414Mud

SID GANDHI, McGill University
NEIL EDELMAN, McGill University

We did a mud as a proof-of-concept for the latest features of the Java programming language with respect to networking.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Design

General Terms: Mud, Network, Java

Additional Key Words and Phrases: TCP Network, Telnet, Client-Server

ACM Reference Format:
Sid Gandhi, Neil Edelman, 2014. 414Mud. *McGill* 1, 1, Article 414 (December 2014), 7 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

Contents

1 Introduction and background **2**

2 Methodology **2**
2.1 Overview 2
2.2 Design Goals and Constraints 2
2.3 Programming Languages and Networking Protocols 3
2.4 Architecture 3
2.5 Summary of Components 3
2.5.1 Package 'gameentities' 3
2.5.2 Room 4
2.5.3 Package main 4
2.5.4 Orcish 4
2.5.5 Commandset 4
2.5.6 Connection 4
2.5.7 FourOneFourMud 5

3 Results **5**
3.1 Features 5

4 Discussion **5**
4.1 Challenges Encountered 5
4.2 Thinking Ahead 6

Fall 2014 – 304-414-001 – Introduction to Telecom Networks.
Sid Gandhi – 260415580, Neil Edelman – 110121860.

DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION AND BACKGROUND

By definition, a MUD or a Multi-User Domain is usually a networked, text-based game, which could range from a simple chat oriented application with elements of virtual reality to an extensive role-playing fantasy game populated by fictional characters, non-player characters, and many other game world elements.[1]

This concept was borrowed and modified slightly for 414MUD, which was intended to be a proof of concept for department or university-wide proprietary chat application with added game elements for user entertainment.

The purpose of this report is to inform the reader of the design process, goals, architecture, tools, and challenges which were encountered in the process of building the networked application. It is organized by sections which cover all the major aspects of the effort with the intent to walk the reader along the path from this projects conception to its realization.

2. METHODOLOGY

2.1. Overview

414MUD is a multi-tier application which includes a chat server, a game controller, and a client-tier using Telnet. The chat server is responsible solely for handling connections – opening, closing and managing clients connected to it using Java Sockets and transferring bytes of data between clients and the game controller. The controller interfaces with the chat server and provides the game related services such as creating players, accounts, and other game logic.

The following section will attempt to illustrate and explain the inner workings of the application, as well as the design process which was followed and the goals and constraints which shaped it.

2.2. Design Goals and Constraints

The goals for this application were mainly to function as a proof of concept for a department or even university-wide chat server with fantasy elements of varying complexity. In order to fulfill this role, the following capabilities were necessary:

- The ability to accept many simultaneous connections
- Concurrent processing to enable communication between the chat server and the game layer
- A protocol to facilitate the sending and receiving of messages at each client connected to the game
- A layered architecture to enable rapid expansion and maintainability

Our ability to implement these features was subject to the following constraints:

- Time (perhaps the biggest factor, this particular constraint had an impact on all others)
- Pre-implemented networking features in the programming language chosen to develop this application in. In addition, the language had to be user-friendly and include a ¡good¿ debugger; it also had to be one the developers already had a high level of familiarity with
- The protocol chosen for communication had to be easy to use and preferably included as a default package with most operating systems (as time and distribution concerns constrained the ability to develop a specialized client front-end application)

An iterative process was needed in order to fine-tune the system as time went on. The idea behind the application started as only a general direction; specific game elements and the details of the implementation were not decided initially. Additionally, an iterative design process and a layered application allows for features to be added down the line as the prototype matures into a real-time MUD functioning on the McGill domain.

2.3. Programming Languages and Networking Protocols

In order to satisfy the above design constraints, the Java programming language was chosen. Java sockets are used in conjunction with the networking protocol. Additionally, ANSI, VT-220 and other terminals are transparently implemented in Java.[2; 3] Even backspace worked without any modification by Java magic. For the networking protocol, Telnet worked for us as it is ubiquitous, installed by default on most machines, and relatively easy to use.[4]

2.4. Architecture

As mentioned above, a layered architecture separating the game layer with the chat and networked layer was critical to this application because the goal was to add many more game elements down the line while leaving the networked core untouched.

However, minimal planning was done at the onset, and application evolved as more and more features were added. The initial design document, Figure 1, consisted only of the following rough sketch of what the game logic would look like.

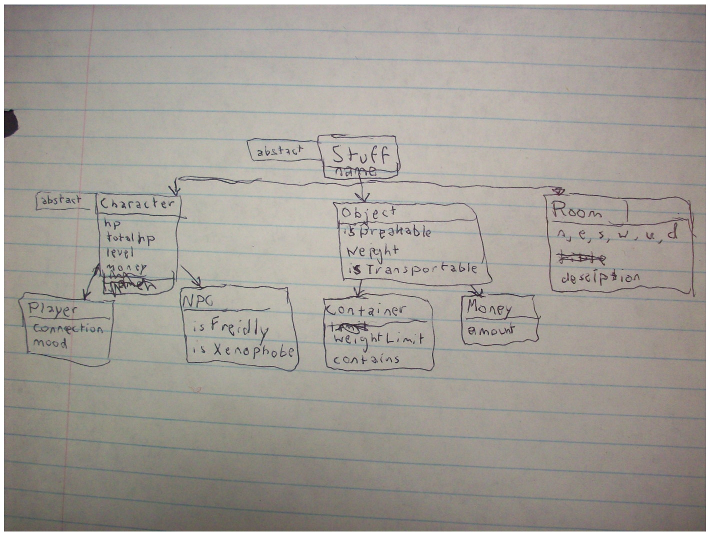


Fig. 1: The draft entities map of the game.

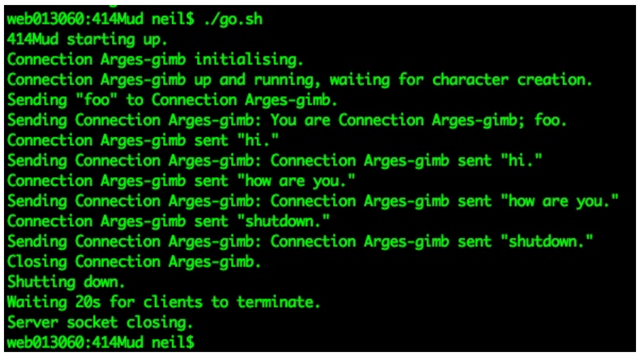
The following diagram, however is the final class architecture for the game system. As is clear, our design ideas were informal from the offset, but it was a suitable process as there were only two developers and the system was relatively simple.

The sections below summarize the main functions of the packages and some technical nuances which the developers felt were both challenging and interesting implementations to handle some expected and unexpected issues alike.

The Java SDK was invaluable in implementing these.[5] Figure 2 shows an in-development screenshot of the application.

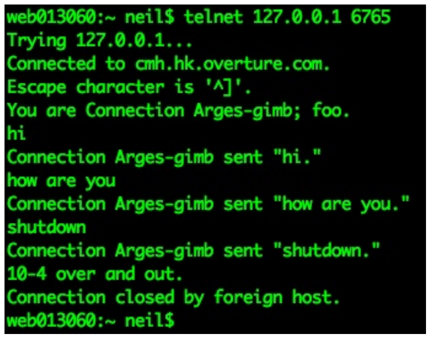
2.5. Summary of Components

2.5.1. Package 'gameentities'. A package containing the different types of game elements, such as `Players`, `Rooms`, etc, this was organized as a hierarchical structure similar to our initial plan. However, instead of having each object implement contents separately, the `in` and `contents` were placed in `Stuff`, forming a completely recursive structure and the contents were Iterable.



```
web013060:414Mud neil$ ./go.sh
414Mud starting up.
Connection Arges-gimb initialising.
Connection Arges-gimb up and running, waiting for character creation.
Sending "foo" to Connection Arges-gimb.
Sending Connection Arges-gimb: You are Connection Arges-gimb; foo.
Connection Arges-gimb sent "hi."
Sending Connection Arges-gimb: Connection Arges-gimb sent "hi."
Connection Arges-gimb sent "how are you."
Sending Connection Arges-gimb: Connection Arges-gimb sent "how are you."
Connection Arges-gimb sent "shutdown."
Sending Connection Arges-gimb: Connection Arges-gimb sent "shutdown."
Closing Connection Arges-gimb.
Shutting down.
Waiting 20s for clients to terminate.
Server socket closing.
web013060:414Mud neil$
```

(a) Server



```
web013060:~ neil$ telnet 127.0.0.1 6765
Trying 127.0.0.1...
Connected to cmh.hk.overture.com.
Escape character is '^]'.
You are Connection Arges-gimb; foo.
hi
Connection Arges-gimb sent "hi."
how are you
Connection Arges-gimb sent "how are you."
shutdown
Connection Arges-gimb sent "shutdown."
10-4 over and out.
Connection closed by foreign host.
web013060:~ neil$
```

(b) Client

Fig. 2: An early client-server showing remote shutdown.

2.5.2. Room. Each `Room` entity contains a `Direction` enum. In addition to providing the set { `N, E, S, W, U, D` }, it provides additional functionality, for example
`W.getBack() = E;`
You could name it in a String and it is automatically converted into its long name (used in game.) Additionally, rooms implements reflection, so you can do this `getRoom(Direction dir)` { `return dir.getRoom(this);` }; thus double connecting a room is accomplished in two lines,
`this.setDirection(dir, target); target.setDirection(dir.getBack(), this);`

2.5.3. Package main. The networking `heart` of the application, this package contains all the elements vital to the proper functioning of the game, including the `FourOneFourMud` class which implements the console server application, the `Connection` class which handles all the individual connections and has sole access to all their data, etc. For example, when `Player` objects need to send messages to each other and communicate, they do so through the `Connection` object associated with that `Player` object. The 414Mud class maintains a list of all active Connections, each of which is identified by its `Name` generated using the `Orcish` class. Interesting technical nuances of some classes are specified below:

2.5.4. Orcish. Random name generator, as mentioned above. The reason to use Names to identify connections instead of just a number was to make for easier identification during debugging and gameplay.

2.5.5. Commandset. This class contains a Commandset.Level enumeration with three possible values depending on difficulty. One of these is specified in its constructor and produces the appropriate set of available commands for that difficulty. The method public void interpret(final Connection c, final String command)breaks a user input string into the command and the arguments and eventually passes the interpretation to the game engine, which then performs the related tasks.

2.5.6. Connection. Connection implements Runnable to allow our thread pool to execute it. The constructor stores stores the value of the parent FourOneFourMud for accessing the map and all the public methods. Connections `run()` method implemented a very simple inner loop: it simply reads the connection line-by-line and feeds it into `Commands.interpret(this, input)`. This reduces overhead as there's no polling involved; it creates another potential issue: connection termination is blocked. As a workaround, the connection is closed by a separate thread. The `getFrom()` method implements readLine().

2.5.7. FourOneFourMud. As mentioned above, the `FourOneFourMud` class implements the main server application. It maintains a list of children Connections and also contains a handle to the starting `Room`. The run() method waits for a connection and uses a fixed thread pool to improve performance. A load() private method builds up a rudimentary map (currently limited but works adequately) Lastly, shutting down the server is achieved through the shutdown() method and requires a command set associated with an `Immortal` level. This method closes the server and Connection sockets.

3. RESULTS

The overall result was a successful prototype of our MUD. We did not implement as many game elements as we had initially planned, mainly due to time limits additional game elements were given a lower priority as the ability to create a MUD network using Telnet was the main goal with this prototype.

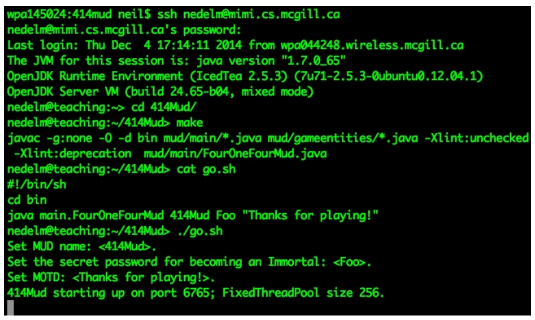
3.1. Features

The game successfully implements chatting in a virtual text-based fantasy universe. We accomplished the following initial goals:

— Handle simultaneous multiple connections
— Concurrency using Threads, as described in the Architecture section
— Layered architecture to enable rapid expansion and development

The application is quick, works smoothly and the server is debugger friendly. It lays a good groundwork to build on for a CS or Electrical Engineering (or both) department-wide MUD.

Compiling the Mud is shown in Figure 3; server-side 414Mud starting up is shown in Figure 4 (as the reader can see, the thread pool size is fixed to 256, which may cause some issues as more and more demands are placed on the application.)



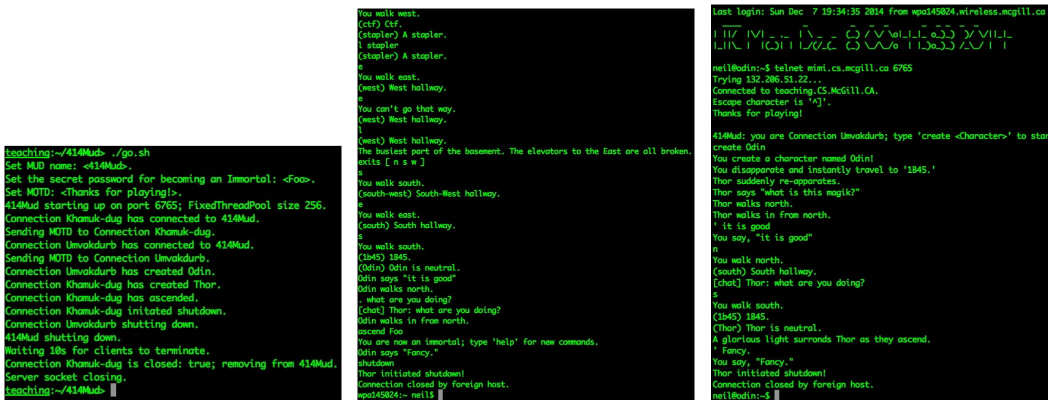
```
wpa145024:414mud neil$ ssh nedelm@mimi.cs.mcgill.ca
nedelm@mimi.cs.mcgill.ca's password:
Last login: Thu Dec  4 17:14:11 2014 from wpa044248.wireless.mcgill.ca
The JVM for this session is: java version "1.7.0_65"
OpenJDK Runtime Environment (IcedTea 2.5.3) (7u71-2.5.3-0ubuntu0.12.04.1)
OpenJDK Server VM (build 24.65-b04, mixed mode)
nedelm@teaching:~> cd 414Mud/
nedelm@teaching:~/414Mud> make
javac -g:none -O -d bin mud/main/*.java mud/gameentities/*.java -Xlint:unchecked
 -Xlint:deprecation  mud/main/FourOneFourMud.java
nedelm@teaching:~/414Mud> cat go.sh
#!/bin/sh
cd bin
java main.FourOneFourMud 414Mud Foo "Thanks for playing!"
nedelm@teaching:~/414Mud> ./go.sh
Set MUD name: <414Mud>.
Set the secret password for becoming an Immortal: <Foo>.
Set MOTD: <Thanks for playing!>.
414Mud starting up on port 6765; FixedThreadPool size 256.
```

Fig. 3: Mimi compiling the programme.

4. DISCUSSION

4.1. Challenges Encountered

Some decisions related to interfacing between the game and networking layers did not work as well as planned and backfired. The nature of object oriented programming made it difficult to choose, among all the possible choices, the most optimal and efficient way to link different objects together within the application. For example, should sending messages be handled by the `Player` object or by the `Connection` object? Should we embed a `Connection` inside a `Player` or a `Player` inside a `Connection`? These decisions were taken lightly at the



```
teaching:~/414Mud> ./go.sh
Set MUD name: <414Mud>.
Set the secret password for becoming an Immortal: <Foo>.
Set MOTD: <Thanks for playing!>.
414Mud starting up on port 6765; FixedThreadPool size 256.
Connection Khamuk-dug has connected to 414Mud.
Sending MOTD to Connection Khamuk-dug.
Connection Umvakdurb has connected to 414Mud.
Sending MOTD to Connection Umvakdurb.
Connection Umvakdurb has created Odin.
Connection Khamuk-dug has created Thor.
Connection Khamuk-dug has ascended.
Connection Khamuk-dug initated shutdown.
Connection Umvakdurb shutting down.
414Mud shutting down.
Waiting 10s for clients to terminate.
Connection Khamuk-dug is closed: true; removing from 414Mud.
Server socket closing.
teaching:~/414Mud>
```

(a) Mimi

```
You walk west.
(ctf) Ctf.
(stapler) A stapler.
l stapler
(stapler) A stapler.
e
You walk east.
(west) West hallway.
e
You can't go that way.
(west) West hallway.
l
(west) West hallway.
The busiest part of the basement. The elevators to the East are all broken.
exits [ n s w ]
s
You walk south.
(south-west) South-West hallway.
e
You walk east.
(south) South hallway.
s
You walk south.
(1b45) 1B45.
(Odin) Odin is neutral.
Odin says "it is good"
Odin walks north.
. what are you doing?
[chat] Thor: what are you doing?
Odin walks in from north.
ascend Foo
You are now an immortal; type 'help' for new commands.
Odin says "Fancy."
shutdown
Thor initiated shutdown!
Connection closed by foreign host.
wpa145024:~ neil$
```

(b) Thor

```
Last login: Sun Dec  7 19:34:35 2014 from wpa145024.wireless.mcgill.ca
  ____          _          _   _  _          _  _ _  _  _
| ||/  |\/| _ ._  | \ _  _  (_) / \/ \o|_|_|_ o_)_)  )/ \/||_|_
|_||\_ |  |(_)| | |_/(/_(_  (_) \_/\_/o  | |_)o_)_) /_\_/ |  |

neil@odin:~$ telnet mimi.cs.mcgill.ca 6765
Trying 132.206.51.22...
Connected to teaching.CS.McGill.CA.
Escape character is '^]'.
Thanks for playing!

414Mud: you are Connection Umvakdurb; type 'create <Character>' to star
create Odin
You create a character named Odin!
You disapparate and instantly travel to '1B45.'
Thor suddenly re-apparates.
Thor says "what is this magik?"
Thor walks north.
Thor walks in from north.
' it is good
You say, "it is good"
n
You walk north.
(south) South hallway.
[chat] Thor: what are you doing?
s
You walk south.
(1b45) 1B45.
(Thor) Thor is neutral.
A glorious light surronds Thor as they ascend.
' Fancy.
You say, "Fancy."
Thor initiated shutdown!
Connection closed by foreign host.
neil@odin:~$
```

(c) Odin

Fig. 4: The server, mimi, and the clients, Odin (Odin on Physics,) and Thor (laptop in Trottier.) Thor is a player with the `Immortal` command set, and hence has the ability to shut down the MUD.

beginning but the performance differences noticed were not insignificant and warranted a re-think of several aspects at least once.

Additionally, there was no way to determine reliability but to test it in as many different traffic situations as possible. Due to this, development was carried out with only a limited inkling of which way to implement a feature would work better than other ways, resulting in possible unexpected issues in circumstances with a high number of connections, for example.

These challenges, however, are a part of any product or application design process, and our iterative development process helped us overcome them for the purposes of making the prototype function efficiently.

4.2. Thinking Ahead

Steps we would like to take now with this application differ in the scope of what they try to achieve. Some involve implementing entirely new protocols to facilitate other tasks, like File Transfer (for example), while others involve modifying existing implementations within the game to make the application more interactive. For example, we wanted to implement a timer which calls the MUD at specific intervals (every second or so) with which the non-player characters could move around and interact with the world. This feature would also enable a host of modifications related to game enhancement a combat system and timed events, to name a few. The ability to create a map from a file was another such idea; however it creates some challenges related to object referencing, which have not yet been resolved.

On the implementing new components front, implementing an FTP layer would be a nice addition, even if it only let players send simple image or text files between each other. However, the existing architecture of the game would have to be modified a considerable amount to tie in both the Telnet and FTP on one server and preferably on a single connection. It would require the development of a customized client side application which would handle switching between the two protocols for each of the different tasks, it may also involve serializing and deserializing files in some manner.

On the whole, much more can be done with the MUD itself, but in our opinion, the most important focus would be deploying a MUD server and inviting people (including professors) to connect. The anonymity offered by offered by a random Orcish name might result in more interesting interactions between participants. However the application would need to be stress tested to deduce the maximum allowable connections under the current infrastructure and perhaps even to know whether or not any core components need to be changed.

Finally, if we were to really let our imagination loose, we don't see why it shouldn't become an open source application encouraging and letting many McGill students contribute to it over the course of a few years and letting the project evolve on its own. Perhaps, if we tried to re-invent the wheel, we might find that spherical works better than cylindrical!

REFERENCES

[1] R. Penton, *MUD game programming.* Premier Press, 2003.
[2] J. Postel and J. K. Reynolds, "Telnet protocol specification," *RFC 854*, 1983.
[3] J. VanBokkelen, "Telnet terminal-type option," *RFC 1091*, 1989.
[4] G. Von Laszewski, I. Foster, J. Gawor, and P. Lane, "A java commodity grid kit," *Concurrency and Computation: practice and experience*, vol. 13, no. 8-9, pp. 645–662, 2001.
[5] Oracle, "Java platform se 8," *http://docs.oracle.com/javase/8/docs/api/*, 1993, 2014. accessed Dec, 2014.

Received December 2014; revised December 2014; accepted December 2014

APPENDIX

The 414Mud is located at https://github.com/billybandawe/414mud.