Abstract

The purpose of this report is to document the evolution of the Cyberscape Project and the functionality I have added to fulfill the requirements for my honours project. This paper will provide suitably detailed descriptions of what Cyberscape is, how it works, the adjustments I have made and future development concerns. It will also document the obstacles faced and the design decisions which were necessary to surmount them.

The Cyberscape Project began its existence as the joint honours project of Benjamin Hall and Jason St. Cyr, two fellow Computer Science students. Although this report does include an overview of the system, more detailed descriptions of components which are not included in this document may be found in Jason's and Ben's reports respectively.

This report will begin with a complete overview of Cyberscape's communication system, both before my contribution and after. This will be followed by a detailed overview of the different servers used by the system, including their responsibilities and how they communicate with each other. Other new Cyberscape elements which are not communication-related will be also highlighted. The report will conclude with any observations I have made and challenges I have faced over the course of this project.

Acknowledgements

Of all those to whom credit and thanks is due, Benjamin Hall and Jason St. Cyr are the most deserving. Without their patience and support, this project could never have been completed. Contributing to Cyberscape has provided me with a challenging but enjoyable learning experience, which is the most any student could ask of their honours project. I am grateful to them for the opportunity they presented me and their guidance as I shaped it.

I would also like to thank my advisor, Dr. Micheal Weiss, for his understanding and willingness to take me as an honours student. I selected my honours project topic at a later date than is usual, but Dr. Weiss was generous in his offer to advise me despite the fact that he was already supervising a large number of students. His interest and enthusiasm were essential, and his non-invasive style of supervision was most appreciated.

The designers and programmers of the Eclipse IDE deserve a special commendation for their efforts. As a former Rational employee who assisted in the development of Rational XDE for the Eclipse platform, I was in a special position to witness its evolution from a somewhat raw and sparse tool into a superior development environment. The fact that Eclipse can be downloaded at no expense is an added advantage for frugal students such as myself. The development of this honours project would have been far more difficult without Eclipse' s assistance, and for that I am grateful.

Table of Contents

Table of Illustrations

1. Introduction

If it ain' t broke, don' t fix it.

It takes a certain type of person to ignore this well-worn axoim, and both Ben Hall and Jason St. Cyr fit the bill. Dissatisfied with the manner in which people view the internet, and have viewed the internet since its conception, they decided to completely redesign it. Instead of the current two-dimensional approach, where users view pages on seemingly disjoint servers and use search engines to make sense of the system, a three-dimensional world is created instead. It provides an accurate view of the internet by showing the relationships between servers (through their IP addresses) and providing users with a navigation mechanism which is more intuitive than a search engine or a series of links.

Cyberscape' s original design also included the concept of a community, where users could see and interact with one another. Despite Ben and Jason' s best efforts, their design was simply too broad to implement completely within the time limit assigned to an honours project and the community aspect had to be foregone. In essence, this report picks up where Ben and Jason' s work left off. It will document the implementation of a Cyberscape community, complete with user profiles, a tracking system and instant messaging.

Not surprisingly, a community requires communication. During the implementation phase, it became apparent that the bulk of the project had to be concentrated on abstracting the current Cyberscape communication system and creating a new set of clients and servers, each with responsibilities for a certain aspect of the Cyberscape experience. As a result, the focus of this report will be on the communication system: its components, how they interact and their responsibilities. In addition, this document will also include an overview of the Cyberscape system, a description on other new elements which are not communication-related and any notes or items which should be mentioned.

2. An Overview of Cyberscape

2.1. Client, Server and VRML

Cyberscape components can be divided into two categories: what resides on the Cyberscape server and what resides on a Cyberscape client' s machine. The Cyberscape server is responsible for answering user requests, receiving data from the client, updating other clients with that data (more on this later) and inserting that data into a database for future use. The server runs several separate programs, either smaller "servers" or programs designed to execute in the background, to accomplish these tasks.

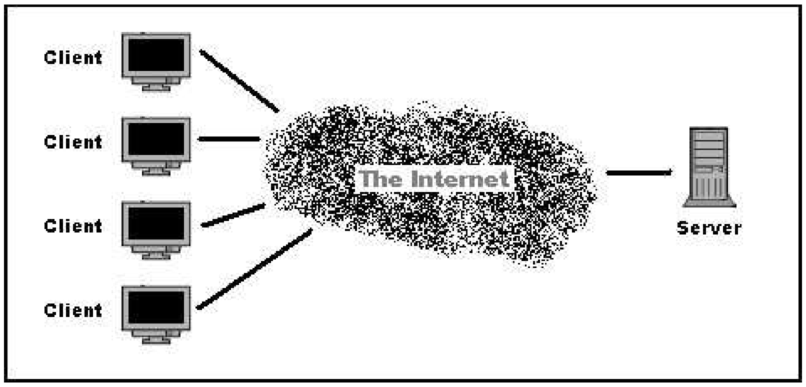
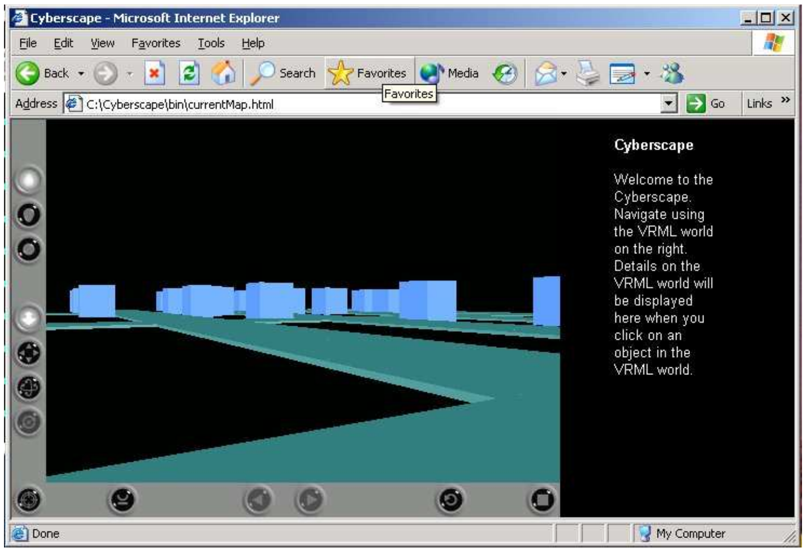


Illustration 1 - The Basic Cyberscape Architecture

The future use in question comes from a Cyberscape client, whose primary function is to request information from the server to build a three-dimensional VRML world which represents the internet. The world consists of streets and buildings where street positions are based on IP addresses and buildings bordering those streets represent web sites hosted on those IP addresses. A user can navigate the world' s pathways and click on buildings to select web sites and launch them automatically in a browser.

2.2. How Worlds are Built

A Cyberscape client requests a map from the server by sending it the IP address of the "area" of the internet which the user wants to visit. The server retrieves information from the database, creates an XML file from that data and sends that file' s URL back to the client. The client is then responsible for using an XSLT stylesheet to transform the XML file into two different files: a WRL file which contains the VRML world and an HTML file which will be used to display information about the objects in the VRML world. Another HTML file which contains links to the previous two files is then launched in a web browser, thus allowing the user to navigate and explore.


Cyberscape - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Favorites
Address C:\Cyberscape\bin\currentMap.html
Go Links
Done
My Computer

Cyberscape

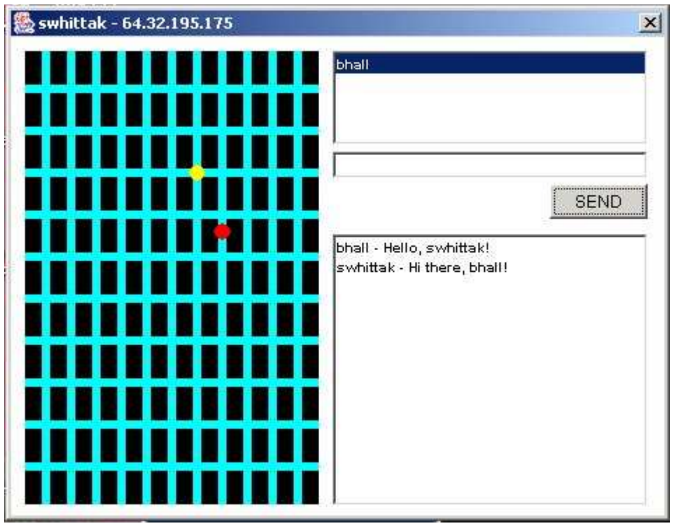
Welcome to the Cyberscape. Navigate using the VRML world on the right. Details on the VRML world will be displayed here when you click on an object in the VRML world.

Illustration 2 - The Cyberscape VRML World

While walking around in the world, a client-side program transfers information about where the user is moving and what websites he or she is browsing back to the server. The server organizes and stores this information, then uses it to build future worlds. This process allows the server to provide the world with features such as popular paths (highlighted in different colours) and popular web sites (given in hit statistics).

2.3. Users and User Interaction

In this latest version of the Cyberscape project, a user must now log into the server with his or her indentity and password (both stored in the server's database) before he or she may request a VRML world. A user is also able to see other Cyberscape users who are in the same world and communicate with them through the use of a "GPS" system. This system provides an arial view of the VRML world



bhall - Hello, swhittak!
swhittak - Hi there, bhall!

Illustration 3 - The Cyberscape GPS Viewer

with the current positions of this user (in red) and every other user in the same world (in yellow). Every user' s GPS receives position updates from the server, which simply routes them when it receives a position update from a client in the same world.

The GPS also allows users in the same world to communicate with one another. A list of potential chat- mates is included at the side of the GPS - all the user has to do is select a recipient, type a message and send it to them. All messages are first passed through the server, which then directs the text to the indended recipient. Hence, GPSs also receive message updates from the server in addition to position updates.

3. Communication: Pre-Project

Before this project was completed, a Cyberscape client would only communicate with the server for one of two reasons: to request a map and to report navigation data.

3.1. Requesting a Map

3.1.1. Work Items and Worker Pools

A work item is a simple abstract object with two purposes: to perform a task and to return the data resulting from that task. It' s that simple. Work items must be added to a worker pool, which provides the ability to store and execute a series of items in a specific order. Any object which adds an item to the worker pool can choose to wait for the work item to finish its task, or choose not to.

For implementation purposes, any class that wants to qualify as a work item must implement the interface WorkItem, with methods performWork() and getData(). The WorkerPool class is a self-contained entity; any class can create an instance of a pool and add items to it which need to be executed.

3.1.2. Map Session Threads

A session thread is responsible for monitoring a socket which any two Cyberscape objects are currently using for communication. It is vital to note that a session thread does not create or destroy its socket - it's the responsibility of the object creating the session to create the socket as well. There are two basic types of session threads: a receive session and a transmit session. As their names suggest, a receive session is responsible for waiting for a packet to arrive on the session socket and a transmit session is responsible for sending a packet on the session socket.

In terms of the implementation, the abstract class MapSessionThread provides basic functionality for sending on and receiving from sockets, while MapRequestSession extends MapSessionThread and provides map request-specific data handling. The final two classes which are actually used, ReceiveSession and TransmitSession, both extend MapRequestSession. Again, none of these classes create the sockets they use; all of them have a parameter in their constructors for the session socket.

The MapSessionThread class implements the WorkItem interface, but the work which it performs is determined by the extending subclass. In this case, a ReceiveSession's task is to attempt to receive and a TransmitSession's task is to attempt to transmit.

3.1.3. Clients and Servers

The physical Cyberscape server, which is a server in the truest networking sense, runs several smaller programs to perform its duties. One of these (in this version of the project) is known as a "server" as well; henceforth, when this section refers to a server, it is referring to this smaller program. A server's responsibilities are to create a socket, listen indefinitely on that socket for a request from a Cyberscape client and handle that request accordingly. The server stops monitoring its socket only when it is told to shut down, at which point it destroys the socket and terminates. If the server receives a request from a client, it is expected to send a response back. To do this, the server uses a transmit session initialized with the socket it created and had been monitoring.

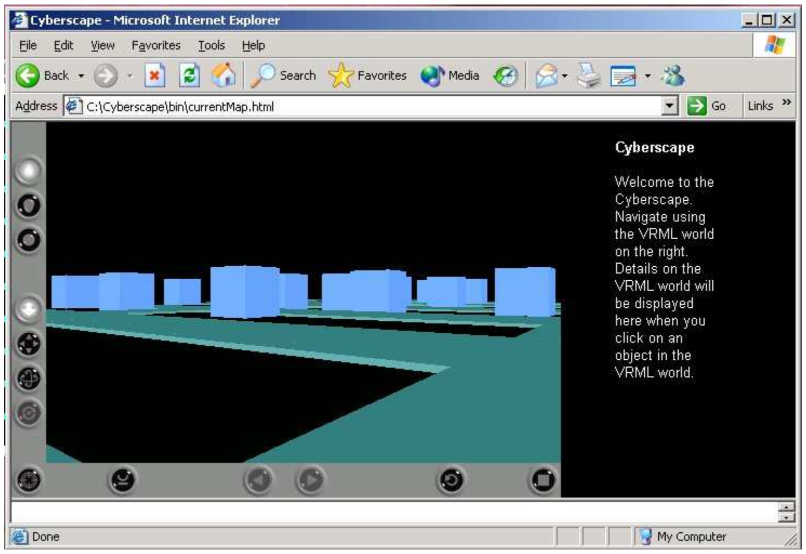
A client is designed to be a self-contained mechanism for communication with the server.   It only requires the information contained in the request from the object that wishes to send it to the server. The client is responsible for creating a session socket, sending the request to the server and then using a receive session (with the session socket) to wait for the server' s response. The client then returns the result of the request to the object that requires it and terminates.

In terms of the implementation, the abstract class MapRequestAbstraction provides core functionality for building sockets, creating a worker pool, starting up and shutting down. Both MapRequestClient and MapRequestServer extend this class and provide a port number on which their sockets are created. A MapRequestClient is responsible for sending the IP address which the user wishes to load to the MapRequestServer. In return, the MapRequestServer is responsible for creating an XML file with the map' s information and returning the URL of that XML file to the MapRequestClient. The MapRequestClient and the MapRequestServer add a ReceiveSession and TransmitSession to their respective worker pools to assist in their communication.

3.2. Reporting Navigation Data

In this version of the project, reporting position and/or website hit data back to the server is far simpler than requesting a map. The process begins with the VRML file itself, which inserts motion triggers into the world. These triggers are assigned to street segments and are activated when the user changes from one segment to another; there are no triggers on the black gaps between streets because those are reserved for buildings. Once the trigger is activated, the issue is how to get that information back to the Cyberscape server.

When a user is navigating in the VRML world, a web browser is helping him or her do that (as shown in Illustration 1). The browser only appears to have two frames loaded: one for the Cyberscape VRML world and one to display navigation and/or hit data. In reality, it has three frames – the third runs along the bottom of the broswer and is set to be "invisible" by the HTML file which the browser loads.


Cyberscape - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Cyberscape\bin\currentMap.html Go Links
Done
My Computer

Cyberscape

Welcome to the Cyberscape. Navigate using the VRML world on the right. Details on the VRML world will be displayed here when you click on an object in the VRML world.

Illustration 4 - The Browser's Invisible Frame Made Visible

When a user activates a trigger, the VRML file sends a command to the invisible frame in the browser to load a URL which contains the internet address of the Cyberscape server and the file "reportHit.pl" which resides on that server. In this version of the project, reportHit.pl requires one parameter: the user' s position. This value is provided by the VRML file, which assigns an IP number to each street segment in the world. When calling reportHit.pl, it simply includes the IP address that was assigned to the street segment whose trigger was activated.

On the Cyberscape server, reportHit.pl simply creates a new text file which contains the position it was provided with as a parameter and exits. A different helper program is later executed on the server which collects all of these files, reads their contents and inserts their position data into the server' s database. This data will be used to construct future maps, which will contain updated information about popular and not-so-popular routes and destinations in the world.

4. Communication: Post-Project

4.1. Why Change It?

As was stated in the introduction, a community requires communication. The previous system worked well for a single map-request-response scenario, but the addition of user profiles and awareness of other users brings several new requirements with it. Among them are the following:

- additional servers must be introduced to handle different user activities
- the system must be able to receive and make more transmissions to accomodate high-traffic functionality
- the system must be able to handle new communication scenarios

Cyberscape' s new communication system retains much of the design of the original. Most of the work went into finding elements of the original classes which would be useful to other objects and formatting them into abstract classes with functionality that could be shared. However, this still involved a major reworking and the changes require documentation.

4.2. Something Old, Something New

The core concepts surrounding work items, worker pools, session threads, clients and servers have all been preserved in this version of the system because they are sound principles which have proven to work well. To briefly recap,

- a work item must perform a task and return the resulting data
- a worker pool stores and runs a group of work items
- a session thread monitors a session socket on a communicating entity
- receive sessions listen for arriving packets and transmit sessions send packets along
- a server creates a socket, monitors it for requests and performs the necessary actions to fulfill those requests

- a client is self-contained communication entity responsible for transmitting a request to a server

However, several aspects of the original system have been modified to allow for scalability and different communication scenarios. The following alterations should be noted:

- one "smaller" server exists on the primary Cyberscape server for each user activity that requires database queries or affects how other users see their world
- servers don't have to exist solely on the primary Cyberscape server – a remote Cyberscape client can now launch a server as well to receive transmissions
- not all user-server communications require a response on the part of the server
- not all communication is user-server based; server-server is also necessary as some user actions may affect more than one server
- every server has a matching client which will handle all aspects of communication with the server and establishes exactly what requests can be sent

4.3. Packets

In addition to the changes described above, a new component has been added to the system to assist in socket communication. Before a client can send data using a socket, that data must first be converted into a string. Likewise, when a server receives data from its socket, it's in string form. Since a server can handle several different requests, there is no guarantee as to the type of data contained in the string. In addition, changing the contents and format of the data string would be messy and difficult; alterations would have to be done in both the client and the server because that's where encoding and decoding are performed respectively.

The packet object is designed to relieve the client and the server of this burden. Every server and each client which waits for a response has a corresponding packet which is responsible for determining the format of the transmitted data. The packet stores the type of request/response and the data necessary for the server/client to perform its duties. Most importantly, the packet is responsible for converting itself into a string before being sent through a socket and converting itself back into a packet once it' s received.

Thus, a typical transmission between a client and a server would occur in the following manner:

1. The client creates a new packet that corresponds to the request and sets the necessary data in the packet
2. The client converts the packet into a string and sends it across the socket
3. The server receives the string transmission from its socket
4. The server converts the string back into packet form and extracts the type of request and the necessary data

4.4. Implementation

4.4.1. Abstract Classes

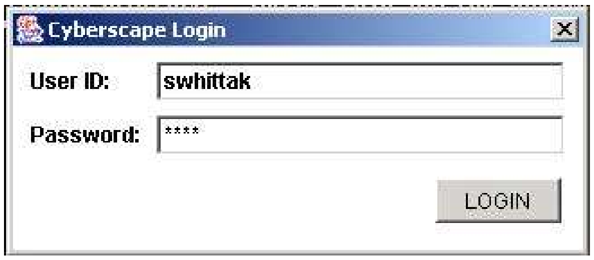
Of all the abstract communication classes, Packet is by far the simplest. The only implementation it requires of its extending subclasses is the method convertToString().

The abstract class SessionThread provides basic initialization and the functionality to send and receive data on a session socket.. This socket must be provided as a parameter in its constructor along with a Packet whose data is handled differently depending on the session thread' s type. A TransmitSession simply sends the data through the socket, while a ReceiveSession sends the data *only* if it did not receive a response on its socket in a certain period of time. SessionThread also implements the WorkItem interface, which allows any extending subclass to be executed using a WorkerPool.

RequestAbstraction is the top-level abstract class for all clients and servers. It is responsible for creating the object' s session socket and worker pool and providing the functionality to destroy the socket when the object is shut down. The abstract class RequestClient implements the ability to send and receive or to just send data on its session socket. It makes the decision on which to do based on the type of session thread which the extending subclass returns from abstract method createSendSession(...): with a TransmitSession, the client simply sends; with a ReceiveSession, the client sends and waits for a response. The abstract class RequestServer implements the functionality to continuously monitor its session socket for transmissions until the server is shut down. A RequestServer only uses TransmitSessions to respond to requests - it doesn' t expect (and should not receive) a confirmation transmission from a recipient. Both RequestClient and RequestServer use their worker pools to execute their session threads and both choose not to wait for a TransmitSession to complete its task. However, a RequestClient does wait for a ReceiveSession to execute so it can examine the data contained in the response.

4.4.2. Login Request Classes

Cyberscape now requires each user to log into the system and expects an update whenever he or she logs out. The LoginRequestServer is responsible for handling both of these events. While a login request requires a response from the server with the results of the attempt, a logout update from the user does not require a response.



User ID: swhittak

Password: ****

Illustration 5 - The Cyberscape Login Panel

The packet classes used for these transmissions are LoginRequestPacket and LoginResultPacket. The former of these must contain the user' s name, password and whether he or she is logging in or logging out.. The latter contains a single boolean which represents the result of the user' s login attempt.

To perform a login request or a logout update, a LoginRequestClient is initialized with the data necessary for the action. The client then creates a LoginRequestPacket and does one of two things:

- if the client is performing a login request, it sends the packet on its session socket, creates a new ReceiveSession with the same packet and adds the session to its worker pool to wait for the LoginRequestServer' s response
- if the client is performing a logout update, it creates a TransmitSession with the packet and adds the

session to its worker pool; note that in this case, the LoginRequestClient does not wait for a response

Once the LoginRequestServer receives the data on its socket, it is converted back into a LoginRequestPacket and examined. The server then does one of two things:

- in the case of a login request, the server checks that the user is not already logged in and that his or her password matches the password stored in the database. If both of these conditions are met, the user' s request is granted and their name is added to a list of users who are logged in. The result of the attempt is then put into a LoginResultPacket and sent back to the user by adding a TransmitSession with the packet to the server' s worker pool
- in the case of a logout update, the server simply removes the user from the list of logged-in users and returns to monitoring its socket

In both of these cases, the LoginRequestServer also has to send one-way updates to the PositionUpdateServer to tell it when users and entering and exiting the system. A complete description of this process will be included in section 4.4.4.

Once the remote Cyberscape client receives the result of the user' s login attempt, it either allows the user to begin requesting VRML worlds or it displays an error message and asks the user to make another login attempt.

4.4.3. Map Request Classes

In order for a remote Cyberscape client to build a three-dimensional VRML world around a website or an IP address, it needs the server to provide it with the relevant data contained in the database. The MapRequestServer is responsible for receiving such requests, collecting the information from the database, creating an XML file containing the data and returning that file' s URL to the remote Cyberscape client.

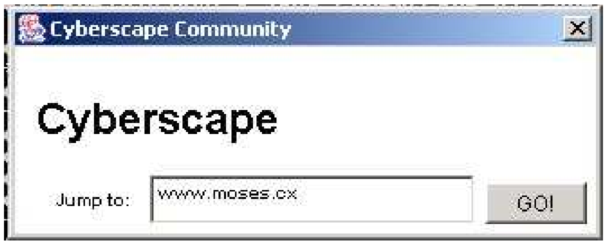


Illustration 6 - The Cyberscape Map Request Panel

The packet classes used for these transmissions are MapRequestPacket, which contains the user name and the IP address that the user is requesting, and MapResultPacket, whose only member is the URL of the file which contains the retrieved data.

To perform a map request, a MapRequestClient is initialized with the data necessary to create a MapRequestPacket. It then sends the packet on its session socket, creates a new ReceiveSession with the same packet and adds the session to its worker pool to wait for the MapRequestServer' s response. Once the server receives the data on its socket, it converts it back into a MapRequestPacket and retrieves the requested URL or IP address. The MapRequestServer then extracts the data from the database, creates the XML file and adds the file' s URL to a MapResultPacket. This packet is then sent to the remote Cyberscape client by adding a TransmitSession with the packet to the server' s worker pool.

Once the URL is received by the remote client, the XML file is read and the information in it is used to build a WRL file which contains the VRML world and an HTML file which contains information about the objects in the world. Another HTML file with frames linking these two files is then loaded into a web browser for the user.

When a user requests a map, the MapRequestServer also has to send a one-way update to the PositionUpdateServer to tell it that a user will be entering a new map and possibly exiting an old one. A complete description of this process will be included in the next section.

4.4.4. Position And GPS Update Classes

Cyberscape now provides users with a complete "GPS" system (shown in illustration 3) to facilitate awareness of and communication with other users who are also in the same world. The GPS has two sections: an arial view which displays every user' s position in the map and an instant messaging area which allows users to communicate with one another. Obviously, a user' s GPS would required updating if a user in the same map changed position or sent a message.

The PositionUpdateServer and GPSUpdateServer work in conjunction to handle these events and update each user' s GPS system. However, that they both do it from different locations; the PositionUpdateServer runs on the primary Cyberscape server while each remote Cyberscape client runs its own GPSUpdateServer. It is also vital to note that all communication with these servers is strictly one-way. No one who sends an update to these servers should expect a response and no one who receives an update from them should send a response either. As a result, both the PositionUpdateClient and GPSUpdateClient classes create TransmitSessions with a server-specific packet, add them to their worker pools and do not wait to receive anything.

The packet classes used for transmissions with these entities are, not surprisingly, named PositionUpdatePacket and GPSUpdatePacket. Unlike the other packets wich have been described previously in this document, these packets can represent one of many update types and the information they contain can vary greatly depending on the type. Descriptions of each type of packet and the data they contain will be included later in this section.

4.4.4.1. PositionUpdateServer

The PositionUpdateServer is responsible for the following duties: tracking which users are currently in which worlds; storing every users' remote address so their GPSUpdateServer can be contacted; handling position updates for any user who moves and the other users in the same map; and routing messages from the sender to the recipient. The PositionUpdateServer receives updates through its session socket from the following entities and performs the following handling activities:

- The LoginRequestServer sends a "user logged in" update packet which contains the user' s name and remote address. Once received, the PositionUpdateServer adds the information to its list of current user data
- The MapRequestServer sends a "user enters map" update packet which contains the user' s name, the requested map ID and the user' s initial position in the map. In response, the PositionUpdateServer

must send a flurry of GPSUpdatePackets:

  - one telling the user's GPSUpdateServer to close its old GPS viewer
  - one telling the user's GPSUpdateServer to open a new GPS viewer with the user's position and the other users' positions contained in the packet
  - one telling the GPSUpdateServer for each user in the previous map (if it exists) to remove the user given in the packet from their GPS viewers
  - one telling the GPSUpdateServer for each user in the new map to add the user to their GPS viewers at the position contained in the packet

- The Perl script "reportHit.pl" (changes to that part of the system will be described later in the document) sends a "user changed position" packet which contains the user's name and new position. Once received, the PositionUpdateServer must route this information to the GPSUpdateServer for each user in the same map (including the one who moved) and tell them to update the given user's position to the given value in the GPSUpdatePacket
- The GPS viewer sends a "user sent message" packet which contains the name of the sender, the name of the recipient and the message text. Once received, the PositionUpdateServer retrieves and uses the remote address of the recipient user to forward the message through a GPSUpdatePacket to the recipient's GPSUpdateServer
- The LoginRequestServer sends a "user logged out" update packet which contains the user's name. In response, the PositionUpdateServer removes the user from its list of current users and tells the GPSUpdateServer for each user in the same map to remove the user contained in the GPSUpdatePacket from their viewers

4.4.4.2. GPSUpdateServer

The GPSUpdateServer performs the following handling activities in response to the updates it receives from the PositionUpdateServer on its session socket:

- a "close GPS" update prompts the server to dispose of the current GPS viewer
- an "open GPS" update prompts the server to create a new GPS viewer which contains the users at the positions included in the GPSUpdatePacket
- a "user changes position" update prompts the server to update the user's position to the value given in the packet
- a "user enters map" update prompts the server to add the user to the GPS at the position given in the packet and to add the user's name to the list of possible message recipients
- a "user leaves map" update prompts the server the remove the user from the arial view and the list of possible message recipients

Although one may assume that the GPSUpdateServer is also responsible for sending "user sends message" updates to the PositionUpdateServer whenever the user wishes to transmit a message, that is not the case. The actual GPS viewer uses a PositionUpdateClient to send the update to the server when the user selects the "SEND" button in the GUI.

4.5. reportHit.pl

4.5.1. Changes to the Script

Section 3.2 of this document included a detailed description of how a user's naviagtion data is reported to the primary Cyberscape server using the script reportHit.pl in the previous version of the project. In fact, very little about this process has been altered for this version of the project. To briefly recap,

- the VRML world assigns a motion trigger and an IP address to each street segment in the map
- the browser which displays the VRML world contains an "invisible" (or hidden) frame for this purpose
- when a user activates a trigger, the VRML world tells the invisible frame to load reportHit.pl's URL

However, reportHit.pl has undergone several changes. First, it now requires both the user' s name and new position as parameters, instead of just the position. Second, it is no longer in charge of writing the position data out to a file. Instead, the script performs a local socket transmission to the PositionUpdateServer with the user' s name and new position as the data.

These changes raise two important questions: how does the VRML world provide report.pl with the user' s name and who now handles writing the received position to a file? The answer to the second question is simple: a separate thread is executed by the PositionUpdateServer to write the information to a file. The answer to the first question is a little more complex and involves returning to the stage when the user has just logged in. Section 2.2 gave a complete analysis of how the VRML world is built using a client-side XSLT stylesheet and a server-side XML file. The XSLT file contains the following line:

```
        field MFString userName <a name>
```

Whenever a login attempt is successful, the user name included on this line is overwritten with the name of the user that just logged in. Thus, when the XSLT file is used to create the VRML file, the VRML file will also contain this line and be able to include the user' s name as a parameter in its call to reportHit.pl.

But why change the XSLT file when it' s the VRML file that ultimately needs to contain the user' s name? First, the XSLT file is rather short and a typical VRML file is rather long; it doesn' t take as much time to edit the XSLT file. Second, a user may load multiple maps during his or her time using Cyberscape; it is far more efficient to change the XSLT file once upon login than to change a VRML file every time a user requests a new URL.

4.5.2. Why Use This Method?

Although this method has proven itself, there are many different options and variations which could have been implemented instead. Several of these were investigated, but eventually rejected because they each introduced a flaw in the system. The following paragraphs will described each option that was explored and explain why it was not implemented.

The first thing to consider is that although VRML does provide motion sensors (which are extremely handy), the world is limited in what it can do once a sensor has been triggered. Since the world is loaded through a web browser, the options are limited to what can be done in that browser. As a result, the "invisible" frame was introduced to allow the VRML world to use CGI to report on a position change. This fact only leaves two options: a Perl script and a Java applet

Unfortuantely, a Java applet is not a good option; it is slow to execute because it needs to start a virtual machine. In the VRML world, users can run around fairly quickly. This may cause motion sensors to be triggered so fast that an applet cannot complete its task before another sensor is triggered and the applet is reloaded. The result would be a series of lost updates which are never reported to the PositionUpdateServer and a group of users with out-of-sync GPS viewers.

An obvious question at this point would be why the reporting mechanism has to be executed on the Cyberscape server. Would a client-side mechanism not be more efficient because it could update the user's GPS viewer locally before sending an update to the PositionUpdateServer for the rest of the users? That would be an ideal situation, but it is not feasible at this point. Assuming that a Java applet could function quickly enough to overcome the motion sensor issue, it would fail again because of communication problems. Because an applet runs inside a virtual machine, it has very few permissions with the actual machine it is operating on. As a result, it would not be able to connect to a socket or communicate with the PositionUpdateServer. But what about a Perl script – would it not be able to perform socket communication? Yes it could, but there is no guarantee that the user has the Perl interpreter on his or her machine. Without a Perl interpreter, the script couldn't run and the PositionUpdateServer would not receive any updates at all.

Cyberscape's current system performs its duties well; reportHit.pl executes quickly and sends updates to the PositionUpdateServer reliably and efficiently. It will definitely stay in place until a more efficient option presents itself.

5. How To Create a New Server

5.1. Basic Concepts

The key to defining a request-specific series of session threads, clients and servers to to first create a request-specific packet, and a response-specific one as well is the client requires it. A packet defines the format of communication between a client and a server (or vice-versa): the types of requests which a client may send, the data necessary for the server to fulfill that request and the information the server will send in its response. All packets must extend the abstract superclass Packet to be sent by a TransmitSession and a ReceiveSession. The packet should also implement a static method which converts it from its string form (obtained by calling convertToString()) back into its packet form.

Once the packets has been created, the session threads should be defined. The types of session threads which are implemented will determine the sequence of the communication that takes place. Remember, a client which needs to receive a response will need a ReceiveSession that takes a request-specific packet and returns a response-specific packet, but the server will only need a TransmitSession to send the response-specific packet. However, a client that does not need a response will only need to use a TransmitSession for the request-specific packet and the server will not need a session thread because it is not transmitting anything back. All session threads should extend the abstract classes TransmitSession or ReceiveSession and require a request or response-specific packet in their constructors.

The last objects which should be created are the client and the server. Of these, the server' s behaviour is by far the most complex because it will need additional functionality to handle the requests it receives. However, RequestClient and RequestServer provide all of the functionality for socket communication – the most important thing is to make sure they both use the request and response-specific packets at the right time.

5.2. Example: AdvertisingRequestServer

This system was designed to make is fairly simple to implement a new client-server relationship. It is apparent that as Cyberscape expands, new features will have to be added and those features will most likely require communication between the remote Cyberscape client and the primary Cyberscape server. To illustrate how simple this process is, consider the following hypothetical situation.

Cyberscape has been launched and is a huge worldwide success, but the creators of the project have not been paid for their development efforts or for hosting the servers necessary to run the system. Concerned about paying the rent, they decide to implement an advertising feature which will provide the revenue to keep them afloat. How do they do it?

The first step is to analyze the data needed by both the client and the server, and the sequence in which transmissions will be made. In this case, the client needs to tell the server that it requires an image which represents the advertisement, and the server needs to send the URL of an advertising image back to the client. From this, the following classes can be defined:

- AdvertisingRequestPacket which has no parameters (just sending it tells the server what the client needs)
- AdvertisingResponsePacket which has one parameter for the URL of the advertisement' s image
- AdvertisingReceiveSession which requires an AdvertisingRequestPacket (in case it needs to resend) and attempts to receive an AdvertisingResponsePacket
- AdvertisingTransmitSession which requires an AdvertisingResponsePacket to send
- AdvertisingRequestClient which creates an AdvertisingRequestPacket, sends the packet on its session socket, adds a new AdvertisingReceiveSession to its worker pool and waits for the result
- AdvertisingRequestServer which receives an AdvertisingRequestPacket, randomly chooses an image, creates an AdvertisingResponsePacket and a new AdvertisingTransmitSession and adds the session to its worker pool to send the result

Once the AdvertisingRequestClient receives an AdvertisingResponsePacket back from the server, it simply returns the information contained in the packet to the class that executed the client and then terminates. The class that executed the client then loads the URL into a new popup window, the advertising takes place and the creators of Cyberscape get to pay their bills.

6. Other Cyberscape Objects

Although the bulk of the work was focused on the communication system, a number of other important items were designed and implemented to assist in the completion of the project' s goals. These objects perform essential tasks, and although the concepts behind many of them may seem simple, that fact does not diminish their importance or the frequency of their use.

6.1. PropertyHelper

The purpose of the PropertyHelper class is to provide a centralized and static means of accessing Cyberscape properties (strings, integers and booleans) from the file *cyberscape.properties*. Currently, this property file include the values of VRML tags, file names, database table identifiers and dialog labels.

Any class which needs access to these values can use one of the PropertyHelper' s static methods for value retreival. This class also provides static methods for setting values and for re-writing them to the cybserscape.properties file. The advantages to this system are impressive: values can changed without having to edit and recompile the code (which is extremely useful with respect to internationalization) and values entered by the user can be saved for later use. While the use of this class is presently limited, it lays the foundation for individual user settings to be implemented in the future.

6.2. XMLFileAdjuster

Section 4.5.1 of this document includes a description of the how the XSLT file which translates the server' s XML file into the VRML world must be altered to contain the user' s name. It' s the

XMLFileAdjuster class that is responsible for this task in the implementation. The method in which the XMLFileAdjuster performs its responsibility is not complex: it opens the file, reads the contents, inserts the user' s name (given in its constructor) and writes the data back to the file. However, it' s simplicity does not diminish the fact that if this is not done properly, the user' s world will not load and no user interaction will be available.

6.3. CyberscapeApplication

In the previous version of the project, the AWT dialog CyberscapePanel (seen in illustration 6) was in essence the Cyberscape remote client; launching the dialog allowed the user to request maps from the server and closing the dialog terminated the session. This version of the project requires a great deal of synchronization: the login dialog, map request panel and client-side GPS server must all be launched at the right time and in the correct order.

It is now the responsibility of the CyberscapeApplication class to represent the remote client and handle every action required of a Cyberscape session. Currently, this involves the following:

- create and display the CyberscapeLogin dialog
- exit if the user' s login atempt is unsuccessful
- use the XMLFileAdjuster to insert the user' s name into the XSLT file
- launch the GPSUpdateServer
- create and display the CyberscapePanel dialog for map requests
- shut down the GPSUpdateServer when the user decides to exit

6.4. CyberscapeLogin and CyberscapePanel

The CyberscapeLogin class is an AWT dialog which allows the user to log in and begin a new Cyberscape session (shown in illustration 5). It validates the user' s information using a helper class (CyberscapeLoginHandler), which in turn uses a LoginRequestClient to contact the remote Cyberscape server' s LoginRequestServer.

Although the CyberscapePanel was implemented in the previous version of this project, it underwent a noteworthy adjustment in this version of the project. As a result, both it and CyberscapeLogin share one important feature: they are now modal dialogs. The reason for this change is the fact that the CyberscapePanel no longer represents the remote Cyberscape client; the CyberscapeApplication does. As a result, the CyberscapeApplication needs to wait for both of the dialogs to complete their execution. Specifically, the application needs the result of the user' s login attempt from the CyberscapeLogin panel before it launches the CyberscapePanel for map requests. The application must also wait for the CyberscapePanel to close before it can consider the user' s session terminated and perform cleanup.

6.5. CyberscapeGPS and GPSGridCanvas

The CyberscapeGPS class is an AWT dialog which provides the user with an arial view of his or her VRML world and the ability to talk to other users in the same map. It is created and updated by the remote client' s GPSUpdateServer, which in turn receives updates from the Cyberscape server' s PositionUpdateServer. The dialog is comprised of two different areas: the GPS area (controlled by the GPSGridCanvas) and an instant messaging area where the user can type a message and send it to another user in the same map (shown in illustration 3). The CyberscapeGPS is responsible for sending these messages to the PositionUpdateServer using a PositionUpdateClient; the server will then route them to the recipient' s GPSUpdateServer.

The GPSGridCanvas class extends the AWT Canvas class and provides a drawing surface to display the streets and the users in the current map. It redraws itself when the CyberscapeGPS dialog is redrawn as a whole or when GPSUpdateServer alerts the CyberscapeGPS that a user had changed position in the map. The GPSGridCanvas is also responsible for calculating each user' s position in the viewer given user' s position in the world, which is given as an IP address. All of the properties necessary for this calculation are retreived from the PropertyHelper, and hence are easily adjusted.

7. Project Goals (Accomplished and Otherwise)

Every honours project proposal contains a detailed list of requirements, both for the project and for personal education and experience. The proposal for this project was no different, but it was slightly over-ambitious in its stated goals. The result is that this project satisfies the original vision and contains the components necessary to fulfill all of its requirements. However, the time-consuming tasks outlined in the previous sections prevented one of the stated goals from being implemented. In addition, the limitations associated with VRML resulted in some adjustments to the manner in which the interactive system was implemented and in the education received from this project. The details surrounding these two exceptions will be explored in this section.

7.1. Stated Goals

The following requirements have been taken from the original honours project proposal in verbatim.

7.1.1. Project Goals

- To transform the Cyberscape virtual world from a map which remains static once the user has requested it to a map which is dynamically updated with other users' positions
- To allow Cyberscape users to see other users in the same map
- To allow Cyberscape users to communicate with other users in the same map
- To allow Cyberscape users to remain anonymous or invisible while inside the virtual world

7.1.2. Personal Education Goals

- To gain a deeper understanding of how the Internet functions and the standards which Cyberscape uses to gather information from the Internet and create its virtual world (i.e. XML, VRML)
- To learn how to mediate and co-ordinate in real-time between multiple clients and a central server
- To determine how to authenticate individual users and ensure no one can masquerade as another Cyberscape user

7.2. Project Goal Status

From the detailed descriptions which have been given in the previous sections of this document, it is apparent that all of the project goals have been met except for the very last: the ability for Cyberscape users to remain anonymous and/or invisible while inside their world. As was stated at the beginning of this section, it was a lack of time which prevented the implementation of this item. However, all of the components which are necessary to perform this user function are in place. In fact, this aspect of the system could be implemented in the following manner:

- Include two radio buttons in the CyberscapeLogin dialog: one for anonymity and one for invisibility
- Transmit the selection status of these options in the LoginRequestPacket to the LoginRequestServer
- Forward the selection status of these options from the LoginRequestServer to the PositionUpdateServer with the "user logged in" update which is currently sent anyway
- Store the selection status of these options in the PositionUpdateServer with the user data that is currently kept (user name and remote address)
- Check the selection status of these options for each user when the PositionUpdateServer is formatting an update for a client's GPSUpdateServer – an anonymous user's name should be omitted and an invisible user should not be shown in the GPS whatsoever

7.3. Personal Education Goal Status

In addition to all but one of the stated project goals being met, all but one of the personal education goals has also been fulfilled. However, this does not mean that the experience gained through this project remains bound to the items listed in the proposal. Instead, a vast amount was learned about other areas which were not originally envisioned but are nonetheless useful and noteworthy.

The original design for the community aspect of the Cyberscape system involved using the VRML world to display other users walking around in the same map. Unfortuantely, VRML contained too many limitations (which will be discussed later in this document) to allow the project to be implemented this way. As a result, the GPS system was installed in its place and a "deeper understanding" of VRML was achieved, but not in the form of practical experience.

Instead, the implementation of the GPS system provided an opportunity to learn more about the Java AWT utility (included in the JDK) for building GUIs and about network socket communication. Both of these were investigated extensively while rebuilding the new communication system and creating the drawing system for the GPS viewer. With respect to personal education, the information learned about these areas has already proven useful for other assignments and will continue to be worthwhile.

8. Observations and Challenges

8.1. The Limitations of VRML

As was stated in the previous section, the original project proposal indicated that the existing VRML world would be used to implement the Cyberscape user community. However, upon further investigation it was concluded that VRML could not properly perform this task and the GPS system was implemented instead. But what exactly are the reasons behind this decision?

The primary issue which prevents the VRML world from performing Cyberscape community functions is its lack of an external updating mechanism. VRML does provide scripting facilities which can be used to track events, such as a user's position. There are also ways to implement changes to objects in the the world in response to an event. However, this functionality must be programmed into the VRML file at load time and the event must originate from inside the world. Although an extensive online search was conducted, no examples of stimulating events from external sources could be located, nor any viable examples of multi-user VRML environments.

The fact is that when a VRML file is loaded, that's it. For its part, the file remains static and the web browser's VRML plugin simply reads the file and then runs the world on its own. As a result, the current communication system (which involves the remote client receiving updates from the server) would not perform well because an updated VRML world would have to be completely reloaded in the browser each time an update is received, thus wreaking havoc on the user' s ability to navigate. Similar updating suggestions which were given by interested parties, such as streaming user data to the client, would also not work for the same reason.

One ingenious suggestion involved making the VRML world which the user loads small enough to cover a single street segment. Changing position to a different segment would involve loading a new map which would contain updated user information. Unfortunately, this idea is not viable. First, the user would only see the segment around him or her - none of the other Cyberscape objects or users would be visible. Second, it takes time to load a VRML world; users would not really be able to walk so much as step, stop, step, stop.

Although VRML has performed its duty in both the initial and current version of the Cyberscape project, it is apparent that the time for a replacement is fast approaching. One option would be to use a utility such as Java3D. It allows the programmer to create a 3D graphics environment, provides optimization through hardward-independent techniques and contains the functionality to dynamically alter the world at runtime. One small drawback is that Java3D cannot provide picture-quality graphics, which may prove tricky with respect to inserting advertising (especially billboards) into the world. Another option could be to use an existing video game engine; it would definitely be able to handle the necessary interaction between multiple users, but may introduce other limitations such as restricting the user' s choice of operating system.

8.2. Error Handling

At the moment, Cyberscape does not have a single mechanism for handling exceptions or any other errors which may occur on a remote client or on the primary server. Although exceptions are caught and errors are detected, the accompanying message is simply written to STDOUT instead of being properly recorded. It should be considered a priority in future versions of this project to create a logging mechanism for both the client and the server, and an automated reporting mechanism which allows the client to send errors (perhaps in an email message) to the server as they occur. This would create an extensive and informative resource for Cyberscape developers when they attempt to troubleshoot and debug the system.

8.3. Scalability

Implementing multi-user functionality in Cyberscape is one thing; making it work for more than a couple of users at a time is quite another. There are several aspects of the system to consider when investigating how Cyberscape could support a large number of users at the same time, but the following subset should hightlight the most important items.

The first thing to consider in a system featuring high-traffic socket communication is the size of the data that is being sent and the frequency of the messages being sent and received. The current communication system performs extremely well on this point – both the data being sent and the number of transmissions is minimized. The use of the Packet class ensures that all of the data necessary to perform a request or a response can be converted to and from a string by a single entity. This translates into fewer transmissions because all of the data can be sent in one fell swoop. In addition, the Cyberscape' s position updating mechanism is selective in which users it contacts – only those who are in the same map as the user who changed position receive an update. This is far more efficient and scalable than a system which chooses to broadcast its updates, regardless of whether a user needs it or not.

The second aspect of the system which requires attention is the user of worker pools by each of the smaller server programs which reside on the primary Cyberscaper server. When a worker pool is initialized, it creates a certain number of "worker" objects which are responsible for executing any work items which are added. If every one of a work pool' s workers are busy, the work item is added to an ordered collection to wait its turn. The problem which may arise from a large number of users is that a server' s worker pool could receive work items at a faster rate than it can execute them. Worker pools are extremely useful and vital to keeping the system synchronized. Hence, a series of experiments should be conducted to determine an optimal number of work items; not so many that the worker pool is physically bogged down, but not too few that the worker pool falls behind in its responsibilities.

The third major item to consider is the viability of using a web browser to execute Cyberscape' s three-dimensional world. The current version of the project uses Internet Explorer to load the VRML map and it is prone to occasional sluggish and choppy navigation when multiple users are also using the system and the client-side GPS server is receiving position updates. It should be consideration in future versions of the project to investigate alternate software for displaying the world, especially if a switch is made away from VRML to another graphics system.

9. Conclusion

Athough not every single goal given in the proposal was met, the fact remains that the completed project does fulfill the original vision and meets all of the core requirements which were expected of it. Cyberscape users can now see each each other; they can now communicate with each other; and they can now watch each other navigate the world and move through the map with abandon while the GPS maintains an updated viewpoint. And perhaps more importantly, the underlying communication system has been extensively rebuilt to facilitate future development and expansion.

When Ben Hall and Jason St. Cyr first undertook this project, they were both looking for more than a half-credit and a decent grade. They both wanted to focus their efforts on a project which interested them and which they could pursue once their university careers were completed. It has been a pleasure to continue their work and help to complete their ambitious vision. As a learning experience, this project has been invaluable and that is the most a student could expect from any honours project. Hopefully, Ben and Jason will consider this contribution to their work invaluable as well.

References

Java 3D.ORG FAQ (2003). http://www.j3d.org/faq/intro.html.

Gagnon, Real (2000). Java How-To. http://www.rgagnon.com/howto.html.

Hall, Benjamin (2002). Cyberscape: A Three-Dimensional View of the Internet.

Hall, Benjamin (2003). Private Communication.

Pillai, Premshree (2002). Socket Programming in Perl. http://premshree.resource-locator.com/articles/perl_sockets.htm.

Reed, John A. Simple Example of Components, Containers, Layout Managers and Graphics Classes. http://memslab.eng.utoledo.edu/~jreed/mime-6150/18nov97.pdf.

St. Cyr, Jason (2002). The Cyberscape Project: Pushing Internet Browsing and Data Mining to a New Level.

St. Cyr, Jason (2003). Private Communication.

Weiss, Michael (2003). Private Communication.