**Socket Communication using Java**
**Modernizing the Dark Ages Talker**
**CS690 Computer Science Project**
**Jeffrey K. Brown**

## I. Introduction

As a Computer Science freshman, I had my first experience with socket communication. Using the computer systems of the University of New Haven, I discovered a popular, text-based, online computer game called a MUD, or Multiuser Dungeon. Along with the MUD, there was another type of multiuser online system called a Talker. Rather than fighting dragons and dungeon creatures, the talker served as a type of chat room. I found myself frequenting Talkers more and more until I obtained my own Talker source code and began customizing it to my own liking. The Talker world has introduced me to many different people around the world. To this day, I continue to own and operate my own Talker, Dark Ages.

While I, along with about 50 regular users, continue to enjoy the Dark Ages Talker, we face some real problems if we want to continue operating in the future. I think this project may be an excellent opportunity to solve some of these problems. Some of the problems that I would like to address are the user interface, development accessibility, and documentation. Another concern that I have been thinking about is security. Through this project, I may even have an opportunity to address some of those issues as well. Let me take a closer look at some of these problems.

The first problem is the user interface. The talker uses the same commands as other talkers, so people familiar with them have no difficulty adjusting. The general population, however, is not quite as skilled at picking up the talker commands. As a result, we have a user base consisting mostly of experienced talker users. The variety of users from all around the world, which created such a draw toward talkers in the past, has stagnated. New users looking for conversational variety discover the similarity of users and lose interest. At one time, to try and make the system more accessible, I implemented a Java Telnet Applet that provided a means of accessing via a Java Applet. The Applet's interface was just as unintuitive as the normal means of connecting, so it did not work. A better web-based interface may help us overcome this hurdle by reopening our talker to the less computer-experienced users who offer more variety of conversation, since they may be more experienced at other things.

Next on my list is the development accessibility. While I was a Computer Science sophomore, I found that experimentation with talker source code was an excellent place to apply some of my newly-learned Computer Science training, especially since I was already interested in the system from a user point of view. Even today, new Computer Science students sign on and talk about programming their own talkers. Two hurdles exist before one can program for and run a talker. First, talkers use the Unix socket APIs with C. Historically, to run a talker, you must have access to a Unix or Linux system. Second, the C socket API is

often confusing, especially to new Computer Science students who are interested in programming one. These two challenges may be overcome by implementing a talker using Java. Java boasts cross-platform compatibility, which may ultimately allow one to run on a Windows system. Additionally, Java has the capability of dealing with sockets at a much higher level than the Unix C Sockets, making it less confusing and more understandable.

Documentation about talkers has always been scarce. Reengineering a talker from the ground up will provide an opportunity to decide how things should be put together. The documentation for the talker project can be created as the talker is built, stating reasons why things are done the way they are. Moving back on the list of challenges, to the developer accessibility, I made the mention of developing the project in Java. Java offers an interesting documentation tool called Javadoc. I think the documentation in a web-based form like Javadoc will be readable and understandable to anyone wishing to do additional development on the system.

Finally, I have some concerns about security. Talkers generally work by connecting directly to the server's socket and talking directly to the system. The system replies back to the user's socket connection. Using a packet sniffer of any sort will reveal both sides of a conversation as it passes the one point on the network. Java offers the use of encryption. This project may be a good way to figure out how to work encryption into the talker system. Additionally, a web-based system could possibly use Secure Socket Layer modules to provide encryption for that interface as well.

As I have discussed, there are many ways in which the Dark Ages talker can be improved. Implementing some of these ideas will take some time and effort. I think this would be a great opportunity to apply some of the knowledge I have gained as a Computer Science student along with researching and implementing knowledge gained from other sources. The end result will be an improved system for existing users and a new and user-friendly experience for new ones. With these broad ideas, I suggest this proposal for a University of New Haven, Graduate Project.

## II. Project Statement of Objectives

Modernize the Dark Ages Talker by providing a web-based means of access, improved documentation, improved cross platform compatibility, and an effort at improving security. Implementing these changes will result in a better, more understandable talker that others can use as a learning tool and a recreational system.

## III. Project Description

First, I will need to model the data that will be in the system. This data will consist of user information, system information, communications, and atmosphere information. The user information will include characteristics like settings, user names, and passwords. The system information will consist of ports, configuration information, and server properties. Communications consist of a sender

and a set of recipients. Atmosphere information are the attributes of the actual chat rooms. I will design a database to replace the file-based means of storing this data.

Next, I will interview different users on the system, along with different users on other systems to find out what types of features are the most important parts of the user experience. Additionally, I will perform tests on the present Dark Ages system to get figures on which commands are used the most often. Based on this information, along with commands that are important to the management of the talker, I will design these features in an appropriate manner for each user interface.

Once I have determined how the system will be designed, I will implement the data portion of the system into a database. I want to make the system as database independent as possible, and leave it up to the one who implements the system to decide. During development, I will most likely use Postgres or MySQL.

The Dark Ages Talker is presently implemented in C using the Unix Socket API, so I will need to recreate the socket server in Java. I will need a server piece that will accept connections, and a client that can make connections. Even though most users have their own favorite talker clients, I will want to create one for demonstration purposes. The individual client connections will need to be able to communicate back and forth with each other.

Once I have the socket system successfully handling connections, I will link the socket system to the database. The database will store the originator of the message along with special sending attributes. The talker will use this information to determine whether the message was an online mail message, a private message or a broadcast message to the other users on the system.

When the system is successfully using the database to handle the connection information, I will attempt to implement a web-based client. Rather than relying on talker commands, this user interface will look more like popular chat-room website software. Communication from one type of interface to the other should not cause any difficulty. I will probably use Java for this portion, but I may also attempt to use the PHP, Hypertext Preprocessor, programming language to implement this section.

In order to keep from writing the same code twice, once for the socket interface, once for the web interface, I will probably need to create another program that runs in the background that will parse the user input and create the correct output. I will need to investigate how exactly I would like to do this.

I will use Javadoc to create some documentation about the classes used to create the Java portions of the system. If I decide to use PHP for it, I may use the PHPdoc software, too. While this section is an ongoing one as the system gets developed, when I finish the implementation of it, I will go back and refine my comments and produce some actual documentation for the system and then some instructions on how the users will utilize the system.

Finally, if there is time, I would like to investigate the possibility of adding encryption to the system or providing a Microsoft .NET implementation of a client. These two items are not critical to the project, but they may be interesting to attempt to implement.

## IV. Resources and References

At present, I have the following reference materials that I will be using. I can only imagine that the list will grow larger and longer as I research different aspects of the project.


1. Walton, Sean. Linux Socket Programming. Indianapolis: Sams Publishing Company, 2001.
2. Gay, Warren W. Linux Socket Programming By Example. Indianapolis: Que Publishing Company, 2000.
3. Silberschatz, Abraham and Galvin, Peter B. Operating System Concepts, Fourth Ed. New York: Addison-Wesley Publishing Company, 1994.
4. Sommerville, Ian. Software Engineering. New York: Addison-Wesley Publishing Company, 1996.
5. Peterson, Larry and Davie, Bruce. Computer Networks: A Systems Approach. New York: Morgan Kaufmann Publishers, Inc., 2000.
6. Sobell, Mark G. A Practical Guide to the Unix System, Third Ed. Reading: Addison-Wesley Publishing Company, 1995.
7. Java 1.4 API
8. PHP API
9. Manpages and Software Documentation


## V. Hardware and Software Requirements

The software requirements are a list of the software that I will be using to produce the final project. The hardware requirements are my development systems. I will produce documentation of the hardware and software requirements of the client and server software when it becomes known.

**Software Requirements:**
Java Development Kit 1.4, including Javadoc – for Development and Documentation
Java Runtime Environment – Needed to run the Server software
Apache Web Server or IIS Web Server - for implementation of the web interface
Database – Probably MySQL or Postgres, but I would like to be database independent.
Microsoft Visual Studio – Possibly to create a C# .NET client
mod_ssl – Secure Socket Layer module for Apache Web Server
mod_php – PHP module for Apache Web Server

**Hardware Requirements:**
Chimaera – Development System running SuSE Linux
Viper - Second Development System running SuSE Linux

Pegasus – Windows 2000 Development System
Darkfantastic – Production Slackware Linux System.

## VI. Schedule

This is the chronological order of the tasks needed to implement the talker system. I would like to have the project complete by September 1st, 2003.

1. Model the data in the system
2. Using the data model, design a database that would contain this data.
3. Interview users and perform analysis on the existing features to find out user requirements.
4. Design user requirements and features to work with both main interfaces.
5. Implement a database according to database design above.
6. Implement multiuser socket server system.
7. Create a basic socket system client.
8. Connect the socket server to the database and store connection specific data in tables.
9. Use the database to control message attributes.
10. Create "handling" system to process input and determine whether it is a command or message and produce the correct response for the system.
11. Implement a web-based client in Java or PHP that uses the database.
12. Refine each interface and implement additional user features.
13. Create web-based documentation with Javadoc.
14. Create more formal documentation and directions.
15. If time, see if we can implement some means of encrypting communications or implementing a client using .NET.

## VII. Final Product and Deliverables

At the end of this project, I would like to have a workable talker server. The server will certainly run on Linux and Windows 2000. The talker server will communicate with a database. I have yet to discover whether or not the system will communicate with a database running on Windows 2000. I would like to have produced in Java or PHP a more user friendly web interface that will allow users inexperienced with talkers to participate. I would like to create a talker client in Java for users that would like to run one. I would also like to take a look at creating a talker client using C# .NET. Finally, I would like to evaluate how I could integrate security measures, like encryption to the system and if time, implement them.

Along with the system, I would like to produce documentation of the source code itself, along with user and administration directions on the applications developed in the project along with the use of the talker itself. In addition to the documentation and actual program, I will submit a report that evaluates the design choices that I faced and ultimately decided upon. The report will also discuss some of the knowledge I have gained through the project and future recommendations if ever had to implement this project again. Finally, I will attempt to model the system using UML or another formal description language.