1) Abstract

This report recounts my contribution to the Cyberscape Project, a joint Honours Project for both myself and Jason St. Cyr. It provides an overview of the Cyberscape system, and then investigates my specific contributions in more detail. Because this project represents our combined efforts, please refer to Jason St. Cyr's report for a better overall understanding of client-side development.

While time constraints did not allow development of Cyberscape to the originally envisioned point, a strong foundation has been laid for future development, and many of the needed design decisions have been made. The resulting software is indeed usable, and shows the potential of Cyberscape.

This report first provides an overall overview of Cyberscape, and why the project was attempted in the first place. Next, a detailed overview of the server components is provided. Finally, some of the more interesting features and limitations are discussed. Throughout the document, the challenges faced as well as the successful and unsuccessful approaches to solutions are discussed.

2) Acknowledgments

First and foremost, I would like to thank Jason St. Cyr for inviting me to join him in developing his Cyberscape vision. He has provided me with focus, and given me an opportunity to make use of many of the skills I have developed over the past four years. While I could have found another honours project, this has proved to be a unique challenge, and an interesting experiment.

Our project adviser, Dr. Michael Weiss, has also been a tremendous help. He has guided us, advised us and has given us space to investigate things on our own. Dr. Weiss has shown much support and enthusiasm for this project.

In order to make any reasonable headway in the short time frame allowed for an honours project, this project has drawn substantially from existing software infrastructure. The data collection was made possible through modifications to the "Muffin" HTTP proxy software and all database work is done using the excellent PostgreSQL RDBMS with it's accompanying JDBC module for our Java connectivity. The choice of PostgreSQL was a fortunate one, as in addition to providing a solid RDBMS, it also has excellent documentation supporting it. I would like to take this opportunity to thank author Bruce Momjian for first writing "PostgreSQL: Introduction and Concepts" and then providing it free over the Internet. It has been an invaluable resource.

Finally, I would like to thank Chris Corman and the rest of the webHancer development team. Cyberscape's architecture draws heavily on experiences gained during my co-op term there. webHancer's highly distributed system has successfully scaled to several million users. Exposure to this system greatly influenced many design decisions and has helped us to create an equally scalable system.

3) Table Of Contents

4) Introduction

Since the commercialization of the Internet all of our lives have radically changed. Businesses have climbed to spectacular heights and plummeted to new depths. However, at the end of the day, all we are left with is a vast collection of unrelated servers each in their own world, providing unrelated services with little or no understanding of the bigger picture. What is the Internet? What does it look like? What shape is it? Is there a better way to use this tool and view this world? These questions are what drives the Cyberscape project. Cyberscape hopes to give a three dimensional representation of what the Internet is and how all these disparate servers are in fact related to one another. While search engines attempt to bring order to the chaos, Cyberscape aspires to be something more than just a series of links. Cyberscape will represent how the physical machines sit in the virtual world that we have created.

This virtual world would be capable of using visual metaphors to display the relationships between websites, the size of a website, the popularity of various websites, paths traveled by other users, other users in the system, and other information which may be desired.

Because this concept is so broad, the development of Cyberscape is being done jointly as the honours project for both myself and Jason St. Cyr. His focus is on actually creating and displaying the 3D world, while my focus is on implementing the server-side of the Cyberscape architecture.

5) Goals Accomplished

The original vision for the Cyberscape project has been achieved. While the finished project falls short of our original lofty goals, the system does meet all stated requirements. The foundation has been laid for a robust, distributed system that could one day offer the rich user environment that was originally envisioned. At this point, data is successfully collected on the client machine, is transmitted to the Cyberscape server and is then used to provide the 3D view of the Internet.

In addition to the core requirements being met, several libraries have been written that may have applications outside Cyberscape. Also, great care has been taken to use abstraction and encapsulation wherever possible, resulting in a very cleanly designed system.

From the initial proposal, the following stated goals have been met:

- Gather information to provide information needed to construct a persistent world:
  - Server component to retrieve overall information from existing web servers
  - Client-side component to acquire information specific to Cyberscape users
  - Collect actual, physical network information to provide the foundation for the Cyberscape world. Methods for acquiring this information are still under investigation. Options at this time include calculating the data manually, using a server-side module, or making use of a proxy system for data collection.

- The above information will be transmitted to the Cyberscape client, which will construct the 3D world. Please refer to the proposal by Jason St. Cyr for more information as to the applications of the collected data.
- Enough data should be collected independent of any modules residing on the actual web servers to create at least a basic 3D world. Only additional "nice to have" information should be gathered through per-server modules.
- Above information will be used to create "paths" for use by the Cyberscape client. A path would represent common routes through the Internet that users of the Cyberscape client travel. For instance, a user may commonly travel from IP address 192.168.1.1 to 192.168.1.2 or from ebay.com to yahoo.com.

In addition, we have also accomplished the following:

- The client-side proxy will collect basic usage data (for example, hits for a given website.) This will be the foundation for how we show the popularity of a given site, which is one of the criteria for the appearance of the site in the VRML client.
- The user should be able to somehow search the world. Perhaps initially searching for a given domain name will provide a list of similarly named domain names that will allow the user to "teleport" to the requested domain's IP address. While the search isn't overly complex, we do allow users to travel to arbitrary domain names or IP addresses.
- Since the ultimate goal of Cyberscape was to provide a useful user community, we have integrated a web-based Internet Relay Chat (IRC) client as well as a IRC server as a part of the Cyberscape server component. This does provide basic user communication.

6) Architecture Overview

The Cyberscape system can be broken down into client and server halves, illustrated below. Both halves consist of several sub-programs that make up the overall system. On the client side, there is a proxy, a transfer agent, as well as the actual Cyberscape client.

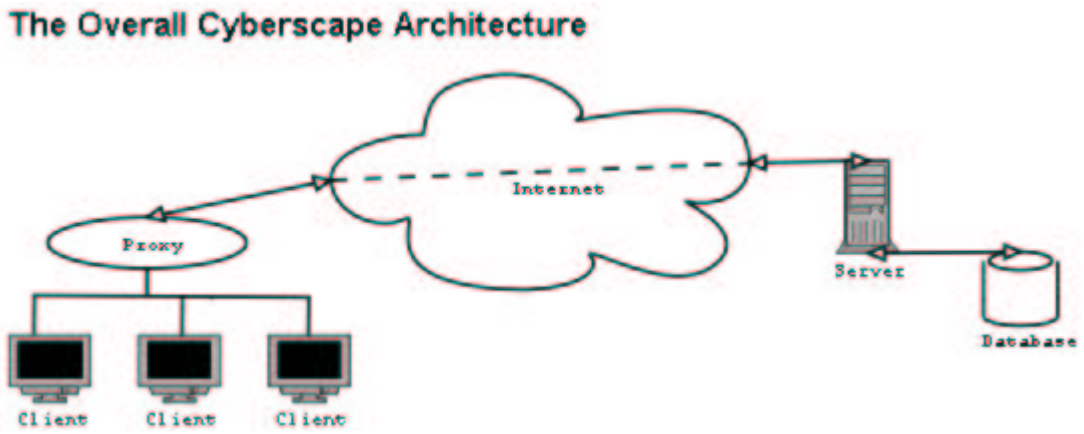

FIGURE 1: Cyberscape Architecture Overview

The Cyberscape architecture has been designed to be as scalable as possible. The main method of data collection is done on the client PC regardless of whether or not the Cyberscape client is in use. This distributed method of data collection has both advantages and disadvantages, as will be discussed in the next section. However, it was determined to be most comprehensive and efficient method of data collection available.

The data collected is used to display a 3D VRML world based on the IP addresses and domain names that make up the Internet. This world can then be traversed, as a Cyberscape user "walks" from IP address to IP address. This 3D world is what we generally refer to as the Cyberscape Client. If an IP address hosts a website, the user can click the building, which will launch a web browser on the web site.

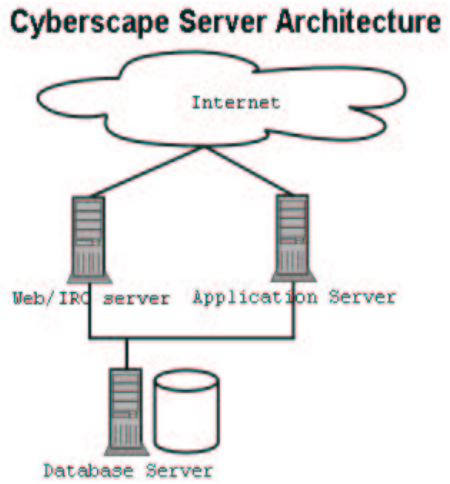

FIGURE 2: Cyberscape Server Architecture

Cyberscape collects client-side data which is automatically transmitted back to the appropriate server. Once the client-side data collection program has finished it's job, this data serves as the foundation of the Cyberscape experience. The data is read by a Java program which inserts it into the SQL database. The server side consists of a database, a mediator-program responsible for receiving incoming proxy data, a program responsible for handling client requests, and a data-collection program design to run automatically in the background.

This data is then used by the Cyberscape launcher to request a map which is built by querying the database. A URL is sent back to the client which then downloads the generated XML map. Next, the client applies an XSLT stylesheet to convert the XML file into WRL and HTML components, which reside locally on the client system and are what the user ultimately sees as the Cyberscape world.

Once within the VRML world, the user walks from building to building, triggering touch and proximity sensors that transmit data back to the server. This data, received by a perl script, is placed in a text file, which is loaded into the database by the same program responsible for proxy data files. This process provides the next Cyberscape user with that much more data, building and changing the world through use.

6.1. The Division of Labour

As has been mentioned, this project represents the combined efforts of both myself and Jason St. Cyr. Generally, Jason has been responsible for the client side, and I have worked on the server side. Specifically, Jason has developed the VRML client, the XML format, the XSLT translation of the XML files, the actual code that eventually generates the XML file, and the communication layer between the Cyberscape server and client halves.

My tasks have included the complete design of the SQL database, client-side data collection using MuffinCS, the communication layer between the SQL database and the XML data creation. I have also setup and tested the IRC server and clients, and have developed the applications that are responsible for reading received data files and loading them into the database. Finally, I wrote the IPLookup program that gathered much of our initial data, and continues to be useful in quickly building portions of the Cyberscape landscape.

7) Investigation of Main Components

The first three components which were investigated deal with data collection. Initially it was assumed that, by making use of DNS, it would be possible to retrieve all domains associated with a given IP address, and therefore easily obtain a complete list of domain and IP address pairs. However, due to the extremely distributed nature of DNS, this is simply not possible. Given that this information makes up the cornerstone of the Cyberscape landscape, a multi-faceted approach has been used to construct as much of this information as possible. At present, two main methods are used to pair IP addresses and domain names. First, a Java program, IPLookup, performs reverse lookups on the IP addresses to obtain primary ownership of the domain. Second, a proxy program sits on each client PC. It collects browsing information through use of an HTTP proxy, while the initial client-side collection made use of the SOCKS protocol. Each of these components will now be examined in more detail, along with a third method that was attempted at the start of the project.

All collected data is stored in an SQL database. After investigating the current data collection methods, the initial base schema and classes responsible for managing the database from Java will be examined, as well as the program which loads the data files into the database. Finally, performance issues and the integrated Internet Relay Chat component will be discussed.

7.1. Server-side Data Collection: IPLookup

IPLookup is a Java program that spawns multiple threads, each of which are responsible for doing a reverse lookup of several "a" portion IP addresses. That is, if an IP address is broken up into the following: a.b.c.d where a,b,c and d are between 0 and 255, each thread is responsible for a through a+i IP addresses. This means that each thread must examine millions of IP addresses with the eventual goal of collecting data on all of the 4,228,250,625 possible IPv4 IP addresses.

As an example, to collect on all possible IP addresses, 51 threads would be spawned, each responsible for 5 "a" portion addresses. This would mean that each thread would be required to collect data on 82,906,875 IP addresses in order to get a complete map of the Internet.

On startup, each of these threads queries the database to retrieve the last IP address that it worked on, and it then continues where it left off. As new IP address /domain name pairs are discovered they are inserted into the database, using the classes discussed in the Database section of this document. The domain name/IP pairs provide the foundation on which the Cyberscape world relies.

This brute-force method of data gathering is far from the preferred method, however, it was necessary to allow the Cyberscape world to have any meaningful structure quickly.

While some optimizations can be made by removing blocks that are known to be private (such as class C networks starting with 10 or 192) this is still an extremely slow process. Furthermore, simply looking up the "owner" of an IP address in no way paints a complete picture of the situation. Many exceptions and problems exist with this method, several of which are listed below:

- Given the vast amount of possible IP addresses, the process is very slow
- Many domain names can share the same IP address
- Many IP addresses can share the same domain name
- Many IP addresses are "owned" by specific companies or organizations
- Many IP addresses do not run websites, so simply collecting domain/IP pairs does not guarantee that the IP address is of any interest
- The code is designed entirely to work with IPv4 only, this is considered a limitation, but an acceptable one at this point, given the slow adoption of IPv6.

There are more concerns than this, but these are the greatest issues encountered thus far. At the time of writing, IPLookup has successfully collected data on over four million domains. This data has been collected over the course of roughly one full week of data collection, spread out over one month. The current implementation of IPLookup certainly has flaws, but the data collected thus far has been enough to serve as a starting point. IPLookup has always been seen as a secondary method of data collection, while the client-side proxy provides us with the most valuable and realistic data.

7.2. Server-side Name-Based Collection

Early on there were efforts to discover domain/IP pairs by creating randomly generated domain names and searching for a corresponding IP address. This method, while initially quite promising, turned out to be of limited use in discovering pairs where the domain name was greater than five characters long. It is quite easy to understand why, as the number of possible strings of length five with characters a-z, 0-9 and "–" is approximately 1.3742x10^11. In actual runs, as little as one pair would be found for each 10,000 attempts.

It is quite possible that a less random approach could have worked fairly well. For instance, looking up real English words or combinations thereof would be a less daunting task, but given that this is such a small component of the overall system, this method was eventually abandoned in favour of the less random IPLookup and proxy programs.

7.3. Client-side Data Collection: MuffinCS

Two different attempts at a client-side proxy have been investigated for use with the Cyberscape architecture. The first attempt was to modify a SOCKS 4/5 proxy called Windsock. While this initial attempt looked promising, the underlying SOCKS protocol operates at a lower level than was required, and provided no more data than the IPLookup program discussed previously. More specifically, the SOCKS protocol handles only the actual TCP/IP requests, while the required data comes from the higher-level HTTP1.1 protocol requests.

Once the limitations of this implementation were discovered, it was decided that an actual HTTP proxy was required to collect the correct data. This lead to the modified "Muffin" proxy discussed below.

MuffinCS is a modified version of the "Muffin" HTTP proxy. Written in Java, MuffinCS runs in Windows, Linux, MacOS and any other system with at least a JDK1.1-compliant virtual machine. Current modifications of the Muffin proxy simply allow it to collect domain names requested by the client browser. With the domain names collected, the proxy then does a lookup to retrieve the domain's corresponding IP address. This information is then transmitted to the Cyberscape server, which stores this retrieved information in the database.

At this point, the Muffin HTTP proxy collects almost the same information as IPLookup, but from opposite ends. IPLookup collects domain info from the IP address while MuffinCS collects IP info from the domain name. This data is far more useful, however, as IPLookup does not tell at this point if the domain name/IP pair actually represents a valid website.

Also, any IP address may have more than one domain, IPLookup has no way of querying all domains that a given IP address is responsible for. Finally, since many IP addresses exist that do not serve web pages, IPLookup collects a lot of empty data. Since MuffinCS collects real web usage, the data it collects is never empty.

Furthermore, because the HTTP proxy resides on the client side, it has already been enhanced to provide more depth to the Cyberscape world. The first such data that has been collected is domain hits. This information directly impacts the way a building looks in the VRML world created by the Cyberscape client. Having a client-side program doing the data collection allows us to capture very rich data in the future. This data could have many uses other than providing data for the Cyberscape world, and is one of the more commercially viable aspects of Cyberscape.

Presently, the collected data is stored in a text file that is successfully transmitted to the server machine using the TFTP protocol. The TFTP program in use on both the client and server sides had been written previously for another course by Jason St. Cyr and is discussed in his report. The TFTP client and server are both written in Java, and provide a fairly light and platform-neutral way to transfer data between the client and server.

7.4. Data Collection Within the Cyberscape Environment

In addition to collecting client browser traffic, the Cyberscape world is itself full of data to collect. In order to show the paths mentioned previously, the generated VRML file contains touch and proximity sensors throughout the Cyberscape environment. These sensors are triggered automatically as a user walks the world, or when they click on a building. The triggers call a simple Perl script residing on the Cyberscape server. The data transmitted includes what street segment or intersection the user has walked over. The data is then loaded into the SQL database in the same manor as the proxy data files. This is discussed in more detail in section 7.7.

7.5. Cyberscape Database Information

For the project, PostgreSQL 7.2 has been chosen as the RDBMS to hold the collected data. PostgreSQL has been chosen for many reasons over other RDBMSs available. First, Oracle is very large, and seemed overkill for the project. MySQL was also investigated, but fails to scale past a few clients writing concurrently. PostgreSQL, on the other hand, is small, reasonably fast, free, portable, and has many features, such as foreign keys, lacking in other free relational database management systems like MySQL.

While PostgreSQL has managed fairly well thus far, certain queries such as SELECT COUNT(*) FROM IP_MAP, or queries containing sub-selects take a very long time when the table contains several million of rows. These issues are discussed at length in section 7.9, but is should be noted that if further tuning and workarounds prove insufficient, as the size of the data grows, moving to a larger RDBMS may be needed.

7.5.1. Entity Relationship Diagram for Initial Database Schema

The following ER diagram describes the database, with the corresponding tables appearing in section 7.5.2. The database schema has been broken up into the entities which comprise the core types of stored data objects, followed by the entities used to build relationships between these entities.

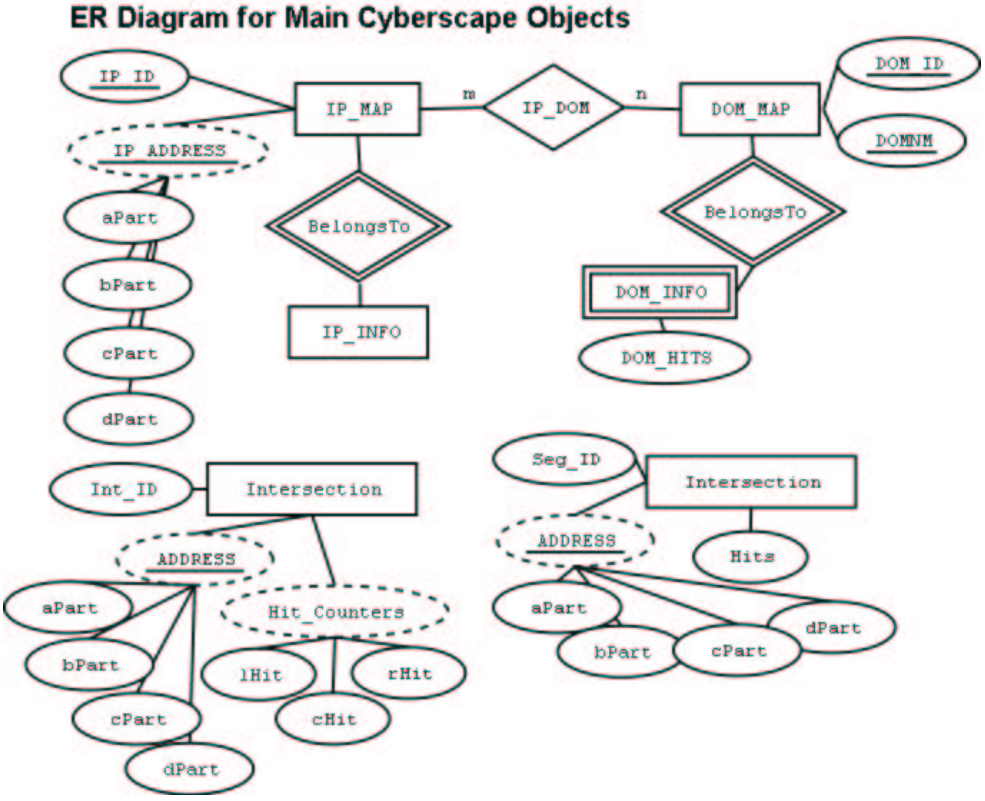

FIGURE 3: Entity-Relationship Diagram of Main Objects

As shown, the core tables store data on IP addresses, domains, street segments and intersections. The address of each street segment corresponds to the position that it belongs to in the VRML world, and is relative to the IP addresses. Intersections are a special type of street segment in that each one consists of three parts, a left part, center and right. Each of these parts must hold distinct hit data, as an intersection links multiple street segments and separating the hits out allows for a more accurate tracking of popular paths.

Next we look at activity tables. Each object type has a corresponding activity table that links some characteristic of the object to a specified activity class. The activity class is then used to retrieve a unique VRML colour string from the VRML_Color_Guide table.

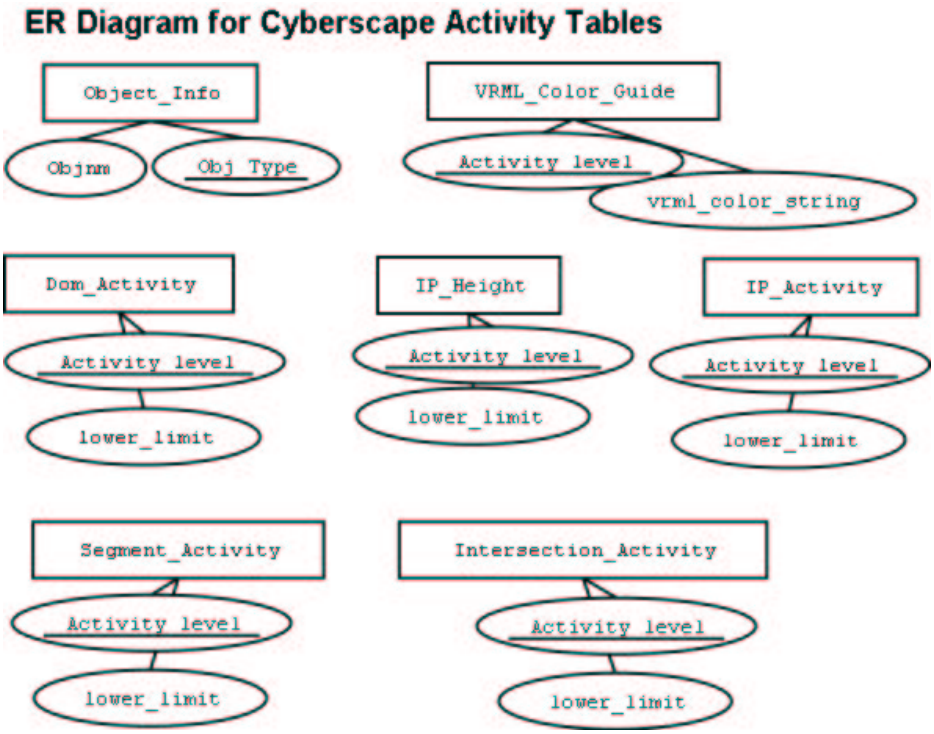

FIGURE 4: Entity-Relationship Diagram of Activity Levels

The only exception to this is IP_Height. In addition to a colour, IP addresses also have heights. For now, the number of hits for a domain (which is calculated as the sum of all domain hits) indicates the height of the building, while the number of domains that a website hosts represents the colour. Given that almost all IP addresses host at most a single domain, this decision may be changed at some time in the future.

Finally, it should be noted that changing the VRML world to use bitmap images instead of simple colours would only require changing entries in the VRML_Color_Guide table, or changing which table is called by a single Java method. This means that the entire look of the Cyberscape world can be changed very easily by only making minor modifications in one of two places.

As the client develops, this could allow us to offer low or high bandwidth versions of Cyberscape that would offer the same content but would be presented differently. Of course, the very fact that the data stored has so little VRML or Cyberscape-specific data means that as the usage and quality of Cyberscape grows, the stored data could take on other unrelated uses, such as understanding the relative popularity of sites, and monitoring trends in usage over time in different physical areas of the Internet.

7.5.2. Database Tables

The following tables correspond to the ER diagram above. Note that the primary key for IP_MAP is currently the complete IP address. While the IP_ID would make a more logical key, making the IP address a key is the only way to ensure uniqueness of the address in PostgreSQL.

IP_MAP: Maps IPID to an IP address (which means storing integer not a varchar)

Ipid	Apart	Bpart	cPart	dPart

DOM_MAP: Maps an integer to a domain name (stores and queries on an int)

Domid	Domnm			

DOM_INFO: Stores information for domains (Hit stats only for the time being.)

Domid	Domhits	Links	(to be added)	(to be added)

Intersection: The main Intersection table

Intersection_id	aPart	BPart	cPart	dPart	lHit	cHit	rHit

Segment: The main segment table

segid	aPart	BPart	cPart	dPart	Hits		

OBJECT_INFO: Stores the valid types of objects

OBJ_Type	Objnm			

DOM_Activity: Stores activity levels for domains

Activity_level	lower_limit			

IP_Activity: Stores activity levels for IP addresses

Activity_level	lower_limit			

IP_Height: Stores height classifications for IP addresses

Activity_level	lower_limit			

Segment_Activity: Stores activity levels for segments

Activity_level	lower_limit			

Intersection_Activity: Stores activity levels for intersections

Activity_level	lower_limit			

VRML_color_guide: Given a activity level, a VRML colour string is returned

Activity_level	Vrml_color_string			

7.6. Connection to the Database From Java

Communication between the various Java programs and the database is handled through several classes using existing PostgreSQL JDBC drivers to make the actual connection, and are part of the cyberscape.db package. These classes are used to connect to the database as well as to insert and retrieve all database information. As you will see in sections 7.8, they also allow the rest of the Cyberscape Java code to use the database without actually knowing anything about how the data is stored.

7.6.1. CyberscapeDatabase

CyberscapeDatabase is the base class that handles low-level calls, such as opening and closing connections to the database. It is the only class that is required to know any details about which JDBC driver we load. This means that as the program develops PostgreSQL can be switched out for other more robust RDBMSs with minimal impact to the Java code base. Because a JDBC interface must conform to certain standards, any JDBC compliant RDBMS could be used.

7.6.2. DBManipulation

This class makes use of the basic methods offered in CyberscapeDatabase and adds methods specific to the Cyberscape implementation. At this point DBManipulation is used by the IPLookup, DataFileParser and XML builder programs. All application-specific queries are handled through this class and it is ultimately responsible for building the Cyberscape world from the database.

7.7. Loading Data into the Cyberscape Database

As has been discussed, the data used to build the Cyberscape world comes from a variety of sources. Briefly, data is recorded from the MuffinCS client-side proxy, from within the Cyberscape environment using proximity sensors and a Perl script, and from the IPLookup program. Except for the IPLookup program which directly inputs the data into the database, these data collection methods need to know nothing about the database. Instead, they are dumped to simple text files which are then processed after the fact by the developed DataFileParser program.

DataFileParser is another Java program that takes in command line arguments for the type of data to be loaded and the directory path. It then attempts to load every file in that directory. Once the files are loaded, they are either moved to an archival area or are destroyed. The program then exits.

This program has been written with the intention of being called at regular intervals from cron job which will ensure that the database is updated frequently. If the Cyberscape world ever becomes overly popular, this also means that data entry can be deferred to non-peak periods.

7.8. Data Abstraction When Building the XML File

The process of building the XML data file involves repeatedly querying the database to gain the needed information. These queries retrieve hit data, segment and intersection information, and determine which domains belong to a given IP address and vice-versa. With the data retrieved, the XML file is then actually built.

The interesting part of this is that the database queries are completely hidden from all of the various XML builder classes. All builder classes are derived from the VRMLObject class. This class is the only one that knows anything about the existence of the Cyberscape database. Even then, it consists of abstract methods that are then implemented in each of the builder sub-classes.

The only methods that are called from VRMLObject are methods which also pass in the object type and ID. The object type is one of Domain, IP, Segment or Intersection and the ID is some unique way of identifying this object. For most objects this is a string that represents an IP address or is of that general form. The object type and ID are passed into the DBManipulation methods which then determine what type of builder class is requesting data, makes the appropriate queries and returns the appropriate results.

It should also be noted that in the event of a database failure, the methods that handle requests from the XML builder classes exit in such a way that enough data is passed back to create an XML file that is essentially empty, but enough to build basic a VRML environment.

7.9. Database Performance Issues

One interesting experience during the development of Cyberscape was to see how real world performance can change the way databases are designed and queried.

The most obvious example of this came in the DBManipulation method "getDomainNamesFromIP(int anID)" which ran a query with a subselect. The query in question was "select domid from dom_map where domid in (select domid from ip_dom where ipid = anID)." This is not an overly complex query, yet with approximately 4 million rows in each of these tables, the method, which is called 50 times for each map, was forcing the map building process to take as much as 15 to 20 minutes with a relatively full map section.

Initially, indices were added to various tables in an attempt to speed up the query. While this helped a little, it still was taking more than 10 minutes on average. However, it was discovered that breaking the select statement up into two separate queries actually improved the performance of the method considerably. By first querying for all domain id's, and then looping through an array and retrieving the appropriate domain name, map building performance has improved dramatically, with the entire process taking approximately 15 seconds. This simple example clearly illustrates the shortcomings of various RDBMSs. While this is most likely a PostgreSQL limitation, there is little doubt that each RDBMS has its own idiosyncrasies.

7.10. The Start of a Cyberscape Community: IRC Server and Client

While the building of a Cyberscape community is one of the least mature aspects of this project, an Internet Relay Chat (IRC) server has been setup and tested on the Cyberscape server, and a web-based IRC client has been integrated into the main Cyberscape interface.

The IRC server being used is dancer-ircd which considered the leader in free IRC servers available. The client, CGI:IRC, is a light and easy to use client written in Perl by David Leadbeater. This is not an overly robust client, but it does allow Cyberscape users to interact in a fairly anonymous fashion. It is our hope that future versions of Cyberscape will expand upon the community aspect of Cyberscape.

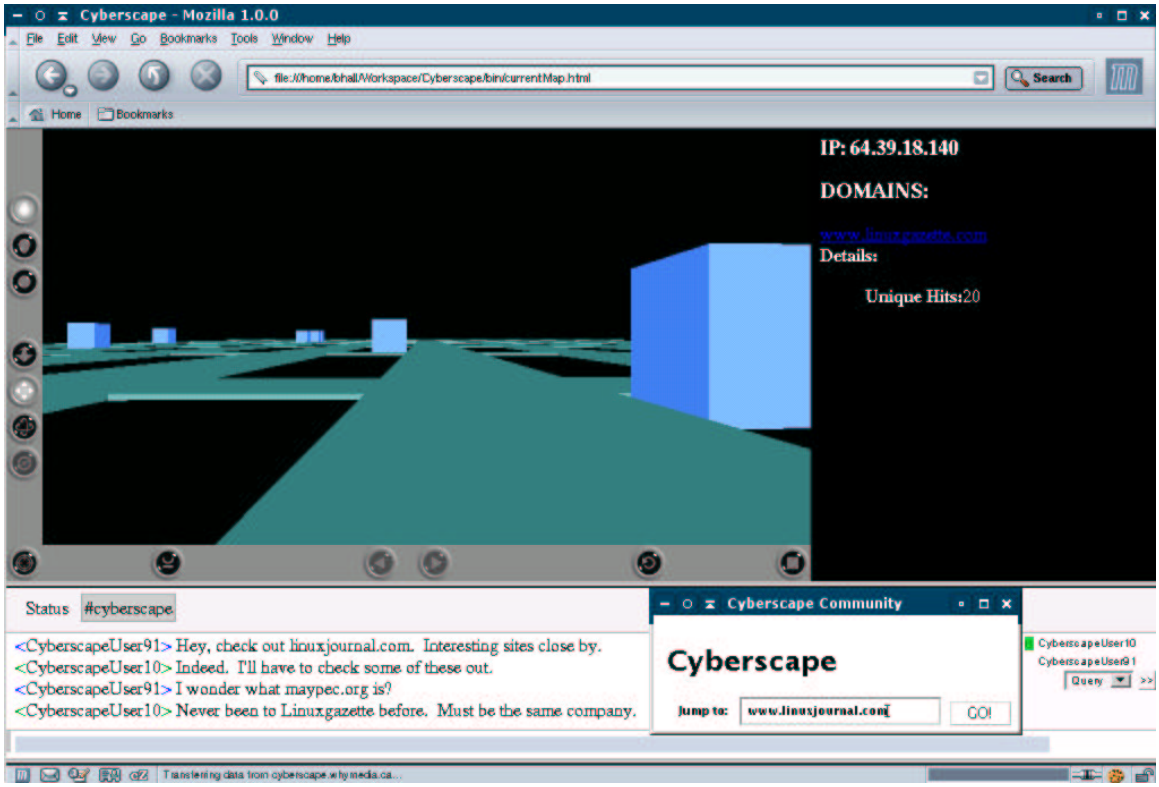

FIGURE 5: The Cyberscape Client illustrating the Integrated IRC Program

At present, Cyberscape does not store any data that could identify specific clients. While this anonymous usage was simpler to develop and offers a certain degree of peace of mind from prospective users, the client could be easily modified in the future to actually store client data within the Cyberscape server database.

If this were to happen, the potential uses of the Cyberscape data would broaden further still, as the browsing habits of specific users could be tracked within the environment. This would open up a plethora of new options from both a marketing and usability standpoint. Through looking at other user's browsing patterns, recommendations could be made on prospective destinations, very targeted advertisements could be provided and more. Because data would be stored server-side, users could create and share path information with each other within the Cyberscape environment.

If this change does occur, security and privacy will become issues, as the potential for information misuse is also quite large. A possible compromise that could offer a balance between functionality and security could be to simply store user data on the client machines. This too has good and bad points, and may dilute the community potential of Cyberscape. As the community aspects of Cyberscape develop, these and other issues will need to be investigated at length.

8) Key Features of the Cyberscape World

Now that all main components of the Cyberscape architecture have been investigated in some detail, the main benefits and design decisions will be discussed.

First off, while the application, database and web servers currently reside on the same machine, this does not have to be the case. Breaking the tasks across several machines was one of the design goals of the system. This coupled with the fact that most of the complex calculations are done client-side, means that the system should scale well to many simultaneous users, making the current design suitable for the foreseeable future.

Also, the collected data which is currently used to display the 3D VRML world could have many other commercial and non-commercial uses. The data is generic enough that it could immediately be used by ISPs to get an overview of what's going on on their networks. A quick look at the Cyberscape data would easily show who is running what servers where, and roughly how much usage the servers experience. With the proxy running on the client-side, the data being collected could be easily modified to provide a myriad of useful information. This data could be used to further enhance the Cyberscape world, or could be used for many other purposes. Cyberscape could become a vehicle for collecting completely unrelated client-side information.

Cyberscape currently makes efficient use of client-server communication. The server can handle a pool of requests. Cyberscape could be easily modified to only generate XML maps when they don't currently exist. This would mean that more popular locations would actually be easier and faster to load as the world is more completely mapped out. Over time, Cyberscape would only need to query the database if significant changes had occurred, making the efficiency of the system actually improve with use.

Because the files actually generated are in XML format, they can easily be transformed into other formats. At the moment the XML file simply serves as an intermediate format for the eventual VRML display, however the data could just as easily be transformed into a spreadsheet, or any other format.

With this system being as modular as it is, areas can be upgraded and improved independent of one another. This should help with system maintenance, and allows for development in stages.

Finally, Cyberscape has been designed to be as abstract as possible. By using Java, XML and SQL as the base, both the client and server halves can be used on a variety of operating systems. The database can be changed from PostgreSQL to any other RDBMS at any point, and by using a different XSLT stylesheet the client display could be easily changed from VRML to something more modern, such as an actual game engine.

9) Limitations Of The Cyberscape World

While there are many aspects of Cyberscape that are quite desirable, the system isn't perfect by any means. While none of these issues prevent Cyberscape from being useful, and all are a result of time constraints, there is certainly no shortage of room for improvement.

The system and software requirements are fairly steep. The client and server must use JDK 1.4. As has been mentioned, we have chosen PostgreSQL 7.2, currently available on Unix, Win32 and MacOS X, as the SQL server. To date, the server has only been tested under Debian Linux, and would likely need some extra time to port to other systems. The client needs to have a browser with a working VRML plug-in, and needs to have enough computing power to quickly translate the generated XML files into the eventual HTML/VRML format. While it should run on any system with JDK1.4 and a VRML plug-in, client testing has been limited to Windows and Linux, so some effort would likely be required to run on other platforms.

More importantly, Cyberscape requires a fair amount of bandwidth. First, there is the constant transfer of data from the VRML environment to the server, as well as the collected proxy data. However, the largest bandwidth consumption comes when the XML map file is generated. The resulting file is somewhere between 400k to 1MB depending on the density of the information, and this is using a basic environment. Because these files are so large, it is recommended that Cyberscape not be used over a dial-up connection.

Cyberscape has also not been optimized in almost any way, and many loose ends have not been tied. XML files are immediately overwritten at each new request, none of the files are compressed and the code likely doesn't handle exceptions as well as it should.

While the program responsible for loading the data into the database is complete, this process is still not as automated as it should be. There is also currently no way to load multiple maps. When a map is requested and generated, that's all. If the user walks to the end of the environment, they simply see empty space. New maps must be loaded manually by the user.

The database is likely another area that will have to be investigated at some point. While PostgreSQL has been a capable RDBMS, as the system grows past several million sites, it may not be fast or robust enough for Cyberscape's needs.

Keeping the collected data current is another concern that has not been adequately researched at this point. IP addresses gain new domain names frequently, and web sites change IP addresses regularly as well. At present, our system will offer a moving snapshot of the Internet, and may not always reflect the realities of the actual network topology. This issue is still under investigation, and will likely require extra programs to monitor and verify that the data in the database still reflects the realities of the Internet as it changes over time.

While progress on the project has been quite encouraging, in retrospect it seems that the original project goals may have been overly ambitious. This has meant that we were forced to exclude several features described in the original proposal. It was initially hoped that time would permit the inclusion of an integrated search and preference facility. Again, while this wouldn't be difficult to add, it is time consuming. This, in addition to the inclusion of data on a domain's hyperlinks, are the only features originally discussed in the proposal that have not been included in the final iteration of the project. In addition to time constraints, hyperlinks were dropped because of the difficulty in both collecting and then displaying the links between sites.

10) Challenges

The first major stumbling block faced was the very limited information that can be gathered through DNS requests. It was assumed that the data collection on domain name/IP pairs could be easily retrieved from simply requesting information from various DNS servers. However, it turns out that this is not the case, and is actually part of the design. While the scalability of the DNS concept speaks for itself, at the very least it was assumed that one would be able to query a DNS server to retrieve information on all domains that it is responsible for. Working around the limitations of DNS required significant effort.

Other challenges included gaining an adequate understanding of the SOCKS protocol and how proxies work in order to complete a working client-side proxy. Once this was established, decisions of security and usability come into play.

Also, there are many exceptions that must be taken into account when designing the system. For instance, a single domain name can belong to multiple IP addresses, and a single IP address can have multiple domains. Deciding how to handle this in both the database and the VRML world were unexpected challenges.

By far, the greatest challenge for both of us was that all of the work required was so new to us. At the beginning of this project, I had done little Java programming outside of a second-year data structures course. My database experience had been limited to a project where I could consult with a very experienced DBA. Jason had absolutely no VRML experience, and neither of us had much experience trying to build the type of system that we had hoped to create. While this is the whole point of an honours project, it was a constant issue that we had to deal with.

11) Conclusion

With the foundation laid, I believe that the Cyberscape project can be seen as a success. If nothing else, I have learned a great deal about how to store massive amounts of data, how to write good wrappers that shield the rest of a system from the database back-end and how to manage the realities of the performance impact of the certain SQL queries.

It is the hope of both and Jason and I that Cyberscape will continue to be developed after the conclusion of our Honours Project. The goals of the complete Cyberscape world could not be adequately met in only four months. It is our hope that, with further development, Cyberscape will continue to grow and evolve past it's current infancy.

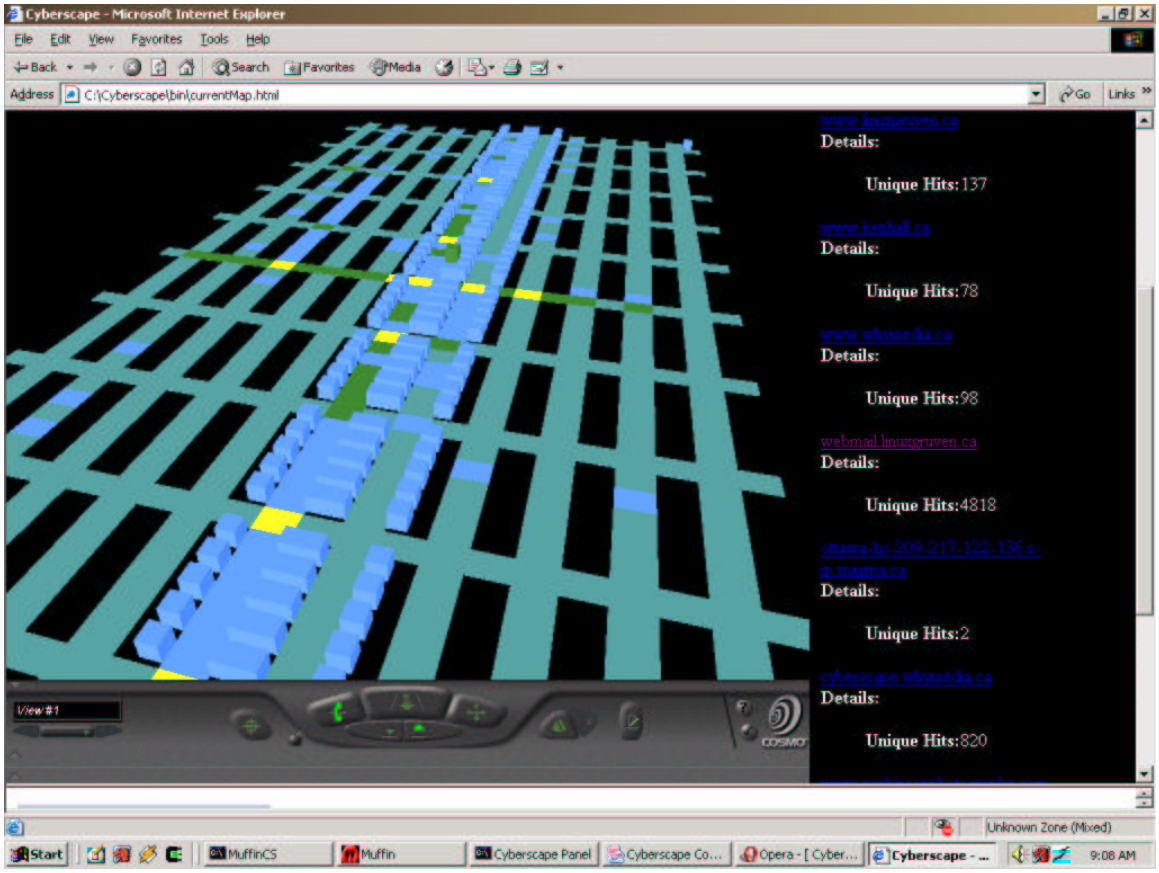

FIGURE 6: Final Cyberscape Client With Overhead View of a Map

12) Resources

On-Line Software Resources:

- CGI:IRC Homepage http://www.cgiirc.visionhost.net/
- Eclipse Project Homepage and Documentation http://www.eclipse.org/eclipse/index.html
- Muffin Homepage http://muffin.doit.org/
- PostgreSQL Website http://www.ca.postgresql.org/
- PostgreSQL JDBC Driver and Documentation http://jdbc.postgresql.org/
- Sun Java JDK 1.4 http://java.sun.com/j2se/1.4/index.html
- VRML http://www.vrml.org/
- Windsock SOCKS4/5 Proxy http://www.wiw.org/~drz/software.html

Database Resources:

- Momjian, Bruce. PostgreSQL: Introduction and Concepts, Addison-Wesley., 2000 http://www.ca.postgresql.org/docs/awbook.html
- Nel, Louis. Introduction to Database Systems: 95.305 Course Notes, Carleton University, Fall 2000
- Worsley, John and Drake, Joshua. Practical PostgreSQL, O'Reilly & Associates Inc., Sebastopol, Califiornia, 2002 http://www.commandprompt.com/ppbook/book1.htm

Java and Other Programming Resources:

- Albitz, Paul and Liu, Cricket. DNS and BIND, O'Reilly & Associates Inc., Sebastopol, California, 2001
- Lanthier, Mark. Introduction to Programming: 95.105 95.106 Course Notes, Carleton University, 1998
- Morgan, Michael. Java 2 for Professional Developers, Sams Publishing, Indianapolis, Illinois, 1999
- Schwartz, Randal and Christiansen, Tom. Learning Perl, O'Reilly & Associates Inc., Sebastopol, California, 1997