FACULDADES BARDDAL
CURSO DE SISTEMAS DE INFORMAÇÃO

FERNANDO COSTENARO SILVA

MIDDLEWARE DE COMUNICAÇÃO PARA JOGOS *MULTIPLAYER*: UM ESTUDO DE CASO NO PROJETO SUBVERSE

FLORIANÓPOLIS/SC
2008

FERNANDO COSTENARO SILVA

MIDDLEWARE DE COMUNICAÇÃO PARA JOGOS *MULTIPLAYER*: UM ESTUDO DE CASO NO PROJETO SUBVERSE

Monografia apresentada como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação do curso de Sistemas de Informação Esta monografia foi submetida ao Curso de Sistemas de Informação ministrado pelas Faculdades Barddal.

ORIENTADOR: Msc. Saulo Popov Zambiasi

FLORIANÓPOLIS/SC

FERNANDO COSTENARO SILVA

MIDDLEWARE DE COMUNICAÇÃO PARA JOGOS *MULTIPLAYER*: UM ESTUDO DE CASO NO PROJETO SUBVERSE

Esta monografia foi submetida ao Curso de Sistemas de Informação das Faculdades Barddal, como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação e considerada aprovada pela banca abaixo.

Profª. Drª. Mirela Sechi Moretti Annoni Notare
Coordenadora do Curso de Sistemas de Informação

BANCA EXAMINADORA

Prof. Msc. Saulo Popov Zambiasi - Orientador
Faculdades Barddal

Profª. Drª. Mirela Sechi Moretti Annoni Notare
Faculdades Barddal

Prof. Msc. Bruno Panerai Velloso
Faculdades Barddal

Florianópolis SC, 08 de dezembro de 2008.

Dedico este trabalho à meus pais, parentes e ao meu orientador por todo o apoio dado a mim.

AGRADECIMENTOS

Agradeço o meu orientador Saulo Popov Zambiasi por ter acreditado, incentivado e me orientado ao longo da realização deste trabalho. Agradeço a minha mãe, avó, namorada e família por todo o apoio, e incentivo e compreensão pelos momentos em que estive ausente. Agradeço também a todos os professores e funcionários das Faculdades Barddal, pelo excelente profissionalismo durante toda a trajetória na faculdade.

“Uma longa viagem começa com um único passo.”

(Lao-Tsé)

RESUMO

O mercado de jogos *on-line* é um dos que mais crescem, ano após ano, na área de entretenimento. Entre os diversos segmentos deste mercado, os jogos multiusuários *on-line* se destacam por proporcionarem a experiência de interatividade entre os jogadores, conectados de regiões diferentes através de uma rede, ou da Internet, como se estivessem no mesmo ambiente real. Porém o desenvolvimento de um jogo desse segmento é complexo e demorado, devido ao grande número de elementos de software que compõem o jogo. Assim, este trabalho propõe um *middleware* de comunicação, para facilitar o desenvolvimento de jogos multiusuários *on-line*, poupando os desenvolvedores de se preocuparem com toda a parte de envio e recebimento de mensagens, fazendo com que a troca dessas mensagens pareça localmente. Para a realização de testes, o *middleware* proposto utiliza como base a atual implementação do Projeto Subverse, da qual ainda não existe implementado em seu Ambiente Virtual essa comunicação entre o Cliente e o Servidor. Os testes realizados mostraram que a implementação foi bem sucedida e alcançou seus objetivos de realizar a conexão e a troca de mensagens.

PALAVRAS-CHAVE: *Middleware*, Soquetes, Sistemas Distribuídos, Jogos *Multiplayer*.

ABSTRACT

The market of online games is one of that most grows, year after year, on entertainment area. Among the many segments of this market, the on-line multiplayer games emphasize itself by provide the experience of interactivity between the players, connected from different regions through a network, or the Internet, like they were on the same real ambient. But the development of a game of this segment is complex and slow, because the many elements of software that compose the game. So, this paper propose an communication middleware, to make easier the development of on-line multiplayer games, saving the developers to worry with all the part of send and receive of messages, so that the exchange of those messages look locally. For testing, the proposed middleware uses as base the current Subverse Project, which does not yet exist implemented in its Virtual World this communication between the Client and the Server. The tests made shows that the implementation was successful and reached its objectives of make the connection and exchange of messages.

KEYWORDS: Middleware, Sockets, Distributed Systems, Multiplayer Games.

LISTA DE ACRÔNIMOS

***MMOG**: Massive Multiplayer Online Game*

***RPG**: Role-Playing Game*

***MMORPG**: Massive Multiplayer Online Role-Playing Game*

***RPC**: Remote Procedure Call*

***RMI**: Remote Method Invocation*

CORBA: *Common Object Request Broker Architecture*

LABSIB: Laboratório de Sistemas Inteligentes das Faculdades Barddal

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O crescimento da indústria de jogos gera um aumento da procura por profissionais especializados na área e, conseqüentemente, aumenta o requerimento por propriedade intelectual gerada pelas universidades e seus estudantes. Zyda (2006) cita que o maior requerimento da indústria de jogos é na área da ciência da computação e que as universidades estão cada vez mais introduzindo disciplinas voltadas a área de entretenimento.

Outro fator que contribui com esse crescimento da indústria de jogos é a popularidade da internet, uma vez que centenas ou até milhares de pessoas de todo o mundo podem interagir através dos jogos *on-line* multiusuários sem sair de casa.

Segundo Dickey (2004), os jogos *on-lin*e multiusuários geralmente utilizam arquitetura de comunicação cliente/servidor. Os clientes enviam mensagens para um servidor responsável pelo processamento do jogo e recebem mensagens processadas para mostrar em uma interface gráfica ao usuário. Porém, quanto maior o número de clientes conectados no servidor, maior a quantidade de informações que necessitam serem processadas, podendo causar atraso nas respostas ou até mesmo perda de dados. (Orthman, 2004)

Segundo Orthman (2004), a disponibilidade do servidor é melhorada com a utilização de supercomputadores que consigam processar pelo menos o limite máximo de clientes suportados. Porém essa solução acaba se tornando inviável para muitos sistemas devido ao alto custo, tanto de aquisição quanto de manutenção de supercomputadores.

Uma alternativa aos supercomputadores é a utilização de sistemas com múltiplos processadores, na qual segundo Machado (2002), é possível reduzir drasticamente o tempo de processamento em aplicações que necessitam de um grande poder computacional, além de um ganho de desempenho, escalabilidade, relação custo benefício, tolerância a falhas, disponibilidade e balanceamento de carga.

Uma subdivisão de sistemas que utilizam múltiplos processadores são os sistemas distribuídos, no qual segundo Machado (2002), o sistema operacional trata os *hosts*, ou sistemas autônomos, como um único sistema centralizado, que tem como principal problema o gerenciamento da comunicação entre os diversos processadores. O sistema precisa oferecer tolerância a falhas de forma transparente e a utilização de recursos concorrentes exige mecanismos mais complexos e lentos para manter a integridade e a segurança nos dados.

Existem vários tipos de comunicação de processos, como por exemplo, Soquetes, RPC, RMI, e CORBA. Cada um desses tipos de comunicação se diferencia basicamente pela aplicação. De acordo com Silberschatz (2001), os soquetes permitem a comunicação em baixo nível, ou seja, entre processos, enquanto que os outros mecanismos de comunicação proporcionam uma comunicação em alto nível, chamando funções de outros processos ou objetos.

Como a comunicação dos processos é o principal problema dos sistemas distribuídos, é possível utilizar um *middleware* de comunicação. Segundo Deitel (2005), um *middleware* promove modularidade de programa, facilitando a programação de aplicações distribuídas, uma vez que o desenvolvedor não precisa se preocupar em escrever o código para gerenciar interações entre os processos. Entretanto, a comunicação entre o *middleware* e processos incorre em custos adicionais se comparado com a comunicação direta entre os processos.

Sob este contexto, diversas aplicações e jogos de computador surgem, utilizando a comunicação por meio destes *middlewares*. Nas Faculdades Barddal, no LABSIB – Laboratório de Sistemas Inteligentes das Faculdades Barddal –, existe um projeto, o Subverse, que é um ambiente de aprendizado multidisciplinar, onde os alunos aplicam os conhecimentos adquiridos em várias disciplinas na prática. Porém, um dos pontos ainda não foi implementado é o *middleware* de comunicação.

Com base no problema proposto, este trabalho apresenta uma proposta de um *middleware* de comunicação para facilitar o desenvolvimento e programação de aplicações distribuídas aplicadas no projeto Subverse.

1.1. OBJETIVOS

De forma a delimitar os objetivos de forma explícita, em conformidade com a problemática apresentada, é necessário a descrição dos objetivos gerais, como uma descrição genérica, e os objetivos específicos na forma de uma descrição mais exata de cada ponto a ser atingido. Assim, estes pontos são descritos a seguir.

1.1.1. OBJETIVO GERAL

Este trabalho tem por objetivo o desenvolvimento de um protótipo parcial de um middleware de comunicação para jogos de computador multiplayer com estudo de caso no Projeto Subverse. Este protótipo busca poupar os desenvolvedores de se preocuparem com toda a parte de envio e recebimento de mensagens, fazendo com que a troca dessas mensagens pareça acontecer localmente.

1.1.2. OBJETIVOS ESPECÍFICOS

Com base no objetivo geral descrito acima, torna-se necessário a definição de alguns objetivos específicos para nortear o presente trabalho:

- Obter conhecimento sobre formas de comunicações distribuídas por meio de uma revisão bibliográfica;
- Definir e conceituar middleware, com ênfase em comunicação;
- Obter uma modelagem do sistema;

- Implementar um protótipo parcial do sistema com base na modelagem alcançada no item anterior;

- Testar o middlware implementado para validação do mesmo.

Desta forma, o trabalho deve concluir-se com a apresentação dos resultados em conformidade com o objetivo geral definido.

1.2 JUSTIFICATIVA

O mercado de jogos *on-line* é um dos que mais crescem a cada ano após ano na área de entretenimento. *MMORPG*, acrônimo de *Massive Multiplayer Online Role Playing Game*, é um estilo de jogo que permite a interação de até milhares de pessoas conectadas ao mesmo tempo em rede, onde cada jogador tem que interpretar seu personagem e agir de acordo com o enredo proposto (Orthman, 2004).

Este segmento de jogos tem se destacado na quantidade de vendas no mercado e no número de usuários ativos. O jogo *World of Warcraft* (Blizard, 2004), por exemplo, passou dos 10 milhões de assinantes no mundo inteiro no início do ano de 2008, pouco mais de três anos após seu lançamento (TERRA, 2008). A primeira expansão do jogo bateu o recorde de cópias vendidas nas primeiras 24 horas após estar disponível para compra, com 2,4 milhões de cópias e a segunda expansão, com 2,8 milhões de cópias (BLIZZARD, 2008).

Com o constante crescimento no segmento de jogos multiusuários *on-line*, cada vez mais são necessários profissionais capacitados ou, pelo menos, com uma noção mínima da estrutura e funcionamento dos jogos multiusuários. Porém o desenvolvimento de um jogo multiusuário pode levar meses ou até anos, devido á complexidade envolvida entre a comunicação entre os jogadores clientes e o servidor, criação da história, dos personagens,

cenário, programação do mecanismo do jogo, interação entre os jogadores, entre outros componentes presentes.

Como uma das partes mais importantes nos jogos multiusuários *on-line* é a comunicação entre os clientes e o servidor, é necessário um ambiente de desenvolvimento para implementar o *middleware* de comunicação.

Nas Faculdades Barddal, no LABSIB, o Subverse (ZAMBIASI, 2008) proporciona um ambiente de aprendizado para o desenvolvimento de aplicações que utilizam desde computação gráfica, técnicas de desenvolvimento de jogos de computador (JESUS, 2008), sistemas distribuídos e até agentes que utilizam algoritmos com inteligência artificial para interagir no mundo virtual (BENIN, 2007). Porém toda a parte de comunicação ainda não foi implementada, oferecendo assim, um ambiente ideal para a implementação de um *middleware* de comunicação.

Assim, a proposta desse trabalho é o desenvolvimento de um *middleware* de comunicação utilizando como estudo de caso o projeto Subverse, fazendo com que o desenvolvimento de jogos multiusuários seja mais ágil, uma vez que o programador não precisa se preocupar com o envio, recebimento e tratamento das mensagens trocadas entre o servidor e os clientes.

1.3 METODOLOGIA

Para responder os objetivos propostos, o processo de coleta de dados e das informações é realizado através de pesquisa bibliográfica com revisão de literatura sobre o assunto, consulta a dados digitalizados disponíveis na Internet que contemplem a produções na área da educação e tecnologia da informação, discussão teórica sobre o tema. Segue-se a seguinte estrutura:

- Pesquisar sobre sistemas distribuídos, comunicação distribuída, protocolos de comunicação, arquitetura cliente servidor, modelos de referências e *middleware* em livros, artigos e demais materiais disponíveis na Internet.
- Estudar o ambiente virtual Subvese para a implementação do protótipo. Modelagem, documentação.
- Dividir o Subverse em dois ambientes, o cliente e o servidor.
- Implementar o *middleware* de comunicação tanto no lado do cliente como no lado do servidor.
- Testar a comunicação para afirmar a sua aplicabilidade.

1.4 ESTRUTURA DO TRABALHO

De maneira a atingir os objetivos propostos, este trabalho está estruturado da seguinte maneira:

O Capítulo 1 apresenta a introdução junto com a problemática, os objetivos, a justificativa do tema escolhido e a metodologia utilizada no desenvolvimento deste trabalho.

O Capítulo 2 apresenta a fundamentação teórica da computação distribuída, onde são abordados sistemas fracamente e fortemente acoplados, sistemas distribuídos, sistemas operacionais de rede, comunicação distribuída, estrutura cliente-servidor, protocolos de comunicação, modelos de referência e *middleware*.

O Capítulo 3 apresenta as características e fundamentação sobre o ambiente Subverse.

O Capítulo 4 apresenta toda a estrutura do middleware implementado, com descrição dos métodos utilizados tanto no lado do cliente quanto do lado do servidor.

O Capítulo 5 conclui este trabalho, discorrendo sobre os resultados alcançados e contribuições da pesquisa e as propostas de continuação deste trabalho.

2 FUNDAMENTOS DA COMPUTAÇÃO DISTRIBUÍDA

O presente capítulo apresenta um estudo sobre computação distribuída e seus respectivos meios de comunicação a fim de obter conhecimento mais específico na área e encontrar uma solução apropriada para o desenvolvimento do *middleware* de comunicação proposto. Primeiramente são discutidas algumas definições de sistemas distribuídos, alguns dos principais tipos de comunicação utilizados atualmente, protocolo de comunicação TCP/IP, modelos de referência, e em seguida a definição de *middleware*.

2.1 SISTEMAS FRACAMENTE E FORTEMENTE ACOPLADOS

Deitel (2005) explica que os sistemas com vários processadores podem ser divididos em sistemas fortemente acoplados e sistemas fracamente acoplados, dependendo de como os recursos são compartilhados e gerenciados.

Segundo Machado (2002) a principal diferença entre eles é que nos sistemas fortemente acoplados uma memória principal é compartilhada entre os processadores e os dispositivos são gerenciados por único sistema operacional, enquanto que nos fracamente acoplados, cada sistema funciona de forma independente, tendo apenas uma linha de comunicação em comum para troca de mensagens entre os processadores.

A difícil gerência de uma memória compartilhada por todos os processadores e a redução da flexibilidade com o acréscimo de processadores num sistema fortemente acoplado sugere a escolha de um sistema fracamente acoplado, onde cada componente funciona independentemente, tornando o sistema mais flexível e escalável, entretanto menos eficiente, devido à troca de mensagens por uma linha de comunicação ser mais lenta que utilizar memória compartilhada.

Os sistemas fracamente acoplados são subdivididos, segundo Maia (2002) em sistemas distribuídos e em sistemas operacionais de rede, dependendo do grau de interação dos *hosts* da rede.

2.1.1 SISTEMAS DISTRIBUÍDOS

Segundo Silberschatz (2001) um sistema distribuído é uma coleção de processadores que não compartilham memória nem um relógio e fornecem a seus usuários acesso aos vários recursos compartilhados, permitindo maior velocidade de computação, assim como melhor disponibilidade e confiabilidade de dados.

Tanenbaum (2003a) discute que o acoplamento fraco dos computadores em um sistema distribuído é uma vantagem devido à grande diversidade de aplicações possíveis e ao mesmo tempo uma desvantagem, pois a falta de um modelo de uma plataforma comum dificulta sua programação.

Tanenbaum (2000) argumenta que um sistema distribuído é dividido em nodos, que são computadores completos, com todos os periféricos, como CPU, RAM, uma interface de rede e talvez um disco rígido para paginação. Estes nodos podem estar espalhados ao redor do mundo, executar sistemas operacionais diferentes, ter seus próprios sistemas de arquivos e estarem sujeitos a gerenciamentos diferentes.

2.1.2 SISTEMAS OPERACIONAIS DE REDE

Segundo Machado (2002), os sistemas operacionais de rede permitem que um host compartilhe seus recursos, como uma impressora ou diretório, com os demais hosts da rede.

Um exemplo deste tipo de sistema são as redes locais, onde uma estação pode oferecer serviços de arquivos e impressão para as demais estações da rede, dentre outros serviços. Os usuários têm o conhecimento dos hosts e seus serviços.

2.2 COMUNICAÇÃO DISTRIBUÍDA

Segundo Deitel (2005), processos que estão localizados em sistemas finais diferentes se comunicam através de troca de mensagens por meio da rede de computadores, onde primeiramente um processo cria e envia mensagens e outro processo as recebe e possivelmente responde com outra mensagem.

Existem várias estratégias diferentes que permitem a comunicação entre aplicações distribuídas. Alguns dentre os diversos métodos existentes utilizados são soquetes, *RPC*, *RMI* e *CORBA*.

2.2.1 SOQUETES

Segundo Silbershatz (2001), os soquetes são as extremidades de um canal de comunicação por onde um processo se comunica com outro em uma rede. Geralmente é utilizado em arquiteturas cliente servidor, contendo um endereço IP e o numero de uma porta, por onde o servidor fica esperando por requisições de clientes. Recebido um pedido, o servidor aceita uma conexão.

Uma aplicação de rede, segundo Deitel (2005), consiste basicamente em pares de processos que se comunicam por meio da troca de mensagens por meio de uma rede, onde o processo que inicia a comunicação é considerado cliente e o processo que fica esperando a chegada de mensagens é o servidor. A Figura 1 mostra onde está situado o soquete em uma aplicação distribuída entre dois sistemas finais diferentes.

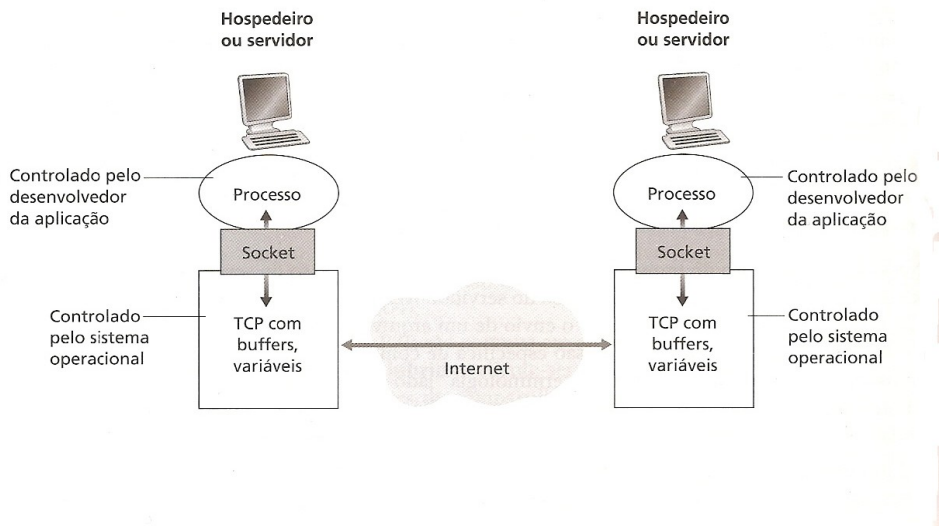

Figura 1 - Localizaçao do soquete em uma aplicação distribuída.
Fonte: (Kurose, 2006, pg. 62)

Os soquetes, por ser um método de comunicação rápido e de simples implementação, são uma ótima escolha para comunicar processos em aplicações que não apresentam uma grande complexidade, porém é preciso um controle e tratamento do envio e recebimento das mensagens por parte do *middleware* de comunicação.

2.2.2 RPC

O método de comunicação *RPC*, ou chamada a procedimento remota, conforme Kurose (2006), permite uma abordagem estruturada, de alto nível, onde um computador cliente chama um procedimento remoto, junto com os parâmetros desse procedimento, e o computador servidor executa o procedimento e envia o resulta para o computador cliente. Isso permite que o cliente chame um procedimento remotamente parecendo que está localmente.

Porém o RPC apresenta alguns problemas, como a falta de segurança e confiabilidade, não suporta variáveis globais, transparência e capacidade são limitadas e a sua implementação é complexa. (Kurose, 2006)

2.2.3 RMI

O RMI, ou invocação de método remoto, é um recurso Java que permite que um computador cliente invoque um método de um objeto remotamente. A principal diferença entre o RMI e o RPC, é que o primeiro permite a transmissão e recebimento de objetos entre os processos remotos. (Kurose, 2006)

RMI é uma ótima solução para a comunicação de sistemas distribuídos onde tanto o cliente quanto o servidor forem escritos na linguagem Java, devido ao suporte orientado a objetos que permite a envio de objetos por meio da troca de mensagens entre métodos remotos.

2.2.4 CORBA

CORBA, ou *Common Object Request Broker Architecture*, é um padrão aberto que foi elaborado para habilitar interoperação entre programas em sistemas heterogêneos. Suporta objetos como parâmetros em procedimentos remotos durante a comunicação inter processos como o RMI, porém é independente de linguagem e de sistema, diferentemente do RMI que é especifico para a linguagem de programação Java. (Silberschatz, 2001).

Aplicações escritas em linguagens de programação diferentes e em sistemas operacionais diferentes podem comunicar entre si por meio da *IDL*, *Interface Definition Language* ou linguagem de definição de interface, que permite aos programadores definir os parâmetros e métodos da interface. (Kurose, 2006)

Aplicações escritas em qualquer linguagem podem se comunicar por meio de CORBA seguindo a especificação *IDL* de cada objeto.

2.3 TIPOS DE REDES

A comunicação de sistemas distribuídos necessita que os computadores estejam interligados em rede. Segundo Coulouris (2001) “sistemas distribuídos utilizam redes locais, redes de longa distância e internet para comunicação”.

Conforme Deitel (2005) uma LAN, ou rede de área local, tem alcance limitado, podendo estar em ambientes como uma sala ou até um campus. Utiliza equipamentos de interconexão de alta velocidade com protocolos otimizados e não apresentam problemas de colisão. Dantas (2002) cita que uma LAN fornece uma comunicação fácil e de alta velocidade entre dispositivos, processadores numa área pequena, como um prédio.

Um tipo de LAN muito difundido é a Ethernet, que de acordo com Dantas (2002) foi desenvolvida por dois engenheiros no centro de pesquisa da Xerox, com uma velocidade inicial de 3 Mbps e recomendada como rede padrão em 1980 com velocidade de 10 Mbps. A melhora na relação custo-benefício e o avanço da tecnologia de telecomunicações aumentaram a taxa de transmissão para outros padrões, como *Fast Ethernet* de 100 Mbps, *Gigabit Ethernet* de 1 Gbps e mais recentemente, segundo INTEL (2003), 10 *Gigabit Ethernet* com taxas ainda mais elevadas de transmissão, chegando à velocidade de 10 Gbps e expandindo o espaço da ethernet tornando compatível com WAN.

A WAN, ou rede de longa distância, possui taxas de transferência menor que uma LAN, porém permite a comunicação de nodos situados em cidades, estados, países ou até mesmo continentes diferentes. A comunicação entre nodos é intermediada por roteadores, onde as mensagens e pacotes sofrem atrasos de milissegundos em cada ponto (Colouris, 2001).

Outra forma de realizar a comunicação entre os processos é por intermédio da Internet. Kurose (2005) descreve a internet como “uma infra-estrutura de rede que provê serviços para aplicações distribuídas”. Kurose (2005) explica ainda que milhares de computadores,

equipamentos de computação e até mesmo sistemas domésticos eletrônicos e de segurança, como uma torradeira ou câmera de vigilância, são interconectados pela internet, onde esses equipamentos estão divididos como sistemas finais ou hospedeiros.

Por estarem na periferia da rede, os computadores pessoais, dispositivos móveis, servidores, entre outros equipamentos são denominados sistemas finais. Aplicações distribuídas como compartilhamento de arquivos, correio eletrônico, áudio e vídeo em tempo real e jogos distribuídos podem trocar dados entre si através da internet (Kurose, 2005).

O fato de a internet permitir que um sistema distribuído se comunique mesmo que seus nodos estejam em ambientes físicos separados facilita uma futura expansão, alem de que a velocidade de transmissão dos dados na internet vem crescendo constantemente e por ser uma tecnologia muito difundida, é de fácil instalação e manutenção.

2.4 ARQUITETURA CLIENTE SERVIDOR

Um cliente se comunica com o servidor criando um soquete se conectando a porta onde o servidor está escutando. Ele pode simplesmente estabelecer uma conexão *telnet* com a porta 5155. Por exemplo, o comando *telnet 127.0.0.1 5155* abre um soquete e solicita uma conexão com o servidor no endereço 127.0.0.1 na porta 5155 (Silberschatz, 2001).

Os clientes são utilizados pelos jogadores para jogar o MMOG, ou *Massive Multiplayer Online Game*. Cada cliente interage com um servidor de jogo e fornece-lhe atualizações em atividades dos jogadores e recebe atualizações nas atividades vizinhas. O servidor de jogo é o software que armazena o estado do mundo do jogo e coordena a atividade dos jogadores no jogo. (Balan, 2005)

2.5 PROTOCOLOS DE COMUNICAÇÃO

Segundo Riso (2004), os protocolos de comunicação constituem num conjunto de

regras e convenções que permite a troca organizada de informações entre diferentes entidades.

A Internet constitui, hoje, uma ferramenta que permite a integração e de difusão de conhecimento, onde estudantes, pesquisadores e o publico em geral possui o acesso rápido a informações, de modo barato e atrativo.

O serviço de comunicação oferece às entidades de protocolo, um conjunto de recursos, para que essas entidades possam se comunicar através da troca ordenada de mensagens. Esses serviços são possíveis através das primitivas de serviço *Request, Indication, Response* e *Confirm* e ocorre em diferentes pontos de acesso disponibilizados pelo Provedor de Serviço correspondente. (Riso, 2004)

2.6 MODELOS DE REFERÊNCIA

A comunicação entre o cliente e o servidor necessita de protocolos padrões, utilizados tanto pelo sistema operacional quanto pelas placas de redes dos mais diversos fabricantes, para que uma mensagem enviada por um emissor possa ser recebida por um destinatário.

Segundo Dantas (2002), os modelos de referência dos protocolos representam uma abstração contendo especificações de como o ambiente de comunicação deve funcionar. Nessa estrutura existe desde o detalhamento da função de cada nível até a relação entre as interfaces das camadas e dos protocolos, porém não existe menção de uma implementação de um protocolo específico.

O modelo de referência mais utilizado e conhecido atualmente, no ambiente Internet, é o TCP/IP (*Transmition Control Protocol/Internet Protocol*), onde o modelo original foi projetado em quatro camadas, entretanto com a ascensão do ambiente WWW (World Wide Web), alguns autores dividiram a camada de acesso ao meio na camada física e de enlace (Dantas, 2002).

Outro modelo de referência utilizado é o RM-OSI (*Reference Model – Open Systems*

Interconnection), proposto pela ISO (*International Organization for Standardization*), devido à falta de interesse e aceitação do protocolo TCP/IP por parte dos grandes fabricantes de computador, que era rotulado como protocolo eficiente para universidades e centros de pesquisas (Dantas, 2002).

Um comparativo entre as camadas presentes entre os modelos de referencias citados é apresentado na Figura 2.

	RM-OSI		RM- TCP/IP		RM- TCP/IP Modificado
7	Aplicação	4	Aplicação	5	Aplicação
6	Apresentação				
5	Sessão				
4	Transporte	3	Transporte	4	Transporte
3	Rede	2	Inter-Rede	3	Rede
2	Enlace	1	Subcamada de acesso ao meio	2	Enlace
1	Físico			1	Físico

Figura 2 - Comparação dos modelos de referencia RM-OSI e RM-TCP/IP.
Fonte: (Dantas, 2002, pg. 116)

Segundo Tanenbaum (2003b), a princípio, o modelo TCP/IP não diferenciava claramente os conceitos de serviço, interface e protocolo e não faz distinção entre as camadas física e de enlace de dados. Onde, a camada física está relacionada às características de transmissão, tanto do fio de cobre, fibra ótica e sem fio, e a camada de enlace de dados está relacionada em delimitar o início e final dos quadros e que eles cheguem ao destino com o grau desejado de confiabilidade.

O fato da camada 1 do modelo de referência TCP/IP não possuir uma definição bem aceita resultou no surgimento de diversas soluções proprietárias no mercado para as redes locais. A fim de solucionar a falta de padronização por parte das empresas, a Sociedade de Computação do Instituto de Engenheiros Elétricos e Eletrônicos (*Computer Society* do *IEEE*) publicou uma série de padrões que ficou conhecido como modelo IEEE 802 (Dantas, 2002).

2.7 MIDDLEWARE

Sistemas distribuídos são usualmente compostos por computadores heterogêneos, ou seja, que executam sistemas operacionais diferentes, que utilizam hardwares diferentes e utilizam diferentes arquiteturas e protocolos de redes para se comunicarem. Para habilitar interações de uma aplicação entre esses computadores é preciso que cada computador tenha o *middleware* instalado. (Deitel, 2005)

Com freqüência, uma camada de software sobre o sistema operacional, denominada *middleware*, é responsável pela implementação, em um sistema distribuído, de um modelo que apresenta aos usuários o conjunto de computadores independentes parecerem ser, como se fosse um único sistema coerente. Um exemplo bem conhecido de sistema distribuído é a *world wide web*, na qual tudo tem a aparência de um documento (uma página *web*). (Tanenbaum, 2003b)

Segundo Deitel (2005), o programador tende a gastar muito tempo e a cometer erros para fornecer as rotinas de conversão de mensagens entre os computadores. Softwares conhecidos como middleware ajudam a fornecer portabilidade, transparência e interoperabilidade em sistemas distribuídos. A Figura 3 mostra a localização do middleware em um sistema distribuído.

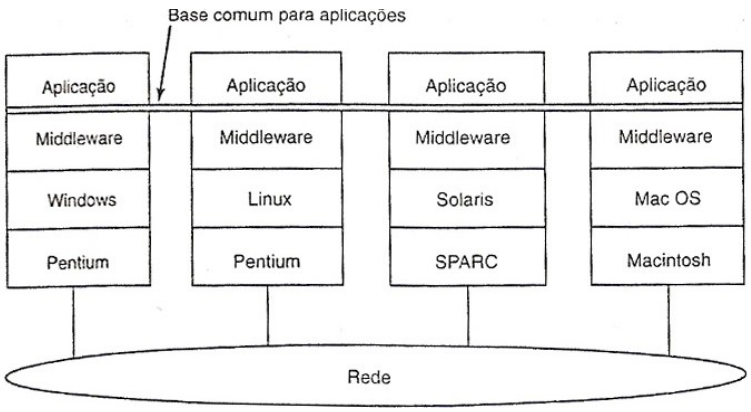

Figura 3- Posicionamento da *middleware* em um sistema distribuído.

Uma das vantagens do uso de *middleware* é que sua utilização facilita a comunicação e cooperação entre os vários componentes de um sistema distribuído ocultando detalhes de implementação e protocolos comuns que formam a espinha dorsal de muitos sistemas e faz com que chamadas a procedimentos em outro computador pareçam chamadas a procedimentos locais. (Deitel, 2005)

Colouris (2001) define *middleware* "como uma camada de software com propósito de mascarar a heterogeneidade e promover um modelo de programação conveniente para a aplicação dos programadores". Assim, o desenvolvedor não precisa se preocupar em como a mensagem vai ser enviada ou recebida, simplesmente vai precisar chamar uma função do lado do cliente e criar uma função correspondente no lado do servidor que processe a mensagem.

Uma desvantagem é quando aplicações distribuídas invocam inteiramente os serviços fornecidos pelo *middleware* disponível para auxiliar no compartilhamento comunicação e dados, tornando a aplicação dependente do *middleware* e assim menos flexível. (Colouris, 2001).

O estudo dos fundamentos da computação distribuída ajudou a criar uma base teórica para o desenvolvimento do *middleware* de comunicação. Foi possível analisar: o funcionamento e as principais características de sistemas distribuídos. As diferenças entre os métodos de comunicação distribuída, assim com onde podem ser aplicadas. Os tipos de rede mais utilizados em jogos *on-line*. A arquitetura cliente servidor e sua utilização nos jogos multiusuários. Os principais modelos de referência, com protocolos de comunicação para utilizar no *middleware*. E por fim, algumas definições de *middleware*, com vantagens e desvantagens de seu uso.

3 SUBVERSE

O Subverse é um dos projetos do LABSIB – Laboratório de Sistemas Inteligentes das Faculdades Barddal – comandados pelo Prof. Msc. Saulo Popov Zambiasi. Este projeto visa apresentar aos alunos um ambiente de aprendizado para o desenvolvimento de aplicações que se utilizam de: (i) computação gráfica, para a visualização do mundo virtual e dos agentes inseridos neste; (ii) técnicas de desenvolvimentos de jogos de computador; (iii) sistemas distribuídos com agentes conectados ao ambiente remotamente e (iv) Inteligência Artificial com a implementação de algoritmos inteligentes nos agentes que devem interagir no mundo virtual. Algumas dessas implementações envolvem técnicas de inteligência artificial distribuída, redes neurais artificiais, lógica, lógica difusa, sistemas especialistas e outros.

No meio acadêmico, nem sempre é interessante que o aluno apenas trabalhe em uma plataforma pronta, apenas interagindo no ambiente ou, quando possível, desenvolvendo algoritmos para gerenciar os agentes do ambiente. É interessante que o aluno possa participar do processo de desenvolvimento do ambiente.

Este aprendizado apresenta ao aluno desafios reais, tais como: a prática da modelagem de sistemas; aprendizado de técnicas de gerenciamento de mensagens e computação distribuída; aprendizado no desenvolvimento de interfaces gráficas e jogos de computadores, entre outros.

A interface visual é um ambiente gráfico baseado em jogos de computadores para fornecer aos usuários uma forma de observar visualmente os agentes no mundo virtual.

Algumas características são necessárias para que os agentes possam interagir com o mundo virtual. A arquitetura do Subverse é composta por sistemas e módulos que permitem a interação dos agentes, como mostrados na Figura 4.

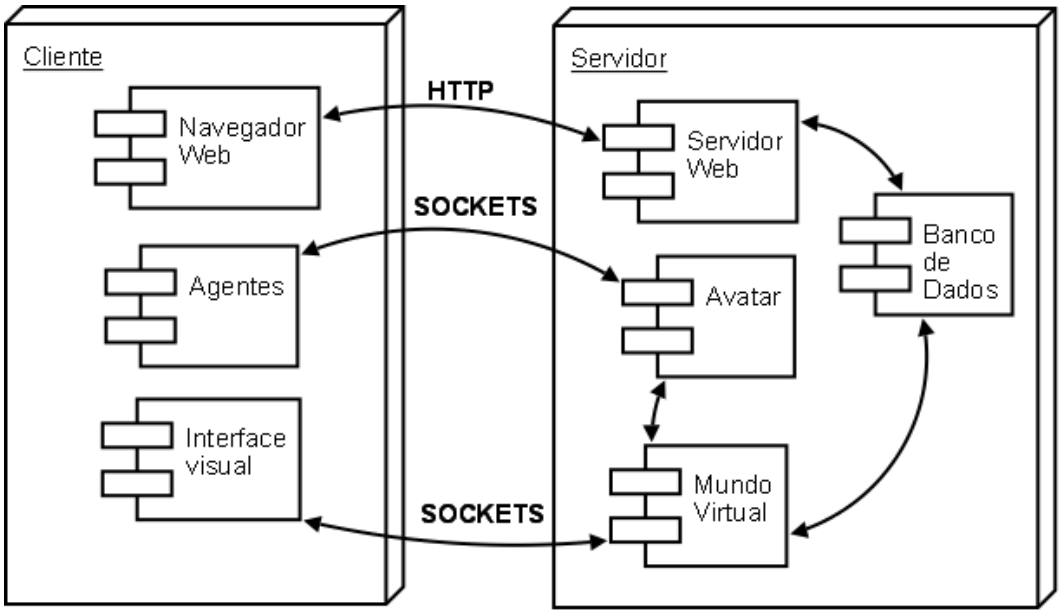

Figura 4 - Arquitetura do Subverse

Toda a parte de comunicação entre o Cliente e o Servidor ainda não foi implementada, tornando viável a implementação do *middleware* de comunicação proposto neste trabalho.

A Figura 5 mostra onde é aplicado o *middleware* de comunicação na arquitetura do Subverse.

Figura 5 - *Middleware* de comunicação aplicado no Subverse

A fim de ter uma visão de como o Subverse está estruturado, na Figura 6 é mostrado o diagrama de classes, com os métodos mais relevantes que se encontram implementados no Projeto original.

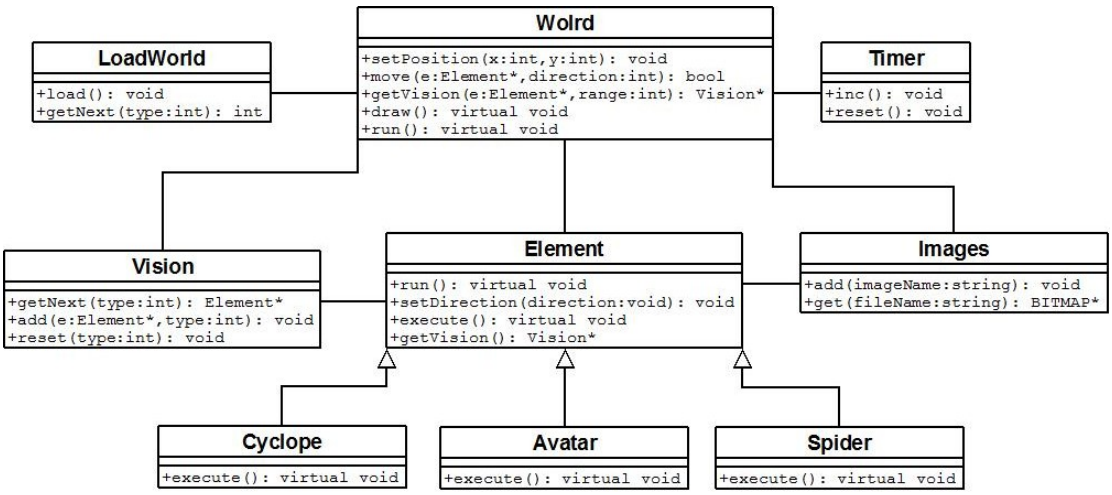

Figura 6 – Diagrama de classes do Subverse

Nos subcapítulos subseqüentes são descritos os principais módulos existentes no Subverse, com suas estruturas e funcionamento, a fim de um melhor entendimento do algoritmo do Subverse, para a implementação do *middleware* de comunicação.

3.1 CLASSE WORLD

A função *Main* do Subverse inicializa todas as configurações de áudio, vídeo, teclado, mouse e temporizadores da biblioteca gráfica ALLEGRO[1] e logo em seguida estância um objeto do tipo *World*.

World é a classe principal do Subverse, onde toda a interface com o usuário é criada e atualizada a cada ciclo do programa. Pode-se dizer que é o Mundo Virtual, propriamente dito, pois é a base de execução para a leitura das informações persistentes e onde os agentes irão interagir. Este é caracterizado como o núcleo de processamento e comunicação dos agentes.

Ao ser instanciada, a classe *World* chama a função *loadWorld*(*fileName*) onde é carregado o mapa com nome *fileName*, localizado no diretório do programa e apresenta extensão do tipo ".map". Como o sistema ainda não possui suporte a banco de dados, a

[1] Allegro é uma biblioteca de programação de jogos para C/C++ distribuída gratuitamente, disponível no endereço http://alleg.sourceforge.net/.

configuração do mundo virtual assume esse arquivo que contêm o tamanho do mapa e duas matrizes de dados referentes aos elementos que compõe o mapa. A primeira matriz é referente aos elementos que podem interagir diretamente com o jogador. A segunda matriz é referente aos elementos de cenário que podem servir apenas de obstáculo ou de ajuda para o jogador.

Após o mapa ser inicializado, onde foram instanciados todos os elementos do mapa, é chamada a função "run()", onde o Subverse é executado dentro de um loop enquanto a tecla ESC não for pressionada. Dentro da função *run*(), a matriz do mapa que foi carregado é percorrida e em cada posição é verificada se existe elemento. Se existe elemento, esse elemento é executado através da chamada da função *run*() do elemento correspondente. Senão verifica a próxima posição do mapa. Depois de analisar todas as posições do mapa, o Ambiente é atualizado, sendo redesenhado com as interações realizadas pelos elementos. Assim um ciclo do Mundo foi executado e logo sem seguida começa outro ciclo, como mostra o fluxograma representado na Figura 7.

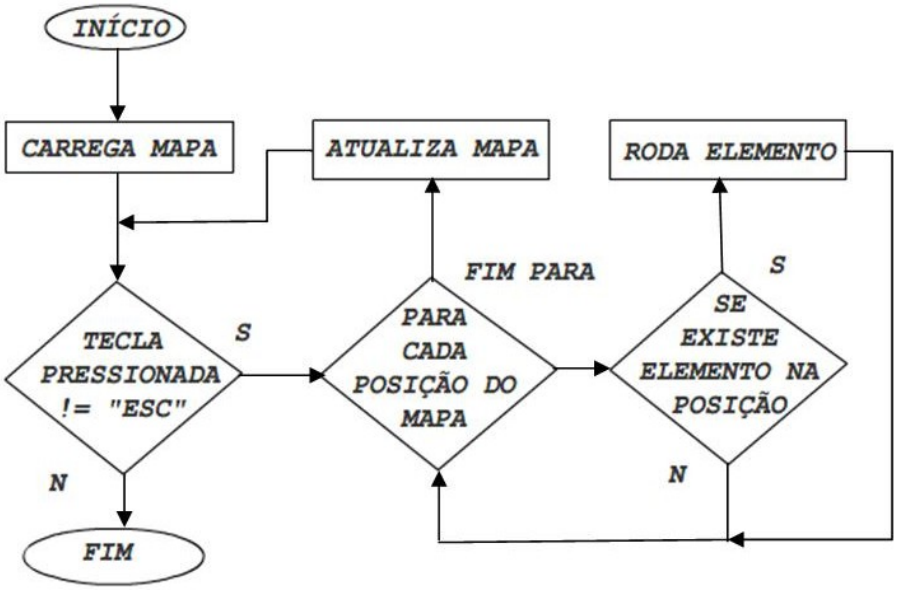

Figura 7 – Fluxograma do funcionamento da classe *World*

Essa classe é fundamental para o *middleware* de comunicação, pois é nela que os clientes e o servidor executaram os agentes do Subverse. Porém, quem envia e recebe as mensagens são os agentes, ou elementos do Subverse. Assim, a seguir é descrito o funcionamento da classe *Element* e de suas classes filhas.

3.1 CLASSE ELEMENT

A classe *Element* é uma classe genérica, onde possui variáveis que armazenam diversas características dos agentes assim como funções para que os elementos possam agir no Mundo Virtual.

Existem diversas características do elemento, como a imagem do tipo BITMAP correspondente, o tipo, a direção em que está virado, a vida, o campo de visão, se esta andando, se esta vivo, se esta sobre ataque, se é uma criatura ou parte do cenário, entre outras.

Várias outras classes herdam essas características, para que elementos com comportamentos distintos possam integrar o Mundo Virtual. Segundo Benin (2007) os elementos são vistos como agentes no Mundo Virtual e estão divididos em:

- **Avatar**: Agente controlado pelo próprio usuário que faz uso do teclado para movimentar o personagem.
- **Ciclope**: Agente controlado por computador que possui características particulares iniciais como a habilidade de reconhecer um agente do tipo Avatar dentro do cenário virtual.
- **Spider**: agente controlado por computador que possui características particulares iniciais como a habilidade de reconhecer um agente do tipo Ciclope dentro do cenário virtual.
- **Elementos de Cenário**: Os elementos de cenário são todos os que compõem as características físicas do ambiente virtual. Tais elementos são compreendidos

como árvores, fogueiras, água, terra, grama, paredes e ladrilhos. As características dos elementos de cenário são duas: Prover obstáculos de colisão aos agentes e também ser uma fonte de comida aos mesmos.

A Figura 8 mostra os elementos que se movem e podem interagir com os jogadores.

Figura 8 – Elementos ativos do Subvrse. 1: Avatar; 2: Aranha e 3:Ciclope.

Com o estudo realizado no código do Subverse, foi possível entender seu funcionamento e definir onde e como pode ser implementado o *middleware* de comunicação proposto. Inicialmente o Subverse será dividido em duas partes, a primeira considerada Cliente e a segunda Servidor. Em seguida, o Avatar será modificado a fim de permitir a interface com o *middleware*. E por fim, será implementada a parte do middleware, onde o Cliente deve conectar no Servidor e depois iniciar a troca de mensagens entre eles.

4 IMPLEMENTAÇÃO DO *MIDDLEWARE*

O presente capítulo tem como objetivo apresentar um protótipo parcial de um *middleware* de comunicação para jogos de computador multiplayer, utilizando como estudo de caso o Projeto Subverse para realização de testes e avaliação do mesmo.

Para o desenvolvimento do protótipo parcial foi utilizado à linguagem de programação orientada a objetos C++. Alguns fatores foram decisivos para a utilização desta linguagem:

- O projeto Subverse foi inicialmente implementado na mesma linguagem;
- Maior conhecimento em relação a outras linguagens de programação;
- Apresenta melhor desempenho em relação a outras linguagens orientadas a objeto.
- O compilador utilizado no desenvolvimento é de distribuição livre[2];
- Apresenta todas as características das linguagens orientadas a objeto.

A escolha de soquetes para realizar a troca de mensagens do *middleware* em vez de um dos outros métodos de comunicação, citados no capítulo 2, se deve a diversos fatores:

- Como utiliza chamadas diretas ao sistema operacional, é muito mais rápido que os outros métodos, o que é fundamental em jogos *multiplayer*.
- Independência de linguagem programação. Basta a linguagem oferecer bibliotecas para sua implementação.
- Utilização de protocolos muito difundidos. Como é o caso do TCP/IP.

Para que o *middleware* funcione, é necessário que exista pelo menos um computador como cliente, para enviar mensagens, e um computador como servidor, para receber as

[2] O compilador de distribuição livre utilizado é o Dev-C++ 5.0 beta 9.2 (4.9.9.2) with Mingw/GCC 3.4.2, disponível no endereço http://www.bloodshed.net/dev/devcpp.html.

mensagens e processá-las e retornar ao cliente. Ambos devem estar conectados obrigatoriamente na mesma rede.

4.1 O LADO DO SERVIDOR

O *middleware* foi implementado no Servidor através da classe "Servidor". Nela estão todas as funções de inicialização do Servidor e de envio e recebimento de mensagens através de soquetes de comunicação. O Subverse, ao instanciar um objeto da classe *World* chama a função "ServidorInitConnect( )" e assim inicializa o soquete, que possibilita iniciar as trocas de mensagens. A Figura 9 explica na forma de algoritmo a inicialização do Servidor que ocorre dentro da função "ServidorInitConnect( )".

```
Servidor inicia conexão. [WSAStartup( )]
Cria um soquete. [socket( )]
Associa uma porta e endereço para o soquete. [bind( )]
Coloca o soquete em modo passivo, para escutar a porta. [listen( )]
```

Figura 9 – Algoritmo da inicialização do Servidor.

No Windows, para a utilização dos soquetes, é necessário chamar a função *WSAStartup*( ), que especifica a versão do Windows Socket utilizada e aponta para uma estrutura do tipo *WSADATA*, que armazena detalhes da implementação do Windows Socket.

Em seguida é criado um soquete com a função *socket*( ), onde é especificada a família de endereço, o tipo do soquete e o protocolo a ser utilizado com o soquete. A família de protocolos escolhida foi a *AF_INET*, que utiliza a família de endereço do tipo *IPv4*. O tipo do soquete foi o *SOCKET_STREAM*, para transferência orientada á conexão. E o protocolo foi o *IPPROTO_TCP*, para o protocolo TCP. Outros parâmetros podem ser escolhidos, dependendo da aplicação do soquete.

Depois de inicializado, é preciso associar uma porta e um endereço para o soquete utilizando a função *bind*( ). É passado como parâmetro o soquete a ser associado, uma

estrutura do tipo *sockaddr*, contendo informações da porta, do endereço e da família de protocolo, e o tamanho dessa estrutura do tipo *sockaddr*.

Uma vez configurado, o soquete é colocado em um estado no qual fica escutando por conexões com a função *listen*( ). O soquete a ser colocado nesse estado e o número e o tamanho máximo da fila de conexões pendentes são passados como parâmetros nessa função. Foi utilizado *SOMAXCONN*, que atribui o número do tamanho máximo para a fila de conexões pendentes.

Foi criada uma nova classe com nome Jogador no Servidor que herda as características da classe *Element*. Essa classe foi criada para se diferenciar do elemento do tipo *Avatar*, que pode ser controlado diretamente do servidor, enquanto o elemento do tipo Jogador só pode ser controlado por um Cliente conectado no Servidor

O algoritmo que representa conexão de clientes no Servidor e recebimento e envio de mensagens é explicado na Figura 10 e está implementada dentro da classe *World*, com exceção do recebimento, processamento e envio de mensagens, que está implementada na classe Jogador.

```
Enquanto (Servidor Rodando)
      Verifica se uma conexão foi solicitada por um cliente.
      Se conexão foi solicitada
            Estabelece conexão.
      Fim Se
      Se conexão estabelecida
            Cria novo elemento do tipo Jogador( )
            Retorna uma mensagem de confirmação para o Ciente
      Fim Se
      Se existe elementos do tipo Jogador( )
            Verifica se tem mensagem do Cliente
            Se existe mensagem
                  Processa a mensagem
                  Envia resposta para o Cliente
            Fim Se
      Fim Se
      Se uma conexão foi interrompida
            Remove o elemento do tipo Jogador( ) correspondente
      Fim Se
Fim Enquanto
```

Figura 10 – Algoritmo do funcionamento do Servidor

Depois que a parte do servidor foi configurada, o jogo começa a ser executado. A cada ciclo de atualização do ambiente é chamada a função ”ServidorVerificaConnect( )”, onde a função *select*( ), verifica se um cliente enviou um pedido de estabelecimento de conexão.

O uso da função *select*( ) é necessário pois a função *accept*( ), que aceita uma conexão, é do tipo bloqueante, ou seja, o programa fica travado até que um pedido de conexão seja enviado por um cliente com a função connect( ). O mundo virtual não pode ficar esperando por um cliente a cada novo ciclo do programa, por isso, a função *select*( ) permite que essa espera seja de no máximo de alguns milissegundos e não até que um cliente conecte.

O primeiro parâmetro utilizado na função *select*( ) é 0, pois só é necessário para que o soquete seja compatível com soquetes do tipo Berkeley. O segundo é um ponteiro para uma variável do tipo FDS, no qual é associada com o soquete com a função *fd_set( )*, e permite que o soquete seja verificado como leitura. O terceiro é nulo, pois é um ponteiro para uma variável do tipo FDS, porém associada com um soquete para permitir a escrita. O quarto parâmetro é nulo também, pois é um ponteiro opcional que associa um soquete á uma variável para verificar possíveis erros. O quinto e mais importante parâmetro é uma variável do tipo *struct timeval* onde é guardado o tempo em microssegundos em que a função vai ficar travada.

Em seguida é analisada e aceita uma conexão com a função *accept*( ). Se a conexão foi estabelecida com sucesso, um elemento do tipo Jogador( ), que representa o Cliente no mundo virtual situado no Servidor, é criado no servidor e é enviada uma mensagem para o cliente confirmando a conexão.

Se existem elementos do tipo Jogador( ) é verificado se foi recebida alguma mensagem do Cliente correspondente a esse elemento com a função “ServidorVerificaNovaMensagem( )“, onde um *select*( ) verifica se alguma mensagem chegou no soquete e recebe a mensagem com a função *recv*( ).

Se tiver alguma mensagem, ela é acessada com a chamada da função "getLastMessage( )" e então processada pelo elemento do tipo Jogador. Em seguida é enviada uma mensagem com a resposta para o Cliente com a função "ServidorRetornaMensagem(bool b)", onde a função *send*( ) envia a resposta que é do tipo verdadeiro ou falso.

Uma vez que a conexão foi interrompida, seja por solicitação do Cliente ou por algum erro qualquer, o elemento correspondente é eliminado.

A Figura 11 mostra o diagrama de classe das classes Servidor e Jogador que foram implementadas.

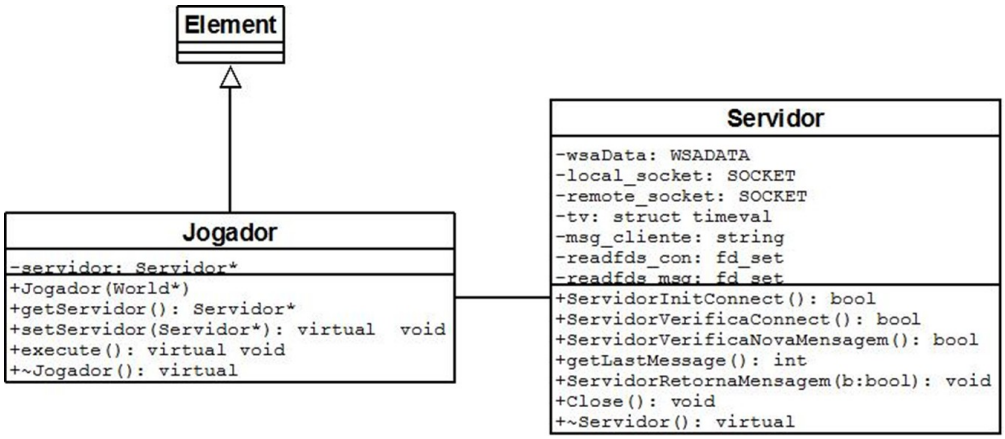

Figura 11 – Diagrama de classe das classes implementadas no Servidor

Na classe Jogador, as funções getServidor( ) e setServidor( ) servem para que o jogador possa acessar o Servidor que foi criado no inicio da execução do Subverse na classe World. A verificação de novas mensagem e o processamento das mesmas é realizado dentro da função execute( ) que sobrescreve a função da classe Element.

Na classe Servidor, é mostrada as variáveis globais utilizadas e funções que permitem a troca de mensagens com o Cliente. A função getLastMessage( ) retorna um inteiro, que corresponde a direção para onde o Cliente deseja se mover. A função ServidorRetonaMensagem( ) envia resposta do processamento para o Cliente.

A seguir é descrita a implementação realizada na parte do Cliente.

4.2 O LADO DO CLIENTE

Semelhante ao lado do Servidor, o Cliente é inicializado quando um objeto da classe *World* é instanciado. Em seguida, é instanciado um objeto da classe Cliente e é chamada a função "ClienteInitConnect(char* ip)", onde é passado por parâmetro o IP do Servidor. A Figura 12 explica na forma de algoritmo a inicialização do Cliente que ocorre dentro da função "ClienteInitConnect(char* ip)".

```
Cliente inicia conexão. [WSAStartup( )]
Cria um soquete. [socket( )]
Conecta no Servidor. [connect( )]
```

Figura 12 – Algoritmo da inicialização do Cliente

O cliente só consegue estabelecer conexão se o Servidor estiver iniciado e o soquete do mesmo estiver inicializado.

No Windows, para a utilização dos soquetes, é necessário chamar a função *WSAStartup*( ), que especifica a versão do Windows Socket utilizada e aponta para uma estrutura do tipo *WSADATA*, que armazena detalhes da implementação do Windows Socket.

Em seguida é criado um soquete com a função *socket*( ), onde é especificada a família de endereço, o tipo do soquete e o protocolo a ser utilizado com o soquete. A família de protocolos escolhida foi a *AF_INET*, que utiliza a família de endereço do tipo *IPv4*. O tipo do soquete foi o *SOCKET_STREAM*, para transferência orientada á conexão. E o protocolo foi o *IPPROTO_TCP*, para o protocolo TCP. Outros parâmetros podem ser escolhidos, dependendo da aplicação do soquete.

Após a criação do soquete, é realizado uma solicitação com a função *connect*( ). Os parâmetros passados nessa função são o socket previamente criado, um ponteiro para uma estrutura do tipo *sockaddr* contendo informações da porta, do endereço e da família de protocolo e o tamanho dessa estrutura do tipo *sockaddr*.

Uma vez enviada a solicitação de conexão com a função *connect*( ), o Cliente espera a resposta de confirmação do Servidor. Em seguida o jogo é inicializado no lado do Cliente e ele está pronto para se comunicar com o Servidor.

O algoritmo que representa o funcionamento do Cliente é explicado na Figura 13 e está implementada na classe Avatar, com exceção do loop Enquanto situado na classe *World*.

```
Enquanto (Conectado no Servidor OU Jogo Rodando)
      Verifica se alguma ação foi realizada pelo Cliente
      Se foi solicitado alguma ação pelo Cliente
            Uma função equivalente a ação é chamada
            O middleware trata essa função e envia uma mensagem
            O middleware espera uma mensagem de resposta do Servidor
            A ação é executada ou não dependendo da resposta
      Fim Se
Fim Enquanto
```

Figura 13 – Algoritmo do funcionamento do Cliente

Após a conexão, o jogo começa sua execução. O jogador controla um elemento do tipo Avatar, porém, ao invés de chamar as funções localmente para se locomover ou realizar outra ação, ele chama uma função correspondente no *middleware*, que converte na mensagem apropriada e envia para o Servidor, que retorna confirmando ou não a ação.

A Figura 14 mostra o diagrama de classe das classes Cliente e Avatar, no qual a primeira foi implementada e a segunda modificada.

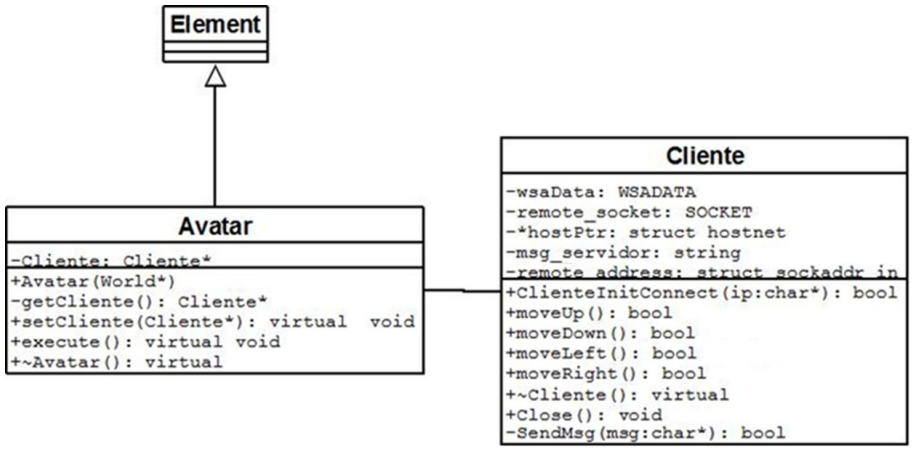

Figura 14 - Diagrama de classe das classes implementadas no Cliente

Na classe Avatar, as funções getCliente( ) e setCliente( ) servem para que o avatar possa acessar o Cliente que foi criado no inicio da execução do Subverse na classe World. A solicitação de envio de novas mensagem é realizado dentro da função execute( ) que sobrescreve a função da classe Element.

Na classe Cliente são mostradas as variáveis globais utilizadas e funções que permitem a troca de mensagens com o Cliente. As funções moveUp( ), moveDown( ), moveLeft( ) e moveRight( ), chamam a função, do tipo privada, SendMsg( ), evitando o avatar tenha acesso direto ao envio de mensagens.

Foi utilizado o caractere '+' no final de cada mensagem enviada como protocolo, a fim de ter o controle de sempre ler mensagens completas enviadas através do soquete. Assim a mensagem enviada pela função moveUp( ), por exemplo, é "UP+", onde "UP" é a direção em que o cliente quer se mover e '+' o caractere de final da mensagem.

Como o a proposta é o funcionamento correto da comunicação entre o Cliente e o Servidor, foi implementada no *middleware* apenas as funções referentes ao deslocamento do Avatar. Porém, para que outras funções também se comuniquem através do *middleware*, basta criar uma função do tipo pública, isto é, que possa ser acessada por outros objetos, e essa função enviar uma mensagem correspondente a ação para o Servidor. No Servidor, é preciso criar uma função que decodifique esta mensagem e realize a ação desejada pelo Cliente, e em seguida retornar a resposta. Como a proposta é a implementação de um protótipo parcial de um *middleware* de comunicação, foi implementada a conexão de apenas um Cliente ao Servidor, objetivando o correto funcionamento da troca de mensagens entre eles. Assim, o *middleware* de comunicação funcionou de forma esperada, permitindo a conexão e a troca de mensagens entre o Cliente e o Servidor.

5 CONSIDERAÇÕES FINAIS

O desenvolvimento de jogos permite que o interesse do aluno seja dirigido a diversas áreas, como ciências da computação, mídia digital, arte eletrônica e arte de estúdio (Argent, 2006). O aluno que pretende desenvolver jogos possui uma grande variedade de opções ao escolher sua profissão, pois ele aprende várias áreas de conhecimento específicas como programação, criação de história e enredo, edição de imagens, gerenciamento dos dados, gerenciamento de rede entre outros.

Entre os diversos segmentos, os jogos multiusuários *on-line* se destacam no mercado de jogos, pois proporcionam que um grande número de pessoas, que podem estar espalhadas pelo mundo, se encontre num Mundo Virtual e consigam interagir entre si. Porém o desenvolvimento desse segmento é complexo e demorado.

O *middleware* parcial desenvolvido ajuda a minimizar esse problema, pois simplifica o desenvolvimento de jogos multiusuários on-line uma vez que os desenvolvedores não precisam conhecer detalhadamente como é realizada a comunicação dentro do *middleware*, apenas qual função precisa chamar e quais parâmetros passar.

Os testes realizados mostraram que a implementação do *middleware* no Subverse foi bem sucedida, em que a conexão e a troca de mensagens entre o Cliente e o Servidor alcançaram seus objetivos.

5.1 TRABALHOS FUTUROS

Como trabalho futuro, torna-se de grande interesse alguns itens:

- Implementação de um chat de comunicação entre os jogadores, onde o *middleware* pode ser utilizado para envio das mensagens.
- Implementação de outra classe específica para o tratamento das mensagens, tanto no Servidor quanto no Cliente, evitando que o desenvolvedor reprograme o *middleware*.

- Implementação de threads no *middleware* para possibilitar a conexão de múltiplos clientes simultaneamente.

REFERÊNCIAS

ARGENT, Lawrence; DEPPER, Bill; FAJARDO, Rafael; GJERTSON, Sarah; LEUTENEGGER, Scott T.; LOPEZ, Mario A.; RUTENBECK, Jeff. **Building a Game Development Program**. IEEE Computer Society, junho de 2006.

BALAN, Rajesh Krishna; EBLING, Maria; CASTRO, Paul; MISRA Archan. **Matrix: Adaptive Middleware for Distributed Multiplayer Games**. Carnegie Mellon University, Pittsburgh, USA. IBM Research Watson, USA, 2005.

BENIN, Max Ricardo. **Evolução de NPC´s e Adversários em Jogos de Computador Usando Algoritmos Genéticos**. Florianópolis, 2007.

BLIZZARD. **World of Warcraft®: Wrath of the Lich King™ Shatters Day-1 Sales Record.** Blizzard Entertainment, Inc. Irvine, California, 2008. Disponível em: http://www.blizzard.com/us/press/081120.html acesso em: 08 nov. 2008.

DANTAS, M. **Tecnologia de Redes de Comunicação e Computadores**. 1. ed. São Paulo: Axcel Books. 2002.

DICKEY, Chris G., ZAPPALA, Daniel, LO, Virginia. **A Distributed Architecture for massively multiplayer online games**. University of Oregon, Department of Computer Science, Eugene, Oregon

INTEL, **10 Gigabit Ethernet Technology Overview**. Disponível em: http://www.intel.com/network/connectivity/resources/doc_library/white_papers/pro10gbe_lr_sa_wp.pdf

JESUS, Aeolo de; **Adaptação do Teste de Turing como Forma de Avaliar a Inteligência Artificial de Agentes Inteligentes em Jogos de Computador**. Florianópolis, 2008.

MACHADO, F.B.; MAIA, L.P. **Arquitetura de Sistemas Operacionais**. Rio de Janeiro: LTC, 3ª Edição, 2002.

RISO, Bernardo G.; MACIEL, Cristiano; SOUZA, Ivanise V.; ROSSI, Leila L.; NOTARE, Mirela S. M. A.; TONIN, Neilor A. **Engenharia de protocolos com LOTOS/ISO**. Editora da UFSC, 2004.

SILBERSCHATZ, Abraham; GALVIN, Peter; GAGNE, Greg. **Sistemas Operacionais Conceitos e Aplicações**. Editora Campus. 6ª tiragem, 2001.

ZAMBIASI, Saulo Popov; **SUBVERSE** Disponível em: http://sourceforge.net/projects/subverse/ acesso em: 21 nov. 2008.

TANENBAUM, Andrew S.; WHOODHULL, Albert S. **Sistemas Operacionais: Projeto e Implementação**. 2ª ed. Porto Alegre: Bookman, 2000.

TANENBAUM, Andrew S.. **Sistemas Operacionais Modernos**. Editora Pearson Education, 2ª Edição, 2003a.

TANENBAUM, Andrew S. **Redes de Computadores**. 4 ed. RJ: Editora Campus, 2003b.

TERRA. **Word of Warcraft ultrapassa 10 milhões de assinantes**. Disponível em: http://games.terra.com.br/interna/0,,OI2267707-EI6499,00-World+of+Warcraft+ultrapassa+milhoes+de+assinantes.html acesso em: 19 ago. 2008.

ZYDA, Michael. **Educating the Next Generation of Game Developers**. IEEE Computer Society, junho de 2006.