An Entity-Component model for Extensible Virtual Worlds

Abstract

We propose a model with basic building blocks on which arbitrary virtual worlds applications can be built. It consists of Entities which can have any set of Components attached, and application developers can define own components to add arbitrary data that they need for the specific functionality. The components are basically just data stored in the Attributes of the component. The values of the attributes can be automatically synchronized among the participants in the networked environment.

Additionally we present the idea of Entity Actions. ...
http://groups.google.com/group/realxtend-dev/browse_thread/thread/42fed16befd2b9b7/bba9f67d00b60371?lnk=gst&q=

Introduction

A virtual world can be potentially anything. An anatomical simulation of the internals of a biological body, where you can fly around inspecting the functionality of various organs and blood vessels etc. has certain kind of entities, and astrological simulations or solar systems and galaxies has quite different ones. A game of hockey has the concepts of attack, neutral and defending zones, certain conventions for which kind of physical interactions between the humans in the game are allowed, and of course the notions of a goal and a puck. Figure skating has none of these, even though the physics are identical -- a skating ring with people skating. The user interface controls and functionality in those two skating games are largely different, for example in the team based hockey you want good controls for passing the puck to team mates, whereas in figure skating it might be something like elaborate controls for the jumps and the skating itself, and higher level selections for how to proceed in the pre-planned choreography of the performance.

(All these cases are easily virtual worlds, without streching the definition. We could go even further to define abstract worlds with arbitrary behaviours, like a "world" where in one part there is a space where text characters can exist, and another area where there are buttons with actions like load and save. A text editor application, that is. In fact, one definition of a computer is that it is a virtual world generator [Deutch]_, so following that perhaps would make all software virtual worlds. This line of thinking might not be fruitful, but the extreme genericity is something to concider still. For example in games not all geometries are euclidian, perhaps VW archs should not hardcode that.)

What should a generic virtual world architecture provide to facilitate the implementation of all those, and many quite different, applications? It seems we need both arbitrary custom data and functionality. For example in the astronomical simulation the data would be mass and heat and other properties of the planets and suns. In a functional anatomy simulation you might need chemical information (...). To be able the fluently control and navigate these vastly different applications, specific user interface controls would be useful. In the skating games, the custom user interface actions would need to be tightly coupled with how the character animations (e.g. a slapshot or a (kolmoishienohyppy)) are executed. All have common needs too: need to represent visual 3d objects, move and animate them, and to synchronize all the data among the participants and to store it for continuing use.

We propose an architecture that meets these requirements using three main elements:

1) A scene model based on the notion of a generic entity, to which any kind of components which can contain arbitrary data can be added.

2) A core API for basic functionality like showing graphics and getting user input, and manipulating the scene to e.g. create new entities. And a module and scripting system that can be used to add arbitrary code that uses the core API to be executed in all parts of the system (e.g. the client and server).

3) An extensible network protocol suitable for real time use, which allows applications to define own custom messages.

This has been implemented within the open source realXtend project next generation architecture effort, resulting in a generic virtual world software framework in the Naali application. Naali was first developed as a viewer client application, to replace the previous realXtend viewer prototype which was based on the Linden labs Second Life (tm) viewer. The server counterpart is Opensimulator (or Opensim), a modular open source application for generic virtual worlds which features a SL (tm) compatible server implementation out of the box but targets supporting other kinds too. The original realXtend functionality(*) on the server side was already implemented as an Opensim module (ModRex), and we have now added the support for arbitrary scene data using the Entity-Component model there too. The opensim+modrex combination, the counterpart of Naali(*) is called Taiga. Naali is written in C++ and uses the Ogre3d graphics rendering engine and the Qt toolkit for UI and internal architecture like supporting dynamic languages (currently Python and Javascript, both as optional modules, and more can be added in additional modules).

(* original realXtend functionality, first published in February 2008: Ogre meshes with animations and complex materials, mediaurls ('web pages as textures'), ..)

(* Naali is the Finnish name for the arctic fox -- Naali strives to be a generic 3d web client app, the Firefox of virtual worlds)

Now (since October 2010) Naali has also the capability to create a scene by itself and optionally run a simple server module, so it can run standalone also without Opensim and provides an alternative server implementation using a different, more efficient and featureful extensible protocol than the LLUDP provided by current Opensim. This new protocol & server experiment is called Tundra, as an alternative for the Opensim based Taiga, and it uses a new networking library called kNet (or Kristalli) which is similar to eNet but with certain advantages. If the Tundra experiment proves succesfull, and there are use cases that need both the kNet protocol and Opensim features, one possibility is to add a kNet client stack to Opensim. The Naali client works against both server implementations, and for application code that e.g. defines new Entity-Components and uses existing ones, the protocol doesn't show as the data is gotten and manipulated with the same protocol agnostic API. Tundra uses ECs extensively to define everything -- there is nothing hardcoded about what a world likes, unlike with LLUDP where the Second Life (tm) application is hardcoded: every user has an avatar, there is a single certain kind of terrain etc. With the new generic realXtend architecture demonstrated with Tundra that actually becomes data: a description of a certain kind of a scene with default entities.

This article is structured as follows:

First there is an overview of related work. This covers analyses of pre-existing virtual worlds solutions like Second Life and the vanilla Opensimulator server and the original realXtend prototype made using those. Then an overview of literature about solving the extensibility problem, suggesting the Entity-Component model, based on which we selected that for the new realXtend architecture. We also take a look at other related systems utilising ECs and custom networking, especially the proprietary Unity3d game engine and an open source virtual worlds framework called Syntensity.

Then we present the realXtend architecture in more detail, the designs of the three main elements: the extensible Entity-Component scene model, the core API and the extensible network protocol. Using these to make custom applications is then illustrated with simple highlevel descriptions of some examples. One of the examples is a reproducing the basics of the Second Life (tm) like featureset using the generic API. Other examples feature different worlds, ones without avatars nor land nor controls to move single entities around.

Chapter X is a brief overview of the implementation, covering the main technologies used in Naali: Ogre, Qt and kNet. How these were used to implement the ECs and the core API. (also the module system etc. of course).

(something more still? then discussion/evaluation, future roadmap, conclusion)

Related work

This covers analyses of pre-existing virtual worlds solutions like Second Life and the vanilla Opensimulator server and the original realXtend prototype made using those. Then an overview of literature about solving the extensibility problem, suggesting the Entity-Component model, based on which we selected that for the new realXtend architecture. We also take a look at other related systems utilising ECs and custom networking, especially the proprietary Unity3d game engine and an open source virtual worlds framework called Syntensity.

Architecture design

1) A scene model based on the notion of a generic entity, to which any kind of components which can contain arbitrary data can be added.

2) A core API for basic functionality like showing graphics and getting user input, and manipulating the scene to e.g. create new entities. And a module and scripting system that can be used to add arbitrary code that uses the core API to be executed in all parts of the system (e.g. the client and server).

3) An extensible network protocol suitable for real time use, which allows applications to define own custom messages. http://clb.demon.fi/knet/index.html

Functionality example

Avatar

...

Presentation

(the way I started writing the pres example ended so long that putting it to a separate page now, [[Platform Extensibility Working Group/Presentation Application]]

The Naali implementation

a brief overview of the implementation, covering the main technologies used in Naali: Ogre, Qt and kNet. How these were used to implement the ECs and the core API. (also the module system etc. of course).

Discussion: evaluation, future work

Conclusion

References

- A unified component framework for dynamically extensible virtual environments, http://portal.acm.org/citation.cfm?id=571878.571889&type=series
- This seems like one to check for the article, and open source project by some uni with publications,

http://oldsite.vrjuggler.org/papers.php has things like "VR Juggler -- An Open Source Platform for Virtual Reality Applications" latest release is from march 2008, i guess the project is still living. the full proceedings of IEEE VR from the past couple of years would be the thing to check .. that one was there in 2001

- the bimonthly presence journal seems relevant, had this framework thing in 2003 .. has been cited this year too: http://www.mitpressjournals.org/doi/abs/10.1162/105474603763835314