Abstract

We present an overview of the software architecture for a shared virtual environment that supports both user-generated content and time navigation. The novel combination of these features into a shared virtual environment necessitates fundamental augmentation to established techniques for building such software systems. A survey of related work describes how some of these ideas are present in existing projects. Our software architecture is based on a foundation of object-oriented system analysis and design patterns. We present our current status and outline the directions for future design that must be completed prior to prototype implementation.

1 Introduction

Shared virtual environments (SVEs) are software systems that allow multiple users to communicate and collaborate in a virtual world. Although the domain of SVEs includes massively-multiplayer on-line games such as World of Warcraft or Everquest, SVEs are not necessarily games. Software systems such as Second Life and MPK20 provide virtual worlds for collaboration without being “games” per se. Virtual worlds have been used extensively in industry and academia to foster creativity and learning. Many of these environments allow their users to create and contribute content to the world. For example, many companies have moved meeting spaces and advertising campaigns into Second Life. However, there are design limitations to the current generation of SVEs that inhibit users’ creativity. This work defines a software architecture for the next generation of virtual worlds taking into account two new features: fully-realized user-contributed
content and time navigation.

Many SVEs allow users to create and contribute content to the world, but there are no systems that allow a user to fundamentally change the nature of a world. That is, the content a user creates sits atop an immutable set of rules in the world. This misses an important abstraction of computer science and system modeling, namely that “content” is data that can be both virtual items as well as concepts and ideas. To fully realize this idea requires that content be \textit{free} in the sense of free software: it must be available for study, modification, and distribution.

Time moves inexorably forward in physical reality, and all the related work we have reviewed follows this same model for time. However, this places an unnecessary limitation on the usage of virtual worlds. Consider, for example, an impromptu meeting in a virtual world: while it could be captured using existing recording techniques, there is no mechanism for a user to later go back and review the meeting from another perspective. This is potentially important considering the immersive nature of such worlds, where a single perspective may not capture the fullness of a virtual meeting.

This work draws much of its motivation from the needs of computer science education. Tools for teaching have changed significantly in the last few decades as computer technology has become more affordable and widespread. We have gone from blackboards to overhead projectors to digital projectors showing PowerPoint slides. In computer science, students have gone from programming punch cards to text-based interfaces to “new media” computing. Today’s typical computer science student is accustomed to playing the games that continue to push the limits of consumer graphics technology. However, many textbooks and teaching techniques stuck in the text-based computing environment of the past. There are many different schools of thought regarding computer science pedagogy, and educational research has failed to find a silver bullet, if one exists at all.

One of the fundamental complications of modern computer science is that there are many layers of abstraction between an end user and physical hardware. Industry demands software engineers who know how to use these abstractions, but there is no research that supports the argument that teaching the use of abstractions is equivalent to teaching the creation of abstractions. The system we describe would be useful for teaching the use and creation of abstractions in an engaging virtual environment. The architecture and implementation of our system would be open to students to explore and modify, but other users would not need to learn the details of the system in order to contribute content.
2 Related Work

The following applications allow collaboration over an environment between users. Modification is one of the key design points of our shared virtual environment. One problem which we face, is coming up with an implementation which allows commands to be passed between users, objects, and objects created dynamically. We evaluated the following applications, analyzing how they worked and how we could apply what already exists to our design.

MUD, Mush, and Moo are fantasy/adventure games that use a text based interface. The network architecture is based around one central server, with each player acting as a client. They can be thought of as precursors to games such as World of Warcraft and Everquest or virtual socializing environments like Second Life. Most of these systems are designed with a scripting language built into them to allow a player or administrator to manipulate objects and locations within the system, but they do not allow the player to do more intricate things such as expand the functionality of the server or client. The system places an emphasis on user interaction and denies the possibility for collaboration on expanding the system beyond its original design.

Subversion and Git are content management systems which allow the developers to track the history of their projects. These tools give developers a way to branch off from the base code to experiment with new ideas without having to sacrifice the integrity of the base. Later the branches could be merged to incorporate the changes. The software has features that allow the developer at any point to go back to an earlier revision and discard changes or start a new branch from that point on. This ability makes the development process more dynamic and eases the integration of changes. The process by which these programs keep track of the branches and merging them back together gives us a starting point for an implementation of time traversal within an environment. Time traversal is standard within these tools, in the context that the developer can regress back to an earlier state of the code. In a system with dynamic objects this ability has not been
Multiverse is an engine designed to allow a way for users to create subscription based shared virtual environments. It utilizes an independent client server architecture where the single Multiverse client can connect to the different independent worlds. Each world appears to be dictated by the constraints that the developers create before the system is launched. Rather than a system that promotes user collaboration and open expression of ideas, this system allows users to create their own unique virtual environments for profit.

Second Life (Figure 2) is an online virtual environment that allows users to socialize and capitalize on their creations. Due to the motivations behind 2nd life to make a profit, it does not promote the ideas of code and resource sharing. Through its proprietary scripting language (linden script) you are able to create your own objects within the world. Objects created can be shared within the system but not extended through code modification by users other than its originator. Users are not allowed to modify the visual representation such as changing the environment to 2-dimensional, or extend the 2nd life server for variable physics for a different "feel".

The Croquet Project (Figure 3) runs on the Croquet virtual machine, which allows it to run identically across multiple platforms. Taking advantage of this, Croquet is able to synchronize your virtual machine with collective virtual machine in order to bring your machine up to date upon logging in. The Croquet Project
is an open source multi-user shared virtual environment. It uses a peer-based network architecture which allows users to collaborate and share resources. This system is similar to what we are trying to accomplish, but it has an emphasis on the squeak programming language. One of the strong points is you have access to core code of the environment and are able to extend the world around you, adding new capabilities.

Project Wonderland (Figure 4) is another virtual environment similar to Second Life or Croquet. It is an open sourced product sponsored by Sun Microsystems. Its design is directed more toward business applications. Like croquet you can embed applications within the virtual environments. At the time of this publication it is still in a very early release stage.

3 Requirements Analysis

Creating a truly shared virtual environment is particularly difficult because of the different common problems needed to be resolved along with the new problems introduced by our particular requirements. Some of the more common problems that exist in large scalable environments are an effective user account management system and a viable solution to the $N^2$ communication problem, which arises when each of the $N$ users needs to update all other $N - 1$ users.

3.1 Time Navigation

Time navigation in a virtual environment allows for many unique ideas. Such a system would allow a user to go back in time to a previous state of a certain region of the virtual world. Motivations for a user to do this may be to witness a past event from a different perspective, or to create variations on past events. Each case possesses unique considerations. Implementation for a system such as this could be solved with clever logging and playback facilities. Similar in approach to Subversion and CVS, if the user requires a change to be made to the world a branch point from the world would be created at a certain point in time.
from which we wish to change. Creating a copy that is accurate up to some fixed point, and after which there is no determined future. Programs such as Subversion or CVS do not easily allow users to see how a program is created, other than incrementally going back and examining the source code to see what has been done. With our system, if you want to see how a user spawned a new world and created an adventure type game you could traverse back through the history timeline and interactively watch as the user creates the new objects and rooms. With the ability to examine the source code of any of the objects and rooms to see exactly what is being done.

3.2 Free Content

Richard Stallman describes the four software freedoms as the freedom to run, study, redistribute, and improve software. This is the fundamental philosophy behind the free software movement, especially in the Free Software Foundation. Similarly, Lawrence Lessig has advocated for a freedom of cultural works, arguing that currently-popular models that hoard intellectual property are a significant danger for cultural growth. His work is manifest in the Creative Commons licensing scheme.

One cannot deny that their respective models promote creativity through the sharing of creative works. We have adopted a similar philosophy and embrace free content for virtual worlds. The concept behind free content is that the user is free to use, study, adapt, and learn from any content in the world. The market is one of ideas, and not one bound to economic strength in physical reality: one who creates inspiring and lasting works in the virtual world will be recognized, while others can learn from and adapt it, much like the Open Source movement advocates for source code.

We have specifically investigated a licensing scheme akin to the Creative Commons attribution licenses: if Bob creates a derivative of Jane's work, the new work will have a reference to Bob, Jane, and Jane's original work, as well as any historic predecessors it has. Enforcing this concept through the technology of virtual worlds will be a difficult engineering task, and like Digital Rights Management, any scheme can be circumvented with enough effort. However, we envision a culture like that in software, where those who violate license agreements such as the GPL are given bad press and shunned by the community.

1 http://fsf.org
2 http://creativecommons.org
4 Software Design

The architecture for our virtual environment is based around a few key ideas; being able to create or extend objects at runtime, and the ability to traverse through the timeline history of the system. Creating objects at runtime will need to utilize a technique which makes the system aware that they exist so that objects can communicate with each other. We intend to solve these problems through the use of several design patterns. The chain of responsibility combined with the command pattern gives a satisfactory way to handle messages being passed between objects and enables the system to be extended dynamically by the clients. If client creates an object, without a way to receive and send messages, then the object will not be able to interact with the system. We will also keep a history of all commands acted on by an object in such a way that the object can be recreated from scratch by starting with a base object and applying all commands to it that was applied to the original object. The world as well as every room within the world will also have a history list which will log events and commands which happen.

4.1 Modifiable World using Design Patterns

User modification is one of the key design points of our shared virtual environment. One problem which we face, is coming up with an implementation which allows commands to be passed between users, objects, and objects created dynamically.

The Chain of Responsibility design pattern would allow us to route commands from one object to the next. With this pattern you would create a series of connected handlers which can accept commands. Then either handle the command or modify it and pass it along to the next handler in the chain. With the ability to dynamically add or remove these handlers, we have a solution to interfacing dynamically created objects to the world. For instance, when a user wants to create an object and let it receive and modify messages, the user can add custom handlers to the object. This architecture is shown in figure 5.

The use of history lists helps to solve the problem of time traversal, by giving a way to traverse back into history and examine the state of the world at that point. For example, user1 enters a room where there exists a tree with a sword sitting beside it, figure 6.

User1, the tree, the sword, and the room all have a separate history list which will be used to log events which modify the attributes, state or events which happen to that object. So once the user enters the room, the rooms history list would log that user1 entered the room, and user1’s history list would log that he entered the room. User1 picks up the sword and chops down the tree. Now User1’s history list logs the
Identify the Chain of Responsibility Pattern (and other patterns) using collaboration notation.

Figure 5: UML Class Diagram for the Command Handler

Figure 6: Step 1. History lists of user interaction with the world. The room logs the player entering it, and the user logs that he has entered the room.

sword being picked up, and the event of the user chopping down the tree. Likewise, the tree’s history list logs that it has been chopped down by user1. Also, the sword’s history list will log that it was used to chop down a tree, figure 7.

Next, user2 enters the room and wants to find out how the tree was chopped down. To do so, he can navigate through the rooms history list to the point before the tree was chopped down. At which point there are two options, creating a branch point, or spawning a "read-only" copy of the particular region of the world. In this case user2 wants to find out how the tree was chopped down. By creating a spawned copy of the this region, figure 9, we could then regress each object within this region to the time specified by the

Figure 7: Step 2. As time move on the history list grow in size logging events which happen to the objects. Since the user picks up the sword it no longer exists in the room; it now exists on the player.
Figure 8: Step 3. The Sword logs that it has attacked a tree and the user logs that he has attacked a tree. Notice that it is not logged that the user has chopped down a tree. That is an artifact of the attack, so the tree logs that it as been chopped down.

Figure 9: Step 4. The user spawns a copy of the room which is read only and private to that user. This can be accomplished by either incrementally removing items from the history list and performing and undo type operation on the object for that particular event logged, or just removing all the logged events back to the point specified then ”rebuilding” the object by starting at the beginning of it’s history list and applying all the events to it, which would result in the object be regressed to a previous state of existence. If user2 would have instead chosen to create a branch point from the point right before the tree was chopped down, then a new copy of the region of the world he is in would be created. This new region would inherit all the objects that are in the original region and have its own unique history list.

5 Conclusions and Future Work

The architecture we have described addresses the primary problems involved in creating a shared virtual environment that fully supports user-created content and time navigation. A more thorough formalization and algorithmic analysis is merited, especially for time navigation, which has exponential space requirements in a naïve implementation. Research to alleviate this problem could take two directions: either limiting the realization of time navigation or limiting users’ interactions with a system. The former reduces the effectiveness of our models, and the latter opens a potential security vulnerability into the system. A formal system analysis and prototype testing will be required in order to determine the best combination of these
Supporting user-contributed content will require significant systems research for the implementation language. Our initial explorations have been based on Java since it already has an effective security model and dynamic class loading. However, it is not immediately clear how one would avoid the “grey goo” attacks that have plagued Second Life, in which a user creates infinitely self-replicating objects. A technical solution may exist that is more robust than Second Life’s user agreements and script-protected zones.

Once a prototype is complete, it will need to be tested on a range of users with various skills. Although we imagine prototypes that are limited in size and scope, such as text-based interfaces or support for only a few dozen users, we believe that such a system will demonstrate the benefits of our architecture.