

FINAL REPORT

End-User Client for Creating Indexes for an Internet Search Engine

Industrial Sponsor:
Industrial Advisor:
Faculty Advisor:
Project Engineer:
Date:

Internet Applications Laboratory
Dr. Anthony Beavers
Mr. James Reising
Mr. Jeffrey Carlyle
April 24, 2001

Table of Contents

Table of Contents.....	1
List of Figures.....	2
Acknowledgements.....	3
Acknowledgements.....	3
Introduction.....	4
Statement of Problem.....	4
Design Approach.....	5
Results.....	7
Conclusion.....	7

List of Figures

Figure 1 – Noesis Client operating on sample tree.....	6
Figure 2 – The File menu.	6

Acknowledgements

I would like to acknowledge the work Dr. Anthony Beavers, director of the IALab. This work was also aided by the IALab staff which includes: Sid Naidu, the Noesis search engine software; Josh Burger, initial work on the Perl module known as "Collections;" Kyle Michel, the web interface; Antonio Tourino, maintenance of server software and work on distributed servers. IALab interns Arun Chacko, Sada Garba, and Michelle Morse thoroughly tested the software, and their help is much appreciated. Arun Chacko has also created a user manual for the Noesis client.

Introduction

The Internet Applications Laboratory at the University of Evansville, under the direction of Dr. Anthony Beavers, has created a number of limited area search for the Internet. A limited area search engine differs from general Internet search engine like Google, AltaVista, or HotBot in that it is intended to focus on documents related to a particular subject instead of the entire Internet. By focusing on a particular subject area, a limited area search engine eliminates a number of unwanted references.

Noesis 2.0 is a limited area search engine with a focus on all of philosophy. The URL for Noesis 2.0 is <<http://noesis.evansville.edu/>>. One of the features of Noesis 2.0 is that certain registered users are able to create topic trees that relate to topics in philosophy. These trees are created using a series of complex web-based forms. This process, known as link gathering, is not particularly user friendly and it places a heavy workload on the server due to the number of complex web pages that must be generated.

Dr. Beavers proposed a Windows-based client be created in order to ease the process of creating trees. The client would allow users to create collections of documents in a tree structure. Users would be able to drag and drop new items into the client and make modifications to existing data. The client program would communicate the tree data with the Noesis server programs.

The creation of the Windows-based Noesis client is an implementation of the client-server relationship. In the client-server relationship, a client serves as an entry-point or access point to a service. The server provides the service and responds to the requests of the clients. Servers generally communicate with multiple clients. Server computers are generally of higher quality and are more expensive than client (also known as workstation) computers. Servers also run specialized software and hardware for dealing with the requests of clients.

Statement of Problem

My goal in this project was to create a client-server package with a Windows-based client program and a Linux-based server program. The client program would allow users to interactively create and modify Noesis trees. The server program would add the user created trees into the Noesis server-side databases.

The creation of the Noesis client program was an extension to the existing Noesis project, and as such, there was already a lot of pre-existing code. The pre-existing code for modification of trees was based on a web interface and was optimized for make only one change to the tree at a time. Much of this code would need to be rewritten because multiple changes to the tree could be made on the client side and arrive at the server at the same time. This meant that the code would need to be "re-optimized" for making multiple changes to the database at the same time. So much work needed to be done to the back end systems of the Noesis project.

The project also required that I learn network programming on both Unix and Windows platforms. Another consideration was the protocol to be used for communications between the client and the server. This protocol, which is in effect a language used to communicate between the client and server, would need to be optimized to include as little data as possible because network bandwidth is a

limiting factor. The more data that needed to be sent, the slower client-server communications would be.

Another problem that needed to be addressed was the question of speed. Since users do not generally want to wait to view their data or make small changes, the client and server programs would need to be very responsive to user requests. This meant that the data structures and algorithms used by the client and server would need to be optimized for speed.

Design Approach

The first step I took in creating the Noesis client was to examine the pre-existing Noesis software to see what need to be changed or improved. After our experiences with Noesis 2.0, the IALab staff decided that the backend software that makes direct changes to the database needed to be separated from the user interface code. Since Noesis was written in Perl, we created Perl modules that were responsible for updating particular parts of the database. For instance, among many other modules, we have a Users module and a URLs module. Since the Noesis code was separated into clearly defined modules, I was quickly able to identify and examine code that might need to be changed. Work on this part of the project began in May of 2000.

The next step I took was to learn some of the basics of network programming. I did this by creating a Unix based "echo" client and server. The client program simply accepted keyboard input and sent it to the server. The server would then echo its input back to the client.

The echo program accessed network services using TCP/IP. TCP/IP is the network service protocol that is used by Internet software. Also the program used BSD sockets for network programming. TCP/IP and BSD sockets provide an API for using network services that is fairly common across platforms. The "echo" program was written using GNU g++ on the Linux platform.

My next step was to create the Noesis server program. Unlike the "echo" server, I wrote the Noesis server program in Perl. By writing the server in Perl, I was able to access the same Perl modules used by the Noesis web interface. Additionally, Perl's access to the TCP/IP and sockets API is very similar to the C++ API so there was not a lot of new code to learn. It was during the creation of the server program that I developed the client-server communications protocol. I tested the initial version of the server by using the Unix telnet program to connect to the server. I then entered data into the server by hand.

After thoroughly testing the server, I began work on what was to be the most complex part of the project: the creation of the client. I wrote the client using Microsoft Visual C++ and the Active Template Library. I used the Active Template Library because it provided easy access to Component Object Model (COM) library. COM is method of interprocess communication in Windows. I used COM to allow users to drag URLs and other data from Internet Explorer and Netscape Navigator and place it in the client. The ATL also provides a number of predefined classes for maintaining user interface objects. I had created a number of Windows programs and used the ATL previous to starting this project so I was easily able to create the basic user interface and COM interactions of the project.

After implementing the basic user interface of the project, I began on the network communications portion of the client. Fortunately, the Windows API for

network communications is based on and very similar to the BSD sockets API I used on the Linux platform.

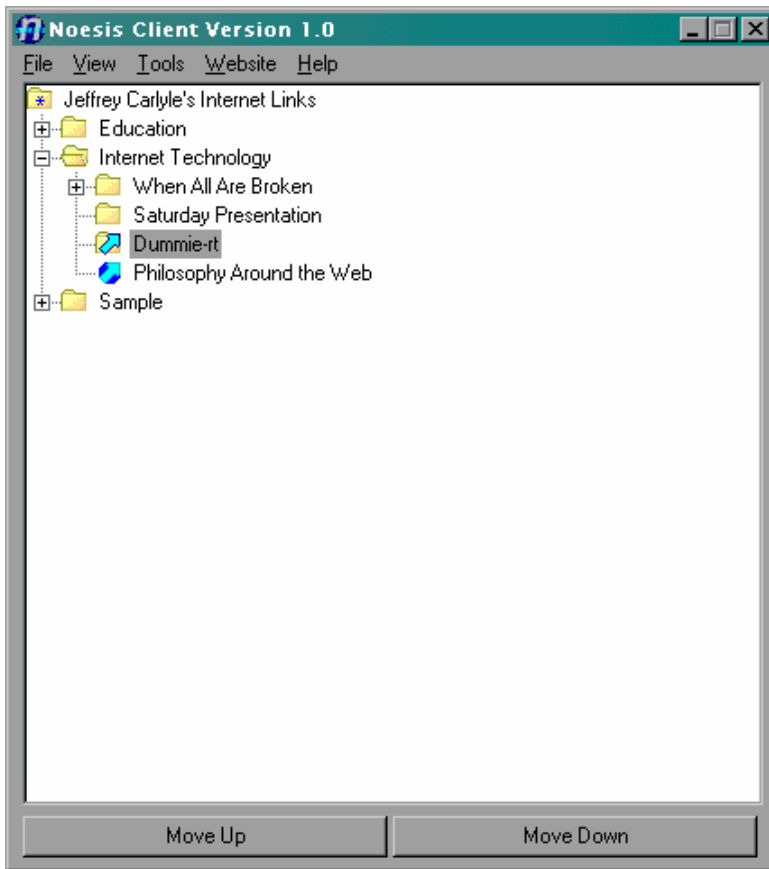


Figure 1 – Noesis Client operating on sample tree.

By the end of August 2000, I had created the first generation of the client and server. The client was a fully featured Windows program. Not only could users create and modify trees, but the client also allowed users to register for Noesis and provided online help. Instead of using the traditional Windows help, the Noesis client accessed help documents stored on the Internet.

The process by which changes made to the user's tree on the client were transferred to the server became known as synchronization. In the first generation of the client, the client would send the entire tree back to the server. The server would then examine the data for changes, make the changes using the Noesis Perl modules, and the

entire tree would be sent back to the client. While this meant that both the client and server would always have the most recent and error checked versions of the tree, it was also very slow. The speed of synchronization needed to be improved, so I decided to create a second generation of the client.

For the second generation of the client, I redesigned the network protocol used by the client and server programs and then rewrote the server to use the new protocol. After testing the server using telnet, I rewrote part of the client. In the second generation of the client, the client scans the tree for modifications and then sends only the modifications to the server. After every fifth synchronization, the client requests the entire tree from the server. The tree is requested to ensure that any server side changes (such as broken links) are transferred to the client. This greatly reduced synchronization time.

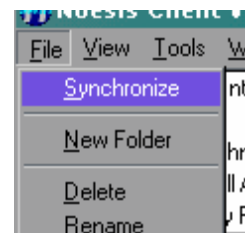


Figure 2 – The File menu.

Results

The Noesis client and server are operational; however, thorough testing has uncovered a few bugs that still need to be fixed. Data is sometimes lost during synchronization. This is apparently because some unknown sequence of events results in data being marked as unchanged when it has actually been changed. Also the client and server hold the entire tree in memory during when they are active. This means that data can quickly be accessed and changed. This works well with small trees on the order of 1000 items; however, IALab employee Michelle Morse has created a tree with over 10000 items. Loading the entire tree into memory is very lengthy process. These problems need to be addressed in future versions of the client.

Conclusion

This project has demonstrated that such a program is possible and has addressed and overcome many problems associated with such a design. The Noesis client program has addressed many of the problems that will be faced by a proposed commercial version of the Noesis client.