A Distributed Social MUD to Enhance Reliability and Scalability

Ryan McDonald and Nick Montfort
{ryantm,nickm}@cis.upenn.edu

23 April 2003

Abstract

MUD (Multiple User Dungeon) systems are client-server. When the single server goes down, everyone is kicked off and no one can log on. Also, even if there are only a handful of people talking in a given room, a large overall number of users can slow the server and cause lag (higher latency) which makes fluent communication impossible. We introduce *Happy Fun MUD,* a distributed, peer-to-peer MUD that implements the essential communication commands of simulated online worlds. A *gateway* allows peers to load the map and to enter the first room. Peers can then move about; the first peer in a given room is designated its *innkeeper.* The innkeepers coordinate activity between rooms in the MUD. When an innkeeper leaves a room, the responsibility is assigned to someone else, if there is anyone left in that room. Peers send what they say directly to other local peers; they whisper private messages directly to the peer who is addressed in the communication. The system is more robust to the failure of peers or the gateway than the client-server system is to a server failure. The busiest node is also much less busy than is the server in the client-server system.

1 Why Create a Distributed Social MUD?

1.1 MUDs Are Social Worlds

A MUD is an all-text virtual environment. It is called a Multiple User "Dungeon" after a free Fortran version of the interactive fiction *Zork* that was released in 1979, called *Dungeon.* Although many people think of puzzle-solving and fantasy quests in relation to MUDs, it is actually the "multiple user" aspect that makes them appealing to many people. Many MUDs serve as social spaces where people have the same sorts of conversations they do "in real life," talking with friends on the phone or chatting with coworkers at the water cooler. The simulated world that the MUD provides allows for persistent objects that can be part of this communication process — the most important objects in MUDs are often things like guest books, which different users can sign, rather than things like magic swords. There are plenty of specific examples of MUDs and MOOs that are used for social and communication purposes. These include systems for education, learning, and academic events, including LinguaMOO, TecfaMOO, MOOSE Crossing, ExploreNet, and Penn's very own PennMOO. These educational MUDs and MOOs do not always host round-the-clock socializing, but many systems do. One prominent example is the famous and much-studied environment LambdaMOO, which has been around for more than a decade.

1.2 Bogged Down in the MUD

The online worlds currently made available to users in MUD and MOO (MUD Object Oriented) systems are implemented using a client-server architecture. This has proven effective enough in many cases, but with this architecture, there are problems with the reliability of such systems. When the single server goes down, everyone who is online is kicked off and no one can log on to use the system. There are also problems with scalability to large numbers of users. This difficulty is particularly seen during popular events (such events are hosted on many MUDs) and during peak times. Even if there are only a handful of people talking in a given room, the large number of people on the MUD or MOO, throughout the whole environment, can bog down the server, which is a bottleneck through which all communication must pass. As the load on the server grows, lag (higher latency) appears and can, at a certain level, make fluent communication impossible. This defeats the main purpose of social MUDs.

In dealing with the reliability and scalability of MUDs and MOOs, we focus on the communication aspects of simulated online worlds. Some MUDs do feature combat, magic, and quests. Many online simulated worlds exist mainly to facilitate communication among users, however. They provide the advantages of a simulated space rather than a simple "channel" for chat, so that many users prefer chatting in a MUD to using Instant Messenger or IRC. We focus on ways that distributed systems can improve reliability and scalability in these sorts of social environments, using a simple text-based MUD system tailored to communication among users. Rather than optimizing for behaviors like exploring the entire simulated world (which are actually uncommon on MUDs), we optimize for the various sorts of communication between users that make up most of the activity in social MUDs.

1.3 New Social Spaces

Improving the reliability and scalability is important since users are bothered by the failure of a server or high levels of lag. One motivation for a distributed implementation is simply to make existing communication-based MUDs more robust, so that they do not suffer as much from server failures and high loads.

However, a distributed implementation might also allow new sorts of online social worlds to arise. Many of the chat-based MUDs that are in use today host little more than one or two dozen users at a time, but they are tiny islands in a network of many such systems. Proprietary MMORPGs

(Massively Multiplayer Online Role-Playing Games) such as *EverQuest* and *Anarchy Online* already have tens of thousands of users online at once. We can't be certain of what social MUD users might want, but it's reasonable to think that people not as interested in quests might still enjoy having a large, distributed, chat-based MUD where different rooms and areas would host different groups and conversations that they could move between seamlessly. With the advent of wireless networks, people also might wish to join local chat-based MUDs at conferences, coffeehouses, or parks. This might provide another motivation for using a peer-to-peer architecture rather than a client-server architecture: no central server would have to handle the flow of communications between people in these more spontaneous virtual environments. While the system we present does require one or more known, fixed processes to act as gateways, we hope it is a step toward more fluid way of establishing virtual environments that might be appealing to users in these sorts of situations, and perhaps even in surprising new ways.

2 Introducing Happy Fun MUD

We have designed and implemented Happy Fun MUD, a very simplified MUD system in Java. The implementation includes the most basic, essential features of more complex systems such as LambdaMoo (C), PerlMUD 2.1 (Perl), and WolfMUD (Java). We chose not to modify the open-source Java system WolfMUD because of its strong role-playing game orientation. Some essential features of chat-based MUDs that are important to our project (such as the ability to whisper and communicate something privately to another user) are not implemented in that system.

To make the project tractable, we implemented only a very small subset of MUD features. For instance, the ability to create new characters, change passwords, and manage characters is not implemented; we simply allow anyone to enter a name and log on as that character. (At present, there is not even a way of requiring users to use unique names.) Similarly, the ability to add new rooms and objects is not implemented; there is a hard-coded map instead. Because we are focusing on communication issues, matters of mutual exclusion for access to objects was not a priority, and although we describe one way this could be accomplished, we have not implemented such exclusive access to objects. Instead, we have focused on implementing the ability for characters to move between rooms, to notice who else is in the room, to speak to one another publicly within the room, to speak privately to one another by whispering, and to say something to everyone in the entire MUD by hollering. We are also not interesting in scaling to larger and increasing more complex simulated worlds; Happy Fun MUD would not be very good at that since each peer keeps track of the entire state of the "physical" environment, remaining unaware only of what users are in remote areas and what they are saying. The sort of scalability we are interested in involves handling very large numbers of users in a reasonably-sized simulated world that can host many large, concurrent conversations.

Happy Fun MUD is implemented over TCP/IP. Although there are several cases in which multicast would be the most effective way of communicating, such as when a peer wishes to transmit a "say" message to everyone in the room, the practical difficulties with UDP multicast recommended against using it. First, many machines have multicast disabled, as we have found out in trying to complete another CIS 505 project. Also, when multicast is enabled, it is often only practical to use it within local area networks by setting the TTL (time to live) to a low number, to avoid accidentally overlapping with other multicast groups on the Internet. We wanted Happy Fun MUD to use only those technologies which are practical for and easily available to ordinary users. With firewalls in place today, it is difficult enough for many people to even use services on non-standard ports; we certainly did not want to implement a system intended for casual, widespread use that had even more esoteric networking requirements.

The system is composed of two runnable Java classes and two helper classes. It can run on any Internet-connected computer that has a Java VM.

2.1 MUDGateway

The first runnable class is MUDGateway. This is the server that allows entry into the MUD. To connect to the MUD, a peer logs on through the gateway and is able to establish communication with other peers. This is the only time when information is sent from the gateway to a peer.

As implemented, this gateway process is a single point of failure — of a certain sort. If it fails, no one can log onto the MUD until it is brought back up. However, people can stay on the MUD, move about, and talk to one another even if the gateway does go down, which is an improvement over the client-server system. Also, it would be fairly simple to replicate the gateway so that if one gateway process failed, peers could still connect to the MUD using another one.

2.2 MUDPeer

The MUD is peer-to-peer, with each user's computer having a complete map of the simulated world. All peers run by executing the same runnable class, MUDPeer. This peer replaces a MUD client such as Pueblo that is the current way MUD users connect to a server.

The MUD relies on designated peers called *innkeepers* to coordinate movement between rooms and to broadcast statements that are hollered. Every inhabited room has an innkeeper. The first character entering an empty room is designated the innkeeper of that room. This peer hands that responsibility to another peer when he or she leaves, unless the room is then empty. The identity of the innkeeper is unimportant to the user and is not even made visible. Public speech is sent locally by peers to the peers of other characters in the room, while private speech is sent directly to the peer of the addressee. When a person chooses to holler (that is, to say something that the whole MUD will hear, as with the `@wall` command in PerlMUD) a message is relayed via the local innkeeper to other innkeepers, and from there to other players.

Movement between rooms is managed through the innkeepers; if a user, exploring the MUD, moves out of a room, that user's peer gets the innkeeper list before departing. On entering a new room, if the room is inhabited, that user's peer notifies the new room's innkeeper and is able to join the room. If, on the other hand, the user finds that there are no others in the room, the user's peer will declare itself an innkeeper. The identities of the innkeepers are known to everyone so that, for instance, an innkeeper in a busy room can be replaced if that peer drops its connection. However, each peer only needs to keep track of the innkeepers and the others in the same room, not everyone

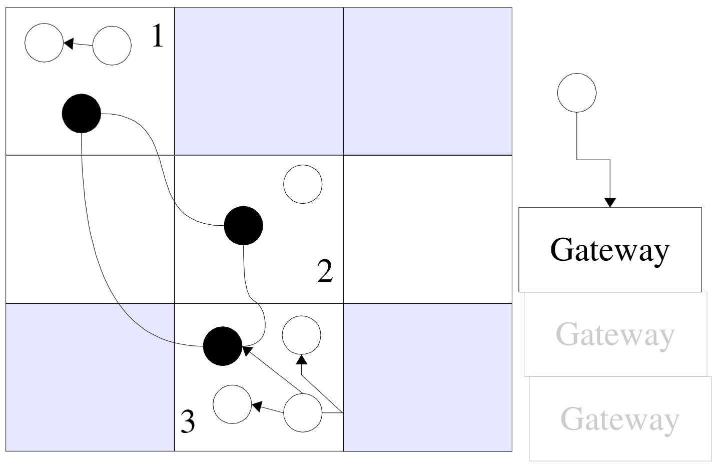

Figure 1: A snapshot of Happy Fun MUD. In general, rooms in a MUD do not need to be arranged in a grid; they are just shown this way so as to make it easy to illustrate different sorts of communication and the role of innkeepers. The peers that are innkeepers are shown in black.

who is online.

One advantage of this sort of distributed setup is that any peers who are not influencing the state of the world do not have to message others. Looking at the surroundings, reading books, and undertaking other sorts of action that are not destructive or reconfiguring does not have to involve any sort of network traffic at all.

Figure 1 shows one possible MUD configuration. One peer is attempting to enter the MUD by sending a request for the current state of the simulated world and the innkeeper list to the gateway server (some machine running MUDGateway). Nine peers are seen who are currently in the MUD, distributed throughout three different rooms. In room one, a private communication (whisper) is occurring between two peers. In room three, a local communication is occurring where one peer is using "say" to speak to everyone else in the room. Both of these communications happen directly between peers. This should reinforce the idea that innkeepers do not behave as servers for a room; most messages do not travel through the innkeeper before reaching the desired peer or peers. Essentially, an innkeeper's main purpose is to always know exactly one occupant of every inhabited room, specifically, the innkeeper of all those rooms. This way peers may move to different rooms and have the ability to communicate with at least one peer in that room in order to learn the rest of the room's occupants. To maintain an up-to-date innkeeper list, all innkeepers are sent a message each time a rooms innkeeper changes.

The circumstances under which messages are sent in Happy Fun MUD are described in Appendix A. The format of messages is described in Appendix B.

2.3 Failure Modes

2.3.1 Non-innkeeper Peer Failures

Ideally, the innkeeper would poll everyone in room at some regular interval, checking with them if they have been idle (as far as public behaviors go) during the previous interval of time. If there was no reply, the player list would be modified and a new player list to everyone in the room. This would prevent "zombies" — entities that seem like other players but are really just crashed or disconnected peers — from hanging around. However, the presence of such zombies is only an annoyance, and does not prevent other communication from taking place, nor does it make the MUD unstable.

In the current system, a peer that has crashed or disconnected without properly exiting is not detected until there is a change of innkeeper for the room. At this point, to avoid handing off the innkeeper responsibility to a crashed peer, the old innkeeper checks to see which of the peers it is possible to connect to. The updated player list is then sent to all the peers in the room, including the new innkeeper

2.3.2 Innkeeper Failures

In the current system, if an innkeeper crashes the corresponding room is essentially doomed. Peers trying to enter that room will also fail. However, other regions of the MUD will not suffer from this failure. We describe a scheme whereby the failure of an innkeeper could be tolerated. It involves using the gateway (or set of gateways) to enable mutual exclusion. As a practical matter, when a richer simulated world is desired in an actual MUD, it will be necessary to implement mutual exclusion for other reasons — to control the state and ensure that a unique object is not held by more than one individual peer, for instance. The following describes a way the gateway can be used to provide a room lock and allow a single peer to become innkeeper:

If a peer cannot reach the innkeeper of the room it is in,

PEER to GATEWAY: Request room lock
If room is not locked,
GATEWAY to PEER: Reply with room lock (which times out after a certain time t)
If room lock obtained, peer designates itself as innkeeper and
PEER to ALL INNKEEPERS & GATEWAY: Current innkeeper list

2.3.3 Gateway Failures

If the gateway fails, no one can join the MUD. However, the gateways can themselves be replicated, which will resolve this problem as long as one gateway remains up. Although the use of the gateway as a source of mutual exclusion and consistency control is easiest to imagine if there is a single gateway, a scheme for distributed control could be used in the case where there are many gateways. For instance, a single gateway could first obtain a room lock among all the gateways by use of a token ring, and that gateway could then reply by granting the lock while ensuring mutual exclusion. Essentially, this idea would push off the need for mutual exclusion to the more well-behaved, simpler, and almost certainly more stable gateways; a scheme that involved coordinating only among the peers would not be necessary.

3 Results

3.1 Reliability Testing

As described aove, the system as implemented was not intended to be robust to innkeeper failures. However, existing users should have been able to survive a gateway failure and continue interacting on the MUD. Also, the failure of a non-innkeeper peer should not have any affect on communications among other peers. We did tests to show that these sorts of failure were tolerated.

We first loaded the system with eight peers and took down peers one at a time, terminating each processes by interrupting it rather than by exiting properly, until only one was left. At each point we ensured that other peers could move around, enter and leave rooms with other people in them, and speak to others publicly and privately, as long as there was anyone else left. We did not terminate an innkeeper process during this test. A separate test showed that a failure of an innkeeper, while it would make communication impossible in the room that the innkeeper was responsible for, would not affect other peers that were elsewhere in the MUD. In a real implementation, the failure of an innkeeper would need to be tolerated to a greater extent.

Also, we ran the system with the usual single gateway and we logged several peers onto the MUD. We then interrupted the gateway process. No one was able to join to MUD after this was done, but the system continued to work for all peers who were already logged on. One improvement would involve making it possible for the gateway to be restarted without having all peers exit the MUD. Although this did not seem extremely difficult, we left this for future work, which could also involve replicating the gateway so that a single failure would not make the MUD inaccessible.

3.2 Comparison of Network Traffic

To determine how our system compared to the usual client-server sort of MUD, we tried a similar communication situation with three, six, and nine users on both a PerlMUD server running on an IRCS computer and our own system, Happy Fun MUD, running on the GE cluster. We had the users go to three different rooms (which hosted one, two, or three users at the different stages of the experiment) and had them undertake a mix of communications (whisper, say, and holler) that represented a realistic mix of social actions in a MUD. Users whispered most frequently, said things to the whole room half as often, and hollered things to the whole MUD only $\frac{1}{3}$ as often as they said things. If anything, our mix of communications overstates how often global communication is employed — many MUDs do not even allow ordinary users to speak to the entire MUD at once, and when they do, this type of speech is usually employed very infrequently, to call for help or to tell other users that the system is going down. However, although we had users move to different rooms initially, we did not cover the case of manipulating objects or otherwise changing the state of the simulated world. We monitored the number of packets handled by the PerlMUD server using IPTraf, a network analysis program for Linux that provided statistics on every packet handled through every socket. We wrote additional code to have our peers report every packet they sent or received, since IPTraf could only be run as root and therefore was not available to us for use on the GE machines. We only went up to a load of nine users because our peers had to each run run on a separate machine and only eleven of the GE machines had their hostnames correctly spelled in their config files and could be used with our MUDPeer code.

We had expected that with 9 users logged on in three different rooms, overall network traffic in our system (neglecting the initial load of data from the gateway) would be about the same as the server's traffic on a client-server MUD, but that the individual groups of peers will have three times less traffic. Specifically, we thought the nodes handling the most traffic (the innkeepers) should have significantly less traffic than the server does in a similar client-server system. We were pleasantly surprised. Not only was the maximum traffic per node significantly less for Happy Fun MUD than was the server traffic on PerlMUD; the rate of growth was also lower. PerlMUD's server traffic grew steeply, while the busiest peer in Happy Fun MUD experienced only a linear increase in traffic.

3.3 Optimizing for Communication

Our system performs extremely well when users are communicating with one another and undertaking the sorts of MUD or MOO behaviors that we have optimized for. We should emphasize, however, that the peer-to-peer system we have implemented, and the concept behind it, is not appropriate in all situations. If players constantly move around the MUD, staying in different rooms and manipulating the environment so as to alter it (for instance, by picking up and dropping items), each peer will constantly be sending updates to every other peer. There will be no advantage seen in this sort of situation in terms of lower traffic — although even here, the reliability benefit will be present.

However, we do not observe these sorts of behaviors in MUDs and MOOs such as LambdaMOO. Instead, communicating with others turns out to be the most important, frequent activity, and

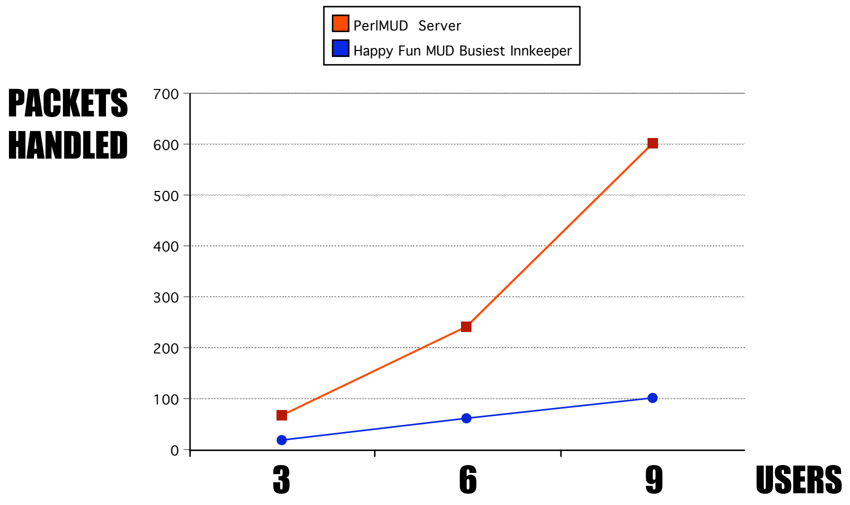

Figure 2: Results of an experiment with identical communicating users, comparing a full client-server MUD with the Happy Fun MUD system.

most users remain in a single room for long periods of time, speaking to others in that room. At the same time, the simulated world remains important setting for the communication; users would not want to give that up for a pure chat system. It is because we have noticed this frequent type of behavior — which does not involve constant modification of the state of the simulated world, only messaging other users with what is said — that we were able to optimize a MUD so effectively by developing this type of peer-to-peer system.

This system requires users to use peers with a larger "footprint" than the typical MUD client, but the memory and processor overhead of such a peer is not prohibitive given the normal computing resources that a MUD user has. More problematic is that the system relies on arbitrary users having adequate bandwidth to serve as innkeepers. If some users connect by dialup and have very little bandwidth available (as happens on MUDs and MOOs now) the innkeeper system as it is now implemented could experience local bottlenecks. This problem is a manifestation of what has been cited [1] as a fallacy of peer-to-peer systems: treating all peers as if they had identical capabilities and distributing service functionality among all peers when a subset is better able to take on certain tasks. To prevent such a situation, some mechanism might allow higher-bandwidth peers to take over as innkeeper, coordinating this through the gateway to make sure that only one such peer was taking over the responsibility at a given time.

4 Related Work

We found very few other projects that dealt with peer-to-peer MUD or MOO implementations. One that was typical was a student project proposed last semester in a class at the University of Illinois Urbana-Champaign [2]. This group employed a different scheme that involved distributing the model of the world among different users, as was suggested for cluster-based server projects in this class. They planned to use XML for the world representation and SOAP for communication. We know of no other attempts to distribute a MUD or MOO environment along the lines of Happy Fun MUD, with each peer keeping a complete map of the environment and sending communications directly to others.

The "gateway" system relates to techniques used in peer-to-peer file sharing to allow users to log on through a server, after which point they can access other peers directly. The "innkeeper" is in some ways similar to the "channel operator" of IRC (Internet Relay Chat). However, we designed the mechanism of innkeeper in such a way that it is not particularly desirable to become one. People battle for "channel ops" on IRC by writing bots that spoof other user's names and log them off the channel so they can gain control of it; we have tried, by design, to prevent the innkeeper from having any special powers that would cause people to struggle for that office. We sought to avoid this problem by not making the innkeeper a "power user" of any sort. It is possible that a full-scale implementation would require a more powerful innkeeper, so that this problem would have to be dealt with at a later stage of development.

Our approach is also related to the cluster-based server approach where each server manages a small portion of the game's players. In these methods, bandwidth is reduced by having each server only interact with the players in its cluster and having the servers interact amongst themselves. Similarly, our approach reduces bandwidth by having players only communicate with other players in their cluster/room. The innkeepers' peers behave in some ways like the servers, managing events between players in different rooms. However, they do not truly function as servers for rooms, because communications that take place in the rooms do not have to pass through them: All peers send public speech to everyone else in the room directly, and they send whispered, private speech directly to the intended recipient. A complete implementation of a peer-to-peer system also would not suffer the same sort of failure modes that clustered servers would. A server failure, even with clustered servers, would either kick off a group of users or eliminate a whole portion of the MUD and render it inaccessible. In the peer-to-peer system, no such problems would exist once the issue of innkeeper failure was handled.

5 Future Work

Converting Happy Fun MUD into a MUD system suitable for ordinary, daily use could be done in three phases.

5.1 Tolerating Peer Failures

The problem of innkeeper failure should be handled first, along with the problem of "zombie" users. Crashes of any sort of peer should be tolerated completely, with at most a delay as innkeeper responsibilities are taken up by a new peer. As soon as a peer discovers a failed innkeeper (which will happen whenever a peer tries to say something to the whole room) it should be able to acquire a room lock from the gateway and designate itself as the new innkeeper. Similarly, innkeepers should regularly poll peers in the room and update the room list when a peer cannot be reached. Of course, there may be situations in which an innkeeper cannot reach a peer but that peer can still reach others in the room, so the situation may be more complex than these suggested modifications assume. So, such modifications may uncover some new difficulties, but we do not anticipate that major new issues would come up in implementing these changes and dealing with these sorts of failures effectively.

5.2 Mutual Exclusion

Peers should be able to access the gateway to acquire other sorts of locks and to atomically manipulate the simulated world. This would allows objects can can be taken and dropped, a malleable environment that could be changed or augmented by users, and other sorts of interaction that are already seen on client-server MUDs. Making the gateway the lock server provides a fairly simple way achieve this exclusion, at the cost of making the gateway its own sort of bottleneck in the entire system. However, it would be doing much less than the server in a typical MUD or MOO even if it did provide locks, so it should be a less problematic bottleneck for activity.

5.3 Gateway Replication

After the above changes are made, the system will once again be fairly reliant on a single process, the gateway, and prone to failure if the gateway crashes or becomes unreachable. However, the gateway itself can be replicated without too much difficulty. That should be the last step in making Happy Fun MUD fully functional and as distributed a system as possible. With replicated gateways, any of the gateways could be used to connect to the MUD. After connecting, any gateway could be consulted for a lock. The gateways would have to be mutually aware and would have to handle the problem of mutual exclusion among peers as it is pushed onto them; any of the schemes for distributed mutual exclusion could be used to accomplish this. The gateways would have to collectively tolerate the failure of one or more gateway processes, also. Once this was accomplished with a better gateway implementation, the system would be robust to all but one gateway crashing and a given peer could endure any number of other peers failing. The complete richness of a MUD environment could be provided in this sort of system, which would easily scale to large, distributed groups of communicating users. We hope that new, compelling sorts of social environments could be enabled by such a distributed MUD.

6 References

[1] Vahdat, Amin, Jeffrey Chase, Rebecca Braynard, Dejan Kostic and Adolfo Rodriguez. “Self-Organizing Subsets: From Each According to His Abilities, To Each According to His Needs.” 1st International Workshop on Peer-to-Peer Systems (IPTPS ’02), 7-8 March 2002, MIT, Cambridge, Mass., USA. <http://www.cs.rice.edu/Conferences/IPTPS02/>

[2] Hurlburt, Nick, Christopher Neihengen, Charlie Pikscher and Andrew Rosenfeld. “Peer to Peer MOO.” Project proposal abstract. <http://wiki.cs.uiuc.edu/cs497rej/Peer+to+Peer+MOO>

Appendix A: Communication through Messages

6.1 On Joining the MUD

PEER to GATEWAY: Request for map
GATEWAY to PEER: Reply with map
PEER to GATEWAY: Request for innkeeper list
GATEWAY to PEER: Reply with innkeeper list

6.2 On Entering an Empty Room

Peer designates itself as innkeeper of the new room
PEER to ALL innkeepers & GATEWAY: Send current innkeeper list

6.3 On Leaving an Otherwise Empty Room

Peer removes itself as innkeeper of the old room
PEER to ALL INNKEEPERS & GATEWAY: Send current innkeeper list

6.4 On Entering an Inhabited Room

PEER to NEW INNKEEPER: Send enter message
New innkeeper adds peer to room list
NEW INNKEEPER to ALL PEERS IN ROOM: Send room list

6.5 On Leaving an Otherwise Inhabited Room

PEER to OLD INNKEEPER: Send leave message
OLD INNKEEPER to PEER: Send innkeeper list
Old innkeeper removes peer from room list
OLD INNKEEPER to ALL IN ROOM: Send room list

6.6 On Performing a "Say" (Speaking Publicly to Everyone within the Room)

PEER to ALL IN ROOM: Send text of utterance, marked public

6.7 On Performing a "Whisper" (Private Speech to a Single Other Player)

PEER to OTHER PEER: Send text of utterance, marked private

6.8 On Performing a "Holler" (Speech to Everyone in the Entire MUD)

PEER to INNKEEPER: Send text of utterance, marked holler
INNKEEPER to ALL INNKEEPERS: Send text of utterance, marked holler
ALL INNKEEPERS to ALL IN ROOM: Send text of utterance, marked holler

Appendix B: Message Formats

- INK:REQ - Request the innkeeper list
- INK:SET:lst - Send a message to set innkeeper list to lst
- BRD:brd - Message containing board configuration
- NEW - Sent to Gateway to inform of wish to enter MUD
- ENT:username - Notification of entering a room
- LEA:username - Notification of leaving a room
- PLA:lst - Send a message saying that the player list has been changed to lst
- SAY:msg - Say msg to all players in the room
- WHP:userX:msg - Whisper msg only to player userX
- HOL:msg - Holler a message to entire MUD