# Peer-to-Peer Architecture and Protocol for a Massively Multiplayer Online Game

Madjid Merabti and Abdennour El Rhalibi
School of Computing and Mathematical sciences
Liverpool John Moores Unversity
Liverpool, UK
a.elrhalibi@livjm.ac.uk ; m.merabti@livjm.ac.uk

*Abstract*— **Massively Multiplayer Online Games (MMOGs) are becoming a very important part of computer entertainment business. With recent development of broadband technologies, the increase of the number of players is putting a strong pressure on this type of applications. Commonly used clients/server systems do not cope well with scalability, limiting the number of players who interact with each other, are not robust enough and might be subject to bottleneck due to their centralized infrastructure. These systems also force the developers to invest enormous amount of money on hardware and time to design complex software systems. To solve these problems we propose a fully distributed, peer-to-peer architecture for MMOGs.**

**In this paper we discuss the issues surrounding MMOGs, the limitations in term of network infrastructure, and the lack of simulation environment to study and evaluate network architecture and protocol. We use a peer-to-peer (P2P) based architecture and protocol to provide a more scalable, flexible and robust technology solution than currently used infrastructures. We have conducted the design and implementation of a modular MMOG: 'Time-Prisoners', using a P2P protocol developed in Java and JXTA. The characteristics of P2P overlays enable to organize dynamically, and in transparent way for the users, the group of players according to their locations in the virtual world, and allow to design scalable mechanism to distribute the game state to the players and to maintain the world consistent in case of node failures.**

***Keywords- Peer-to-Peer Architecture; MMOG; Online Gaming; JXTA; protocol; Agent Design.***

## I. INTRODUCTION

Massively Multiplayer Online Games (MMOGs) [1] are one of the most interesting genres of modern computer games. Born out of the internet boom in the mid to late 90's, they have rapidly gained in popularity. The most popular MMOG, Lineage [2] claims to have over four million active subscribers. MMOGs, as the name suggests, are online games played simultaneously with tens of thousands of players at one time. The games require an internet connection to play. Each player has a copy of the software installed on their machine, which uses the internet connection to connect to a central game server, which in turns keeps all the players up to date with what is occurring in the world. Traditionally, most MMOGs have been Tolkien-esque fantasy role-playing games. While games of this type are still extremely popular, only recently has the full potential of this medium been realized, with games such as Planetside [3], Star Wars Galaxies [4], The Sims Online [5] and EVE [6] appealing to a broad range of player types.

MMOGs nearly always charge a subscription fee to each player in addition to the initial cost of purchasing the game client. This subscription payment is used to cover the costs generated by the game: Customer Service, Patches, Content Updates, Data Storage, Server Maintenance and Bandwidth [7].

The last three make up the largest proportion of the cost. These costs are not directly spent on improving the gameplay for the players, but are a necessity to support the client/server model that forms the backbone of the game. By changing the network topology used to support MMOGs, these costs could be reduced. These savings could be then passed on to the player, greatly reducing the costs, or alternatively spent on developing the game further. The paper discusses the feasibility of using peer-to-peer (P2P) overlays [8] to support a typical MMOG, as a replacement for the client/server model. The technical details of these two topologies will be discussed, along with the issues. P2P infrastructure provides a better, cheaper, more flexible, robust and scalable technology solution for MMOGs.

The rest of the paper is organized as follows: section 2 provides background information about current MMOG network architectures and P2P overlays, section 3 briefly presents the application we have developed to deploy in a P2P overlays, section 4 introduces the P2P architecture we propose, introduce the software architecture meta-model and a protocol, and discusses the issues and possible solutions in the P2P architecture proposed, section 5 introduce some aspects of our communication system implementation, and finally in section 6 we review the concepts presented in this paper, conclude on the viability of the approach and present the future work.

## II. CURRENT MMOG ARCHITECTURE AND TECHNOLOGIES

In this section we discuss some currently available topologies, which are or could be used for MMOGs. We also discuss some of the P2P overlays available.

### A. Client/Server Topology for MMOG)

The client/server network topology is very common in MMOGs [1][9]. Typically there is a group of "client" machines who want to share information, be they financial reports or game data. Each client connects directly to a server, which deals out information individually to each client as it is requested. This kind of topology commonly used in small-scale multiplayer games such as Half-Life [10]. One player chooses to be the server and host the game, then all the other players connect directly to his machine. Whenever a client shoots a gun, moves or performs another action, the data is sent to the server, which calculates the results of that action and forwards the result to all client machines connected.

A single server is fine for small-scale multiplayer games, where the number of players is up to around 64, but a single machine is usually not sufficient to deal with thousands of players synchronously, so typically a MMOG "server" is a group of machines with dedicated responsibilities, as is seen in figure 1. This diagram is only one possible configuration of machines that could make up the server component of a MMOG. Each different machine has a different responsibility to the game. The whole "cluster" of machines operates using grid computing [11] methods to dynamically share resources and ensure consistency across the cluster. When a client first attempts to connect to the game, they connect to the login server, which checks the user exists, has a password, and has a paid up account. They may then be forwarded to the patch server, which checks the client version, and can send any game updates. When the patch server has confirmed the client version is up to date, they are connected to a game server and, perhaps, a separate chat server. In this example the chat server handles all player to player communication, regardless of where they are in the virtual gameworld and the game server handles everything else (e.g. physics, trading, combat). In this example, the "game server" is again split into several smaller sub-servers. These sub-servers could, for example, each handle a certain geographical area of the gameworld.

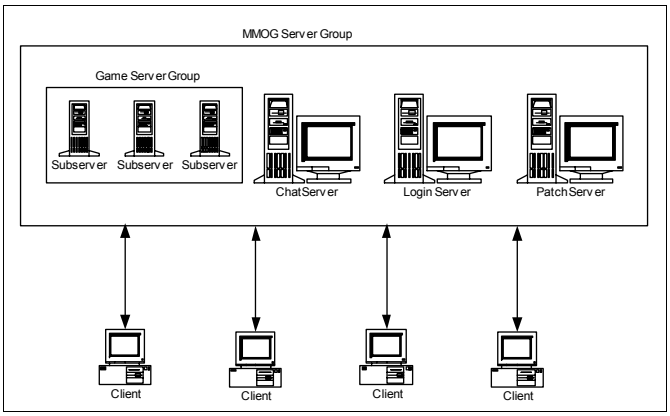



Figure 1. Possible MMOG Server Model

This server model is only one of many possibilities [1]. Some games split the responsibilities of the servers in different ways, for example having dedicated servers for databases, physics or even Artificial Intelligence. One thing common to nearly all modern MMOGs is that the server group acts as the single server in the basic server/client topology, so they all suffer from the cost overhead of running and supporting so many machines. With many modern games there are even many clusters of servers, sometime referred to as "shards", that allow many distinct copies of the game to coexist. Usually these server clusters are located at different places in the real world, allowing players to use the clusters located near to them geographically, thus reducing the effect of network latency or "lag" [1][13].

In client/server infrastructure for MMOGs there are many known issues and solutions related to scalability [14], robustness [8], security and proof-cheating [1][16], bandwidth savings[8], network and transport protocol [15], and delay compensation techniques [13]. However the solutions usually employed are costly and lack of flexibility. For example in the case of the scalability the architecture usually uses server clusters, connected by LANs, or forming a computer grid. Although this architecture scale with the number of players, the server might need to be over-provisioned to handle peaks loads [14].

### B. Peer-to-Peer Topology and Overlays

Peer-to-peer [8][17], or "P2P", networking has become a bit of a technological buzzword in the past few years. It has been popularized by file sharing applications like Kazaa [18], Gnutella[19] and Morpheus[20], who use the technology to build ad-hoc networks to allow sharing of files.

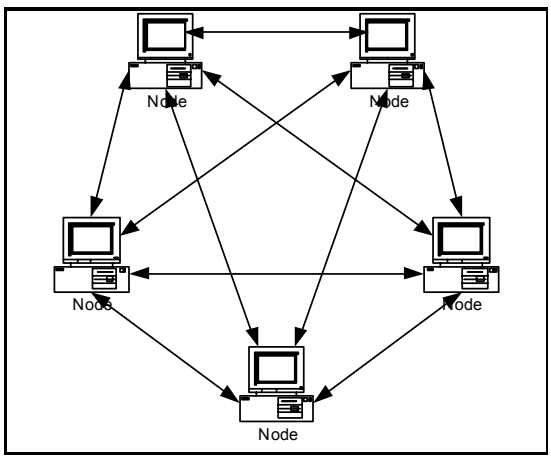



Figure 2. Basic P2P Topology

Figure 2 shows a very basic illustration of how a peer-to-peer network is organized. In essence, each "Node" on the network has exactly the same responsibilities as every other node. It has no requirement for a particular machine to be in the network, and no other node has a requirement for it. This example is simplified, as modern P2P networks do not usually work with this basic a structure. While each node is still capable of the same functions as each other, modern P2P

networks are able to organize themselves into more efficient structures.

Peer-to-peer networks have a lot of advantages over networks using the traditional server/client model used by MMOGs. Firstly there is the issue of cost. The network is distributed among the clients, and does not require a central server in order to work. The computation usually done by the server machine is shared instead by the clients. The peer-to-peer network of client machines can be treated as a giant grid computer, where calculations are performed in different parts of the conceptual "whole". This is similar to grid computing projects such as the SETI@Home[21] projects. If the entire server infrastructure (or even just a fraction) of a MMOG can be replaced by a grid-like system, a massive saving can be made. The amount of bandwidth used is reduced dramatically. Where before two clients wanting to communicate would be required to do so through the server, effectively doubling the amount of bandwidth required in a P2P network where they can communicate directly.

Latency on the network is also reduced thanks to the elimination of the bottleneck caused by a server handling all information between all clients, regardless of who each client is trying to communicate with.

In the next section we discuss available P2P overlays and their suitability for building multi-platform support for MMOGs.

A number of P2P protocols have been recently devised, including JXTA [27], Pastry [17], Tapestry [17], Chord[17] and Can[17]. They are self-organizing, decentralized systems and provide the functionality of scalable distributed hash table (DHT) [17]. The systems balance object hosting and query load, transparently reconfigure after node failures, and provide efficient routing queries [15][24].

We are particularly interested in JXTA [17] which provides a far more abstract language for peer communication than previous P2P protocols, enabling a wider variety of services, devices, and network transports to be used in P2P networks.

JXTA is meant to provide a basic set of services and APIs required for the development of any peer-to-peer application. The architecture of JXTA divides the software into three layers: the JXTA core, JXTA services and JXTA applications [27]. The core implements all the basic concepts involved in P2P communication. In particular, it provides the necessary functions for the management of peers and inter-peer message exchange. Furthermore, the core handles peer discovery and monitoring. The JXTA services layer is responsible for generic services that may be required in common P2P situations, such as file sharing and indexing. The JXTA applications layer is reserved for applications developed by the applications developers. It is in this layer that our application will reside.

Peers are organized into centers of interest called peer groups, which segregate the different communities participating in the JXTA network, and provide a way to control the propagation of broadcast traffic, for example.

Peers communicate by sending messages over JXTA pipes. These provide a transport-agnostic abstraction of the message exchange to client applications. Because pipes make use of the underlying transport protocol, whatever it may be, nothing can be assumed about the reliability of messages sent over JXTA pipes. It is, however, possible for developers to design their application so as to ensure reliability by implementing their own scheme, for example using TCP, UDP or even more specialized recent solutions such as GTP [15] and event synchronization protocol [28].

We can see that because of its rich set of P2P functionality, JXTA is eminently suited to our application. Augmented with a judicious reliability design, it provides an excellent starting point for a P2P based MMOG.

Before presenting the P2P architecture and protocol, we discuss the specification of the Game 'Time Prisoners' we have designed and implemented as a MMOG.

## III. 'TIME PRISONERS' MMOG SPECIFICATION

In this section we briefly present the specification of the game we have developed as a test-bed to deploy and test our P2P MMOG network over the internet.

The first step in the development of an MMOG is to design the gameplay itself, in both its technical and functional [22]. The game we have developed is 'Time Prisoners'. In figure 3, we can see screen shots depicting some of the regions and levels of the game.

The game-world is composed of 'parallel' worlds representing different donjons/regions and time (medieval, modern-war, etc…) in which the player must complete missions which consist in freeing prisoners, and fighting monsters/guards which wander in the world. Each region/time is composed of several levels, with puzzles to solve (maze to access the prisoners, finding trigger or keys to open levels, etc…) and items to collect (potion, ammunition, keys, etc…).

Players can navigate from region/time to region/time and see their character changes appearance and weapons to match the region/time style. There are two types of NPCs the prisoners and the monsters/guards. The AI controlling the NPCs (prisoners and monsters) is implemented using fuzzy finite-states machines [23], path-finding [23], influence mapping [23] and support Dead Reckoning [13].

The interactions (implemented as collision events) are numerous between players, players and NPCs, players and items, prisoners and guards, and all the characters with wall and objects. The world is relatively complex in terms of the events generated and the number of world updates which will be required.

The players can start to play in whatever region they want, and when they first join the game their machine will be either attached as client to an existing peer-group playing in the same region/level, or will be assigned the role of the main node (the server) of a new peer-group of future clients which will be playing in the same region/level.

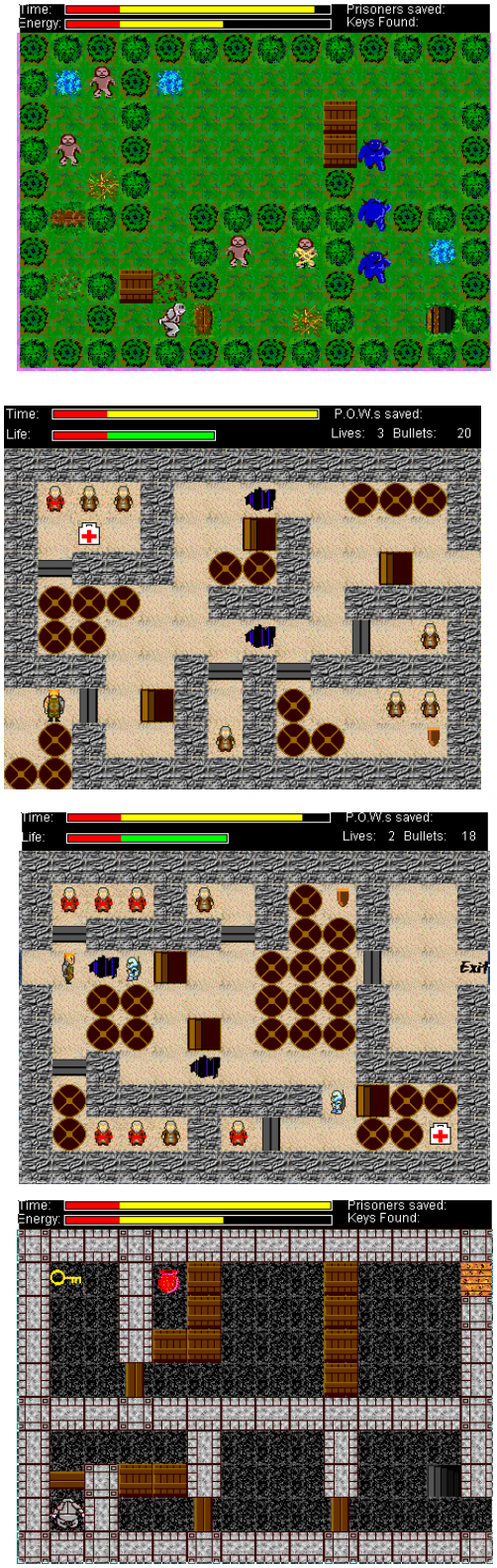



Figure 3. Screen shots of different world and levels in 'Time Prisoners'

Having briefly described our game application, we present in the next section the P2P architecture and protocol.

## IV. A Peer-to-Peer Architecture and Protocol for MMOGs

In section 2, we have mainly been concerned with examining the two extreme topologies available to MMOGs. However there are many hybrid techniques [1] that can overcome some of the issues outlined with the 'pure' topologies.

### *A. A P2P Topology*

The topology we propose is a hybrid solution starting with an initial architecture based on a main server, and building-up a P2P topology as the number of players increases (see Figure 4).

As the players connect to the game, the server delegates more and more of its role as game and network/communication manager to the connected player machines, which self-organize in a P2P fashion. The peers still rely on the initial server to join and leave the game, and to help them discover their peer-group if already created and to receive the game data when a new region is required. However all the in-game communications once the players is connected to his peer-group are done in P2P fashion.

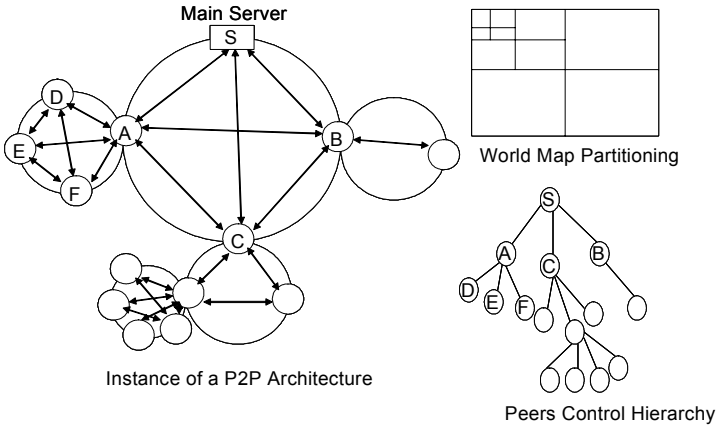



Figure 4. Proposed Architecture

The architecture still allows the developers to maintain direct control and authority over the players account information, which are more secure when subscriptions and personal details are involved. A simple example is that of the 'Login' and account management server are kept away from the peer-to-peer network. They are managed into a separate client/server network style that acts as a gateway into the game. This separate network doesn't involve the client/server issues mentioned as it is only a one-off connection for the players. Once a joining player has been checked, he will granted connection to his peer-group and will be managed in a P2P game session until he leaves the game.

The architecture is flexible, robust and dynamic. Several spatial data structures are used to control peers group and their relations in a dynamic way, and to map the peer-groups.

In the following sections we discuss the meta-model architecture and the P2P protocol for joining and leaving the game. This protocol is the more important as regards to the P2P architecture.

### B. *Meta-Model Architecture*

The first step in our P2P architecture and protocol development is the design of an agent based meta-model architecture. We have defined a high level design based on mobile agent model and suitable for the development of the MMOG and a simulation environment. Our meta-model architecture provides the rules to develop a simulation environment for MMOGs, and to implement an instance of P2P protocol for MMOGs. Using a mobile agent model we can represent all the dynamic and static aspects of our MMOG.

Our agent behavior model consists of three levels of design: the system-level, the host-level, and the agent-level. The system-level design describes an overview of the system and the relationships among hosts in the system. Host-level design uses State-chart to describe the behavior between agents within a host and between hosts, for example communication. Agent-level design uses finite state machines to describe the behavior of a single agent.

We have also incorporated the concepts of agent cloning, host replication, and agent groups within our framework.

To support multi-agent organization, communication and coordination as a P2P infrastructure, we also incorporate the concept of agent groups. Any agent within the agent group may perform multicast or subcast. A multicast communication allows an agent in the group to send a message to all other agents in the group, no matter where the agents are in the system. A subcast allows an agent to send a message to a subset of the group. Agent groups are identified by name, thus an agent may join a group by merely specifying its name.

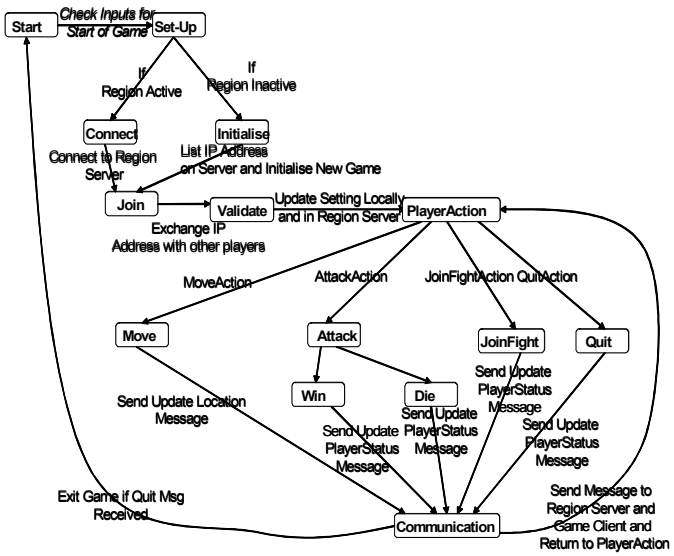



Figure 5. Agent communication level

The overall system behavior can be seen as an emergent property of the concurrent execution of all agents in the system. Our framework enables the understanding of P2P based MMOG design using mobile agents by specifying and simulating behavior on various levels.

## V. HIGH LEVEL DESIGN OF AGENTS ARCHITECTURE

In this section we will concentrate on the user management aspects of the MMOG system. The requirements for the MMOG system are as follows: There exists only one central database server where all usernames and passwords are stored. The Central Database Server is connected to Region Servers (which are the first players/clients logging to region in the virtual world where there are no other players). Each Region Server handles a different region in the virtual world. Once the Region Server is initiated, to join the virtual world, each user uses a client to connect to the Region Server which represents the particular region. Region Servers notify the Central Database regarding to the user's login information. If a new user logs in, the Central Database will register a new account for the user, otherwise, the user will be checked for authorization. After login, users within close proximity in the shared virtual space need to receive state updates about each other in real-time. The state information of each user should be stored on the Region Server, where the user first created his account.

### A. *System-Level Design*

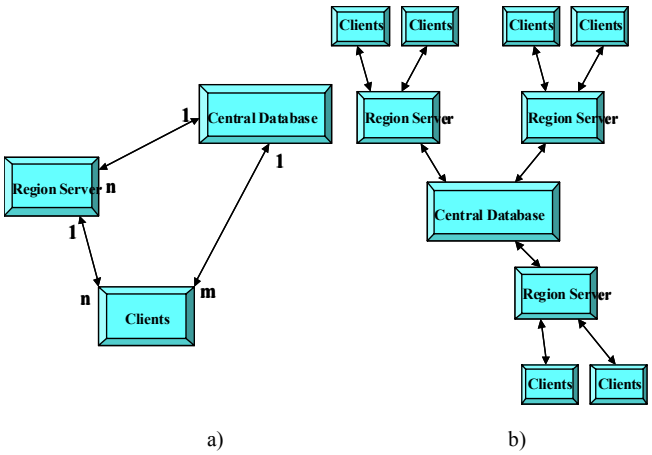
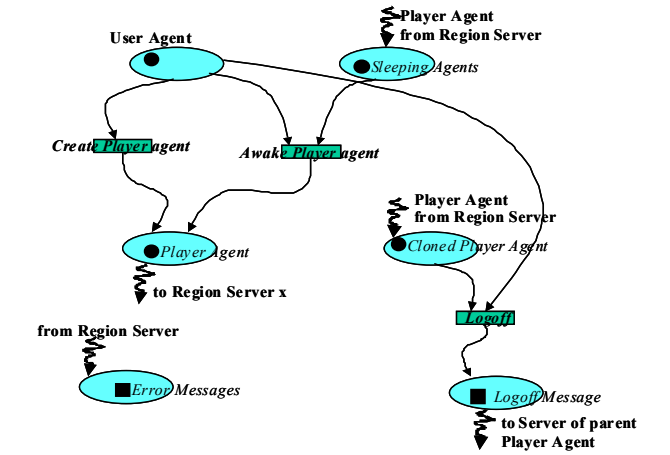


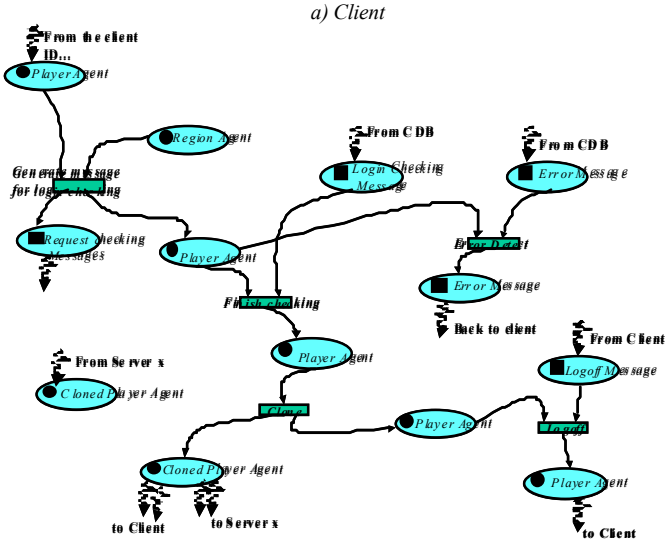

Figure 6. System High level Design of MMOG

To start we need to consider the system as a whole. Thus, we will start by working on the system-level design. In Fig 6.a, we have one central database (unique) along with regional servers (replicable) and clients (replicable). Both regional servers and clients are replicable hosts. A replicable host can have multiple instances of itself. The connections between the hosts are bi-directional system links, which means that the hosts can communicate in both directions. The number and two variables (n, m) describe the relationship between the hosts. In the example, the relationship between the central database to regional server is 1 to n. This signifies that one Central Database Server is connected to multiple Region Servers. Note also that the Central Database Server is unique in the system. Similarly with the regional server and client, one regional server is connected to multiple clients. Fig 7.b is an expanded view of the system-level design which may be more helpful in visualizing the physical configuration of the system.

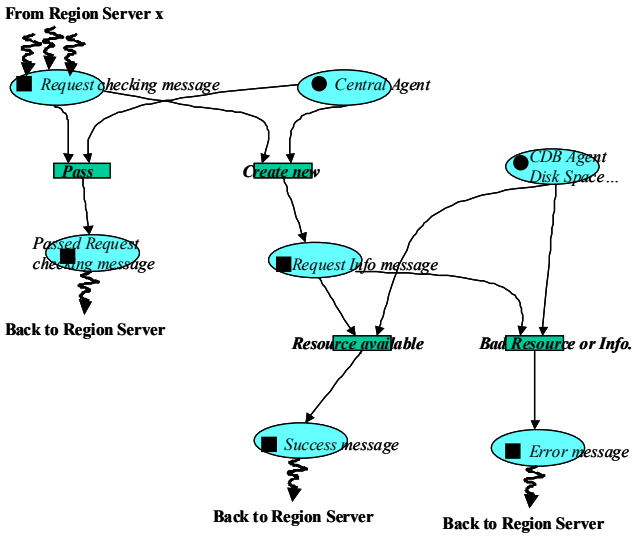
## B. Host-Level Design




a) Client




b) Region Server




c) Central Database Server

Figure 7. Host-Level Design of MMOG

With the system-level view complete, we can now start designing the hosts that were introduced in the system-level design. Host-level design is based on process/event modeling. The order of design of the hosts will not affect the outcome. We will start with the host-level design of the client host.

### 1) Client

The first step is to determine the agents that may reside on or traverse through this host. We introduce a User Agent to represent each user's account information. In MMOG a user is typically able to create multiple characters (players). Thus, the User Agent should be able to create new Player Agents or wake up existing Player Agents already created by the user. The User Agent should also send out a message to the Region Server upon logoff. We can immediately identify that the User Agent will be static, because it will only process jobs that is related to users on that Region Server.

The next step is determining the transitions and places on the Client Host. There are three events: creating the Player Agent, waking up the Player Agent, and logoff. Fig 7.a shows the relationship among the User agent, the active Player Agents and the sleeping Player Agents. The diagrams show two types of arcs, the arcs going from the place to the transition state and then going from the transition state to a place, these are the process arcs; and the arcs incoming or outgoing from a place, which are the migration arcs. Each migration arc will specify where the agent is going to or coming from. If we follow the path for creating or waking up the Player Agents, there is a migration path to a Region Server with id x. The waked up Player Agent migrates to the appropriate Region Server according to the host ID.

Let us migrate with the Player Agent to the Region Server. A Region Agent is already waiting for the incoming Player Agent. Once a Player Agent has arrived, the Region Agent generates a check up message and the Player Agent waits for the login confirmation from the Central Database Server.

### 2) Central Database Server

Let us move our point of view to the Central Database Server. In Fig 7.c, as we can see, there are lot migrations coming in from the regional server. The central agent is already waiting on the host to check for the login identification or create a new account. If the identification is correct, the request-checking message will be sent back to the Region Server identifying that the information is correct. Otherwise, if the user tries to create a new account, the Central Database Disk Space Agent will allocate space to record the information for the new user. If the requesting-information is valid and there is enough disk space, the account will be created, otherwise an error message will be created to notify the user that the information is not valid or there is not enough disk space. The messages will be sent back to the Region Server, as shown in Fig 7.b, near the right hand top where the return messages are received. If the incoming message is an error message the error message is forwarded back to client and eventually the user will have to re-enter the information. If the incoming message is a successful message, the waiting Player Agent can now get

ready to clone. Each player agent then migrates to different Region Servers and also to the client where the player agent came from.

The original agent stays at the Region Server for increased security and improved recovery from client crashes.

At this point the Player Agent stays at the regional server until the client decides to logoff. Let us look back to Fig 7.a, we can see the User Agent decides when to logoff.

Once the user decides to quit, the User Agent will generate a logoff message and then the message will be sent to the home Region Server. Follow with the message to Fig 7.b, where the logoff message comes in. The Player Agent now migrates back to the client. In Fig 7.a, the Player Agent comes back from the Region Server and is put to sleep. The player agent may be awakened later when the user decides to login again and reuse the player.

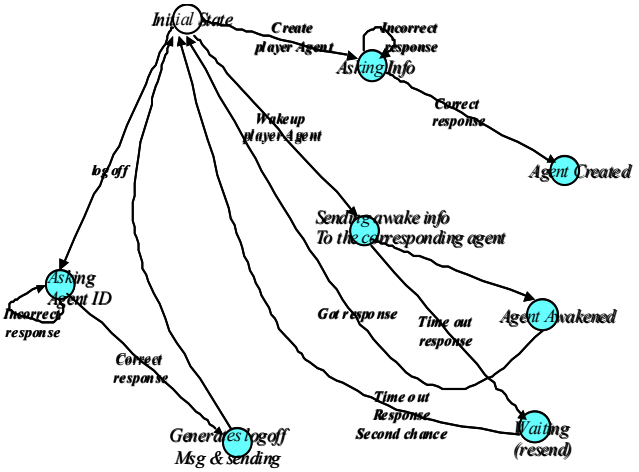



Figure 8. User agent

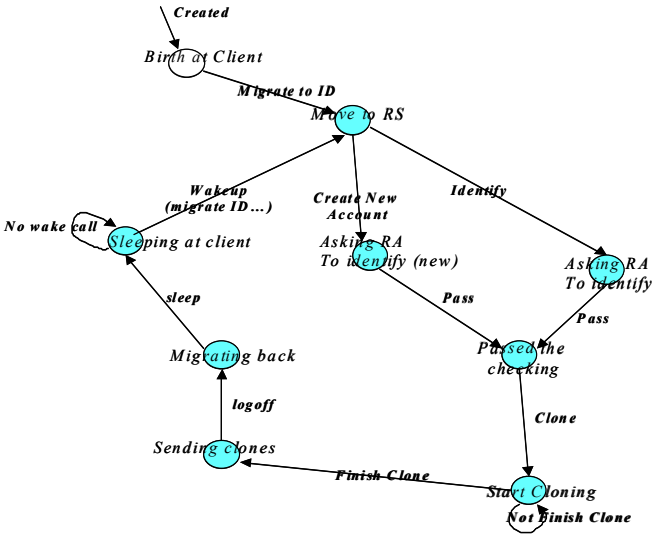



Figure 9. Player Agent

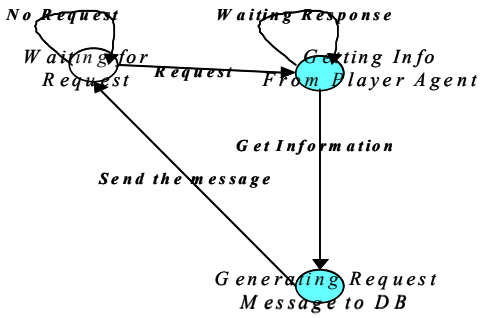



Figure 10. Region Agent

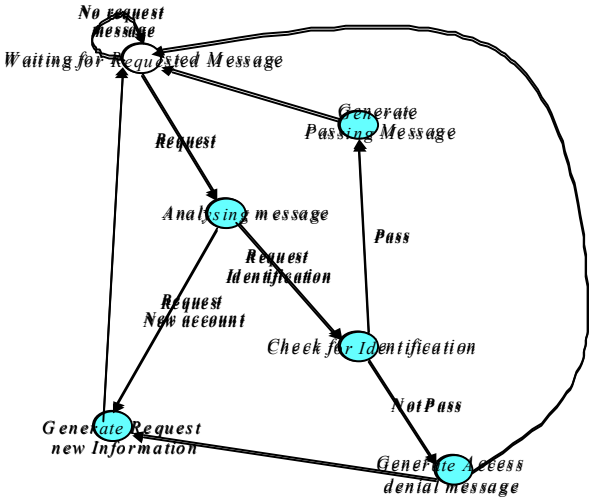



Figure 11. Central Agent

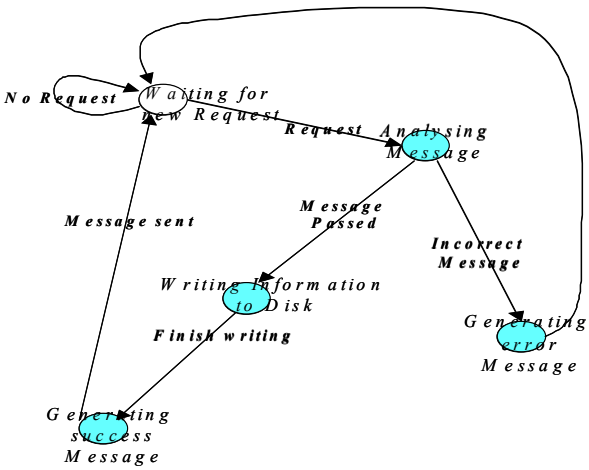



Figure 12. CDB Disk Space Agent

Agent-level design allows us to understand the behavior of individual agents. Let us look into Figures 8 to 12. Fig 8 describes the states of the User Agent. To start, we look at the initial state. If the user decides to create a new Player Agent, the User Agent will then go to the "asking info" state where the user agent will prompt the user for create information. When get the correct response, a Player Agent will be created and the User Agent will go back to the initial

state. If the User Agent decides to use the created Player Agent, it will go to the "sending awake info to the corresponding agent" state and follow the path. This comprises the familiar finite state machine (FSM) of the User Agent.

The individual behaviors of the other agents are shown in Figures 9 to 12. Detailed descriptions are not given here; the diagrams are self-explanatory.

With the combination of these finite state machines at the agent-level (individual behavior), the Process/event model at the host-level (group behavior), and the overall view at the system-level, the complete system design can be seen as a large Process/Event model describing interacting autonomous agents residing in places among a set of hosts.

The meta-model and the high level design of the agents' architecture provide a very suitable framework for MMOGs as dynamic distributed applications, and will be completely described in a future publication.

### C. P2P Protocol for Joining/Leaving

In term of architecture build-up the protocol for joining and leaving are the most important. Below we present briefly the main scenarios in the P2P protocol for joining and leaving the game:

- In the initial state there are no players, no peers, only the main game server (world server) maintaining the full game, the players database, the current game data for existing players, etc…
- If a new player joins the game, he will be connected to the world server and assigned the role of new region server for the region he will play on. He will be sent all the data required to maintain the region, and the identity of any other currently running region server. Any new player joining in the same region will be connected in P2P fashion to the peer-group managed by the region server. The region server will be sent the data of the game, the data of the players joining, and the data of the others existing regions server controlling the others regions.
- If a player moves to another region we have two situations which might occur. Firstly if the region is already controlled by another server region, the new player will join the peer-group associated to this server-region. Secondly if no peer-group for the region exists, the client machine will be assigned the role of server region.
- If a player leaves the game. We have again two cases. If the player is a client in his region peer-group, it will be simply disconnected and the peer-group will be informed. If the leaving player is a region server, the protocol will elect a current peer (client) to become the new region server. The connection with the world server will be re-established. If there are others regions server they will be informed of the disconnection of the leaving region server, and will be given the identity of the new region server. Only the region servers are connected to the world region (i.e. the main server).

Figure 5 shows an example for the agent communication level, part of the communication protocol associated to a peer. Note in particular the states Join and Quit which implement the protocol described in this section.

## VI. DISCUSSION

In this section we discuss the issues which might occurs in a P2P network and the solutions we have devised. The issues are related to scalability, network efficiency, data storage, and policing.

### A. Scalability

One of the usual issues with peer-to-peer networks is their scalability. We have designed our topology to have the ability to organize itself efficiently, to reduce the inherent scalability issues. The most basic of this organization is the creation of "Supernodes". The network can identify which nodes could make good supernodes based on how much bandwidth a node has, and how often and for how long it is usually a part of the network. Supernodes connect to one another, and have a collection of normal nodes connected to them. In larger networks, a collection of supernodes could even set up a super-supernode, connected to other super-supernodes and so on. These scaling techniques have been applied to our MMOG to improve efficiency. For example as in figure 6, each supernode includes a subnet of peers based on their locations in the game world.

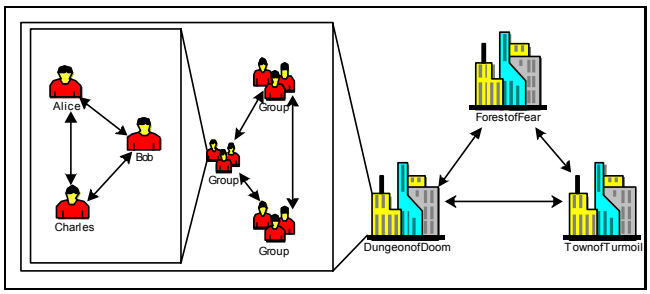



Figure 13. Supernode Hierarchy in P2P MMOG

The diagram does not clearly point out that the "group" and location supernodes would actually be controlled by the nodes within those groups, actually maintained by the player node software itself, but it is how it works.

### B. Network Efficiency

As with the server/client architecture, several methods can be used to make a MMOG more efficiently use its network resources. Most of these techniques can be equally applied to a MMOG running on a peer-to-peer topology. Dead Reckoning [13] algorithms greatly reduce the amount of object position data required to be sent over the network. Multicasting packets [14] is also used instead of traditional unicasting, thereby reducing the load of network traffic.

### C. Data Storage

MMOGs with client/server architecture have a single huge "game state" [22], which is the data of the game world held usually exclusively by the server. This data is extremely important to the game and access and changes to it are strictly monitored by the server software.

In a game running on a P2P network, the storage of the game data must be distributed among the nodes, and it must store a large amount of redundant (replicated) data, so that the game can still function with a minimum number of players.

A way of storing this much data among a collection of nodes hierarchically organized, where information could be required by any node, requires careful designing [24].

We believe there is a solution that can be adapted to be used to store data this way based on the Freenet project [25]. In our system each node "donates" an amount of hard disk space to the software when it is installed. When a node wishes to add data to the network, the network finds nodes with available disk space to store the data. When someone wants to access that data, they request it to be sent. Whenever data is requested by a node, that data, or a portion of that data, is replicated to data stores in nodes closer to the node that made the request, thus increasing the availability, redundancy and bandwidth available for that data and anyone who wants it.

### D. Policing

In a peer-to-peer game of any sort, it is important for each node not to trust any other node. Yet in order for the game to function, calculations must be carried out at some point.

Most game events can be reduced to simple transactions between two players, or between a player and the game. In a peer-to-peer MMOG, this transaction requires that neither player has any control whatsoever. This means that no player data (except a reference to that data) may be held on the node owned by that player, and no transactions pertaining to that player may be calculated on that node. Instead, a group of nodes must all perform that transaction, by finding and checking the data on each player, then each node in the group double checks their result with the rest of the group before storing the data. Given that none of the nodes in the transaction can control which nodes are involved in it, this emulates the existence of a "trusted" server as in current MMOGs.

## VII. Communication System Implementation

We have developed our MMOG in Java and use a peer-to-peer architecture, incorporating JXTA as part of the underlying framework.

Figure 7 shows part of the game logic and communication system class hierarchies.

Figure 8 shows the layered structure of the full application, involving JXTA services and core, JXTA communication interface, the communication system and the game engine.

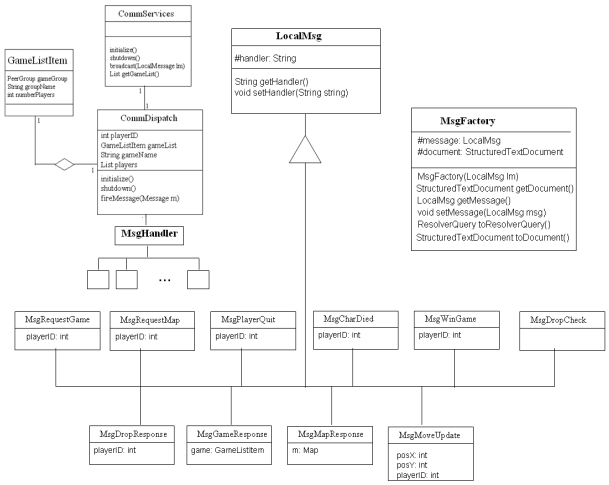



Figure 14. Class Architecture

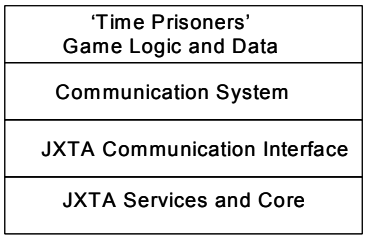



Figure 15. Software Architecture

We are particularly interested in explaining the communication system. The purpose of the communication system is to provide the game with a way to communicate with other peers over the network. This allows for the passing of information between game peers, such as requesting information about games and updating player information.

The communication subsystem provides the following services:

- start a multi-player session game, given the map/region/donjon which will be used
- get a game list
- join a game, given the player to join and an item from the game list
- broadcast a character position update message, given the new position
- broadcast a quit game message
- broadcast a win game message
- broadcast a player died message
- shutdown

The messages that are going to be passed between communication systems are JXTA messages which are XML documents, or binary or both (binary embedded into XML).

They will be constructed by using the player state context and, then sent out by the communication system.

## VIII. Conclusion and Future Work

Massively Multiplayer Online Games have been rapidly gaining popularity and have proven that there is a big future in online gaming. The subscription price model used by most MMOGs is greatly slowing the uptake in new subscribers, and ways need to be developed in order to reduce the overhead of the server upkeep costs and to pass on a saving to the customer, either in monetary terms or in gameplay enhancement.

We have discussed an extreme way that the technology MMOGs used can be altered to vastly reduce the cost of upkeep.

MMOGs by their distributed and incremental structure are natural applications for P2P overlays. Games are different from previous P2P applications that focus on the efficient use of idle storage and network bandwidth, including storage systems, content distribution and instant messaging. Games can use the memory and CPU cycles of peers to maintain and share the game state.

Our protocol offers a fully distributed, fault tolerant and scalable solution to MMOG. It is more efficient than usual infrastructure which relies on a set of dedicated servers that agree to share the workload. This protocol may also be extended for mobile networks [29] where a peer-group manager is a routing agent responsible for clients in its group. The joining and leaving protocol would remain mostly the same for such an application.

Using a peer-to-peer topology to eliminate the server entirely would be a novel way to "float" a MMOG on the internet. However, further research needs to be done into making a viable distributed game over a pure P2P overlay

Ultimately, P2P approach is the only viable means for smaller independent game companies to create massively multiplayer online games (MMOGs) that scale to millions of users.

Our future work will involve the development of an agent-based MMOG simulator to test 100 thousands of simulated players using different transport protocols, larger deployment of 'Time Prisoners' to test the P2P infrastructure based on JXTA, and an extensive evaluation of the simulator and the real system for scalability and robustness.

## Acknowledgment

The authors would like to thank the anonymous reviewers for the insightful and very useful comments.

## References

[1] Smed, Kaukoranta, and Hakonen, "Aspects of Networking in Multiplayer Computer Games", The Electronic Library, 20(2):87-97, 2002.

[2] Lineage, © NCsoft (http://www.lineage.com/nci)

[3] PlanetSide, © Sony Online Entertainment (http://planetside.station.sony.com/ )

[4] Star Wars Galaxies, © Sony Online Entertainment (http://starwarsgalaxies.station.sony.com/)

[5] The Sims Online, © Electronic Arts (http://www.eagames.com)

[6] [EVE: The Second Genesis, © CCP (http://www.ccpgames.com/)

[7] Bauer D., Rooney S. and Scatton P. 2002. "Network Infrastructure for massively Distributed Games". In Proceedings of NetGames 2002, (Braunschweig, Germany, April), pp3-9.

[8] Bjorn Knutsson, Honghui Lu, Wei Xu and Bryan Hopkins "Peer-to-Peer Support for Massively Multiplayer Games", INFOCOM 2004, March 2004, Hong Kong, China.

[9] Smed, Kaukoranta, and Hakonen, "Networking and Multiplayer Computer Games--The Story So Far", International Journal of Intelligent Games & Simulation, 2(2):101-110, 2003.

[10] HalfLife, © Sierra (http://games.sierra.com/games/half-life/)

[11] Tianqi Wang, Cho-Li Wang, Francis Lau, "Grid-enabled Multi-server Network Game Architecture," the 3rd International Conference on Application and Development of Computer Games (ADCOG 2004), April 26-27 2004, City University of Hong Kong, HKSAR.

[12] EverQuest, © Sony Online Entertainment (http://soe.sony.com/)

[13] Y.W. Bernier. "Latency compensation techniques methods in client/server in-game protocol design and optimization". In Proceeding of the Game Developers Conferences, March 2000.

[14] P. Francis, M. Handley, R. Karp, S. Ratnasamy, and S. Shenker. "A Scalable Content-Addressable Network." In SIGCOMM '01, August 27-31, 2001, San Diego, California.

[15] Eunsil Hong, Sangheon Pack, Yanghee Choi, Ilkyu Park, Jong-Sung Kim, and Dongil Ko, "Game Transport Protocol: Transport Protocol for Efficient Transmission of Game Event Data," in Proc. JCCI 2002, Jeju, Korea, April 2002.

[16] Golle, P., and Mironov, I. "Uncheatable Distributed Computations". Lecture Notes in Computer Science 2020 (2001).

[17] K. Kant, R. Iyer and V. Tewari, "A framework for classifying peer-to-peer Technologies", 2nd IEEE/ACM Intl. Symposium on Cluster Computing and the Grid, May 21-26, 2002, Berlin, Germany.

[18] KazaA - http://www.kazaa.com/us/index.htm

[19] Gnutella Protocol Development - http://rfc-gnutella.sourceforge.net/

[20] Morpheus - www.morpheus.com/

[21] Seti@Home - http://setiathome.ssl.berkeley.edu/

[22] Anne-Gwenn Bosser, "Massively Multi-player Games: Matching Game Design with Technical Design", ACM SIGCH Advanced Computer Entertainment Conference, ACE 2004. NUS, Singapore.

[23] Mat Buckland, "AI Techniques for Game Programming" Premier Press, Inc. - ISBN (1-931841-08-X) (October, 2002)

[24] K.L. Morse. "Interest Management in Large Scale Distributed Simulations". Technical Report ICS-TR-96-27, Universiy of California, Irvine, 1996.

[25] Freenet Project – http://www.freenetproject.org

[26] Maniatis, P., Roussopoulos, M., Giuli, T., Rosenthal, D. S. H., Baker, M., and Muliadi, Y. "Preserving Peer Replicas By Rate-Limited Sampled Voting". In SOSP (2003)

[27] JXTA - www.jxta.org/project/www/background.html

[28] Stefano Ferretti, and Marco Roccetti, "A Novel Obsolescence-Based Approach to Event Delivery Synchronisation in Multplayer Games". International Journal of Intelligent Games and Simulation, Vol 3 No 1, 2004, pp7-19.

[29] A. Joseph, J. Kubiatowicz, and B. Zhao. "Supporting Rapid Mobility via Locality in an Overlay Network". University of California, Berkeley.

[30] K.L. Morse. "Interest Management in Large Scale Distributed Simulations". Technical Report ICS-TR-96-27, Universiy of California, Irvine, 1996.