Worlds on Your Desktop

simple yet powerful extensibility for virtual worlds

What if you could edit a visually appealing and highly interactive virtual world just like you edit traditional files? Change them locally, save multiple versions, then publish them on the net as shared environments where anyone can log in? Add your own custom data and functionality using familiar scripting languages? You can already do all of this, using fully open source software.

Introduction

RealXtend is an open source project aiming to speed up the development of standards for 3D virtual worlds. We leverage standards including HTTP, COLLADA, XMPP and open source software such as OGRE 3D, Qt, OpenSimulator, and Blender. Beginning as a collaboration of several small companies that utilize the base technology in different application fields, but which coordinate the development of the common code base together. This has culminated 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 goal 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. A virtual world can be snapped together like Lego bricks, instantly viewed, its a simple and fun process for users of all ages. Editing can be done locally, and the creation published later. This is in contrast to Second Life (tm) (SL) where all edits and additions happen on the servers -- the client application being no more than an interface to server side functionality. Naali can run completely standalone, without the complexity of setting up a separate server for local editing. With OpenSimulator people often run a SL compatible server locally to achieve this local building [opensim-on-a-stick].

Developers will find Naali to be highly extensible with dynamically loadable/unloadable modules. For example, an entirely new scripting language could be loaded as a module without needing to recompile Naali. Naali uses the Apache license so is permissive for businesses to create commerical software based on it. In contrast again to the Second Life viewer which only recently was relicenced partially from GPL to LGPL, with a single company owning and controlling the copyright. The main point of this article, however, is independent of particular viewer and server implementations, but instead about the extensible scene model we have now started using.

Naali uses the so-called Entity-Component (EC) model as a basis to construct extensible scenes. The model was adopted from contemporary game engine architectures [ec-links]. Entities are simply unique identitities, with no data or typing. They aggregate components, which can be of any type and store arbitrary data. Applications built using Naali can add their own components to have the data they need for their own functionality. The code that handles the data exists in preinstalled custom modules or in scripts loaded at runtime as a part of the application data.

The Naali platform provides the basic functionality for all ECs: persistence, network synchronization among all the participants via a server and a user interface for manipulating components and their attributes. In addition, Naali introduces a new concept called "Entity Actions", which are a simple form of remote procedure call (RPC). These are demonstrated in two examples later in this article.

A scene is defined by the entities it has -- there is nothing hardcoded about them at the platform level. This differs essentially from the current OpenSimulator paradigm when using the SL protocol -- where the model is largely predefined and hardcoded in the platform: 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 [VWRAP]. We argue that there is no need to embed assumptions about the features of the world in the base platform and protocols.

There already exists many examples that prove our point, one is the open source Celestia universe simulator that does not have any hardcoded land or sky. Naali is a true platform that does not get in the

way of the application developer; they can create anything from a medical simulator for teachers, to action packed networked games - and always with a custom interface that exactly fits the application's purpose. Rather then being in control of a single avatar, you can for example create a world where the user is an entity controlling the weather conditions of the whole environment, and make a game around that. And ask your friend on-line to test it, using the standard client she already has. The solution is inspired by web browsers, which download code from the websites to run custom user interfaces and logic in the client.

To demonstrate the feasibility of this 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 below to illustrate how the EC model works in practice. First there is an implementation of a SL-like avatar, implemented using a set of pre-existing generic ECs and specific Javascript code that run both on the server and the clients. The second 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 -- they are often described by how the user is in the world as an avatar. The protocol used in Second Life assumes avatars, it 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. Here we describe a proof of concept implementation using the realXtend Entity-Component-Action model. The full source code is available at [tundra-avatar], and a parts of it are included below.

Avatar functionality is split in two aspects here: 1) The visual appearance and related functionality to modify the looks and clothing, and use of animations for communication etc. 2) The model where every user connection is given a single entity as the point of focus and control. The default inputs from arrow keys and the mouse are mapped to move and rotate the own avatar. Here, while covering the basics for the 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 the 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 and finally a Script component to implement the functionality of a single avatar. Additionally, the main application script is also executed on the client, where it adds a new camera which follows the avatar and a keybinding to toggle between camera modes.

Handling new client connections on the server:

```
function serverHandleUserConnected(connectionID, userconnection) {
    var avatarEntity = scene.CreateEntity(scene.NextFreeId(),
                        ["EC_Script", "EC_Placeable", "EC_AnimationController"]);
    avatarEntity.Name = "Avatar" + connectionID;
    avatarEntity.Description = userconnection.GetProperty("username");
    avatarEntity.script.ref = "simpleavatar.js";

    // Set random starting position for avatar
    var transform = avatarEntity.placeable.transform;
    transform.pos.x = (Math.random() - 0.5) * avatar_area_size + avatar_area_x;
    transform.pos.y = (Math.random() - 0.5) * avatar_area_size + avatar_area_y;
    transform.pos.z = avatar_area_z;
    avatarEntity.placeable.transform = transform;
}
```

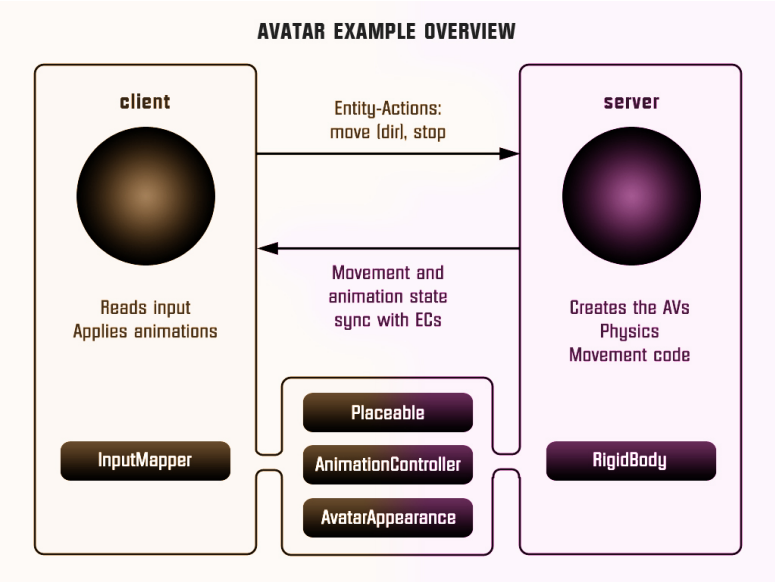
The other script for an individual avatar, simpleavatar.js, adds a few more components: AvatarAppearance for the customizable looks, RigidBody for physics and on the client side an InputMapper for user input. Entity actions are used to make the avatar move according to the user

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. Here the local code 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.

Updating animations, the common code executed both on the client and the server:

```
function commonUpdateAnimation(frametime) {
    var animcontroller = me.animationcontroller;
    var animname = animcontroller.animationState;
    if (animname != "")
        animcontroller.EnableExclusiveAnimation(animname, true, 0.25, 0.25, false);
    // If walk animation is playing, adjust speed according to the rigidbody velocity
    if (animcontroller.IsAnimationActive("Walk")) {
        // Note: on client the rigidbody does not exist,
        // so the velocity is only a replicated attribute
        var vel = me.rigidbody.linearVelocity;
        var walkspeed = Math.sqrt(vel.x * vel.x + vel.y * vel.y) * walk_anim_speed;
        animcontroller.SetAnimationSpeed("Walk", walkspeed);
    }
}
```

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 AvatarAppearance component, which reads an xml description with references to the base meshes used and individual morphing values set by the user in an editor. It uses the realXtend avatar model which was implemented already in 2008 for the first prototype which did not have the entity component system at all, and is used in this demo as is. A more generic and customizable appearance system could be implemented with the ECs, but that is outside the scope of the demo and description here.



The parts of the avatar example:

Symbol	Meaning
Colors brown/purple	Client / Server respectively
Arrows	Network messages
Filled boxes	ECs on client, server or shared by both

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.

With the ability to run custom code also in the client, it is easy to extend avatar related functionality. For example, in one project for schools we added the capability to simply carry objects around as the most simple means for 3D editing. Another possibility is to add more data that is synchronized for animations, even the full skeleton for motion capture or machine vision based mapping of the real body to the avatar pose.

A Simple Presentation Tool

In contrast with the avatar functionality, let's consider an application without avatars at all. Many virtual worlds and games do not have a single character as the locus of control: map applications or astronomical simulations 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 controlling several units, similarly to board games like chess. To get an even more different view to the user controls, 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. 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. The 3d space would be 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 by moving different objects to the view 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 with 2D widgets. The built-in voice capability and text chat can be used for the talk and communicating with the others.

No matter how the presentation view is made, the presenter typically needs the same controls: the default action to advance to the next point, and alternatively to reverse back to the previous one or jump to an arbitrary one. 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. Here we diverge from e.g. the defaults in the Second Life client, where the 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. Moreover, the application designer should be given the choice whether to include avatars 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. Both Naali and the Second Life viewer support showing web pages by default.

However, sharing the custom presentation functionality and the data among the participants would enable useful features for the audience. An outline view could highlight the current position. Participants could browse the material freely in an additional view beside the one the presenter controls. The visual side of showing the outline and hilighting the current position there could be done from the presenter's client, simply by adding more elements to the shared scene. But it might be simpler when the data is shared, and certainly for free individual browsing, having the data in each client is more powerful.

Implementation

The simplest way to get a shared, synchronized view of the presentation slides is to use a static camera which shows a single webpage view. It then suffices for the server to change the current page on that object for everyone to see it. We could do this with a 2D widget, but let's use a 3D scene to illustrate the extensibility there.

So, let's add a new entity called "presentation". For showing web pages, we need a few basic components: 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 for custom data, and a Script to implement the UI and presentation controls. As data we need a list or URLs and an index number for the current position. 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 client code, or send remote actions like in the avatar example. Let's use remote 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 in presentation mode only the designated presenter is allowed to change the shared view. Then if the presentation material is left in the scene for visiting later, control can be freed for anyone. The index attribute is synchronized for all participants so the outline GUI can update accordingly.

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 world platform.

Does this make sense?

Of course just getting a shared view of a set of web pages could 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 ECs 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 animated 3D objects, such as alternative house or car designs, which the presenter would go through and manipulate during the talk. Basically what we are aiming at is making interactive multi-user 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 for things that could work in regular web browsers. This is relevant also because we are writing an experimental WebNaali client which does entity attribute sync and actions over WebSockets. For 3D view it uses WebGL, but for 2D applications the networking part could be used without WebGL as well.

Two Naali clients by the presentation stage of the TOY system, an open source learning environment for the Future School of Finland project. The one on the left just added a web page to the stage, and is currently carrying the object.

Discussion

Related work

We are certainly not the first to propose genericity to virtual world base architectures. For example in NPSNET-V, the system is a minimal microkernel on which arbitrary code can be added at runtime using the the Java virtual machine [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 in Naali is, instead, to lessen the need to invent own protocols for all networked application behaviour when for simple usage using the automatically synchronized attributes suffices. 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 the same Javascript code partially both on the server and clients is identical to a gaming oriented virtual world platform called Syntensity [syntensity]. Also in Syntensity you compose own entities by declaring what so-called StateVariables they have. The data is then automatically synchronized among all participants, they are basically identical with the Naali EC attributes. The Naali implementation is inspired by Syntensity. The difference is that in Syntensity the entities exists on the scripting level only, and the basic functionality like object movements is hardcoded in the Sauerbraten/Cube2 first person shooter platform. In Naali everything is now made with the ECs only, so the same tools work for e.g. graphical editing, persistentence and network sync identically for all data.

The document oriented approach of having worlds as files is of course precedented in 3D file format standards like VRML, X3D and COLLADA. Unlike those, the realXtend files do not have 3D geometry, but describe a scene by referring to external assets, for example meshes in the COLLADA format. Essentially they are a mechanism for also application specific custom data, which is automatically syncronized over the net. They have 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.

The realXtend platform is currently by no means a complete solution for all the problem of virtual world architectures. Naali does not currently address scaling at all, nor is federated content from several possible untrusted sources supported. We started by having power in the small scale, ability to easily make rich interactive applications. In the future, we look forward to continuing collaboration with e.g. the OpenSimulator and Sirikata communities to address the trust and scalability issues. Opensim is already used to host large grids by numerous people, and the architecture in Sirikata seems promising for the long run [sirikata-scaling] [sirikata-scaling2].

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 for the core is however still in very early stages. It can be used, however, with the Naali client when running OpenSimulator with the realXtend (the combination is called Taiga). This works in a limited fashion, as the Second Life protocol and OpenSimulator internals still assume the hardcoded SL model, but you can add arbitrary client side functionality and have the data automatically stored and synchronized over the net via OpenSimulator.

The generic application platform works currently fully when using the so called Tundra server, which a simple server module added to Naali itself [tundraproject]. This allows Naali to run as standalone for local authoring, or for single user applications, but also for using it as a server instead of using OpenSimulator. With Tundra LLUDP is not used, all basic functionality is achieved with the generic EC synchronization. For the transport, we are using a new protocol called kNet which can run either on top of UDP or TCP [knet]. 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 already useful for local authoring and deploying applications with custom functionality. And it serves as an example of how a generic approach to virtual worlds functionality can be simple yet practical.

Regarding the status of the Naali application overall, it is maturing and has already been deployed to customers by some of the development companies. It is a quite straightforward modular C++ application with optional Python and Javascript support. Qt object metadata system is utilized to expose the C++ internals automatically. This covers all modules like the renderer and the UI, and all the ECs. The QtScript library provides this for Javascript support, and PythonQt does the same for Python. There is also QtLua with which Lua support could be easily added. Thanks to the Ogre3D graphics engine, Naali 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. There is also an experimental WebNaali client, written in Javascript to run in a web browser, doing the EC sync over WebSockets and rendering with WebGL.

The most severe missing piece in our current EC synchronization is the lack of security, for example a permission system. A first implementation is probably made soon to cover the basics, similarily to how Syntensity already has attributes that can only change if the server allows.

Conclusion

We have demonstrated how a generic approach to virtual worlds can be simple and practical, yet powerful and truly extensible. We hope this is taken into consideration in upcoming standardization processes, for example if VWRAP proceeds to address in-world scene functionality. In any case, we will continue to develop the platform and applications on top of it. Anyone is free to use it for their needs, and to participate in the development which is mostly coordinated on-line.

References

NPSNET-V	Andrzej Kapolka, Don McGregor, and Michael Capps. 2002. A unified component framework for dynamically extensible virtual environments. In Proceedings of the 4th international conference on Collaborative virtual environments (CVE '02). ACM, New York, NY, USA, 64-71. DOI=10.1145/571878.571889 http://doi.acm.org/10.1145/571878.571889
opensim-on-a-stick	http://becunningandfulloftricks.com/2010/10/07/a-virtual-world-in-my-hands-running-opensim-and-imprudence-on-a-usb-key/
naali-scenes	https://github.com/realXtend/naali/blob/tundra/bin/scenes/
tundra-avatar	Application XML and usage info at https://github.com/realXtend/naali/tree/tundra/bin/scenes/Avatar/ , Javascript sources in https://github.com/realXtend/naali/tree/tundra/bin/jsmodules/avatar/
adam-ecplan	Adam Frisby on Opensim-dev, Refactoring SceneObjectGroup - Introducing Components. The plan PDF is attached in the email, http://lists.berlios.de/pipermail/opensim-dev/2009-December/008098.html
VWRAP	Joshua Bell, Morgaine Dinova, David Levine, "VWRAP for Virtual Worlds Interoperability," IEEE Internet Computing, pp. 73-77, January/February, 2010
sirikata-scaling(1, 2)	Daniel Horn, Ewen Cheslack-Postava, Tahir Azim, Michael J. Freedman, Philip Levis, "Scaling Virtual Worlds with a Physical Metaphor", IEEE Pervasive Computing, vol. 8, no. 3, pp. 50-54, July-Sept. 2009, doi:10.1109/MPRV.2009.54 http://www.cs.princeton.edu/~mfreed/docs/vworlds-ieee09.pdf
sirikata-scaling2	Daniel Horn, Ewen Cheslack-Postava, Behram F.T. Mistree, Tahir Azim, Jeff Terrace , Michael J. Freedman, Philip Levis "To Infinity and Not Beyond: Scaling Communication in Virtual Worlds with Meru." http://hci.stanford.edu/cstr/reports/2010-01.pdf
sirikata_scripting	Bhupesh Chandra, Ewen Cheslack-Postava, Behram F. T. Mistree, Philip Levis, and David Gay. "Emerson: Scripting for Federated Virtual Worlds", Proceedings of the 15th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games (CGAMES 2010 USA). http://sing.stanford.edu/pubs/cgames10.pdf
ec-links(1, 2)	Mick West, Evolve Your Hierarchy -- Refactoring Game Entities with Components http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/
syntensity	http://www.syntensity.com/
knet	http://bitbucket.org/clb/knet/
tundraproject	http://realxtend.blogspot.com/2010/11/tundra-project.html