenter uses to manipulate the basic scene. Naali comes by default with a built-in WebView component which implements showing html pages from the web, and the generic attribute synchronization mechanism shares the URL changes, so the other participants get to see the slide changes.

However, sharing the custom presentation functionality and the data among the participants would enable useful features for the audience. An outline view could show hilight the current position. Participants could browse the material freely in an additional view beside the one the presenter controls. Showing the outline and hilighting the current position there could be done from the presenter's client, for example by simply adding another element to the scene next to the slide display. But it might be simpler when the data is shared, and the free browsing feature is certainly simplest when every participant has the presentation data.

Implementation

So let's add a new entity, and call it "presentation". Entities are just identities which aggregate components. For the simple technique, showing web pages, we need a few basic ones: Placeable to have something in the scene, Mesh to have geometry (e.g. a plane) on which to show the slides, and WebView to render html from URLs. And two additional ones for our custom functionality: a DynamicComponent to hold and sync our custom data, and Script to implement the custom UI handling and presentation controls. As data we need a list or URLs and an index number for the current position, so the DynamicComponent needs two attributes: a stringlist called "slides" and an integer called "index". This custom data becomes part of the scene data and is automatically stored and synchronized among the participants. The Script component is a reference to Javascript or Python code which implements all the logic.

To handle the user input, we have two options: either handle input events and modify the state correspondingly directly in the application code, or use the InputMapper component to trigger entity actions like in the avatar example. Let's use actions again so can use the server as a broker for security, and to get a similar design to compare with the avatar example. So client side code maps right-arrow and spacebar keys to SetPresentationPos(index+1) etc. Server can then check if the caller has permissions to do that action, for example we can make it so that in presentation mode only the designated presenter is allowed to control the shared view. Then if the presentation material is left in the scene for anyone to visit at other times, control can be freed for anyone. Changes to the index attribute are synchronized for all participants so they can update the outline GUI accordingly.

The simplest way to get a shared view is to set the camera in this application to a fixed position. If we then use a simple plane to show the slides, it suffices for the server to change the current one on that object for everyone to get to see it in sync. For the outline view, the code can add a 2d panel with thumbnails of all the slides and hilight the current one. For free browsing, clicking on a thumbnail can open a new window with that slide, while the main presentation view remains. This we way have a simple, complete presentation application implemented on top of a generic virtual worlds platform.

Does this make sense?

Of course just getting a shared view of a set of web pages could be, and is, easily deployed without RealXtend technology just by using regular web browsers with HTML+Javascript and some server backend. The idea here is to illustrate the use of Entity-Components and automatic attribute synchronization for custom functionality in a minimal complete example. Real benefits of the platform would be utilized if the actual presentation material was interactive 3d objects, such as alternative house or car designs, which the presenter would go thru and manipulate during the talk. Basically what we are aiming at is making interactive multiuser 3d applications easy to author, comparable to how single user web pages are today.

It is, however, interesting to consider whether the platform makes implementing this kind of features so easy that it makes sense even for things that could be made to work in regular web browsers too. (W.I.P. NOTE: I think I'll have to implement this example to be able to judge that, is not done yet, just the spec here!). This is relevant also because we are writing an experimental WebNaali client using WebSockets and WebGL for 3d scene use, utilizing the same entity-component attribute and action mechanisms.

Discussion

Related work

We are certainly not the first to propose genericity to virtual world base architectures. For example in the NPSNET-V work, extreme extensibility is achieved by the whole system being built around a minimal microkernel on which arbitrary code can be added at runtime using the mechanisms in the Java programming language [NPSNET-V]_. A contemporary example is the Meru architecture from the Sirikata project, where a space server only knows the locations of the objects. Separate object hosts, either running on the same server or any client / peer, can run arbitrary code to implement the objects in the federated world [sirikata-scaling]_. Messaging is used exclusively for all object interactions [sirikata_scripting]_. The idea with the Entity-Component mechanism here is, instead, to lessen the need to invent own protocols for all networked application behaviour when for simple usage can just use the automatically synchronized attributes. In preliminary talks with some Sirikata developers we concluded that they want to keep base level clean from such high level functionality, but that things like the attribute autosync would be desireable in application level support scripts.

The aggregation, not inheritance, using EC model was adopted from game engine literature [ec-links]_. Running

What differentiates RealXtend Naali now is the combination of relative maturity, simplicity, power and the permissive open source license. It is already being deployed to customers by some of the development companies, and provides a powerful usable GUI for editing the component data also for your own custom components. It is a quite straightforward modular c++ application with optional Python and Javascript support. Thanks to the Ogre3d graphics engine, it runs both on e.g. the N900 mobile phone with OpenGL ES and on powerful PCs with multiple video outputs with the built-in CAVE rendering support.

The document oriented approach of having worlds as files is of course precented in 3d file format standards like VRML, X3D and Collada. The idea with the RealXtend files is to not specify the contents of the files, but they are only a mechanism for the applications to put the component data that they need. An essential element are the script references that implement the functionality of the applications, similar to how HTML documents have Javascript references. But also this is not specified in the file format, it is just how the bundled Script component works. For static content, we support using e.g. Collada assets directly. (W.I.P NOTE: check how x3d and friends do scripting).

Status of implementations

The generic Entity-Component approach was proposed to Opensimulator core and accepted as the plan already in December 2009 [adam-ecplan]_. The implementation is however still in very early stages, only the first steps have been taken to allow refactoring the framework be generalized and the features built with ECs in optional modules. It can be used, however, with the Naali client application both when running against Opensim using the RealXtend add-on module (the combination of opensim+modrex is called Taiga). This works in a limited fashion, as the Second Life protocol and OpenSim internals still assume the hardcoded SL model, but you can still add arbitrary client side functionality and have the data automatically stored and synchronized over the net.

The generic application platform works currently fully when using the so called Tundra server, which a simple server module added to Naali itself. This allows Naali to run as standalone for local authoring, or for single user applications, but also for using it as a server to host worlds on the net instead of using Opensimulator. With Tundra LLUDP is no longer used, but all basic functionality is achieved with the generic EC synchronization. For the transport layer, we are using a new protocol called kNet which can run either on top of UDP or TCP. kNet is similar to eNet but performed better in tests with regards to flow control. The Tundra server lacks many basic features and may never get some of the advanced Opensimulator features, like running untrusted user authored scripts and combining multiple regions to form a large grid. Tundra is however is already useful for local authoring and deploying applications like simple games to production use. And it serves as an example of how a generic approach to allow virtual worlds functionality can be simple yet practical. We hope this is taken into consideration in upcoming standardization processes, for example if VWRAP proceeds to address in-world scene functionality.

References

[opensim-on-a-stick]
http://becunningandfulloftricks.com/2010/10/07/a-virtual-world-in-my-hands-running-opensim-and-imprudence-on-a-usb-k
[naali-scenes] https://github.com/realXtend/naali/blob/tundra/bin/scenes/ [adam-ecplan]