Vi-Char Game
Web Server Development

Scott Dunham, Michael Lim, Eli Mallon, Kimberlee Redman-Garner, Logan Jones

Introduction

The Web Server's goal first is the seemingly simple task of relaying messages between clients in addition to the goal of hosting a web interface that allows many types of user interactions outside of the actual game play such as leader boards, play-by-play and even a type of marketing for the game that will explain the purpose and plot of the game and serve as a public, web-based "face" for Vi-Char.

After the requirements document was finalized, it was time to think more clearly about the concepts behind the web server and how they will be applied to create an architecture that is scalable, flexible and fast. After the class decided on the basic structure of the model for how each component/class group would interact on a functional basis, this allowed for a cohesive idea to form and take shape in our group for the architecture of the server processes. One challenge right now is that the game play itself is still taking shape and therefore hard to visualize in the "big picture" but we hope that our web server will be able to fit whatever ends up happening and we will support other groups' with our own in-class API that will keep them up-to-date on the utility of the server.

As we prepare for the actual implementation, coding and testing of our Web Server module, we have prepared a design of the Architecture that attempts to encompass every component in detail.

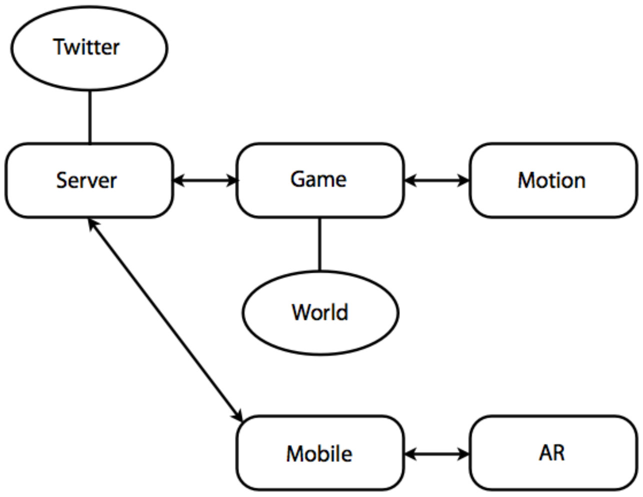
The Messenger module will be doing the bulk of the work in the client/server relationship between the Game Server and the Mobile Devices to achieve a networked, fast and interactive game. For instance, it is vital that the messenger module have a way to open connections to these clients and relay information about the game state to them. This include the need for a structured database, socket connections and a consistent data format that can be read and written between all clients.

The Web Portal is the aforementioned "face" of the Web Server and will be a place where game information will be aggregated, organized and presented to be viewed by the public. Interestingly, we will also have the Twitter gameplay aspect tied into the Web Portal. The major point of the Web Portal is to complete the gameplay experience in a manner that will include all users interested in the game.

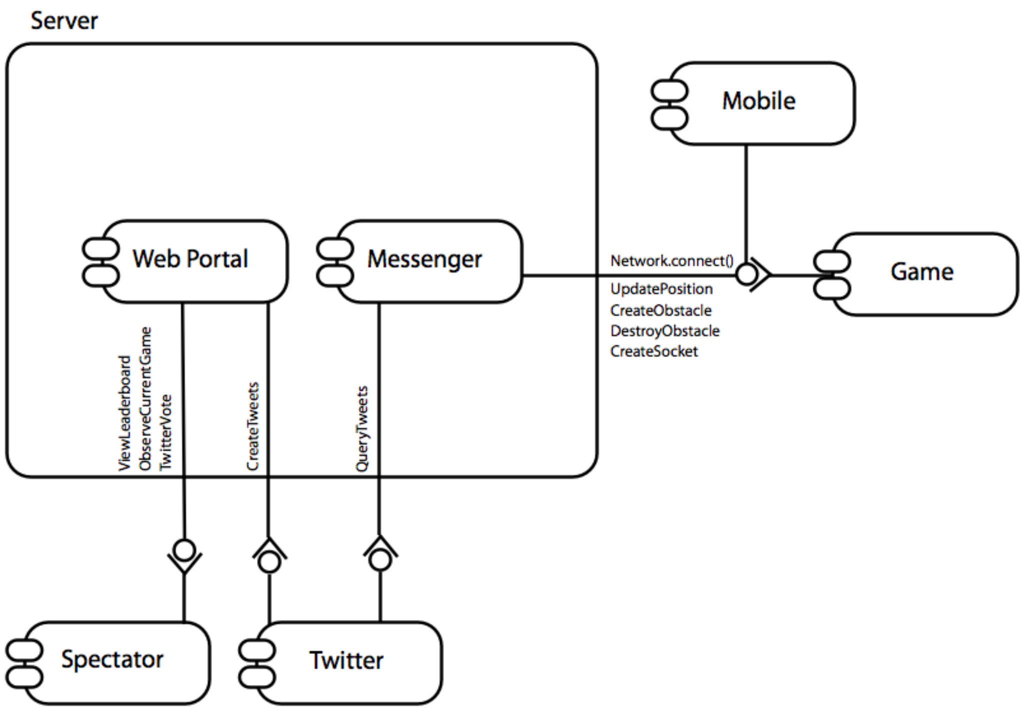
The twitter interface is the module we have designed that will bring Twitter into the Messenger module *and* the Web Portal. The most fascinating aspect of this is that we will be offering Twitter "tweet" information to the Game Engine and Phones to create events in the game itself -- crowd-sourced buffs and obstacles -- how fun!

These three modules make up the higher-level design of the Web Server architecture that will complete the game experience as a networked, multiplayer game and also promote the entire Vi-Char project within the class and beyond.

Architecture Design



On a global level, the web server will primarily interact with the game engine and the mobile devices. The server provides a messaging system that allows game information to be updated by the game and mobile devices as events occur within the game. When game state data is received by the server, it will be sent to the appropriate place.



The architecture of the web server is comprised of three main modules. The messenger will provide an interface that allows game state information to be received from the game engine and mobile devices, and sent to relevant devices. For example, when the player moves, data will be sent from the game to the server. The messenger will then update the stored position coordinates and send them to the mobile devices. The web portal provides interfaces for users. Users will be able to view leader-boards and check the score of the current game. Users will also be able access twitter via the web portal and tweet certain phrases to influences the game. For this reason, twitter is providing an interface to create tweets to the web portal. Twitter is also providing an interface to the messenger to allow tweets to be searched.

When relevant twitter information is received, this will also be send from the server to the game and mobile devices.

Module Design

There are three main modules that make up the interactions between the web server and its users. It is vital to note that these users are both *outside* users, like those accessing the website *and* internal users, such as the game engine and the mobile phone device technology itself (ie, how do we have our server "talk" to the phones?). The messenger module, or what we colloquially call the “Server” is the workhorse that will be hosting the server that will create connections between clients (game engine, phones) which facilitates communication in any direction. The Web Portal is the “face” of the Vi-Char project that will represent the idea of the game as well allow outside users to join in on the game. The Twitter module is the crowd-sourced buffing and obstacle creation that our Server and Portal will facilitate.

MESSENGER MODULE

Purpose Statement:
The messenger module is responsible for coordinating the game components which must stay in communication during the course of a game. Its various provided interfaces allow a game to be started, for players to join that game, open a connection with the web server, specify a name, send commands to the other players, and send/receive updated position for the various objects in the game world.

Description:
The messenger module is designed to allow connections to be opened between the various player devices in the game, and to facilitate the transmission of data between these devices and the web server. The way we are envisioning this working is that, as the various player devices take actions which affect the game world (e.g. creation or destruction of objects, movement of characters/objects), relevant data is sent to the web server via a connection established before the game begins. When the server receives this data, it decides what the appropriate action is and takes it. For instance, if a command is received to create a new object at a particular location in the game, this would be routed on to all other player devices for them to carry out. If instead a command is received to change the position of an object in the game world, this new position data would be added to a central repository of current game state data, which would be periodically polled by the player devices so they can update their rendered version of the game world.

Provided:

- Interface for game engine to connect to the web server to exchange messages with the web server, as well as mobile devices who will join the game
- Interface for game engine to request a new game lobby so mobile players can join the game
- Interface for players to specify their name before joining the game
- Interface for mobile phones to join to existing game and connect to web server to exchange messages with game engine
- Interface for mobile phones to send commands to the game engine
    - *Example commands:* createObject, destroyObject
- Interface for game engine to send commands to the mobile phones
    - *Example commands:* createObject, destroyObject

- Interface for game engine to send position information for objects (e.g. avatar, projectiles, etc...) in the world to mobile phones
- Interface for game engine to send final game data (player scores) to web server

Required:
- Interface from the game engine which allows for the sending and receipt of game commands and position data
    - *Example commands:* createObject, destroyObject
    - Unity, for instance, has an API which communicates with many types of server relationships. We plan to use Node.js as our server which will look similar to the API already provided by Unity 3D to work with .NET. In unity, there is a script command Network.connect().
- Interface from mobile phones which allows for the sending and receipt of game commands and position data
    - *Example commands:* createObject, destroyObject

WEB PORTAL

Purpose Statement: This module primarily provides a way for additional players/spectator to participate in the game without being a part of the puppet group or having a compatible mobile phone. It allows players to view information about past or current games, as well as providing a simple interface for utilizing the Twitter module.

Description: The heart of this module will be a basic website, providing an easy way for potential spectators to get involved with the game. Anyone with a web browser will be able visit our site, and peruse the available statistics (leaderboard, periodic updates of the current game, etc.). One caveat: for now, only people on the campus network will be able to visit our website. When it comes time for these users to interact with the game, they will use a portion of our website devoted to streamlining the twitter integration features. When it is time for a vote to occur, the website will display an easy to use interface, allowing users to send Tweets which will then influence the game state in some way. This will most likely include some “buttons” which can be clicked and will give the user preformatted text that they will then be prompted to Tweet using their Twitter. More details can be found in the Twitter module description, as the web portal merely serves as a “home” for the Twitter module.

Provided:
The web portal will provide an interface for:
- Outside potential players/spectators to log in and access their user accounts.
- Outside users to view a persistent Leader Board
- Outside users to access the social media integration module
- Outside users to check the score of the current game, and see when different teams score points (Think ESPN box scores).
- Mobile phone device users a location to download the App from.

Required:
The web portal will require an interface from:

- The internal web server messenger module to access the final game score and update the Leader Board
- The twitter interface will require an interface from the messenger module to access the status of the current game, to update the page with the current score periodically.

Interfacing Twitter -- Twitter's influence on the game

Purpose Statement:

This module is responsible for interfacing with Twitter and passing all relevant tweets to the Messenger. It interfaces with the public Twitter API on one end and the Messenger model on the other.

Design:

An important distinction: Our integration with Twitter will do minimal parsing and aggregation of data. Rather, we will grab any and all tweets directed at the game (that is, any Tweets that uses our hash-tag) and send them into the Messenger to be routed accordingly. It is the responsibility of the other modules to parse the messages themselves into actionable game data, and they may of course feel free to disregard any messages that don't use the correct syntax.

Basically, this module acts as an intermediary between Twitter and the Messenger that streams a filtered segment of the full Twitter feed into the game.

Provided Interfaces

- Tweet stream. For the entertainment of those watching the game, and for debugging purposes, the web portal will output to HTML a live stream of all tweets incoming to the game.
- A static "influence the game" HTML page to which end users may connect. This will streamline the process of sending tweets to influence the game by providing a button for every possible action. The buttons will redirect the user to Twitter, where they can confirm that they want to send the relevant tweet. (Strictly speaking, this module actually doesn't interface with Twitter directly. They will access the page and then tweet–we will receive the message via Twitter.)

Required Interfaces

- Twitter API. This will entail making an account with Twitter and getting an API key–nothing too arduous. We must be able to poll Twitter regularly for all new tweets.
- Messenger. After we've received relevant data from Twitter, we will turn around and pass this to the main Messenger module.

Other Design Views

The choice currently for the database structure is a non-relational database called MongoDB which uses JSON/BSON and thus JavaScript to code the database. Since we are combining two different types of interfaces to run the game -- the Unity 3D-based Game Engine and the Android mobile devices, we will need a unified data syntax to communicate effectively and fast between these two parts of the game. Ultimately, this means that we will be using the same language for all parts of the Web Server design (that is, JS) which will make faster-to-produce code with less bugs. Even the testing of those bugs will be done in JavaScript so testing can be applied to all parts of the server.

There is a Unity 3D client for Node.js that we will be working from to implement the actually communication but we will also be using the JSON/BSON syntax from Node.js and MongoDB to have it port to the phones with ease.

Design Rationale

The Web Server group aims to serve as an intermediary between the game engine and the mobile devices, as well as a place for outsiders (non game players) to interact with the game and create a full game experience with multiple users from various inputs. In order to provide this service, a server is required to host and transmit data the the various clients. We have outlined the more precise architecture of this server system to fit the needs of the Vi-Char game.

With this in mind, we have chosen a scalable architecture that will function efficiently and effectively from its simplicity. For instance, on the server-side, using Node.js we will be taking advantage of the Event Loop architecture in Node.js to handle many connections and requests at once of different sizes between many types of users (as discussed above). Indeed, this design could be used for many applications but will be tailored to the Vi-Char game experience with optimization for the particular hardware that is available to our group.

We will be a “non-authoritative server.” [See #1 in the Glossary] Meaning that the server will not have an effect on the outcome of the game, instead the clients will process user input and game logical *locally* , send those decisions to the Web Server and the Web Server function will be to relay those messages by synchronizing all of the clients involved to the Game State. The server does no extra processing beyond what is required to relay the messages between clients. This is a more simple design that allows all of the processing to be offloaded where the game is being played.

Thus, meeting the “Messenger” part of our Requirements document will be done through the choice of an API that allows for scalable, flexible, smart ways to send these messages between all devices hooked into the server. In our case, we plan to use the MongoDB database system within the Node framework to have a consistent use of JSON/BSON files that will convey the game state.

The Twitter module is a popular, sleek design that will enhance the user experience and run very cleanly as it has been done many times. We will be using it in a unique way in conjunction with the other components of the Vi-Char game to interact with the game itself by providing the information from Twitter not just as a simple aggregating tool for social network information, but as data for the game to use.

Implementation Notes

From the very beginning we have thought of the many ways in which we plan to implement the design of the server architecture. Because this is a very vast discipline, we have had a lot of choices when it comes to how to serve clients in a multiplayer game setting. Thankfully, this idea is not ground-breaking and many other developers have started us off with many references and ideas.

Version control: Git and Github are our choices for version control. For our own write-up documents and drafts, we have also used Google Drive but not for code.

Server software: From our “Hello World” example, we have expanded on our use of Node.js, a JavaScript-based, server-side implementation of tons of modules for hosting a website, serving a game engine and phone and garnering posts from Twitter and beyond. In the interest of free and open-source, Node.js is a great implementation. There are many adequate choices for such an implementation but because Node.js has a lower barrier to entry and is very popular right now we have chosen it.

Also, in Node.js, there are many built-in modules from the Node Packaged Modules (NPM) Registry that we will be using and building on. The built-in http module let's us create a server that is event-driven through Node.js' Event Loop. The Event Loop is a clever design in Node.js that allows longer operations to be put in the background while the server still serves other, quicker operations. This is not just between separate users, but is something that maximizes efficiency within each user instance. This can be done, theoretically, on a single processor core.

Modules from the NPM Registry we are interested in (external modules):

- Nodemon, a development tool that allows for quick reloading of the server for live coding changes.
- NodeUnit, a unit testing module for Node.js that will allow us to have early testing procedures built right into our server code.
- MongoDB with Node.js. MongoDB uses compatible data types (*.JSON and *.BSON) with Node.js and is also JavaScript based.
- Socket.io, a module that creates client/server connections from the Node.js server (ours) between many clients. There is already a client for Socket.io and Unity projects.
- Other clients may be accessed or built upon through Node.js as the phones will most likely not have Unity running on them, but instead another game engine/rendering client that will require the server to open connections to. Our team is aware of this and ready to work with the mobile and AR groups to assist in finding an appropriate client for development.
- nTwitter, the Node.js module that allows for direct access to the Twitter API for gathering information from recent tweets.
- Express, a Node.js based web app API that will be very useful for the website itself to run right along with the game to allow the website to function in the Node framework.

For ease of transition, Coffee-script can be used where our group needs to write any code to make more elegant programming. JavaScript is known to be verbose and Node.js has a module for Coffee-Script.

For the **website** itself we will be using Node.js with the Connect node-module on top with HTML5, CSS3 and jQuery to make a flexible, useful and fast website.

Glossary and References

1. This document from Unity 3D is extremely helpful in understanding the architecture of a networked game. http://docs.unity3d.com/Documentation/Components/net-UnityNetworkElements.html
2. Node Package Module Registry. The hub for all packaged modules for node. We will be building our own, but this is where we will be getting pre-built modules. -- https://npmjs.org/
3. Node Beginner. No need to reinvent the wheel, this author says. Not only is this a good little tutorial for node.js it is one for javascript and basic client/server relationship understanding as well -- http://nodebeginner.org
4. MongoDB from the company 10Gen has a wealth of information available to those interested in harnessing Node.js with MongoDB for databases and session management. http://www.mongodb.org/display/DOCS/node.JS?showComments=true&showCommentArea=true
5. The Twitter API offers many solutions for developers wishing to harness Twitter's social networking power for their own uses. Here is an example that our website will be using, the

Tweet Button. This will go on our website to allow a quick way for users to vote with hash-tags for their favorite side. https://dev.twitter.com/docs/tweet-button

6. Express is a popular API for Node.js servers for building web apps and pages. Ours will probably not be too complicated as we want to focus on the game, but why not make a useful, elegant website for the leaderboard? http://expressjs.com/
7. Case study: How & why to build a consumer app with Node.js (with an example development stack. Why reinvent the wheel?!) -- http://venturebeat.com/2012/01/07/building-consumer-apps-with-node/