

# **Middleware de Rede para Massive Multiplayer On-line Games**

**Curitiba  
2007**

**Guilherme Moschen  
Romulo De Lazzari**

**Middleware de Rede para Massive Multiplayer On-line Games**

Trabalho de Conclusão de Curso apresentado à UTFPR como requisito parcial para obtenção do título de Tecnólogo em Informática.

**Orientador:**

Prof. Dr. Cesar Augusto Tacla

**Curitiba  
2007**

**Moschen, Guilherme**  
**De Lazzari, Romulo**

Middleware de Rede para Massive Multiplayer On-line Games. X p.

Trabalho de Diplomação – Universidade Tecnológica Federal do Paraná. Curso de Tecnologia em Informática

1. Sistemas Distribuídos
2. Desenvolvimento de Jogos
3. Multiplayer On-line Games

## **Dedicatória**

Dedicamos este trabalho a todos que ingressaram no curso de Tecnologia em Informática, no verão de 2002, e para aqueles que contribuíram de certa forma para a nossa formação acadêmica profissional.

## **Agradecimentos**

Agradecemos ao nosso sócio Tiago Barão e aos outros amigos que passaram por nós nesses cinco anos: Fernando Grott de Carvalho, Fernando Kenji JacoJaco, Franciele da Cunha, Juliano Haus e Moysés Borges Furtado. Às nossas famílias e a todas as outras pessoas que nos ajudaram.

## Sumário

<b>1. INTRODUÇÃO.....</b>	<b>12</b>
<b>2. REVISÃO BIBLIOGRÁFICA.....</b>	<b>14</b>
<b>3. METODOLOGIA DE ANÁLISE E PROJETO .....</b>	<b>16</b>
<b>4. DESENVOLVIMENTO .....</b>	<b>19</b>
4.1. Descrição dos Casos de Uso.....	19
4.2. Diagrama de Classes .....	20
4.3. Diagrama de Estados.....	28
4.4. Diagrama de Componentes.....	28
4.5. Arquitetura do Middleware.....	28
4.6. Tecnologias Utilizadas no Servidor.....	29
4.7. Tecnologias Utilizadas no Cliente .....	29
4.8. Descrição do Protocolo de Comunicação .....	30
<b>5. IMPLANTAÇÃO, TESTES E RESULTADOS.....</b>	<b>33</b>
5.1. Implantação .....	33
5.2. Configuração Utilizada .....	35
5.3. Testes.....	35
<b>6. CONCLUSÕES.....</b>	<b>42</b>
6.1. Resultados Alcançados.....	42
6.2. Dificuldades.....	42
6.3. Trabalhos Futuros.....	43
<b>7. REFERÊNCIAS .....</b>	<b>44</b>
<b>ANEXO I.....</b>	<b>47</b>

**ANEXO II..... 51**

**ANEXO III.....57**

## Lista de Figuras

<b>Figura 1.</b> Diagrama de Componentes.....	28
<b>Figura 2.</b> Diagrama de Representação do Pacote.....	31
<b>Figura 3.</b> Tela do Jogo Teste.....	34
<b>Figura 4.</b> Números de Conexões Bem-Sucedidas.....	36
<b>Figura 5.</b> Mensagens Cliente-Servidor Transmitidas com Sucesso .....	37
<b>Figura 6.</b> Mensagens do Servidor para os Clientes .....	37
<b>Figura 7.</b> Compactação Média.....	38
<b>Figura 8.</b> Mensagens Criptografadas e Descritografadas com Sucesso ...	39
<b>Figura 9.</b> Mensagens Perdidas e Reenviadas .....	40

## Lista de Tabelas

<b>Tabela 1.</b> Descrição das Classes.....	21
---	----

## Resumo

O desafio de desenvolver jogos eletrônicos é maior a cada dia e ferramentas que facilitem o desenvolvimento são bem-vindas. O objetivo deste projeto é desenvolver uma biblioteca de funções (*middleware*) que facilite o desenvolvimento das funções de comunicação de um *Massively Multiplayer Game*. Essa biblioteca, desenvolvida para a arquitetura cliente-servidor, contém as seguintes funcionalidades: compactação, criptografia, conexão confiável e não confiável, transmissão de dados e controle de usuários conectados. O *middleware* desenvolvido foi testado em um ambiente não real, sendo analisados os tempos de resposta do servidor e o número máximo de conexões suportadas. No estágio atual, o *middleware* encontra-se na versão beta.

Palavras-chaves: Sistemas distribuídos, desenvolvimento de jogos, *multiplayer on-line games*.

## **Abstract**

The challenge to develop electronic games is bigger each day and tools that facilitate the development are coming well. The objective of this project is to develop a library of functions (middleware) that it facilitates the development of the functions of communication of a Massively Multiplayer Game. This library, developed for the architecture client-server, contains the following functionalities: compacting, cryptographic, trustworthy and not trustworthy connection, transmission of data and control of connected users. The developed middleware was tested in a not real environment being analyzed the time of response of the server and the maximum number of connections supported. In the current period, the middleware is in the beta version.

**Key Words:** Distributed Systems, Game Developing, Multiplayer On-line Games.

## 1. Introdução

O mercado de jogos está atualmente em expansão e os motivos desse aumento são, em grande parte, os *Massively Multiplayer Games*. Estes são jogos em que milhares de pessoas podem se conectar a partir de qualquer lugar do mundo, necessitando somente de uma conexão com a internet e um computador que possa executar o jogo (Cecin, 2003).

Os jogos *Massively Multiplayer Games* funcionam como um sistema distribuído, sendo que cada uma de suas partes (considerando cada jogador como uma parte) são autenticadas por um servidor central que recebe todas as mensagens enviadas pelos jogadores, as valida e as reenvia para todos os outros jogadores, atualizando, assim, o “mundo” do jogo. Como essas regras são parecidas para a maioria dos *Massively Multiplayer Games*, notou-se a necessidade de construir uma abstração dessa camada destinada aos projetistas de jogos. Com essa camada já pronta, os projetistas do jogo poderão focar nas outras camadas que compõe um *Massively Multiplayer Game*, não mais sendo necessário desenvolver todas as funções de controle e coordenação existentes.

Já existem no mercado alguns *middlewares* de Rede que abrangem também o desenvolvimento de *Massively Multiplayer Games*, mas não foi encontrado nenhum que funcione especificamente para esse tipo de jogo. Todos os encontrados (ver Anexo III) são genéricos e demandam personalização por parte dos desenvolvedores.

Para que esse trabalho de personalização não seja mais necessário, propõem-se a criação de um *middleware* de Rede específico para *Massively Multiplayer On-line Role Play Games (MORPGs)*, uma biblioteca de funções de rede e um protocolo padrão para a distribuição e controle das mensagens trocadas entre as partes do sistema distribuído.

Essa biblioteca de funções deve permitir:

- compactação e descompactação de dados para que as mensagens de tamanho superior ao definido pelo protocolo desenvolvido possam ser transmitidas;

- criptografia de mensagens para transferir dados de maneira segura. Por exemplo, nas transações de login e outras em que seja importante a segurança dos dados;
- conexão e transmissão de dados;
- controle de conexão e aviso de desconexão;
- definição de um protocolo padrão para as mensagens trocadas entre os membros do sistema.

Estima-se que para um jogo típico *Massively Multiplayer On-line Role Play Games*, contando com um número médio de jogadores, o *middleware* (executando com uma única instância) deve aceitar 50 conexões simultâneas com tempo de resposta variando entre 250 milissegundos e 1 segundo, considerando que essas conexões são executadas na internet.

Com as funcionalidades do *middleware*, não só o tempo de desenvolvimento deve diminuir, como também o tempo de testes, considerando que o *middleware* foi testado e validado previamente. O principal objetivo no desenvolvimento do *middleware* é que se diminua o trabalho repetitivo dos desenvolvedores, facilitando o desenvolvimento e liberando-os para se concentrarem em outras partes do jogo, partes essas que são menos repetitivas, como a criação de histórias e de personagens.

Esse trabalho está dividido em 7 capítulos. O capítulo 1 contém a introdução e os motivos para a realização do trabalho. No capítulo 2, apresenta-se uma revisão bibliográfica e os conceitos que serão utilizados ao longo do trabalho. No capítulo 3, mostra-se o processo de engenharia de software utilizado, concentrando-se nas etapas de análise e de gestão do projeto. No capítulo 4 está a descrição do *middleware*, a descrição dos requisitos do sistema e os diagramas de casos de uso. O capítulo 5 dá uma visão geral do desenvolvimento do sistema com as descrições, os diagramas de desenvolvimento, método de implantação e o ambiente de testes. No capítulo 6, são descritos os resultados alcançados, as dificuldades e os trabalhos futuros. O capítulo 7 contém a referência bibliográfica.

## 2. Revisão Bibliográfica

Existem vários estilos de jogos de computador, entre eles podemos encontrar os jogos de Arcade, Lógica, “*Interactive Fiction*”, Simulações, Estratégia, Tiro em Primeira Pessoa (FPS), *Role Playing Games* (RPG), e ainda vários outros menos conhecidos (Perucia, 2005). Esses jogos aceitam um ou mais usuários, existindo várias classificações em relação à quantidade de usuários que um jogo pode suportar. A mais comum das classificações é:

- *Single Player*: um único jogador que interage sozinho com o jogo todo. Essa quantidade de jogadores funciona bem com a maioria dos tipos de jogos, pois em todos eles existe a possibilidade de se utilizar os outros personagens controlados pelo computador.
  
- *Multiplayer*: de 2 a 64 jogadores que podem estar localizados em uma rede local ou distribuídos pela internet. Esse tipo de jogo normalmente funciona bem com jogos de Estratégia, Tiro em Primeira Pessoa e em Simulações, pois são estilos favoráveis à disputa entre jogadores. O tamanho das mensagens enviadas é menor que nos outros. Esse tipo de jogo normalmente é definido como “Muitas pequenas mensagens” (Bauer, 2005).
  
- *Massively Multiplayers*: milhares de jogadores conectados entre si, utilizando a internet como meio de comunicação. O estilo de jogo que mais tem assumido os moldes dos *Massively Multiplayers* são os RPGs (Cyber Creations Inc., 2006). O RPG é um dos jogos mais comuns nos *Massively Multiplayers* pois permite uma maior liberdade de ação para o jogador (que pode fazer praticamente tudo), uma maior personalização do personagem que vai representar o jogador e também não necessita de comunicação em tempo real (o atraso das mensagens nesses jogos pode ser maior, já que as decisões de “vitórias-derrotas” são baseadas em estatísticas dos personagens, não em perícia ou em velocidade de reação).

Para facilitar o desenvolvimento dos jogos de computador, foram criados vários *middlewares* que abrangem muitos dos aspectos dos jogos. Segundo (RNP, 2005), *middlewares* são softwares que conectam vários componentes de software ou várias aplicações diferentes. Segundo (COULOURIS, 2001), *middleware* é uma camada de software que prevê uma abstração na programação.

Uma lista de *middlewares* para jogos pode ser encontrada em (Ogre 3D, 2007), sendo os tipos mais relevantes:

- Física: simula as leis da Física (diferentes ou não do mundo real) no mundo do jogo. Controla a interação física entre os objetos.
- Inteligência Artificial: resolve os problemas como busca por menor caminho e a interação social entre os objetos.
- Gráfica: controla a criação dos componentes gráficos do jogo.
- Áudio: controla a parte sonora do jogo, efeitos sonoros e músicas.
- Entrada: controla os dispositivos de entrada do jogo, tais como, teclado, *mouse*, e *joystick*.
- Rede: cria a parte *Multiplayer* (ou *Massively Multiplayer*) do jogo. Trata do desempenho do jogo em rede, segurança, concorrência e vários outros problemas decorrentes de aplicações distribuídas.

Qualquer um dos estilos de jogos, independente do número de jogadores, pode ser desenvolvido a partir desses *middlewares*. Os *middlewares* não são genéricos para todos os tipos de jogos, mas existem *middlewares* desenvolvidos para praticamente todos os estilos de jogos existentes (Anexo III).

Os *middlewares* são desenvolvidos comercialmente, como os listados no Anexo III, por empresas que têm como interesse vender os próprios *middlewares* como ferramentas para os desenvolvedores. Também existem vários projetos de *middleware* abrangidos pela GNU, e ainda existem os *middlewares* desenvolvidos e vendidos pelos próprios fabricantes de *hardware*.

### 3. Metodologia de Análise e Projeto

O método de desenvolvimento primeiramente planejado para a criação do *middleware* foi o Método da Espiral (Boehm, 1988; Kruchten, 2000). Mas, por motivos de pouca familiaridade e dificuldade de adaptação a esta metodologia, foi escolhido o Método *Extreme Programming* (XP) (Beck & Andrés, 2005; Teles 2005), já trabalhado pela equipe. Esse método prevê um desenvolvimento ágil e uma apresentação de versões pequenas e incrementais para o cliente. Os quatro valores fundamentais do XP são: comunicação, simplicidade, feedback e coragem. Assim, ele possui como princípios básicos: feedback rápido, assumir simplicidade, mudanças incrementais, abraçar mudanças e trabalho de qualidade (Teles, 2005).

Nesta metodologia existe um foco muito grande no escopo do projeto. Sempre são definidas reuniões semanais com o cliente para nova definição do escopo e são redefinidas as prioridades do sistema, assim as funcionalidades menos valiosas podem ser adiadas ou, até mesmo, eliminadas do projeto.

O controle de qualidade do sistema também é considerado muito importante, pois sempre são entregues pequenas versões ao cliente, com apenas algumas funcionalidades do sistema, para que ele aceite e/ou sugira modificações ao projeto.

As práticas que foram utilizadas no desenvolvimento do *middleware* são (Teles, 2005):

- Jogo de Planejamento (*planning games*): o desenvolvimento é feito em iterações semanais. No início da semana, desenvolvedores e cliente reúnem-se para priorizar as funcionalidades. Essa reunião recebe o nome de Jogo do Planejamento;
- Pequenas Versões (*Small Releases*): a liberação de pequenas versões funcionais do projeto auxilia o processo de aceitação por parte do cliente, que já pode testar uma parte do sistema;

- Projeto Simples (*Simple Design*): simplicidade é um princípio do XP. Projeto Simples significa dizer que, caso o cliente tenha pedido que na primeira versão apenas o usuário "teste" possa entrar no sistema com a senha "123" e assim ter acesso a todo o sistema, você vai fazer o código exato para que esta funcionalidade seja implementada, sem se preocupar com sistemas de autenticação e restrições de acesso;
- Posse Coletiva (*Collective Ownership*): o código fonte não tem dono e ninguém precisa solicitar permissão para modificá-lo. O objetivo é tornar a equipe conhecedora de todas as partes do sistema.

Utilizando-se dessas práticas, foi definido que seriam feitas reuniões semanais entre os próprios desenvolvedores (já que não havia um cliente) para novas definições de escopo do sistema e refinamentos das funcionalidades já implementadas. Essas reuniões eram feitas todas as segundas-feiras, exceto nas semanas em que a tarefa era estipulada para ser terminada em mais de uma semana. Nessas reuniões eram definidas a próxima funcionalidade a ser implementada, a melhor forma de implementá-la, definia-se também um escopo para o seu funcionamento e estipulava-se um tempo para desenvolvimento dessa nova funcionalidade. Também se testava e aprovava, ou não, a funcionalidade do sistema que estava para ser entregue, definida na reunião anterior.

O processo de trabalho foi dividido da seguinte maneira: o participante Romulo De Lazzari ficou responsável pelo desenvolvimento das funções relacionadas ao servidor e Guilherme Moschen ficou responsável pelo desenvolvimento das funções relacionadas ao cliente do *middleware*. A tarefa de desenvolvimento do jogo para testar a funcionalidade do *middleware* e os testes ficaram divididos entre os dois participantes. Eventuais correções foram realizadas por cada um na sua respectiva responsabilidade.

Para facilitar a divisão de tarefas e a manutenção dos prazos, foi utilizado o software NetOffice (NetOffice, 2005), que trabalha sobre licença GNU,

desenvolvido em linguagem PHP para ser utilizado em projetos pequenos e médios. Esse software permite definir tarefas e distribuí-las entre os membros da equipe.

## 4. Desenvolvimento

### 4.1. Descrição dos Casos de Uso

O Diagrama de Casos de Uso está no Anexo I e a descrição dos fluxos de cada Caso de Uso está esquematizada no Anexo II. Abaixo a descrição de cada um dos Casos de Uso do sistema:

- Enviar mensagem: envia uma mensagem completa do cliente ao servidor, ou do servidor ao cliente. A mensagem pode ser quebrada em um ou mais pacotes para ser enviada;
- Enviar Mensagem com Comunicação Confiável: comunicação confiável é aquela em que, caso aconteça uma perda de pacote, são adotadas medidas de controle e reenvio de pacotes perdidos, para que nenhuma informação seja perdida;
- Enviar Mensagem com Comunicação não Confiável: comunicação não confiável se define como o envio de pacotes sem nenhum controle de queda ou perda de pacotes;
- Atualização de Arquivos: Implementa a atualização de arquivos no cliente com base nos arquivos escolhidos pelo servidor. Método utilizado para uma possível atualização do jogo (caso de uso não implementado).
- Compactar Mensagem: utilização de um algoritmo de compactação dos dados do jogo;
- Criptografar Mensagem: utilização do algoritmo AES 128 bits para criptografar a mensagem;
- Receber Mensagem: caso de uso que define o recebimento de uma mensagem completa, a junção de todos os pacotes que formam a mensagem, a descompactação e a descriptografia da mensagem;
- Criar Pacote: caso de uso que define um pacote do *middleware* e monta o protocolo do *middleware*;

- Iniciar Servidor: define a preparação do servidor para entrar em modo de espera de mensagens dos usuários;
- Gerenciar Usuários: caso de uso que define as funções de gerenciamento da lista de usuários que estão conectados ao servidor;
- Consultar Usuário: caso de uso que define a pesquisa de usuários na lista de usuários;
- Adicionar Usuário: caso de uso que define a inclusão de um usuário na lista de usuários;
- Excluir Usuário: caso de uso que define a exclusão de um usuário na lista de usuários;
- Abrir Conexão: caso de uso que define a conexão inicial de um usuário com o servidor;
- Fechar Conexão: caso de uso que define o término da conexão de um usuário com o servidor.

## 4.2. Diagrama de Classes

Para efetuar o levantamento das classes foram avaliados, primeiramente, os *middlewares* existentes e artigos relacionados, como (Bauer, 2005) e (Cecin, 2003), com o tema. Dentre a análise proposta pela equipe, as classes levantadas são: *ClientSocketUDP*, *Compact*, *Cryptography*, *ListData*, *ListDataCtrl*, *Packet*, *PacketCtrl*, *PacketList*, *ServerSocketUDP*, *SocketUDP* e o Enumerador *typePacket*. No diagrama proposto, ver Anexo II, não são exibidos os métodos *get* e *set* dos atributos das classes, pois esses métodos são padrões a todas as classes e ao conceito de orientação a objeto. Usamos os nomes em inglês por ser um padrão estabelecido pelo uso. Dentre todos os *middlewares* pesquisados, não foi encontrado nenhum que utilize outra língua como nomenclatura.

<b>Classe</b>	SocketUDP.
<b>Descrição</b>	Classe que define as funções básicas da conexão UDP, envio e recebimento de pacotes.
<b>Atributos</b>	protected socketSender : int – Valor do Objeto Socket que será aberto; protected socketReciver : sockaddr_in – Estrutura de dados do Socket. Armazena o Endereço IP do socket, a porta, o tipo de endereço e o tipo de protocolo.
<b>Métodos</b>	public SocketUDP () – Construtora da classe; public send (pdata : Byte, phost : string, pport : int) : void – Método que envia uma mensagem para determinado host e porta; public send (ppacket : Packet, phost : string, pport : int) : void – Método que envia um pacote para determinado host e porta; public receive () : Byte – Método que espera o recebimento de um (ou vários) pacote e retorna a mensagem recebida.

<b>Classe</b>	ClientSocketUDP.
<b>Descrição</b>	Classe derivada da Classe SocketUDP que representa o cliente que utiliza o <i>Middleware</i> MMO.
<b>Atributos</b>	private dataConnection : int – Armazena os dados recebidos na conexão; public socketSendAddress : sockaddr_in – Estrutura que contém o Endereço IP com o qual o cliente irá se comunicar; private servPort : int – Porta do servidor; private servIP : string – IP do servidor.
<b>Métodos</b>	public ClientSocketUDP () – Construtora da classe;

	<p>public ClientSocketUDP (pIPServer: string, pportServer : int) – Construtora que seta os dados do servidor, IP e porta;</p> <p>public send (pdata : Byte*, compact : Boolean, cryptogrph) : void – Método que envia uma mensagem ao servidor, criptografada e/ou compactada;</p> <p>public receive (pport : int) : Byte* – Espera mensagem do servidor.</p>
--	---

<b>Classe</b>	ServerSocketUDP.
<b>Descrição</b>	Classe que define as funções específicas de conexão para servidor.
<b>Atributos</b>	private list : ListDataCtrl – A lista de usuários conectados no servidor; private port : int – A porta do servidor.
<b>Métodos</b>	public ServerSocketUDP () – Construtora da classe; public SocketUDP (pport : int) : void – Construtora que recebe a porta do Socket que o servidor usará; public sendAll (pdata : Byte*) : void – Método que envia uma mensagem a todos os usuários; public receive () : Byte – Método que fica esperando uma mensagem do cliente.

<b>Classe</b>	ServerSocketUDP.
<b>Descrição</b>	Classe que define as funções específicas de conexão para servidor.
<b>Atributos</b>	private list : ListDataCtrl – A lista de usuários conectados no servidor.
<b>Métodos</b>	public ServerSocketUDP () – Construtora da classe; public sendAll (pdata : Byte*) : void – Método que envia uma mensagem a todos os usuários;

	public receive () : Byte – Método que fica esperando uma mensagem do cliente.
--	---

<b>Classe</b>	ListDataCtrl.
<b>Descrição</b>	Classe que controla a lista de usuário conectados no servidor.
<b>Atributos</b>	private list : vector<ListData> – Objeto da lista de armazenamento que será controlado pela classe.
<b>Métodos</b>	public ListCtrl() – Construtora da classe; public add (pip : string, port : int) : int – Adiciona um usuário, IP e porta, à lista e retorna o identificador gerado deste usuário; public remove (pid : int) : void – Remove um usuário da lista através do identificador do usuário; public clear () : void – Remove todos os usuários da lista; public search (pid : int) : ListData – Retorna um usuário da lista a partir do identificador do usuário.

<b>Classe</b>	ListData.
<b>Descrição</b>	Classe entidade que define um usuário.
<b>Atributos</b>	private id : int – Identificador único do usuário; private ip : string – O IP do usuário; private port : int – Porta do usuário.
<b>Métodos</b>	public ListData () – Construtora da classe.

<b>Classe</b>	Compact.
<b>Descrição</b>	Classe que realiza a compressão dos dados a serem enviados e a descompressão dos dados recebidos.
<b>Atributos</b>	private err : int – Atributo que captura o código do erro que pode ter ocorrido no processo de compressão ou de descompressão;

	<p>private dest : Byte* – Ponteiro para o local da memória onde os dados comprimidos ou descomprimidos serão armazenados;</p> <p>private source : Byte* – Ponteiro para o local da memória onde está a fonte dos dados a serem comprimidos ou descomprimidos;</p> <p>private destLen : uLong – Tamanho do buffer de destino dos dados;</p> <p>private sourceLen : uLong – Tamanho da fonte de dados.</p>
<b>Métodos</b>	<p>public Compact() – Construtora da classe;</p> <p>public compressData (pdata : Byte) : Byte – Método que comprime os dados recebidos como parâmetro e retorna os Bytes que resultaram da compressão dos dados;</p> <p>public uncompressData (pcompresseddata : Byte) : Byte – Método que descomprime uma cadeia de Bytes recebida como parâmetro e retorna a string original.</p>

<b>Classe</b>	Cryptography.
<b>Descrição</b>	Classe que criptografa dados com o algoritmo AES 128 bits.
<b>Atributos</b>	private key : string – Chave da criptografia.
<b>Métodos</b>	<p>public Cryptography (key : string) – Construtora que inicializa a classe que é a chave da criptografia;</p> <p>public encryptData (pdata : Byte) : Byte – Método que criptografa dados;</p> <p>public decryptData (pdata : Byte) : Byte – Método que descriptografa dados.</p>

<b>Classe</b>	PacketCtrl.
<b>Descrição</b>	Classe que controla as listas de pacotes de envio e recebimento de cada usuário. Cada usuário pode ter no

	máximo uma lista de pacotes.
<b>Atributos</b>	private list : vector<PacketList> – Lista de listas de pacotes dos usuários.
<b>Métodos</b>	<p>public PacketCtrl() – Construtora da classe;</p> <p>public add (ppacketlist : PacketList) : void – Método que adiciona uma lista de pacotes de um usuário;</p> <p>public add (pdata : Byte, pcryptography : boolean, compact : boolean, pidUser : int ) : void – Método que recebe uma mensagem, criptografa e/ou compacta, e quebra em vários pacotes para envio;</p> <p>public remove (pidUser : int) : void – Remove uma lista de pacotes através de um determinado usuário;</p> <p>public clear () : void – Remove todas as lista de pacotes dos usuários;</p> <p>public search (pidUser : int) : PacketList – Retorna uma lista de pacotes através o identificador do usuário;</p>

<b>Classe</b>	PacketList
<b>Descrição</b>	Classe que define uma lista de pacotes de um usuário.
<b>Atributos</b>	<p>private list : vector&lt;Packet&gt; – Lista de pacotes;</p> <p>private idUsuario : int – Identificador do usuário;</p>
<b>Métodos</b>	<p>public PacketList() – Construtora da classe.</p> <p>public add (ppacket : Packet) : void – Método que adiciona um pacote à lista;</p> <p>public remove (pidPacket : int) : void – Método que remove um pacote da lista a partir do identificador do pacote;</p> <p>public clear () : void – Método que remove todos os pacotes da lista;</p> <p>public search (pidPacket : int) : Packet – Método que, a partir do identificador do pacote, retorna um pacote.</p>

<b>Classe</b>	Packet.
<b>Descrição</b>	Classe que define um pacote utilizado para a transmissão de dados.
<b>Atributos</b>	<p>public const PACKET_SIZE : int = 256 – Tamanho, em bytes, do pacote;</p> <p>private dataSize : int – Tamanho do campo de dados do pacote. Utilizado para a conferência simples da perda de dados;</p> <p>private data : Byte* – Dados do pacote. Esses dados podem ser qualquer tipo de mensagem trocada durante o decorrer do jogo;</p> <p>private type : TypePacket – Tipo do pacote de dados. Permite a identificação do tipo de mensagem que o pacote carrega. Utilizado para facilitar a organização das prioridades de atendimento das mensagens;</p> <p>private idPacket : int – Identificador único do pacote, é um número seqüencial;</p> <p>private totalPacket : int – Quantidade de pacotes que a mensagem foi quebrada;</p> <p>private isCryptographied : boolean – Se os dados estão criptografados;</p> <p>private isCompacted : boolean – Se os dados estão compactados.</p>
<b>Métodos</b>	public Packet () : void – Construtora da classe.

<b>Enumerador</b>	<u>TypePacket.</u>
<b>Descrição</b>	Enumerador que define os tipos de pacotes que são transmitidos.
<b>Membros</b>	<p>TP_CONNECT_TRUST – Tipo que define que o pacote está sendo enviado através de uma conexão segura;</p> <p>TP_CONNECT_NOT_TRUST – Tipo que define que o</p>

	<p>pacote está sendo enviado através de uma conexão NÃO segura;</p> <p>TP_DESCONNECT – Tipo que define que o pacote representa o fechamento de conexão com o servidor;</p> <p>TP_ACK – Tipo de pacote usado para avisar a entidade emissora de pacote que o mesmo foi recebido com sucesso;</p> <p>TP_AUTHENTICATE – Tipo de pacote que é usado na autenticação dos usuários;</p>
--	---

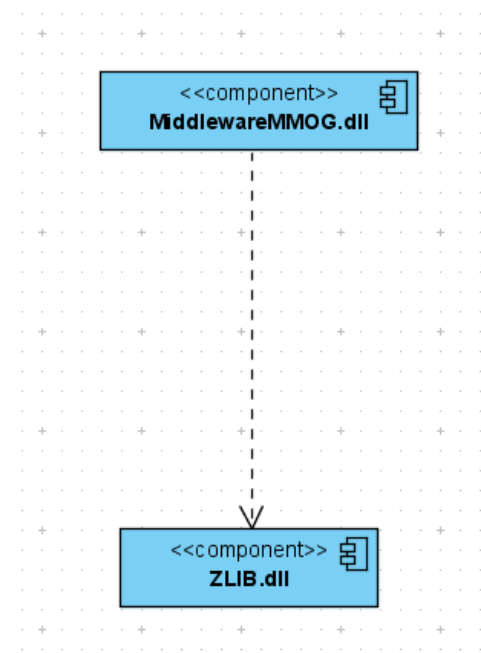
**Tabela 1** – Descrição das Classes.

### 4.3. Diagrama de Estados

Para melhor compreensão do projeto, a equipe desenvolveu dois Diagramas de Estados das classes mais importantes, SocketUDP e ListDataCtrl. Ver diagramas no Anexo II.

### 4.4. Diagrama de Componentes

No Diagrama de Componentes mostra apenas duas entidades: o *middleware* e ZLIB. A ZLIB é um componente que compacta e descompacta as mensagens do jogo.



**Figura 1** – Diagrama de Componentes.

### 4.5. Arquitetura do Middleware

Foi utilizada a arquitetura cliente-servidor, assim a maior parte do processamento fica no lado do servidor, perfeito para jogos com muitos clientes. Isso torna mais fácil a manutenção dos servidores que estarão em apenas um

lugar, não em vários (como na arquitetura *peer-to-peer*). O *middleware* nada mais é do que uma biblioteca de funções que ajudará o desenvolvedor de *MMORPGs* no que diz respeito à comunicação entre máquinas diferentes, “para realizar um objetivo em comum” (Tanenbaum, 2002).

#### **4.6. Tecnologias Utilizadas no Servidor**

No lado das funções de servidor do *middleware*, foi utilizado o sistema operacional Linux por ser um ambiente mais estável para funcionar como um servidor para várias máquinas e também pela sua licença de livre utilização, o que tiraria do desenvolvedor do jogo um custo relativamente alto. O sistema operacional Linux também é funcional em máquinas de configuração mais baixa, o que também diminui os custos do desenvolvimento. A distribuição utilizada é a Ubuntu (Canonical Ltd., 2007), por ser de configuração simples, pois Linux Ubuntu tem como premissa ser utilizável por qualquer pessoa, e também por sua fácil obtenção em qualquer lugar do mundo.

Para o desenvolvimento das funções de servidor, foi utilizada a linguagem de programação C++ por ser uma linguagem bastante utilizada no desenvolvimento de jogos, e a grande maioria dos desenvolvedores a conhece não criando nenhum tipo de dificuldade para a conversão para outra linguagem se esse for o caso específico do jogo que está sendo desenvolvido.

O protocolo de transporte utilizado na parte do servidor é o UDP, um protocolo de transmissão de datagramas não confiável. Este é um atributo necessário nos *Massively Multiplayer Games*, pois é primordial que se transmitam os dados com velocidade e a perda de alguns pacotes não vai afetar o jogo como um todo.

#### **4.7. Tecnologias Utilizadas no Cliente**

No lado do cliente do *middleware*, foi utilizado o sistema operacional Windows XP que, segundo (W3SCHOOLS, 2006), é utilizado por 76% dos

usuários da internet, que por consequência torna o Windows XP o (W3Schools, 2007) sistema operacional largamente utilizado entre os jogadores de *Massively Multiplayer Games*. Mas a biblioteca de funções do cliente foi desenvolvida de modo a ser genérica a todas as linguagens de programação. Para que essa independência de linguagem seja possível foi utilizada a tecnologia COM.

Component Object Model (COM) é um *framework* desenvolvido pela Microsoft para interação de componentes (Carnegie Mellon University, 2001). Através desta tecnologia é possível criar várias interfaces para a comunicação de entidades heterogênicas e encapsular todo o *middleware* em uma DLL.

A linguagem de programação utilizada para as funções de cliente também é o C++, por simples compatibilidade com o servidor, já que algumas funções básicas foram utilizadas nos dois lados (cliente e servidor) e elas não precisaram ser reescritas em outra linguagem.

O protocolo da camada de transporte no sentido cliente-servidor também foi definido como UDP, por motivos de desempenho.

#### **4.8. Descrição do Protocolo de Comunicação**

O protocolo utilizado foi desenvolvido devido às necessidades para os meios de controle dos pacotes (meio de transporte) do *middleware* (o protocolo está esquematizado na figura 2).

O pacote definido tem o tamanho de 256 bytes. Sendo eles divididos em 5 partes, 4 para o cabeçalho e uma para os dados, conforme ilustra a figura 2.

Campo	Tamanho em Bytes
Tipo de Mensagem	1
Tamanho da Mensagem do Jogo	4
Se a Mensagem está Compactada	1
Se a Mensagem está Criptografada	1
Número Seqüencial do Pacote	4
Número de Pacotes	4
Dados	241

**Figura 2** – Diagrama de Representação do Pacote.

O campo Tipo de Mensagem é utilizado para representar o tipo da mensagem entre 5 disponíveis:

- TP\_CONNECT\_TRUST (mensagem de conexão confiável);
- TP\_CONNECT\_NOT\_TRUST (mensagem de conexão não confiável);
- TP\_DISCONNECT (mensagem de desconexão);
- TP\_ACK (mensagem correspondente ao recebimento de um pacote);
- TP\_AUTHENTICATE (mensagem de autenticação de usuário). Esse campo ocupa o tamanho de 1 byte na mensagem.

- Tamanho da Mensagem:

Esse campo contém o tamanho do campo data para que ele seja lido até o seu último caractere, caso seja menor que o seu tamanho máximo. Esse campo ocupa 4 bytes na mensagem.

- Parte da Mensagem, Número Seqüencial:

Caso uma mensagem seja maior que o tamanho máximo, 241 bytes para dados, ela é quebrada em várias mensagens que são enviadas em ordem por um

número seqüencial, que é o número contido nesse campo. Ocupa 4 bytes da mensagem.

- Número Seqüencial do Pacote:

É o identificador único do pacote, cada entidade tem o seu número seqüencial.

- Número de Pacotes:

Quantidades de pacotes que compõe a mensagem.

- Dados:

Os dados a serem enviados pelo usuário do *middleware*. Campo com tamanho máximo de 241 bytes.

## 5. Implantação, Testes e Resultados

### 5.1. Implantação

A implantação do *middleware* foi realizada em um jogo teste desenvolvido pela própria equipe especialmente para os testes. Esse jogo utiliza todas as funcionalidades que foram implementadas no *middleware*.

As seguintes métricas foram utilizadas para analisar o desempenho:

- Tempo de Resposta – tempo entre a saída da mensagem do cliente e a resposta do servidor referente a essa mensagem. Contando o tempo em que a mensagem está no meio e o tempo de processamento no servidor. (Federal Standard, 1996);
- Número de Mensagens Perdidas;
- Requisições por Segundo no Servidor – processar um número de 100 requisições por segundo no servidor e monitorar o seu desempenho.

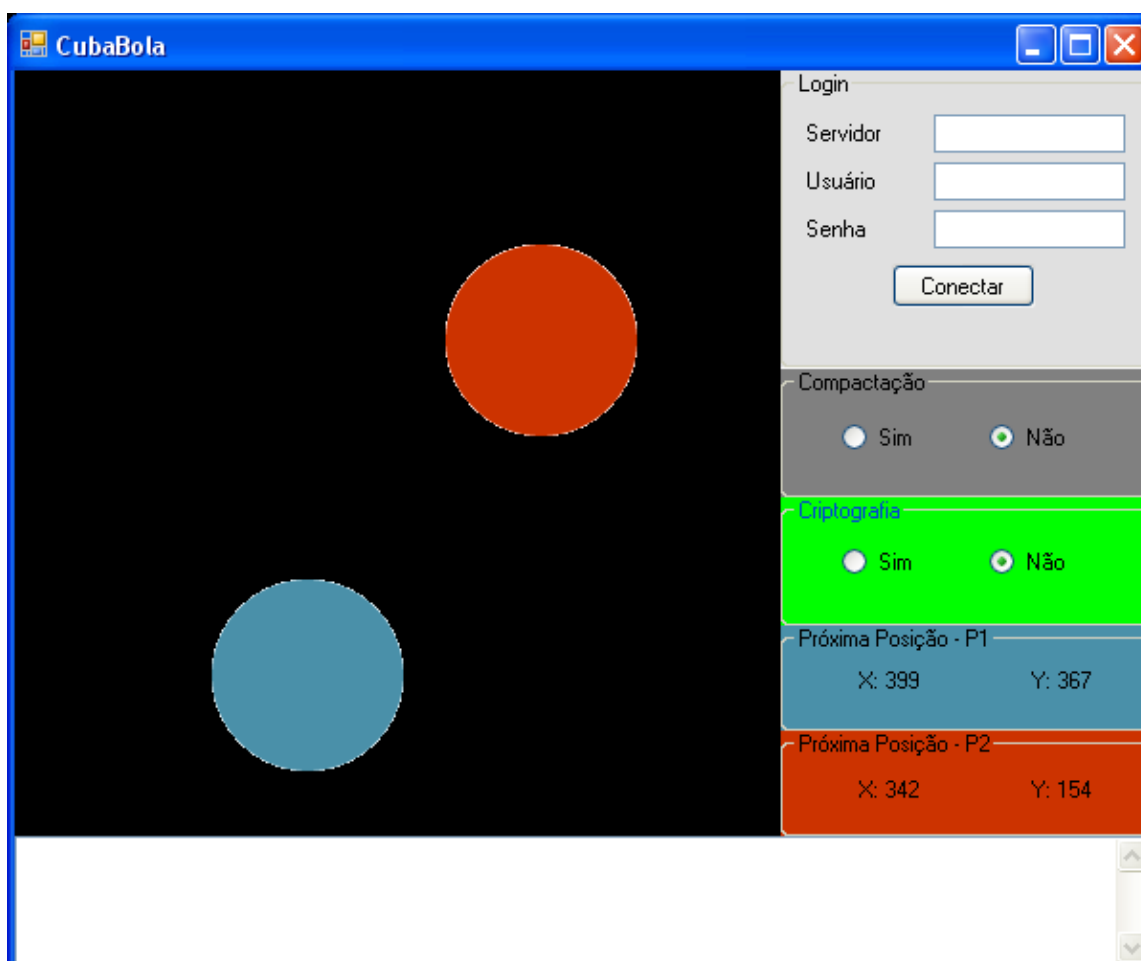
Cada jogador conectado terá uma representação do mundo do jogo (cada jogador é um círculo em uma cor distinta dos demais) que se movimenta aleatoriamente pelo espaço do jogo. Cada vez que o jogador muda de posição, uma mensagem é enviada ao servidor. O servidor a valida, atualizando assim todos os outros clientes conectados, inclusive quem enviou a mensagem.

O servidor é acionado e espera novas conexões. Quando um cliente se conecta no servidor, este envia um aviso a todos os clientes conectados que há um novo jogador.

Quando o cliente recebe o aviso de novo jogador, desenha um círculo colorido para simbolizá-lo. Os círculos, ou jogadores, se movimentam independentemente uns dos outros. O cálculo da próxima posição que os círculos irão se movimentar é feito de duas formas: caso o usuário clique na tela preta, o círculo irá ao local clicado. Se não clicar, é sorteada uma nova posição para o círculo. Um movimento só inicia quando o movimento anterior estiver terminado.

Neste jogo, é possível verificar os pacotes que podem estar criptografados e/ou compactados pelo servidor e pelo próprio cliente. Em todos os clientes há representações de onde os outros jogadores estão localizados no mundo.

O jogo para teste foi desenvolvido na linguagem de programação C#<sup>1</sup>. Essa linguagem foi escolhida primeiramente para demonstrar que o uso do *middleware* não está limitado aos jogos desenvolvidos em C++, por permitir um tempo reduzido de desenvolvimento e finalmente pela familiaridade que os executores do projeto têm com essa linguagem.



**Figura 3** – Tela do Jogo Teste.

1 - CSharp – Linguagem de programação orientada a objetos criada pela Microsoft, parte da sua plataforma .Net.

## 5.2. Configuração Utilizada

Os testes do *middleware* foram realizados utilizando-se quatro máquinas com as seguintes configurações de hardware:

- Servidor – processador AMD Athlon 2.1 Ghz, memória 512 MB, placa de rede Ethernet 10/100 e HD de 80 Gb;
- Cliente 1 – processador AMD Sempron 2.8 Ghz, memória 512 MB, placa de rede Ethernet 10/100 e HD de 80 Gb;
- Cliente 2 – processador Pentium III 800 Mhz, memória 256 MB, placa de rede Ethernet 10/100 e HD de 20 Gb;
- Cliente 3 – processador Pentium III 850 Mhz, memória 192 MB, placa de rede Ethernet 10/100 e HD de 20 Gb.

O sistema operacional da máquina servidora é o Ubuntu Linux Versão 6.10 e o sistema operacional das máquinas clientes é o Windows XP Service Pack 2. Todos os computadores estão conectados em rede local.

## 5.3. Testes

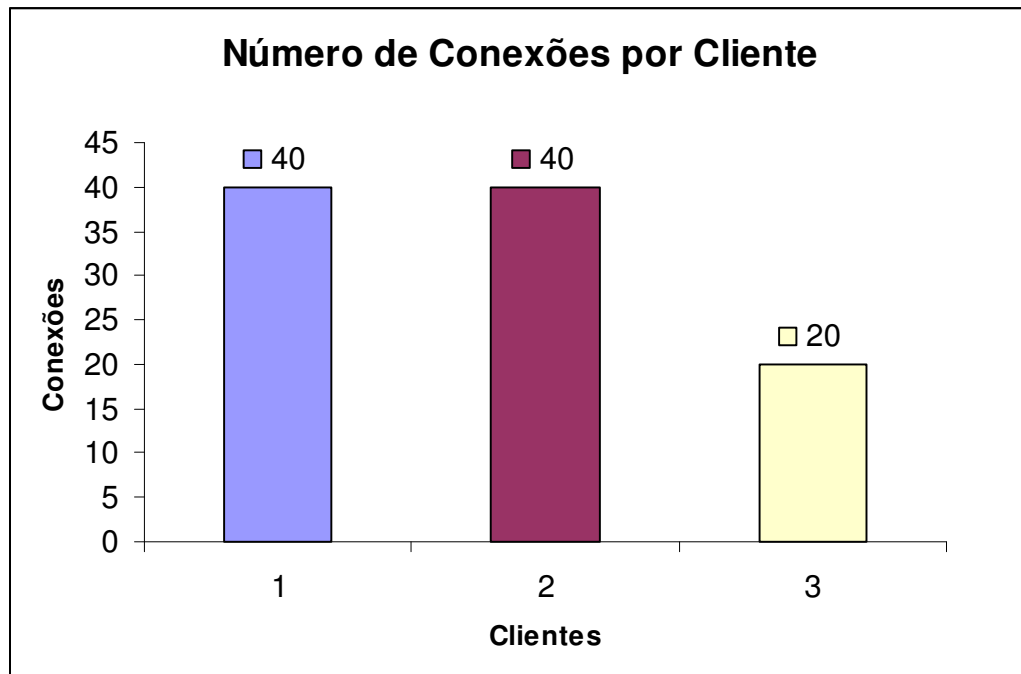
O jogo foi testado em sete sessões (uma por dia), com quatro horas de duração cada. Foi desenvolvido um cronograma diário para os testes, que está descrito abaixo:

- Dia 1

Realizados testes gerais de funcionamento do *middleware*, tais como conexão, envio e recebimento de mensagens:

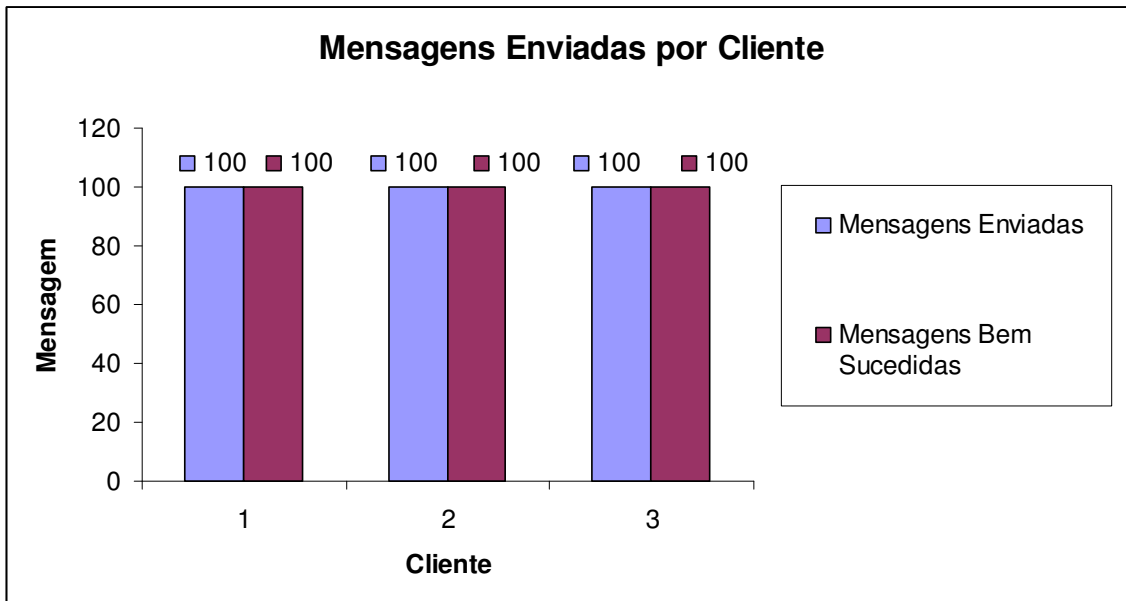
- Números de conexões testadas – 100 conexões simultâneas. O cliente 1 e o cliente 2 abriram 40 conexões cada um e o cliente 3 abriu 20 conexões;

- Números de conexões bem-sucedidas – 100;



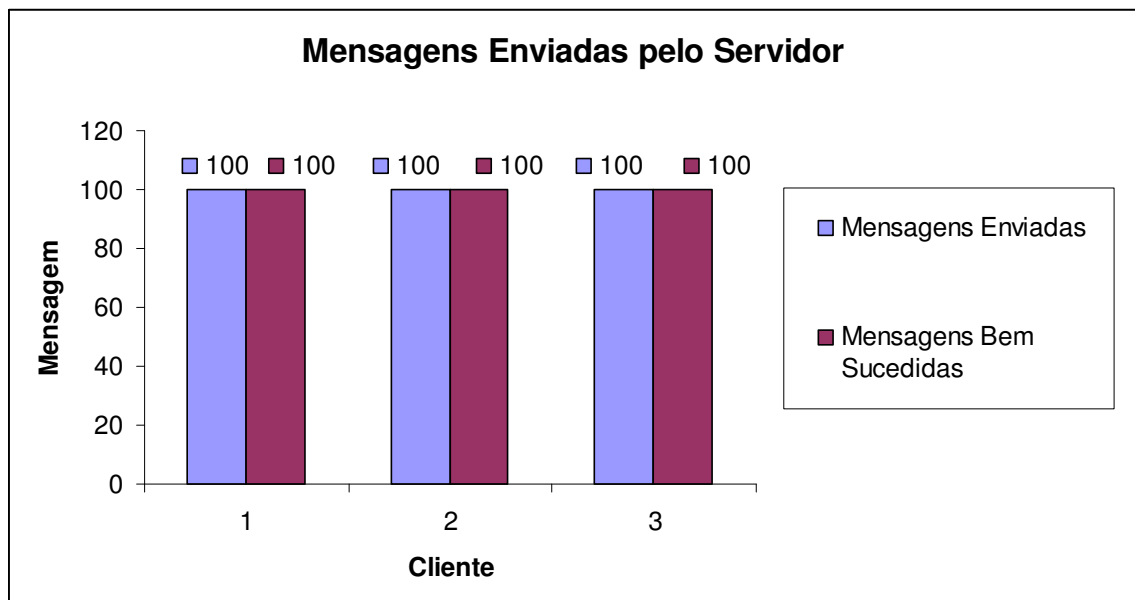
**Figura 4** – Números de Conexões bem-sucedidas.

- Número de mensagens enviadas pelos clientes – 300 mensagens. Sendo 100 mensagens por cada cliente;
- Número de mensagens enviadas pelo cliente bem-sucedidas – 100;



**Figura 5** – Mensagens Cliente-Servidor transmitidas com sucesso.

- Número de mensagens enviadas pelo servidor – 300. Sendo 100 para cada cliente;
- Número de mensagens enviadas pelo servidor bem-sucedidas – 100.



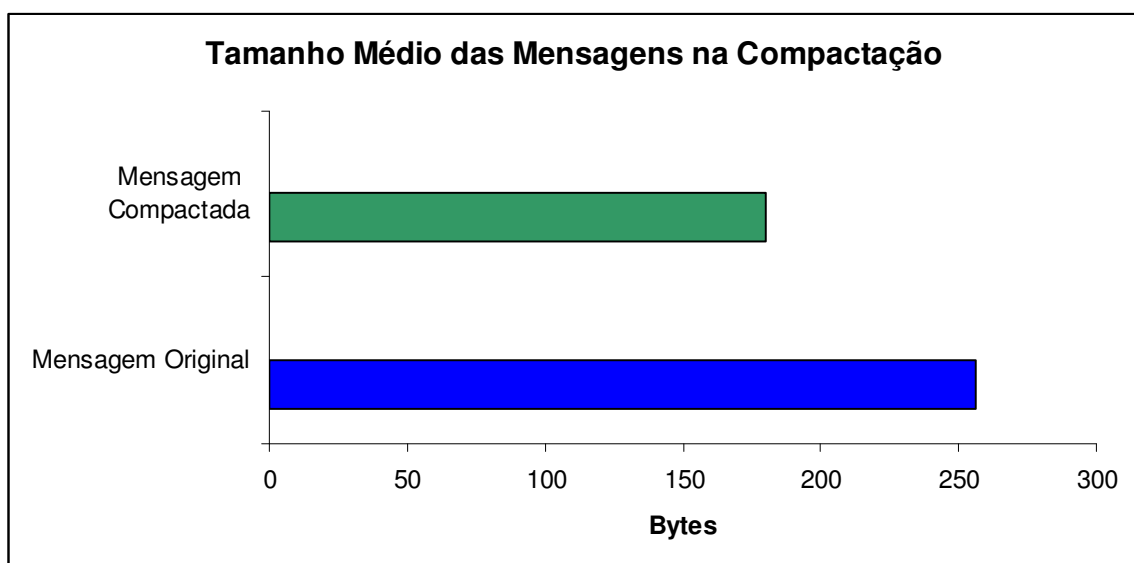
**Figura 6** – Mensagens do Servidor para os Clientes.

- Dia 2

Realizados testes de compactação/descompactação com mensagens de vários tamanhos, e testes de criptografia/descriptografia com mensagens de vários tamanhos e caracteres diferentes para verificar se não há erros ao enviar caracteres especiais para a criptografia. Essas mensagens foram geradas manualmente.

Testes de Compactação:

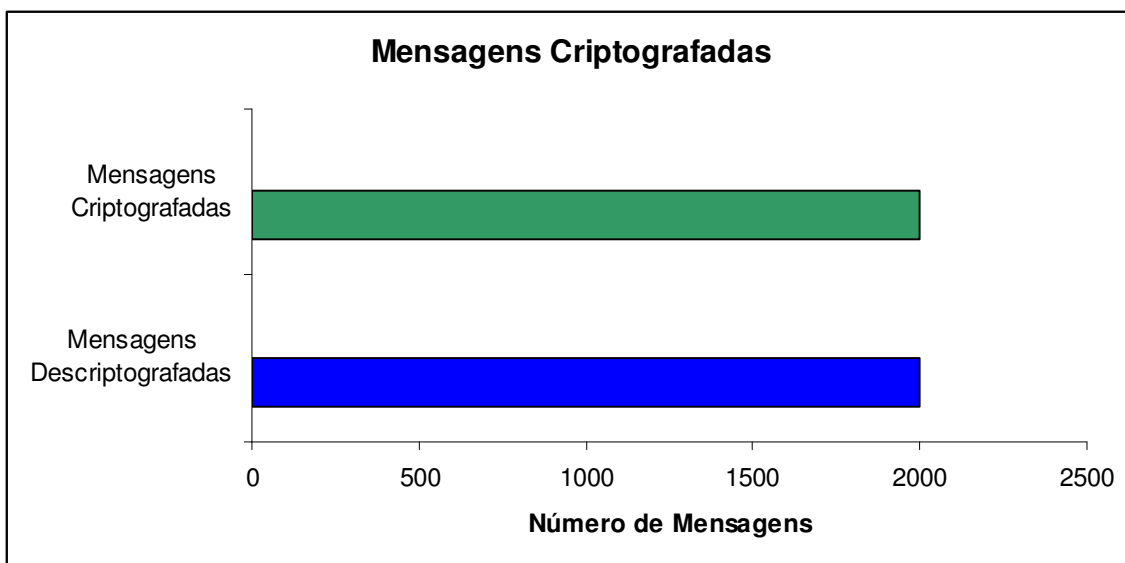
- Número de mensagens compactadas – 2.000;
- Tamanho das mensagens não compactadas – 256 bytes;
- Tamanho médio das mensagens compactadas – 180 bytes.



**Figura 7** – Compactação Média.

Testes de Criptografia:

- Número de mensagens criptografadas – 2.000;
- Número de mensagens descriptografadas com sucesso – 2.000.



**Figura 8** – Mensagens Criptografadas e Descryptografadas com sucesso.

- Dia 3

Realizados testes de quantidade máxima de conexões no servidor. Foram abertos vários clientes em cada uma das três máquinas clientes e o processamento do servidor foi monitorado.

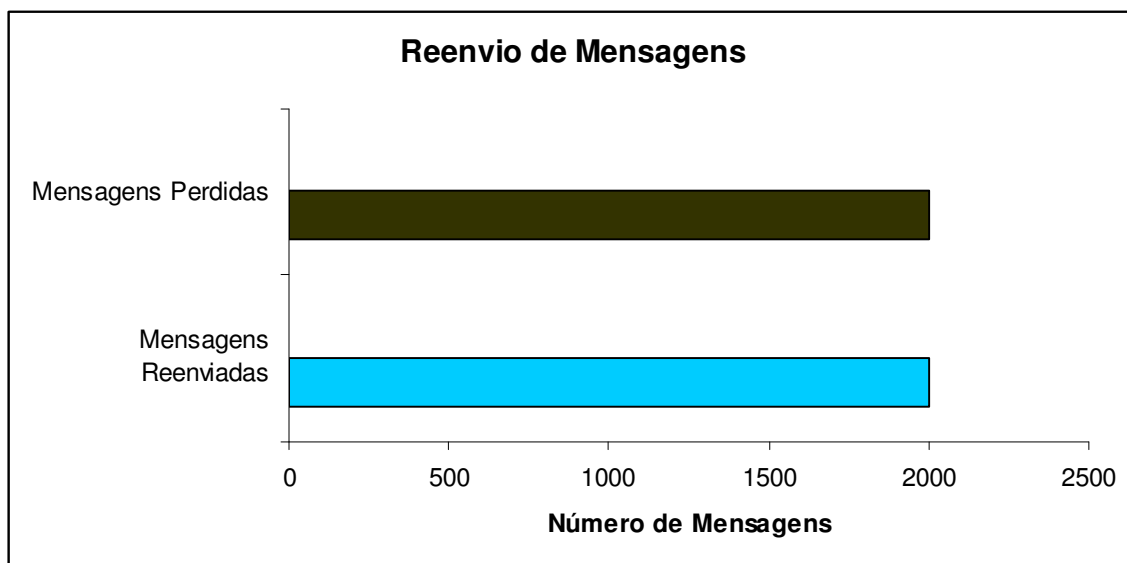
Os testes de quantidade máxima de clientes conectados resultaram em um número máximo de clientes conectados simultaneamente no servidor igual a 300.

Foi verificado que o servidor suportou sem problemas esse número de clientes conectados. Por limitação do número de máquinas de testes para os clientes chegamos a esse número máximo.

- Dia 4

Realizados testes de reenvio de mensagens perdidas. A conexão de rede do cliente foi desligada no cliente e foi monitorado se os pacotes não enviados foram reenviados.

Foi simulada a perda de mensagens tanto no fluxo cliente-servidor quanto servidor-cliente. Foram enviadas 2.000 mensagens nos dois fluxos e o reenvio das mesmas foi constatado nas 2.000 mensagens.



**Figura 9** – Mensagens Perdidas e Reenviadas.

- Dia 5

Realizados testes de tempo de resposta. Foi utilizado um software nativo do Ubuntu Linux para medir o tempo que um pacote enviado demorava para receber a resposta.

Os testes de tempo de resposta seguiram os seguintes critérios. Foram divididos em três grupos para a verificação do tempo de resposta:

- Grupo A – de um (1) a cem (100) clientes conectados;
- Grupo B – de cento e um (101) até cento e noventa e nove (200) clientes conectados;
- Grupo C: de duzentos e um (201) até duzentos e noventa e nove (300) clientes conectados.

Seguem os resultados:

Grupo A – o tempo de resposta médio foi de menos de 1 ms;

Grupo B – o tempo de resposta variou entre 1 ms e 2 ms, mantendo uma média de 1.2 ms;

Grupo C – o tempo de resposta variou entre 2 ms e 4 ms, mantendo uma média de 2.8 ms.

Vale ressaltar que todos os testes foram feitos utilizando uma rede local. O tempo de resposta com certeza aumentará se for utilizada uma conexão via internet.

- Dias 6 e 7

Realizados testes de simulação em um ambiente real, com um jogo desenvolvido com a utilização das funcionalidades do *middleware*.

Esses testes foram realizados por duas pessoas (os próprios desenvolvedores) e mais o auxílio de uma terceira que foi convidada para a validação do resultado dos testes.

Os resultados dos testes realizados foram compatíveis com os requisitos de desempenho previstos. O único teste que não pode ser realizado foi o de conexão via internet por falta de recursos técnicos (havia a necessidade de um número de IP real).

## 6. Conclusões

### 6.1. Resultados Alcançados

Das funcionalidades previstas para o *middleware*, somente a funcionalidade de Atualização de Arquivos não foi implementada. A complexidade dessa funcionalidade não foi prevista no escopo do projeto e, por não se tratar de uma função básica do *middleware*, foi deixada para as novas versões.

Dos requisitos previstos para o *middleware*, a quantidade de usuários simultâneos prevista inicialmente foi alcançada. Foi constatada a conexão de 300 clientes simultâneos, mas não se tem a certeza se esta quantidade prevista será satisfatória em uma conexão via internet. Não foi possível testar o tempo de resposta com confiabilidade, pois o teste que seria necessário – um teste no ambiente da internet – não foi realizado por falta de recursos financeiros. Realizou-se o teste em uma rede fechada, o que não reflete o estado real da internet.

### 6.2. Dificuldades

As principais dificuldades enfrentadas foram as de gestão de projeto, pois foram muitas mudanças durante o decorrer do mesmo e isso impactou na eficiência do desenvolvimento. A mudança principal foi a troca de metodologia: uma troca necessária devido à adaptação complicada à metodologia em espiral (os desenvolvedores do projeto não estavam acostumados com a mesma). Então, a solução foi a mudança de metodologia e o reinício da parte de planejamento do projeto. As outras mudanças foram pequenas alterações de escopo que no final do projeto se acumularam e deixaram a prática diferente da teoria em vários aspectos. Ocorreram também trocas de papéis na equipe de desenvolvimento, que por afinidades mútuas decidiram que havia a necessidade de trocar de atividades em algumas situações. Outra dificuldade de gestão não relacionada a

mudanças veio quando deveriam ser realizados os testes, pois um ambiente de testes não tinha sido planejado no início. Por isso, alguns testes que poderiam ter sido feitos rapidamente, se tornaram um fator de atraso e complicador no final do projeto.

Ocorreram também algumas dificuldades técnicas no que concerne à linguagem de programação utilizada (C++), pois nenhum dos executores tinha experiência prática na sua utilização. A linguagem é muito fragmentada, o que dificultava na busca por informações rapidamente. A procura por referências em livros foi pouco utilizada em casos específicos, pois as informações encontradas nos poucos livros pesquisados são muito genéricas para valer como aprendizado no desenvolvimento.

### **6.3. Trabalhos Futuros**

Está previsto o desenvolvimento da parte de atualização de arquivos, que não foi possível implementar na versão inicial do *middleware*. A Atualização de Arquivos não é essencial para o funcionamento de um *Massively Multiplayer Game*, mas é uma adição à biblioteca de funções, que vem facilitar o desenvolvimento do jogo.

Depois do desenvolvimento dessa função adicional, está prevista a liberação do *middleware* sob licença GNU para que outros desenvolvedores possam utilizá-la sugerindo melhoria nas funcionalidades do *middleware* e também apontando correções que não foram identificadas nos testes realizados.

Depois de dez meses de disponibilidade do *middleware* para outros usuários, está previsto o desenvolvimento de um *Massively Multiplayer Role Play Game* comercial, utilizando-se deste *middleware* para as funções de comunicação.

## 7. Referências

BAUER, DANIEL; ROONEY, SEAN SCOTTON, PAOLO. **Network Infrastructure for Massively Distributed Games**. Bruanschweig, Germany, 2002. Disponível em: <<http://portal.acm.org/citation.cfm?id=566506&dl=ACM&coll=portal>>. Acessado em: 20 de Abril de 2006.

BECK, KENT; ANDRES, CYNTHIA. **Extreme Programming explained: embrace change**. 2. ed. Upper Saddle River: Addison-Wesley, 2005. 189 p.

BOEHM B. W. **A Spiral Model of Software Development and Enhancement**. **Computer**. 1 ed. 1988. Volume 21, número 5.

CANONICAL LTD. **Ubuntu 6.10**. Isle of Man, 2007. Disponível em: <<http://www.ubuntu.com>>. Acessado em: 1 de março de 2007.

CARNEGIE MELLON UNIVERSITY. **Component Object Model (COM), DCOM, and Related Capabilities**. Pittsburgh, 2001. Disponível em: <<http://www.sei.cmu.edu/str/descriptions/com.html>>. Acessado em: 19 de janeiro de 2007.

CECIN, FABIO REIS; BARBOSA, JORGE LUIS VICTÓRIA; GEYER, CLÁUDIO FERNANDO RESIN. **FreeMMG: Uma Proposta Baseada em Peer-to-Peer para Simulação Interativa Competitiva em Tempo Real e Distribuída com Suporte à Elevada Escalabilidade**. Porto Alegre, 2003. Disponível em: <<http://www.inf.ufrgs.br/pos/ppgc/semanacademica/artigos2003/1302.pdf>>. Acessado em: 20 de março de 2003.

COULOURIS, GEORGE; DOLLIMORE, JEAN; KINDBERG, TIM. **Distributed System Concepts and Design**. 3. ed. Upper Saddle River: Addison Wesley, 2001. 772 p.

CYBER CREATIONS INC. **MMORPG**. Disponível em: <<http://www.mmorpg.com/>>. Acessado em: 26 de abril de 2006.

DAEMEN, JOAN; RIJMEN; VINCENT. **AES submission document on Rijndael, Version 2**. 1999. Disponível em: <<http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf>>. Acessado em: 15 de março de 2007.

FEDERAL STANDARD. **Round-trip Delay Time**. 1996. Disponível em: <[http://www.its.bldrdoc.gov/fs-1037/dir-031/\\_4641.htm](http://www.its.bldrdoc.gov/fs-1037/dir-031/_4641.htm)>. Acessado em: 15 de março de 2007.

GAILLY, JEAN-LOUP; ADLER, MARK. **ZLib 1.2.3**. 2005. Disponível em: <<http://www.zlib.net/>>. Acessado em: 1 de setembro de 2006.

KRUCHTEN, P. **The Rational Unified Process: An Introduction**. 2. ed. Upper Saddle River: Addison-Wesley, 2000. 320 p.

NETOFFICE. **NetOffice version 2.6.0**. Disponível em: <<http://netoffice.sourceforge.net/modules/news/>>. Acessado em: 30 de abril de 2006.

PERUCIA, ALEXANDRE SOUZA. **Desenvolvimento de Jogos Eletrônicos: Teoria e Prática**. 1. ed. Novatec. 302 p.

POSTEL, J. **User Datagram Protocol - RFC 768**. 1980. Disponível em: <<http://tools.ietf.org/html/rfc768>>. Acessado em: 16 de março de 2007.

RNP – Rede Nacional de Ensino e Pesquisa. **GT Middleware**. Disponível em <<http://www.rnp.br/pd/gts2004-2005/middleware.html>>. Acessado em: 31 de janeiro de 2007.

TORUS KNOT SOFTWARE LTD. Ogre 3D. Disponível em: <<http://www.ogre3d.org>>. Acessado em: 15 de março de 2007.

SEBERN, MARK J. **STL Vector Class Version 1.4**. 1998. Disponível em: <<http://www.msos.edu/eecs/ce/courseinfo/stl/vector.htm>>. Acessado em: 1 de setembro de 2006.

STANTON, MICHAEL. **Rijndael: o sucessor do DES**. São Paulo, 2004.

Disponível em:

<<http://www5.estadao.com.br/tecnologia/coluna/stanton/2000/nov/06/168.htm>>.

Acessado em: 15 de março de 2007.

TANENBAUM, ANDREW S. **Computer Networks**. 4. ed. Indiana: Prentice Hall. 912 p.

TELES, VINÍCIUS MANHÃES. **Um estudo de caso da adoção das práticas e valores do extreme programming**. Rio de Janeiro, 2005. Disponível em:

<<http://www.improveit.com.br/xp/dissertacaoXP.pdf>>. Acessado em: 10 de maio de 2006.

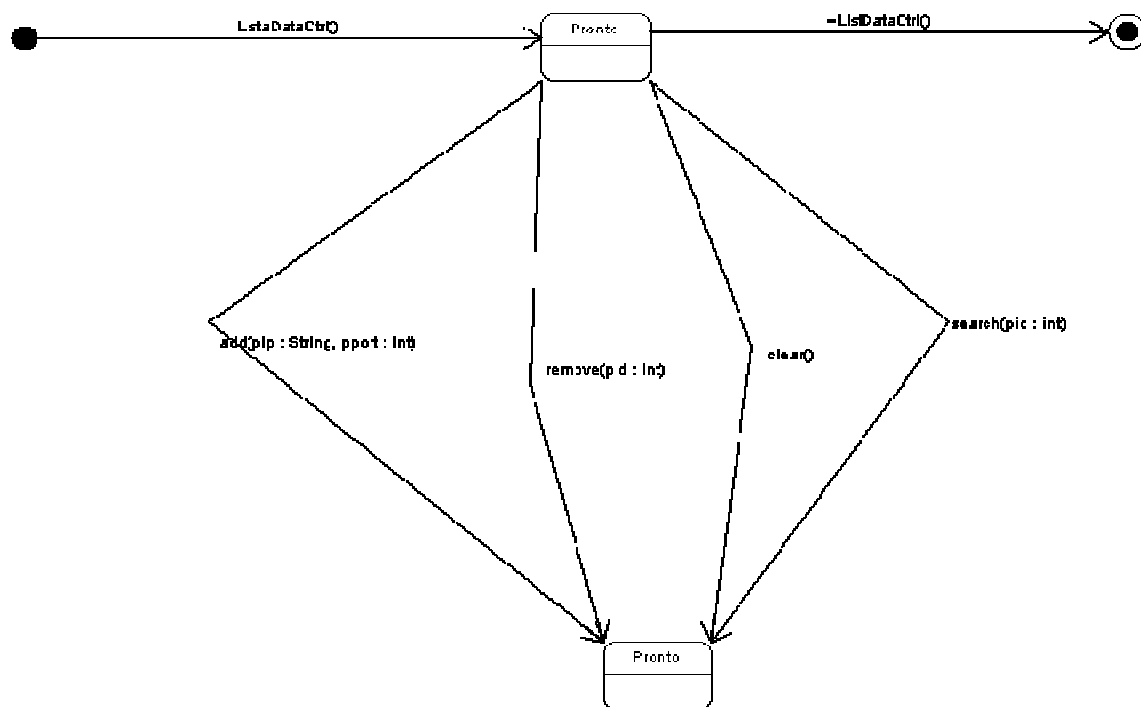
W3SCHOOLS. **Brower Statistics**. 2007. Disponível em:

<[http://www.w3schools.com/browsers/browsers\\_stats.asp](http://www.w3schools.com/browsers/browsers_stats.asp)>. Acessado em: 15 de março de 2007.

## Anexo I

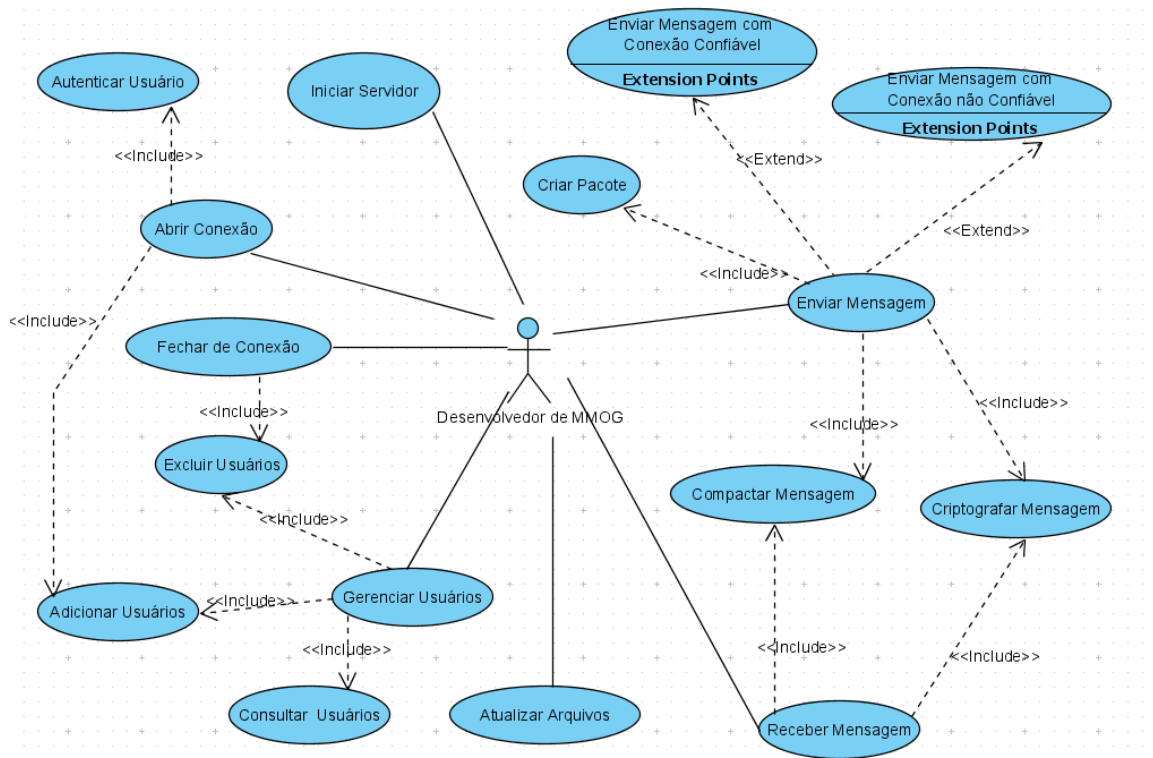
### Diagramas

#### 1. Diagrama de Estados da Classe ListDataCtrl

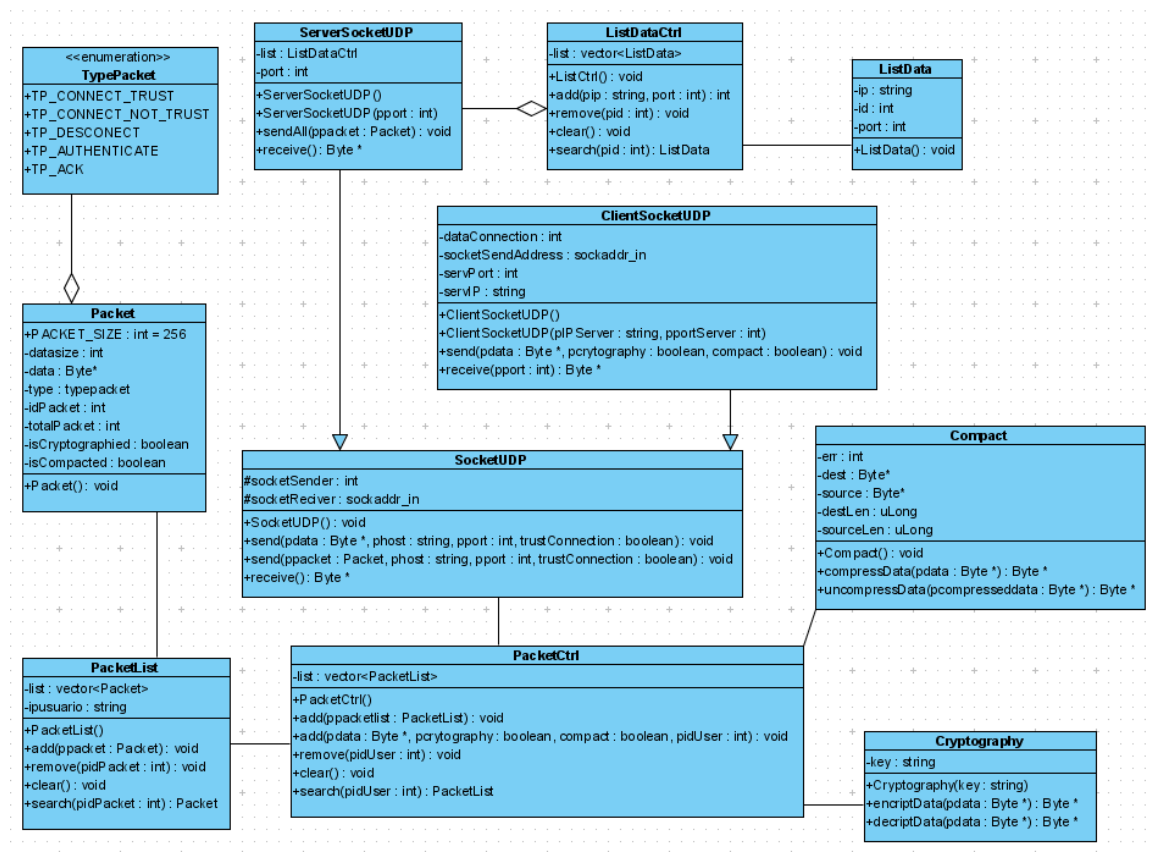




### 3. Diagrama de Casos de Uso



## 4. Diagrama de Classes



## Anexo II

### Descrição dos Casos de Uso

<b>Nome</b>	Enviar mensagem.
<b>Descrição</b>	Envia uma mensagem para o servidor. A mensagem pode ser quebrada em um ou mais pacotes para ser enviada.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Verifica a mensagem (A1) (A2);</li> <li>2. Cria vários pacotes seqüenciais a partir da mensagem;</li> <li>3. Constrói o protocolo;</li> <li>4. Envia os pacotes ao destinatário (A3) (A4).</li> </ol>
<b>Fluxo Alternativo</b>	<p>A1) Se a opção de compactação estiver ativada:</p> <ol style="list-style-type: none"> <li>1. Compacta a mensagem.</li> </ol> <p>A2) Se a opção de criptografia estiver ativada;</p> <ol style="list-style-type: none"> <li>1. Criptografa a mensagem.</li> </ol> <p>A3) Se a opção de comunicação confiável estiver ativada:</p> <ol style="list-style-type: none"> <li>1. Enviar Mensagem com Comunicação Confiável.</li> </ol> <p>A3) Se a opção de comunicação não confiável estiver ativada:</p> <ol style="list-style-type: none"> <li>1. Enviar Mensagem com Comunicação Não Confiável.</li> </ol>
<b>Fluxo de Exceções</b>	Não há.

<b>Nome</b>	Enviar Mensagem com Comunicação Confiável.
<b>Descrição</b>	<p>Comunicação confiável é aquela que, caso aconteça uma perda de pacote, são adotadas medidas de reenvio do pacote perdido.</p> <p>OBS.: toda parte de criação do pacote é feita no caso de uso Criar Pacote.</p>
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Enviar pacote;</li> </ol>

	2. Recebe o ACK (E1).
<b>Fluxo Alternativo</b>	Não há.
<b>Fluxo de Exceções</b>	<p>E1) Caso o ACK não seja recebido em 0,3 segundos:</p> <ol style="list-style-type: none"> <li>1. Reenvia o Pacote Perdido (E2)(E3);</li> <li>2. Volta ao Fluxo Principal 1.</li> </ol> <p>E2) Caso o pacote seja perdido 3 vezes e o servidor é o emissor:</p> <ol style="list-style-type: none"> <li>1. Dispara o evento de queda de conexão com o cliente;</li> <li>2. Realiza o caso de uso Fechar Conexão.</li> </ol> <p>E3) Caso o pacote seja perdido 3 vezes e o cliente é o emissor:</p> <ol style="list-style-type: none"> <li>1. Dispara o evento dizendo que o servidor está inativo.</li> </ol>

<b>Nome</b>	Enviar Mensagem com Comunicação Não Confiável
<b>Descrição</b>	<p>Conexão não confiável se define como o envio de pacotes sem nenhum controle de queda e perda de pacote.</p> <p>OBS.: toda parte de criação do pacote é feito no caso de uso Criar Pacote.</p>
<b>Fluxo Principal</b>	1. Enviar pacote.
<b>Fluxo Alternativo</b>	Não há.
<b>Fluxo de Exceções</b>	Não há.

<b>Nome</b>	Atualização de Arquivos.
<b>Descrição</b>	Não foi implementado, pois não é uma funcionalidade básica do <i>middleware</i> .
<b>Fluxo Principal</b>	Não há.
<b>Fluxo Alternativo</b>	Não há.
<b>Fluxo de Exceções</b>	Não há.

<b>Nome</b>	Compactar Mensagem.
<b>Descrição</b>	Utilização de um algoritmo de compactação dos dados do jogo. Utilizamos a biblioteca ZLIB (GAILLY, 2005).
<b>Fluxo Principal</b>	1. Compacta ou descompacta a mensagem.
<b>Fluxo Alternativo</b>	Não há.
<b>Fluxo de Exceções</b>	Não há.

<b>Nome</b>	Criptografar Mensagem.
<b>Descrição</b>	Utilização do algoritmo AES 128 bits para criptografar a mensagem (DAEMEN, 1999).
<b>Fluxo Principal</b>	1. Criptografa ou descriptografa a mensagem.
<b>Fluxo Alternativo</b>	Não há.
<b>Fluxo de Exceções</b>	Não há.

<b>Nome</b>	Receber Mensagem.
<b>Descrição</b>	Caso de uso que define o recebimento de uma mensagem, a junção de todos os pacotes para formar a mensagem, a descompactação e descriptografia da mensagem.
<b>Fluxo Principal</b>	1. Recebe todos os pacotes (A1)(E1). 2. Monta a mensagem (A1)(A2).
<b>Fluxo Alternativo</b>	A1) Caso o pacote tenha sido enviado com a conexão confiável: 1. Envia ACK do pacote. A2) Caso a mensagem esteja criptografada: 1. Descriptografa a mensagem. A3) Caso a mensagem esteja compactada: 1. Descompacta a mensagem.
<b>Fluxo de Exceções</b>	E1) Caso a espera por receber o próximo pacote seja superior a 250 milisegundos: 1. Dispara o evento de timeout de recebimento de

	<p>mensagem (E2).</p> <p>E2) Caso seja gerado 3 timeout:</p> <ol style="list-style-type: none"> <li>1. Dispara o evento dizendo que o servidor está inativo.</li> </ol>
--	---

<b>Nome</b>	Criar Pacote.
<b>Descrição</b>	Caso de uso que define um pacote do <i>middleware</i> e monta o protocolo.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Monta o protocolo.</li> <li>2. Cria o pacote.</li> </ol>
<b>Fluxo Alternativo</b>	Não há.
<b>Fluxo de Exceções</b>	Não há.

<b>Nome</b>	Iniciar Servidor.
<b>Descrição</b>	Caso de uso que define a preparação do servidor para entrar em modo de espera de mensagens dos usuários.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Cria lista de usuários.</li> <li>2. Espera conexões do usuário.</li> </ol>
<b>Fluxo Alternativo</b>	Não há.
<b>Fluxo de Exceções</b>	Não há.

<b>Nome</b>	Modificar Usuários.
<b>Descrição</b>	Caso de uso que define as funções de gerenciamento da lista de usuários que estão conectados ao servidor.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Consulta a lista de usuários.</li> <li>2. Adiciona um usuário à lista.</li> <li>3. Exclui um usuário da lista.</li> </ol>
<b>Fluxo Alternativo</b>	Não há.
<b>Fluxo de Exceções</b>	Não há.

<b>Nome</b>	Consultar Usuário.
<b>Descrição</b>	Caso de uso que define a pesquisa de usuários na lista de usuários.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Pesquisa usuários.</li> <li>2. Retorna com os usuários encontrados A1).</li> </ol>
<b>Fluxo Alternativo</b>	A1) Caso nenhum usuário seja encontrado: <ol style="list-style-type: none"> <li>1. Retorna vazio.</li> </ol>
<b>Fluxo de Exceções</b>	Não há.

<b>Nome</b>	Adicionar Usuário.
<b>Descrição</b>	Caso de uso que define a inclusão de um usuário na lista de usuários.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Inclui um usuário (E1).</li> <li>2. Retorna o ID do usuário.</li> </ol>
<b>Fluxo Alternativo</b>	Não há.
<b>Fluxo de Exceções</b>	E1) Caso o usuário já exista na lista de usuário: <ol style="list-style-type: none"> <li>1. Retorna a mensagem: “Usuário já cadastrado”.</li> </ol>

<b>Nome</b>	Excluir Usuário.
<b>Descrição</b>	Caso de uso que define a exclusão de um usuário na lista de usuários.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Exclui um usuário.</li> </ol>
<b>Fluxo Alternativo</b>	Não há.
<b>Fluxo de Exceções</b>	Não há.

<b>Nome</b>	Abrir Conexão.
<b>Descrição</b>	Caso de uso que define a conexão inicial com o servidor.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Autentica o usuário (A1).</li> <li>2. Adiciona o usuário à lista de usuário.</li> <li>3. Envia o novo ID do usuário ao usuário, cliente.</li> </ol>

<b>Fluxo Alternativo</b>	A1) Caso o usuário não esteja autorizado a acessar o servidor: <ol style="list-style-type: none"> <li>1. Envia ao usuário a mensagem de falha na autenticação.</li> </ol>
<b>Fluxo de Exceções</b>	Não há.

<b>Nome</b>	Fechar Conexão.
<b>Descrição</b>	Caso de uso que define o término da conexão do servidor com o cliente.
<b>Fluxo Principal</b>	<ol style="list-style-type: none"> <li>1. Exclui o usuário da lista de usuários.</li> <li>2. Envia o aviso de fechamento de conexão do usuário a todos os usuários da lista de usuários.</li> </ol>
<b>Fluxo Alternativo</b>	Não há.
<b>Fluxo de Exceções</b>	Não há.

## **Anexo III**

### **Lista de *Middlewares* pesquisados (OGRE3D, 2007)**

#### **Física:**

Ode – <http://ode.org/>

NovodeX/PhysX – <http://www.novodex.com/>

Newton Game Dynamics – <http://www.newtondynamics.com/>

True Axis Physics SDK – <http://www.trueaxis.com/>

DynaMo – <http://home.iae.nl/users/starcat/dynamo>

The Gangsta Wrapper – <http://sourceforge.net/projects/gangsta>

OPAL – <http://opal.sf.net/>

Bullet – <http://bullet.sf.net/>

#### **Inteligência Artificial:**

OpenAI – <http://openai.sourceforge.net/>

FEAR – <http://fear.sourceforge.net/index.php>

OpenSteer – <http://opensteer.sourceforge.net/>

A\* Tactical Pathfinding – <http://www.cgf-ai.com/products.html#tacastarexplorer>

PathLib – <http://pathlib.hildebrand.cz/pathlib.html>

Garfixia AI Repository – <http://www.dossier-andreas.net/ai/index.html>

MicroPather – <http://www.grinninglizard.com/MicroPather/>

Boost Graph Library – [http://boost.org/libs/graph/doc/table\\_of\\_contents.html](http://boost.org/libs/graph/doc/table_of_contents.html)

FANN Platform independent – <http://leenissen.dk/fann/>

OpenSkyNet – <http://opensky.net.sourceforge.net/>

#### **Rede:**

RakNet – <http://freshmeat.net/projects/raknet>

OpenTNL – <http://www.opentnl.org/>

Zoidcom – <http://www.zoidcom.com/>

HawkNL – <http://www.hawksoft.com/hawknl/>

ENet – <http://enet.cubik.org/>

ZIGE Game Engine – <http://zige.sourceforge.net/>

SDL\_net – [http://www.libsdl.org/projects/SDL\\_net/](http://www.libsdl.org/projects/SDL_net/)

### **Áudio:**

FMOD – <http://www.fmod.org/>

OpenAL – <http://www.openal.org/>

BASS – <http://www.un4seen.com/bass.html>

Audiere – <http://audiere.sourceforge.net/>

### **Entrada:**

SDL – <http://www.libsdl.org/>

LibGII – <http://www.ggi-project.org/packages/libgii.html>

OpenInput – <http://home.gna.org/openinput>

OIS – <http://sourceforge.net/projects/wgois>

### **Gráficas:**

Cegui Layout Editor –

[http://www.cegui.org.uk/wiki/index.php/CELayoutEditor\\_Downloads\\_0.5.0](http://www.cegui.org.uk/wiki/index.php/CELayoutEditor_Downloads_0.5.0)

Open GUI – <http://opengui.rightbracket.com/index.php>

## Autorização

Autorizo a reprodução e/ou divulgação total ou parcial da presente obra, por qualquer meio convencional ou eletrônico, desde que citada a fonte.

Nome do Autor: Guilherme Moschen

Nome do Autor: Romulo De Lazzari

Assinatura do Autor: \_\_\_\_\_

Assinatura do Autor: \_\_\_\_\_

Instituição: Universidade Tecnológica Federal do Paraná

Local: Curitiba, Paraná

Endereço: Rua Sete de Setembro, n.º 3.165, Rebouças

E-mail: guilherme.catarina@gmail.com

E-mail: romulodelazzari@gmail.com