OpenSimulator Virtual World Server Case Study

Introduction
The use of virtual worlds for training, entertainment, and collaboration is growing. These present to a user a shared three-dimensional space where interactions with the virtual world and other users are possible. There has been a lot of work focusing on the design, architecture, and performance of the client, but looking end-to-end at the whole system that supports many users, the server is a significant component to the quality of the user’s experience.

Intel Labs has been researching the scalability of virtual worlds. To meet the increasing demand of rich user experiences, high level of realism, and new usages such as experiencing professional sports games in virtual auditoriums, virtual worlds must scale beyond their current capabilities. The goal of future virtual world architectures is thousands of interacting avatars in a world rich with millions of objects. This goal requires architectural changes to the client, the network, and the virtual world server.

This is the first of several articles exploring some of the design, performance, and execution features of virtual world servers. An open source virtual world simulator is used as an example, and design requirements are deduced from its operation. This first article describes OpenSimulator’s design, and future articles will explore specific facets such as threading design and power savings.

OpenSimulator
OpenSimulator (http://opensimulator.org) is an open source project which is building a 3D virtual world simulator where users can interact and dynamically build the content of the world. Similar to systems like Second Life® or Blue Mars®, users become avatars that move about, chat, collaborate, and play games in a 3D environment created by independent authors. This creates a diverse world where regions and cities differ in architecture, tone, and game play.

The basic operation of the OpenSimulator server is similar to most virtual world systems: object interaction is mediated by a central simulator which coordinates the actions of the clients, executes object scripting, and applies physics to physical items. Users interact in the virtual world by running a viewer application that connects to the central simulator. The communication between the viewer and the simulator includes the 3D description of the objects to display and updates on the movements of the other users and changes in the state of objects in the world.

The simplified block diagram of OpenSimulator is shown in Figure 1. The “Scene” is the data structure which holds all
the data for a region with the “Scene Graph” holding the objects in the region. “Client Views” present displays of the region to the clients.

OpenSimulator is built to be easily extensible, so a system of dynamically loaded modules performs operations associated with a scene. All of the functions that occur in a region are implemented in these dynamically loaded modules. For instance, there are modules which implement avatar-to-avatar chat, modules which provide region object persistence, and modules which compute and track the location of the sun and moon. Users can also supply modules to manipulate the virtual world in the region. Some examples might be the implementation of AI for bots or a complex object interaction (like a protein folding simulation).

Each scene has “client views” into the virtual space -- one “client view” for each user. This code communicates with the user’s client software to relay changes in the virtual space (object descriptions, object updates, avatar movement, etc.) and apply the user’s input to the virtual space.

An OpenSimulator grid service is shown in Figure 2. This figure shows two OpenSimulator servers each running several regions. Each region supports a specific 256 meter square region in the virtual world. The figure also shows some of the backend services which support the operation of the simulator. The support services include User Service (login, authentication), Grid Service (world layout and relationship between region spaces), Asset Service (storage of objects that can be placed in the world), and several user support services (user inventory, profile storage, etc.). Behind all of these backend services is the database for persistent storage of all object, user, and world data.

**Time and Simulation**

OpenSimulator is, at its core, a discrete, real-time simulation engine – time is advanced in step intervals. There is a central loop which provides the heartbeat of the simulation: in every simulation interval, the loop 1) updates all objects based on user input, 2) applies the physics engine for the interval, and finally 3) sends the updated object information to the clients. OpenSimulator’s heartbeat is approximately 10 times per second. This is the period of a world step: the time that physics is advanced and the period when object updates are batched.

The existence of the heartbeat loop has several consequences. One is that updates are computed and sent out every heartbeat period. This means that updates to the world will always be delayed by half a heartbeat period on average. While there are some attempts
to short-circuit this delay (for instance, sending responses immediately rather than integrating with the heartbeat loop), in general OpenSimulator is not built for games which require quick user interaction.

Another consequence of the heartbeat loop is that if any of the operations performed in the loop (updates or physics) become overloaded, everything slows down. So, for instance, if there are too many physical objects in the scene for the physics engine to simulate in the heartbeat period (CPU time greater than the heartbeat time), the updating of objects will also slow down, since the heartbeat loop is slowed down.

One of the reasons OpenSimulator has a heartbeat loop is data structure access locking. The heartbeat loop provides single threaded access to the update and physics data structures. This simplifies the design of locking in the base simulator. A different design could move the physics and the object update functionality out of a single control thread and parallelize the execution of the simulator core. This has two advantages: utilization of the many hardware threads available in modern servers, and the elimination of one service slowing down the operation of other services.

Summary
OpenSimulator is an open source, general purpose virtual world simulator that has some design features that show some of the advantages and disadvantages of different server architectures. The next article will explore stress testing of OpenSimulator and the detailed effects of these architectures.

About the Author
Robert Adams is a software engineer in Intel Labs and is a member of the Virtual World Infrastructure team investigating systems architectures for scalable virtual environments.