Agents-Based Modeling for a Peer-to-Peer MMOG Architecture

ABDENNOUR EL RHALIBI AND MADJID MERABTI
School of Computing and Mathematical Sciences, Liverpool John Moores University, Liverpool, U.K.

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

We discuss the issues surrounding MMOGs, the limitations in terms of network infrastructure and the lack of a simulation environment to study and evaluate network architectures and protocols. We use a peer-to-peer (P2P) based architecture and protocol to provide a more scalable, flexible, and robust technology solution than do currently used infrastructures. We conducted the design and implementation of a modular MMOG, called "Time-Prisoners," using a P2P protocol developed in Java and JXTA. The characteristics of P2P overlays enabled us to organize dynamically, and in transparent way for the users, the group of players according to their locations in the virtual world, and allowed the design of a scalable mechanism to distribute the game state to the players and to maintain a consistent world in case of node failures.

Categories and Subject Descriptors: C.2.1 [**Computer Communication Networks**]: Network Architecture and Design; C.2.4 [**Computer Communication Networks**]: Distributed Systems

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Network topology, network communications, distributed applications, peer-to-peer architecture, MMOG, online gaming, JXTA, protocol

1. INTRODUCTION

Massively Multiplayer Online Games (MMOGs) [PlanetSide, © Sony Online Entertainment] are one of the most interesting genres of modern computer games. Generated by the internet boom in the mid to late 90s, they have rapidly gained in popularity. The most popular MMOG, Lineage [Lineage, © NCsoft] 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 to connect to a central game server, which in turn keeps all the players up to date with what is occurring in their world. Traditionally, most MMOGs have been Tolkien-esque fantasy role-playing games. While games of this type

Authors' address: School of Computing and Mathematical Sciences, Liverpool John Moores University, Liverpool,UK; email: a.elrhalibi@livjm.ac.uk ; m.merabti@livjm.ac.uk

are still extremely popular, the full potential of this medium has been realized only recently, with games such as Planetside [PlanetSide © Sony Online Entertainment], Star Wars Galaxies [Star Wars Galaxies © Sony Online Entertainment], The Sims Online [The Sims Online © Electronic Arts] and EVE [EVE: The Second Genesis © CCP], 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 [Bauer et al. 2002].

The last three items 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, and the savings passed on to the players, greatly reducing the cost to them, or alternatively spent on developing the game further. This article discusses the feasibility of using peer-to-peer (P2P) overlays [Knutsson et al. 2004] 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 article 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 P2P overlays; Section 4 introduces the P2P architecture we propose, the software architecture meta-model and a protocol, and discusses the issues and possible solutions in the proposed P2P architecture; Section 5 introduces some aspects of our communication system implementation; and finally in Section 6 we review the concepts presented in this article, and conclude with a discussion on the viability of the approach and present the future work.

2. 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.

Client/Server Topology for MMOG

The client/server network topology is very common in MMOGs [Smed et al. 2002; 2003]. Typically, there is a group of "client" machines that want to share information, be it financial reporting 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 is commonly used in small-scale multiplayer games such as Half-Life [HalfLife © Sierra]. One player chooses to be the server and to 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 results to all connected client machines.

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 seen in Figure 1. This diagram is only one possible configuration of machines that could make up the server component of a MMOG. Each machine has a different responsibility to the game. The whole "cluster" of machines operates using grid computing [Wang et al. 2004] methods to dynamically share resources and ensure consistency across the cluster. When a client first attempts to connect to the

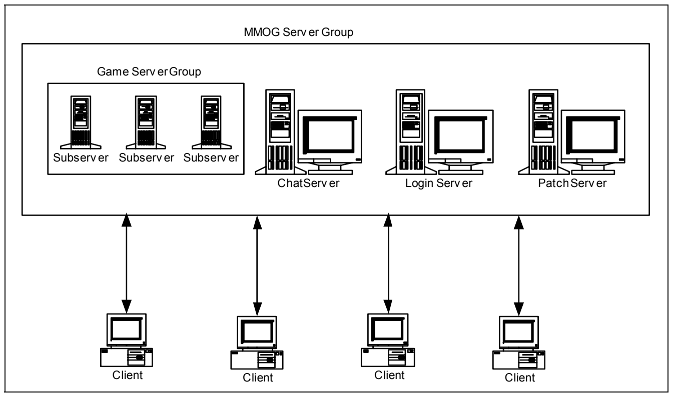


Fig. 1. Possible MMOG server model.

game, he or she connects to the login server, which checks that the user exists, has a password, and has a paid-up account. The user 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 that the client version is up to date, the user is connected to a game server and, perhaps, to a separate chat server. In this example the chat server handles all player-to-player communication, regardless of where the players are in the virtual game-world, and the game server handles everything else (e.g., physics, trading, combat). In this example, the "game server" is again split into several smaller subservers, which could, for example, each handle a certain geographical area of the game-world.

This server model is only one of many possibilities [Smed et al. 2002]. 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, hence 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" [Smed et al. 2002; Bernier 2000.].

In client/server infrastructure for MMOGs, there are many known issues and solutions related to scalability [Francis et al. 2001]; robustness [Knutsson et al. 2004]; security and proof-cheating [Smed et al. 2002; Golle and Mironov 2001]; bandwidth savings [Knutsson et al. 2004]; network and transport protocols [Hong, et al. 2002]; and delay compensation techniques [Bernier. 2000]. However, the solutions usually employed are costly and lack flexibility. For example, in the case of scalability, the architecture usually uses server clusters, connected by LANs or forming a computer grid. Although this architecture scales with the number of players, the server might need to be over-provisioned to handle peak loads [Francis et al. 2001].

Peer-to-Peer Topology and Overlays

Peer-to-peer [Knutsson et al. 2004; Kant et al. 2002], 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 [KazaA]; Gnutella [Gnutella Protocol Development]; and

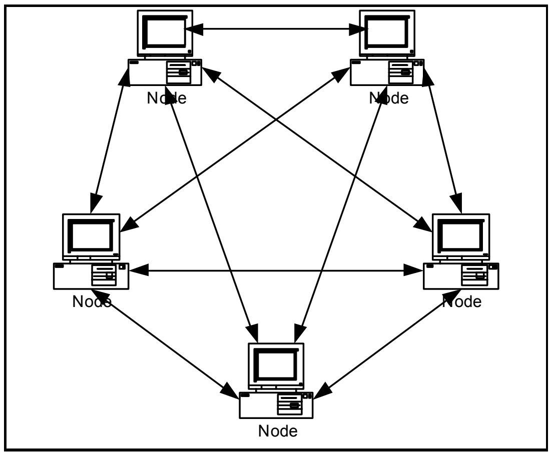


Fig. 2. Basic P2P topology.

Morpheus [Morpheus] that use the technology to build ad-hoc networks to allow file sharing.

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 any 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 [Seti@Home] 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 multiplatform support for MMOGs.

A number of P2P protocols have been devised recently, including JXTA [JXTA]; Pastry [Kant et al. 2002]; Tapestry [Kant et al. 2002]; Chord [Kant et al. 2002]; and Can [Kant et al. 2002]. They are self-organizing, decentralized systems that provide the functionality of a scalable distributed hash table (DHT) [Kant et al. 2002]. The systems balance object hosting and query loads, transparently reconfigure after node failures, and provide efficient routing queries [Hong et al. 2002; Morse 1996].

We are particularly interested in JXTA [Kant et al. 2002], 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 [JXTA]. 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; our application will reside in this layer. 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, for example, control the propagation of broadcast traffic.

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 to ensure reliability by implementing their own scheme, for example by using TCP, UDP, or even more specialized recent solutions such as GTP [Hong et al. 2002], and event synchronization protocol [Ferretti and Rocetti 2004].

We can see that due to 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.

3. "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 game-play itself, in both its technical and functional aspects [Bosser 2004]. The game we have developed is called '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 times (medieval, modern-war, etc.) in which the player must complete missions which consists in freeing prisoners and fighting monsters and guards that wander in the world. Each region and time is composed of several levels, with puzzles to solve (a maze to access the prisoners, finding trigger or keys to open levels, etc.) and items to collect (potions, ammunition, keys, etc.).

Players can navigate from region/time to region/time and see their character change appearance and weapons to match the style of the region and time. 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 [Buckland 2002]; path-finding [Buckland 2002]; influence mapping [Buckland 2002]; and dead reckoning [Bernier 2000].

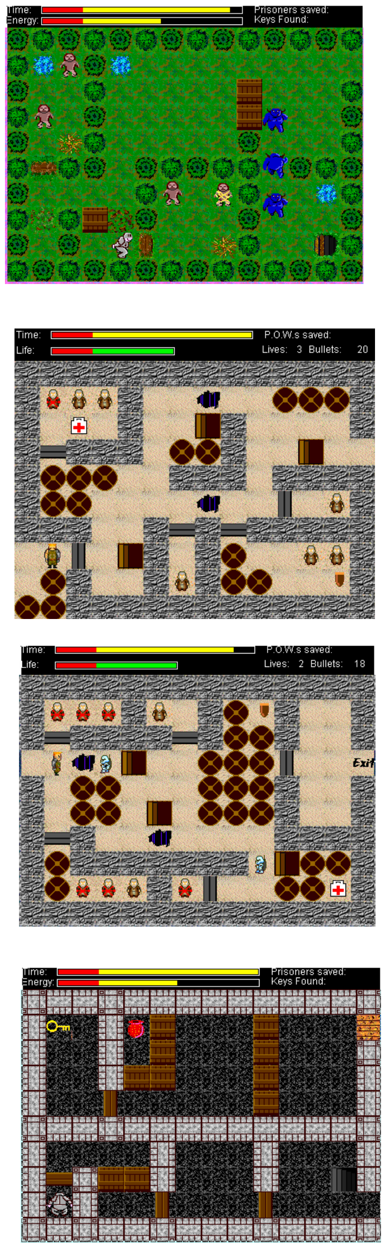


Fig. 3. Screen shots of different worlds and levels in "Time Prisoners."

The interactions (implemented as collision events) are numerous among players, players and NPCs, players and items, prisoners and guards, and all the characters with walls 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. When they first join the game their machines will either be attached as clients to an existing peer-group playing in the same region/level, or assigned the role of the main node (the server) of a new peer-group of future clients who will be playing in the same region/level.

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

4. A PEER-TO-PEER ARCHITECTURE AND PROTOCOL FOR MMOGs

In Section 2, we had mainly been concerned with examining the two extreme topologies available to MMOGs. However there are many hybrid techniques [Smed et al. 2002] that can overcome some of the issues outlined with the "pure" topologies.

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 player is connected to his peer-group, are done in P2P fashion.

The architecture still allows the developers to maintain direct control and authority over the players' account information, which is more secure when subscriptions and personal details are involved. A simple example is that of the "Login," and account management server kept away from the peer-to-peer network. The players' account information are managed in 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 is checked, he will be

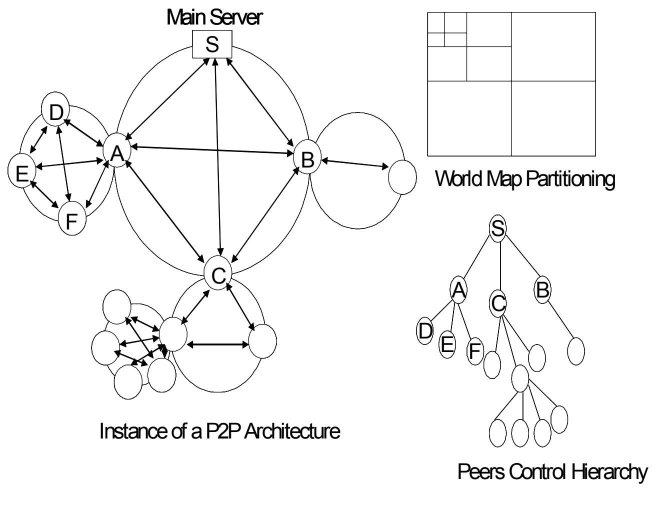


Fig. 4. Proposed architecture.

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 the peer groups and their relations in a dynamic way and to map the peer-groups.

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

Metamodel Architecture

The first step in our P2P architecture and protocol development is the design of an agent-based metamodel architecture. We have defined a high-level design based on the mobile agent model, which is suitable for the development of the MMOG and a simulation environment. Our metamodel 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 a state-chart to describe the behavior between agents within a host and between hosts--communication, for example. 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 multiagent organization, communication, and coordination as a P2P infrastructure, we also incorporate the concept of agent groups. Any agent within the agent

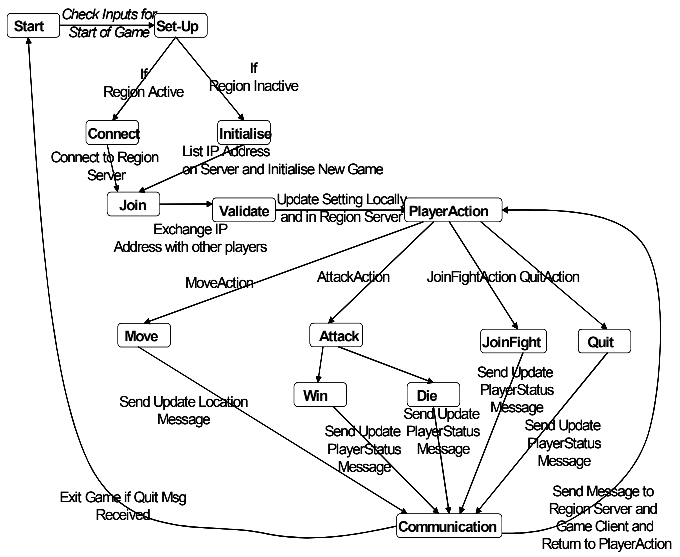

Start
Check Inputs for Start of Game
Set-Up
If Region Active
If Region Inactive
Connect
Initialise
Connect to Region Server
List IP Address on Server and Initialise New Game
Join
Validate
Update Setting Locally and in Region Server
PlayerAction
Exchange IP Address with other players
MoveAction
AttackAction
JoinFightAction
QuitAction
Move
Attack
JoinFight
Quit
Win
Die
Send Update Location Message
Send Update PlayerStatus Message
Send Update PlayerStatus Message
Send Update PlayerStatus Message
Send Update PlayerStatus Message
Exit Game if Quit Msg Received
Communication
Send Message to Region Server and Game Client and Return to PlayerAction

Fig 5. Agent communication level.

group may perform a multicast or a 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.

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.

5. HIGH-LEVEL DESIGN OF AGENT 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 a 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, each user, to join the virtual world, uses a client to connect to the region server that represents the particular region. Region servers notify the central database regarding 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.

System-Level Design

To start, we need to consider the system as a whole. Thus, we start by working on the system-level design. In Figure 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 bidirectional system links, which means that the hosts can communicate in both directions. The number and two variables (n, m) describe the

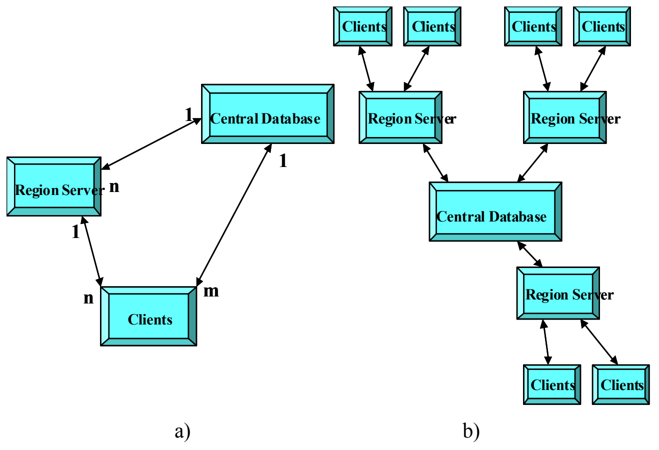
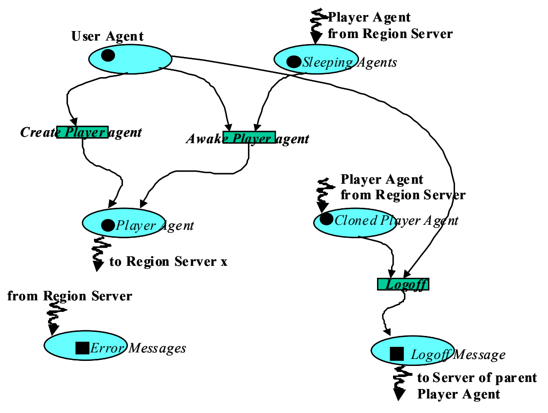
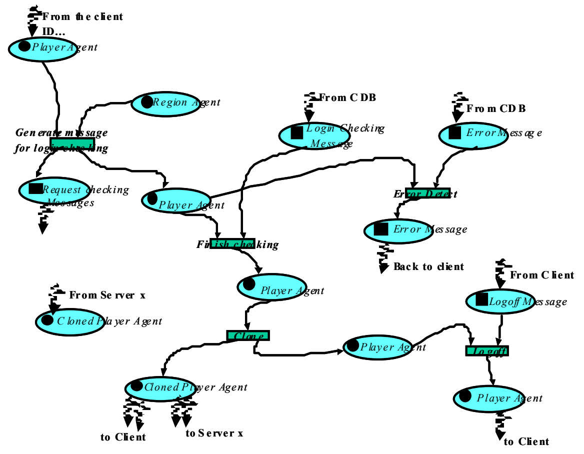


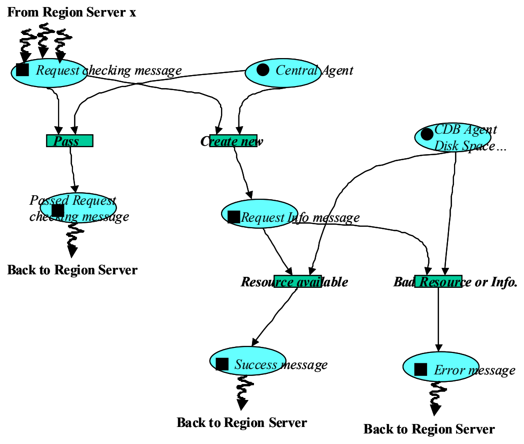
Fig. 6.. System high-level design of MMOG.



a) Client



b) Region Server



c) Central Database Server

Fig. 7. Host-level design of MMOG.

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. Figure 7(b) is an expanded view of the system-level design, which may be more helpful in visualizing the physical configuration of the system.

Host-Level Design

With the system-level view complete, we can 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 start with the host-level design of the client host.

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 a message to the region server upon logoff. We can immediately identify that the user agent will be static, since it will only process jobs that are 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 the player agent, and logoff. Figure 7(a) shows the relationships among the user agents, 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; the arcs incoming or outgoing from a place 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 the player agents, there is a migration path to a region server with id x. The woken-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.

Central Database Server

Let us move our point of view to the central database server. In Figure 7(c), as we can see, there are lot migrations coming 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 Figure 7(b), near the top right-hand where the return messages are received. If the incoming message is an error message, the error message is forwarded back to the 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 Figure 7(a), we can see that 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; to be followed with the message to Figure 7(b), where the logoff message comes in. The player agent now migrates back to the client. In Figure 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.

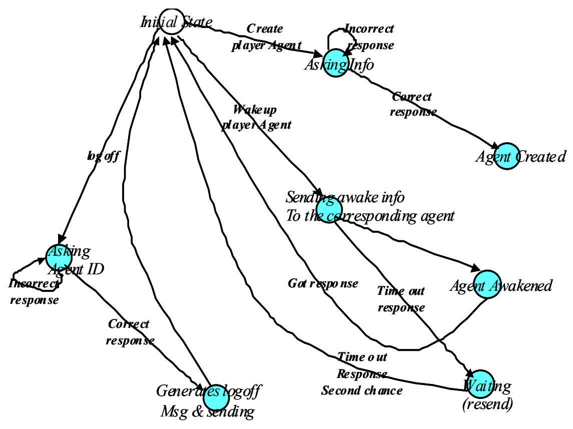


Fig. 8. User agent.

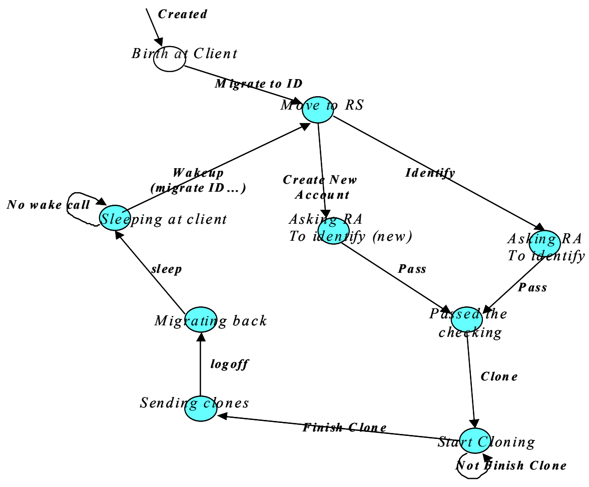


Fig. 9. Player agent

Agent-level design allows us to understand the behavior of individual agents. Let us look at Figures 8 to 12. Figure 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 the create information. When the correct response is received, 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 that describes interacting autonomous agents that reside among a set of hosts.

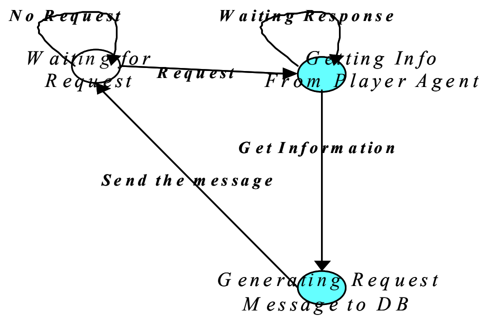


Fig. 10. Region agent

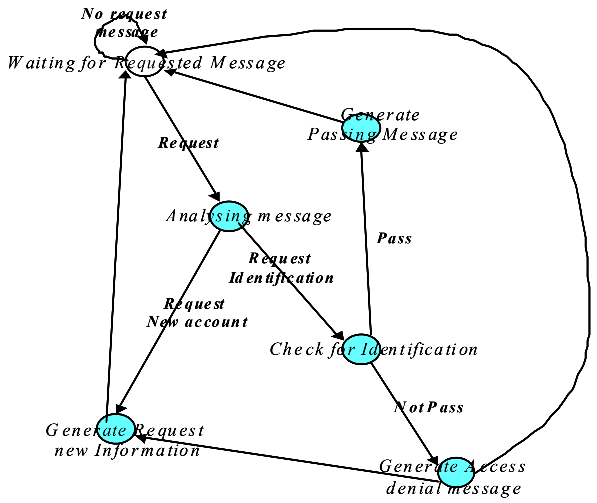


Fig. 11. Central agent.

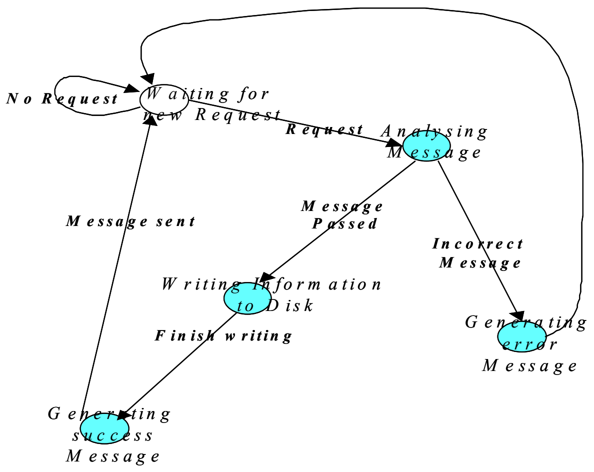


Fig. 12. CDB disk space agent.

The metamodel 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.

P2P Protocol for Joining/Leaving

In terms of building the architecture, the protocols for joining and leaving are the most important. Below we briefly present 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) which maintains the full game, the players' database, the current game data for existing players, etc.
- If a new player joins the game, he or she will be connected to the world server and assigned the role of new region server for the region the new player will play on. He or she 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 other existing region servers that control the other regions.
- If a player moves to another region, we have two situations that might occur. First, if the region is already controlled by another server region, the new player will join the peer group associated with this server-region. Second, 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 again have two cases. If the player is a client in his or her region's peer group, the leaving player will simply be 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 other region servers, they will be informed of the leaving region server's disconnection, and 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 of 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.

6. DISCUSSION

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

Scalability

One of the common issues in peer-to-peer networks is their scalability. We have designed our topology with the ability to organize itself efficiently, to reduce the inherent scalability problems. Most basic to this organization is the creation of *supernodes*. The network can identify which nodes would 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.

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 that is how it works.

Network Efficiency

As in the server/client architecture, several methods can be used to make a MMOG use its network resources more efficiently. Most of these techniques can equally be applied to a MMOG running on a peer-to-peer topology. Dead reckoning [Bernier 2000] algorithms greatly reduce the amount of object position data that has to be sent over the network. Instead of traditional unicasting, multicasting [Francis et al. 2001] is used, thereby reducing the load of network traffic.

Data Storage

MMOGs with a client/server architecture have a single huge "game state" [Bosser 2004], which is the game world's data usually held exclusively by the server. *This data is*

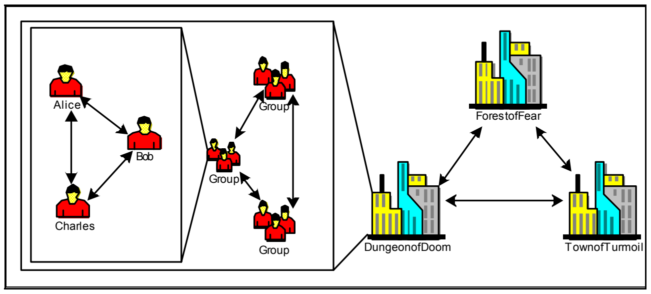


Fig 13. Supernode hierarchy in P2P MMOG.

extremely important to the game and to access; changes to the data 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 the P2P network must store a large amount of redundant (replicated) data, so that the game can still function with a minimum number of players.

To store this much data among a collection of nodes that are hierarchically organized, where information could be required by any node, requires careful design [Morse 1996].

Based on the Freenet project [Freenet Project], we believe there is a solution that can be adapted to store data this way. 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 that it 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.

Policing

In a peer-to-peer game of any sort, it is important for each node not to trust any other node. Yet, 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 have 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.

7. 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 14 shows part of the game logic and communication system's class hierarchies.

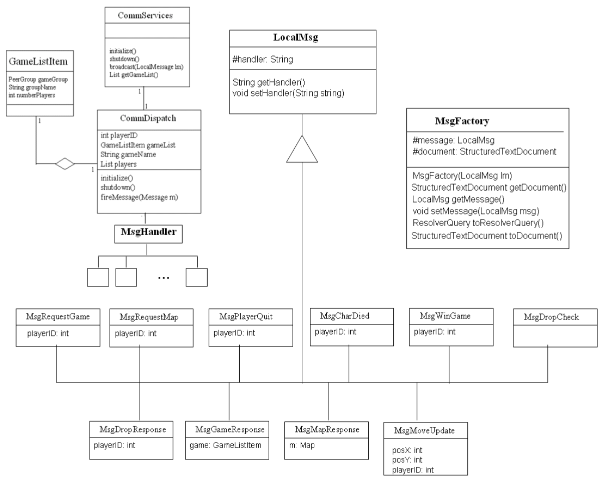


Fig 14. Class architecture.

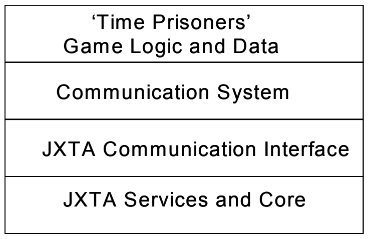


Fig. 15. Software architecture.

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.

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 passing information between game peers, such as requesting information about games and updating player information.

The communication subsystem provides the following services:

- start a multiplayer session game, given the map/region/donjon that 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 that are XML documents, or binary, or both (binary embedded in XML). They will be constructed by using the player state context and then sent out by the communication system.

8. CONCLUSIONS AND FUTURE WORK

Massively multiplayer online games have rapidly grown in popularity, and have proven that there is a big future in online gaming. However, the subscription price model used by most MMOGs is greatly slowing the uptake in new subscribers. Hence ways need to be developed to reduce the overhead in server upkeep costs and to pass savings on to the customer, either with money or in enhanced game-play.

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

MMOGs, due to their distributed and incremental structure, are natural applications for P2P overlays. Games are different from previous P2P applications with a 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 the usual infrastructure which (i.e. the protocol) relies on a set of dedicated servers that agree to share the workload. This protocol may also be extended to mobile networks [Joseph et al. 2002] where a peer-group manager is a routing agent responsible for clients in his or her group. The joining and leaving protocol would mostly remain 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 on making a viable distributed game over a pure P2P overlay

Ultimately, the 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 one hundred thousand 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.

BIBLIOGRAPHY

BAUER, D., ROONEY, S., AND SCATTON, P. 2002. Network infrastructure for massively distributed games.. In *Proceedings of NetGames 2002 Conference* (Braunschweig, Germany, April 2002), 3-9.

BERNIER, Y. W. 2000. Latency compensation techniques methods in client/server in-game protocol design and optimization. In *Proceedings of the Game Developers Conference* (March 2000).

BOSSER, A.-G. 2004. Massively multi-player games: Matching game design with technical design. In *Proceedings of the ACM SIGCHI Advanced Computer Entertainment Conference* (ACE 2004). NUS, Singapore.

BUCKLAND, M. 2002. *AI Techniques for Game Programming*. Premier Press, 2002.

EVE. The Second Genesis © CCP. http://www.ccpgames.com/

EVERQUEST © Sony Online Entertainment http://soe.sony.com/

FERRETTI, S. AND ROCCETTI, M. 2004. A novel obsolescence-based approach to event delivery synchronisation in multplayer games. *Int. J. Intelligent Games and Simulation 1 3,* 1 (2004), 7-19.

FRANCIS, P., HANDLEY, M., KARP, R., RATNASAMY, S., AND SHENKER, S. 2001. A scalable content-addressable network. In *SIGCOMM '01* (San Diego, CA, Aug. 27-31, 2001).

FREENET PROJECT. http://www.freenetproject.org.

GOLLE, P. AND MIRONOV, I. 2001. Uncheatable distributed computations. In *Lecture Notes in Computer Science 2020* (2001).

GNUTELLA PROTOCOL DEVELOPMENT. http://rfc-gnutella.sourceforge.net/

HALFLIFE © Sierra. http://games.sierra.com/games/half-life/

HONG, E., PACK, S., CHOI, Y., PARK, I., KIM, J.-S., AND KO, D. 2002. Game transport protocol: Transport protocol for efficient transmission of game event data. In *Proceedings of the JCCI Conference* (Jeju, Korea, April 2002).

JOSEPH, A., KUBIATOWICZ, J., AND ZHAO, B. 2002. Supporting rapid mobility via locality in an overlay network. Technical Report CSD-02-1216. Univ. of California, Berkeley.

JXTA. www.jxta.org/project/www/background.html.

KANT, K., IYER, R., AND TEWARI, V. 2002. A framework for classifying peer-to-peer technologies. In *Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid*. (Berlin, May 21-26, 2002). KAZA. http://www.kazaa.com/us/index.htm

KNUTSSON, B., LU, H., XU, W., AND HOPKINS, B. 2004. Peer-to-peer support for massively multiplayer games. In *Proceedings of the INFOCOM 2004 Conference* (Hong Kong, March 2004).

Received January 2005; accepted March 2005

LINEAGE © Ncsoft. http://www.lineage.com/nci

MANIATIS, P., ROUSSOPOULOS, M., GIULI, T., ROSENTHAL, D. S. H., BAKER, M., AND MULIADI, Y. 2003. Preserving peer replicas by rate-limited sampled voting.. In *Proceedings of the 2003 SOSP Conference*.

MORPHEUS. www.morpheus.com/

MORSE, K. L. 1996. Interest management in large scale distributed simulations. Tech, Rep. ICS-TR-96-27, Univ. of California, Irvine, 1996.

PLANETSIDE © Sony Online Entertainment http://planetside.station.sony.com/

SETI@HOME. http://setiathome.ssl.berkeley.edu/

SIMS ONLINE © Electronic Arts. http://www.eagames.com

SMED, K. AND HAKONEN. 2002. Aspects of networking in multiplayer computer games. *The Electronic Library 20,* 2 (2002), 87-97.

SMED, K. AND HAKONEN, 2003. Networking and multiplayer computer games--The so far. *Int. J. Intelligent Games & Simulation 2*, 2 (2003), 101-110.

STAR WARS GALAXIES © Sony Online Entertainment. http://starwarsgalaxies.station.sony.com/

WANG, T., WANG, C.-L., AND LAU, F. 2004. Grid-enabled multi-server network game architecture. In *Proceedings of the 3rd International Conference on Application and Development of Computer Games* (ADCOG 2004, City University of Hong Kong, April 26-27, 2004).

Received January 2005; accepted March 2005