DEVELOPING MODULAR MULTI-USER ENVIRONMENTS WITH CARNIVAL

An Honors Project

Presented By

Shaun K. Kane

Submitted
May 2003

Guidance Committee Approval Signatures:

_____

Professor Beverly Woolf, Computer Science
Chair

_____

Sean Wright

TABLE OF CONTENTS

INTRODUCTION

The goal of this project was to create a framework for developing online multi-user games. Creating a fully-featured online game would be a tremendous undertaking. Creating such a game alone in two semesters would be impossible. But I had some new ideas, and a two-semester Culminating Experience seemed a perfect way to test them out.

Even a rough, experimental game would take much effort to build, so I focused on designing a simple, modular project from the very beginning. Given the pitfalls and setbacks of large software design projects, a modular design would allow me to focus on the successful parts of my project and rework or discard unsuccessful parts.

I further simplified the design of this project by using familiar technology whenever possible. I decided to use Macromedia Flash for the main content: after two semesters of interactive media courses, I felt very comfortable using the program to build simple games and applications. My recent summer internship at Microsoft prompted me to use .NET Web Services and SQL Server 2000 on the backend.

The system I eventually designed was called Carnival. The design featured a collection of modular components called *rooms* (think booths at a real-life carnival) that could easily be added to or removed from the whole. Each of these components could potentially exist as a self-contained game, but are linked by the Carnival server to provide a more engaging experience. Carnival modules may store and share information about users, and are arranged geographically to one another to provide an overall "world map".

Due to the time constraints I was under, I decided to focus my effort on developing the Carnival platform itself. Creating original game content is a lot of work,

and there simply was not enough time to make the sort of game I would have liked using the Carnival platform. Some content was clearly needed for demonstration purposes, and so I created a series of simple modules with the common theme of “Monster Island”. These demo applications serve primarily to show what the Carnival platform is capable of, and are quite rough around the edges.

DEVELOPMENT PROCESS

Development on this project was split into four main parts: initial research, system design, system and API specification, and design of the demo world. I had originally planned to do the first two parts during the first semester and complete the design and API specification by the end of the first semester. I then planned to spend the second semester developing the system software and demo environment in parallel.

I followed this plan somewhat closely, though my deadlines were never strictly set and tended to shift throughout the two semesters. I had spent some time over the previous summer developing some initial design ideas, and thus spent the first month or so exploring various ways to implement this project. Having a flexible schedule was very useful here, as I had time to explore various web services platforms and several types of non-relational databases. However, I was not completely satisfied with any of these solutions, and decided to stay with my original choices, Microsoft .NET and SQL Server 2000.

During this time I was also developing the web service API and the database schema. I had hoped to convince some friends to develop simple games for this project later on, so having an easy-to-use programming interface was very important. After deciding upon the types of objects I wished to store in the database, I created a set of about fifteen functions allowing content developers to access these objects.

Programming work on this project was divided between the three main components: the database, the web service, and the Carnival plug-in. In my initial design, the API was defined in each of these components, so that changes to the API needed to be made in three places. I soon realized the problems inherent in this design and attempted

to rectify them. My updated design allowed generic requests to be made through the web service, placing the majority of the game logic solely in the database. This allowed me to add new functionality with little effort.

By December I had created a functional Carnival plug-in within Flash and had written rough API documentation. Unfortunately, I was unable to enlist the help of anyone in developing game content. I decided to continue developing the backend and worry about the demo later.

I worked on the Carnival software from December until March. Developing the database was very straightforward, and was completed within the first few weeks. Due to some strangeness in Flash's XML handling, I had some trouble integrating the Carnival plug-in with the web service, though I eventually managed to piece together a solution. During this time l also wrote some simple helper applications, including the package installer, and added minor features to the API, such as the buddy list system.

By March the backend system was working reliably, so I decided to stop working on the system software and focus on developing game content. I started by developing the UI's navigation bar, itself a Flash application. This allowed me to test the system as a whole and prompted me to make some minor improvements to the API.

Between March and the end of May l completed about ten Carnival modules using Flash. These were all loosely based around the theme of "Monster Island", a theme-park-like environment for children. Although I had originally hoped to spend more time developing a fun place for kids to play online, the demo I created contained a wide variety of activities, including online quizzes, multi-user chat and a functioning e-mail system.

SYSTEM DESIGN

The Carnival system is a multi-tiered application. It contains a client component and server component, as well as several helper applications. A diagram of the complete Carnival system can be seen in Figure 1. Each component is also described in detail below.

Database. A SQL Server 2000 database was used to store user information, room information, shared world items and system settings. The database also contained a set of stored procedures used to access the database. For a complete database schema, see Appendix B.

Package tool. All game content used in Carnival, including rooms, shared world items, and user avatars may be managed using Carnival's built-in package system. Packages allow developers to group related content, and allow administrators to easily add and remove content. A Carnival package is simply a directory containing game content and an XML index file describing that content. Carnival contains a built-in package installer for adding new content.

Web service. An XML web service is used to handle database requests from client-side Flash movies. The web service receives requests over HTTP as SOAP objects, queries the database, and returns a response as another SOAP object. These SOAP objects are then parsed by the Flash plug-in and passed to the requesting application.

Navigation bar. The navigation bar provides a consistent user interface for interacting with the Carnival system. It allows users to get information abut their current status and to move between rooms. The navigation bar uses the Carnival plug-in to interact with the server.

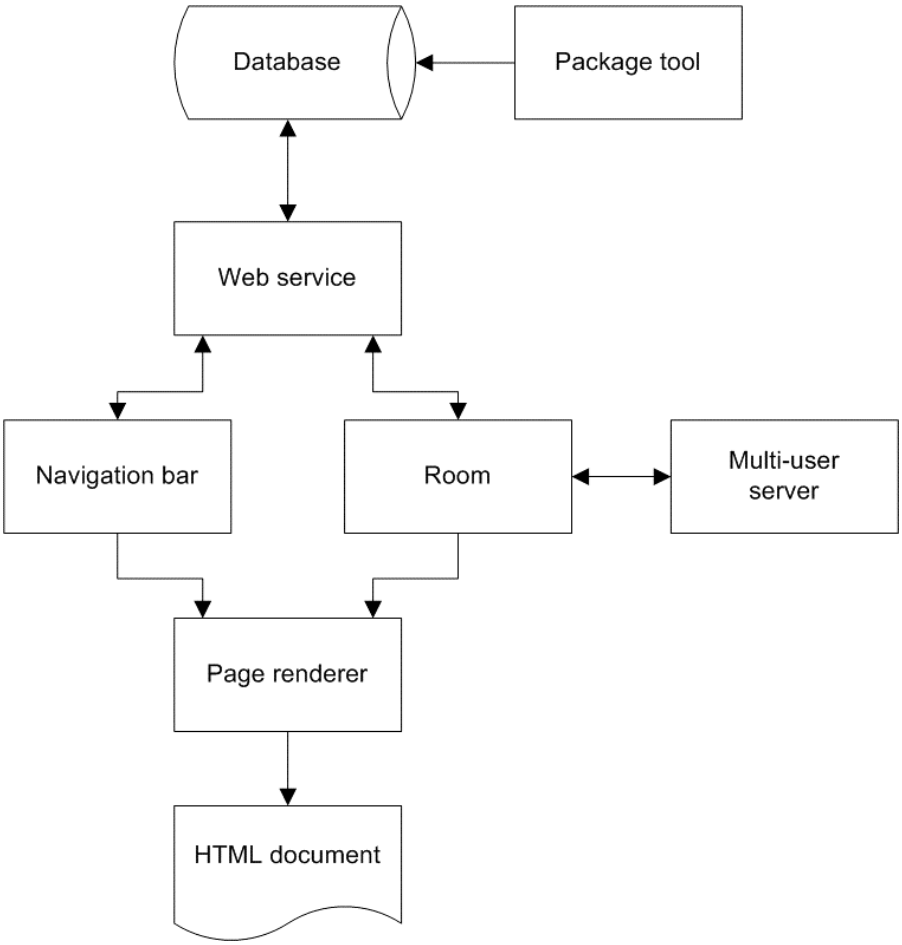


Figure 1. The Carnival system.

Room. The Carnival world is presented as a series of individual Flash movies known as “rooms”. Each room my store information abut itself and its users in the database. Rooms may use the Carnival plug-in to interact with the Carnival server and the multi-user server to allow users to interact with each other.

Multi-user server. The multi-user server enables developers to easily add multi-user functionality to their Flash movies. It uses Flash’s built-in XML socket functionality.

Page renderer. The Carnival user interface is presented in the user's web browser as HTML with embedded Flash movies. The page renderer is a JavaScript object used to generate the HTML for each room. It positions each element on the page and passes each Flash movie the parameters needed for them to run, such as the user's ID.

HTML document. The page renderer outputs the current room as an HTML document, viewable in any web browser that supports JavaScript and Flash.

USER INTERFACE

The Carnival user interface was designed to promote both ease of use for end users and flexibility for content developers. It consists of two parts: the main content area and the navigation bar. The interface is rendered as an HTML document with embedded Flash movies. This allows anyone with a modern web browser to use Carnival. Figure 2 shows a screenshot of the user interface.

Figure 2. Carnival user interface.

The navigation bar is the user's primary means of interacting with the Carnival system. It is organized as a tabbed interface with three tabs: the Room tab, the Friends tab, and the Info tab. The Room tab allows the user to move between rooms. The Friends tab allows users to see who else is in the current room and to manage their buddy list. The Info tab allows users to see information about themselves, such as their current

avatar and the amount of in-game currency, known as “tickets”, currently in their possession. The navigation bar also reports changes in the user’s state to the server.

The navigation bar is designed to be simple and unobtrusive to the end user. The tabbed interface uses up little space and helps keep the screen uncluttered. It also provides a consistent interface for essential functions, so that users do not need to learn a new way of interacting with the system each time they visit a new room.

This method is not without its drawbacks. The navigation bar is always present, and may detract from the continuity of the environment. For example, because links to other rooms are presented in the navigation bar, they should not he presented in the room itself. This means that users must shift their focus from the main content in order to change rooms, and prevents direct linking between rooms. Unfortunately I could find no simple solution to this problem, though I would like to investigate it further.

Because the navigation bar handles all essential interaction with the system, content developers are freed from implementing such interactions themselves. In fact, Carnival rooms do not need to interact with the server at all unless they wish to.

DEMO WORLD

The development of a compelling demo world has been a priority since the beginning of this project. Without a good demo, it would be difficult to show what the system is capable of. The act of creating a demo would also be a great usability test. If Carnival was not easy to develop for, it would be useless.

My initial ideas for a demo world sought to combine education and entertainment. The world would contain educational rooms and entertainment rooms. The in-game currency would serve as a balance between the two: students could earn tickets through educational activities and spend tickets playing games and chatting. I had hoped to create an environment in which students would be rewarded for doing their work, and especially for working together.

I decided that the setting for this demo would be Monster Island, a tropical environment inhabited by cute cartoon monsters. Setting the game in a vacation resort would enforce the idea that this was a place for students to visit frequently. The island's residents could also walk users through difficult tasks. Perhaps most importantly, a giant-monster-themed resort would be fun to design.

Unfortunately, time constraints kept me from developing the elaborate experience I had planned. Although I found developing rooms for Carnival relatively straightforward, my original plans required more original art and animation than I could produce in a month and a half.

Despite these time constraints, I managed to develop a good number of original modules using the Monster Island theme. Below are a few examples of these rooms. Some of the rooms were created for Carnival, while others were adapted to work with the

system. Whenever possible, I attempted to make the rooms themselves modular, so that they could be changed without editing the source code.

Mail center. The mail center is a fully functioning mail client for sending messages to other Carnival users. The mail center has a rough interface, though I think it is an interesting application nonetheless.

Arcade. The arcade is a simple wrapper for Flash games. Administrators may choose the game and the cost to play. Users may then spend tickets to play the game.

Chat room. The chat room module provides multi-user avatar chat rooms. Users are represented by their Carnival avatar images.

Online quiz. Students may take a quiz online to earn tickets. Students earn tickets for each question answered correctly. The list of questions and corresponding point values is contained within an XML file.

FUTURE DEVELOPMENT

Overall, I was very pleased with the results of this project. I believe it is quite possible to build interesting online worlds from small, independent pieces. While I would have liked to have spent more time developing Monster Island, I feel it does a good job of showing what is possible with this technology.

Once I began, I was determined to stick to my original plan as closely as possible. Previous experience with large software projects taught me that it is important to avoid backtracking whenever possible. However, I had plenty of interesting ideas throughout the development process, and made sure to record them. Below are some enhancements that could be made to improve the existing Carnival system.

Consolidate the server software. Currently, a Carnival installation requires multiple pieces of proprietary software and spans several executable files. This setup was ideal for development, but is difficult to set up on other machines. I believe that much of the existing functionality could be rolled into a single executable file, using a web-aware scripting language such as Python and a persistent object library such as Zope.

Improved admin console. While the administrator has a good deal of control over the system, the existing admin tools are sorely lacking. During development I often found it more convenient to work directly on the database, and the state of the admin tools reflects this. Better admin tools would make managing a Carnival server much easier.

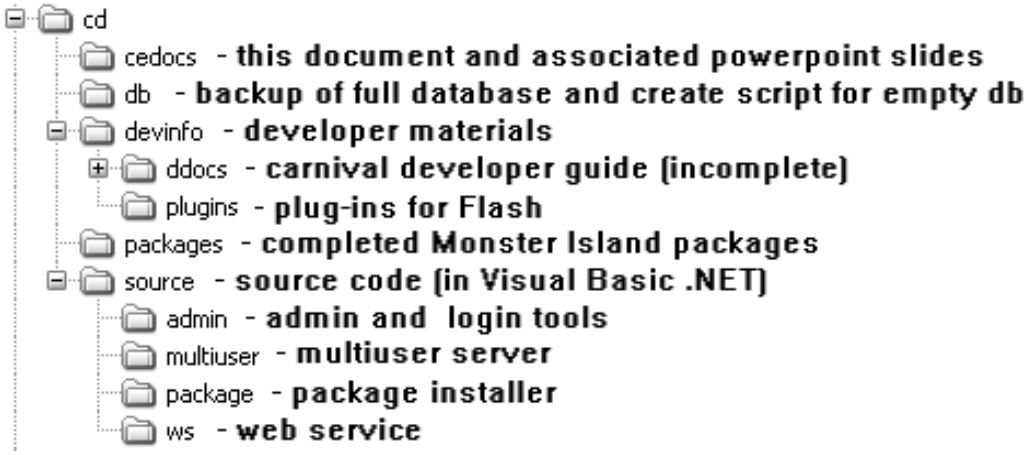
Support other media types. I chose to develop content in Flash because it is fairly simple to use and supported on many platforms. Adding support for other content

types, such as Macromedia Director, would be fairly trivial. Any media type that supports the HTTP protocol could potentially become a Carnival module.

Support mobile devices. An "offline" version of Carnival could be used on mobile devices. Locally-run Carnival applications could record the user's activities, and then report them to the server when an Internet connection is available. For example, students could work on homework assignments using their mobile device, then connect to the Internet and earn tickets for their work.

Collaborative problem solving. While the existing Carnival rooms make interesting demo applications, they do not exercise the full potential of the technology. One potentially interesting application of this technology would be to create collaborative problem solving exercises, in which students work together to solve a problem and earn rewards. These problems could be story-based, and linked together in a somewhat linear order. Much like in an online role-playing game, students could form groups, go out into the world, and solve the next problem together.

APPENDIX A: GUIDE TO FILES ON ACCOMPANYING CD



APPENDIX B: INSTALLING CARNIVAL

System requirements. The following software is needed to install Carnival:

- Microsoft Internet Information Services
- Microsoft SQL Server 2000
- Microsoft .NET Framework 1.0
- Version of Windows that can run the above programs

Installing the server. Everything needed to reproduce the Carnival system is included on the CD. Unfortunately, there is currently no setup script for Carnival, but setup can be accomplished in a few steps.

First, you must create the database. The easiest way to do this is to run the create script named *Carnival.sql.* This will create an empty Carnival database. You will need to input some values for the database to function. Most importantly, you will need to add default values to the Settings table.

Next, you must set up the web scripts. Everything in the *source**admin*\ and *source**ws*\ directories must be put in an IIS server directory. You must then update the Settings table with the location of these files.

Once the web pages are set up, you may use the package tool to install packages. Once a package is installed, you may add rooms using the stored procedure *adminAddRoom.* Make sure that the "root movie" attribute of the settings table points to a valid room.

Once this is done, you need only activate IIS to start the Carnival web service. To start the multi-user server, simply run *Carnival.exe* in the *source**multiuser*\ directory.

Managing an installed server. The *CarnivalAdminPage* project, included on the CD, contains some simple web-based administration tools. These are incomplete and may require modification to work with the current Carnival database. For many functions, you will need to access the database's stored procedures directly. Stored procedures related to administrative functions have the prefix "admin".

APPENDIX C: DATABASE SCHEMA DIAGRAM