UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

TESE DE DOUTORADO

H-N2N: UMA ARQUITETURA HIERÁRQUICA ESCALÁVEL PARA AMBIENTES DE REALIDADE MISTA DE GRANDE PORTE

AQUILES MEDEIROS FILGUEIRA BURLAMAQUI

Orientador: Luiz Marcos Garcia Gonçalves
Co-Orientador: Jauvane Cavalcante Oliveira

Natal – RN
Fevereiro de 2008

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

H-N2N: UMA ARQUITETURA HIERÁRQUICA ESCALÁVEL PARA AMBIENTES DE REALIDADE MISTA DE GRANDE PORTE

AQUILES MEDEIROS FILGUEIRA BURLAMAQUI

Tese de Doutorado apresentada ao programa de Pós-graduação em Engenharia Elétrica – PPGEE – da Universidade Federal do Rio Grande do Norte – UFRN, como requisito parcial para a obtenção do grau de Doutor em Engenharia Elétrica.

Orientador: Luiz Marcos Garcia Gonçalves
Co-Orientador: Jauvane Cavalcante Oliveira

Natal – RN
Fevereiro de 2008

Divisão de Serviços Técnicos

Catalogação da Publicação na Fonte. UFRN / Biblioteca Central Zila Mamede

Burlamaqui, Aquiles Medeiros Filgueira.
H-N2N : uma arquitetura hierárquica escalável para ambientes de realidade mista de grande porte / Aquiles Medeiros Filgueira Burlamaqui. – Natal, RN, 2008.
123 f. : il.

Orientador: Luiz Marcos Garcia Gonçalves.
Co-orientador: Jauvane Cavalcante Oliveira.

Tese (doutorado) – Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Engenharia Elétrica e de Computação.

1. Sistemas distribuídos – Tese. 2. H-N2N Framework – Tese. 3. Sistemas colaborativos distribuídos – Tese. 4. Plataforma Java – Tese. 5. Multimídias colaborativas – Tese. 6. Realidade virtual – Tese. I. Gonçalves, Luiz Marcos Garcia. II. Oliveira, Jauvane Cavalcante. III. Universidade Federal do Rio Grande do Norte. IV. Título.

RN/UF/BCZM CDU 004.75 (043.2)

“Tudo que está no plano
da realidade já foi sonho um dia.”
Leonardo da Vinci

Dedicatória

Dedico este trabalho a meu Pai Marcos, à minha mãe Goretti, a meus irmãos Nestor, Penélope e Tétis, e a minha namorada Akynara, conhecida carinhosamente por Kakay.

Agradecimentos

Agradeço a meus pais (Marcos e Goretti), pela formação e amor que me deram, a meus irmãos (Nestor, Penélope e Tétis) por tudo que aprendi com eles, a meus amigos de faculdade, especialmente a todos que fazem parte do Laboratório NATALNET-DCA, que sempre estiveram ao meu lado para tirar dúvidas, me fazer rir, ajudar nos problemas, apontar defeitos e me fazer mais forte para continuar, aos professores e amigos Luiz Marcos, Guido e Jauvane pelo apoio, incentivo e por terem acreditado em mim. E a Akynara por todas as horas que deixei de estar com ela para tornar possível a realização deste trabalho como ele é hoje.

Resumo

Neste trabalho, propomos uma solução para resolver o problema de escalabilidade encontrado nos ambientes virtuais e de realidade mista colaborativos de grande porte, que usam o modelo cliente-servidor hierárquico. Basicamente, usamos uma hierarquia de servidores. Quando a capacidade de um servidor é atingida, um novo servidor é criado subordinado a ele e a carga atual do sistema é distribuída entre eles (pai e filho). Propomos ferramentas e técnicas eficientes para solucionar problemas inerentes ao modelo cliente-servidor, tais como a definição de agrupamentos de usuários, a distribuição e redistribuição de usuários entre servidores e operações de mixagem e filtragem, que são necessárias à redução de fluxo de informação entre servidores. O novo modelo e técnicas propostas foram testados, em simulação, emulações e em aplicações interativas que foram implementadas. Os resultados destas experimentações mostram melhorias nos modelos tradicionais anteriores, indicando a usabilidade do *framework* proposto em problemas de comunicação todos para todos. Este é o caso de jogos interativos e outras aplicações voltadas à Internet (incluindo ambientes multiusuário) e aplicações interativas do Sistema Brasileiro de Televisão Digital, a serem desenvolvidas pelo grupo de pesquisa.

Palavras-chave: Ambientes Virtuais de Grande porte, TV Digital Interativa, Sistemas Distribuídos

Abstract

In this work, we propose a solution to solve the scalability problem found in collaborative, virtual and mixed reality environments of large scale, that use the hierarchical client-server model. Basically, we use a hierarchy of servers. When the capacity of a server is reached, a new server is created as a sun of the first one, and the system load is distributed between them (father and sun). We propose efficient tools and techniques for solving problems inherent to client-server model, as the definition of clusters of users, distribution and redistribution of users through the servers, and some mixing and filtering operations, that are necessary to reduce flow between servers. The new model was tested, in simulation, emulation and in interactive applications that were implemented. The results of these experimentations show enhancements in the traditional, previous models indicating the usability of the proposed in problems of all-to-all communications. This is the case of interactive games and other applications devoted to Internet (including multi-user environments) and interactive applications of the Brazilian Digital Television System, to be developed by the research group.

Keywords: large scale virtual environments, interactive digital tv, distributed systems

Sumário

Lista de Figuras

Capítulo 1 – Introdução

"Uma longa viagem começa com um único passo."

Lao-Tsé

A área de Realidade Virtual (RV), considerada uma das formas mais avançadas de interface de usuário disponíveis, ganha cada vez mais espaço, através de aplicações que vão desde treinamento e educação até entretenimento. A idéia de representar visualmente um usuário em um ambiente de RV impulsionou o surgimento de avatares (Capin, 98). A implementação desse mecanismo de percepção trouxe novas possibilidades para os ambientes de RV, tornando-os sistemas multiusuário e colaborativos. Dessa forma, tornou-se possível a interação entre vários usuários em um ambiente virtual, surgindo uma nova classe de aplicações, denominadas de Ambientes Virtuais Colaborativos (AVCs) (Zyda, 1996).

Um dos maiores desafios nesta área de pesquisa é o desenvolvimento de sistemas que permitam escalabilidade. Escalabilidade é uma característica desejável em todo o sistema, em uma rede ou em um processo, que indica sua habilidade de manipular uma porção crescente de trabalho de forma uniforme, ou estar preparado para o crescimento do mesmo (Bondi, 2000). Um dos principais fatores que afetam a escalabilidade em sistemas virtuais multiusuário é a troca de mensagens de atualizações, que inclui informações do tipo: identificação dos usuários, localização no ambiente virtual, estado atual, etc. Para contornar este problema, várias arquiteturas de comunicação têm sido propostas de acordo com os requisitos das aplicações. Basicamente existem duas arquiteturas principais: o modelo cliente-servidor e o modelo *peer-to-peer* (Tanenbaum, 2002).

Dentro deste contexto, e visando resolver tais problemas, propomos a arquitetura H-N2N (*Hierarchical N to N*), destinada ao desenvolvimento de uma plataforma para ambientes virtuais/mistos colaborativos de grande porte. Nosso

modelo resolve o problema complexo, enfrentado pelos modelos tradicionais(Tanenbaum, 2002), que é o de lidar com muitos usuários. O uso de outros modelos tradicionais, como torna inviável a comunicação de todos para todos em um ambiente massivo, o que não ocorre com o modelo aqui proposto.

1.1 Motivação

Este trabalho surgiu no contexto dos sistemas *Hyperpresence* (Tavares, 2003) e Torcida Virtual (Tavares, 2004), vinculados aos laboratório Natalnet, da Universidade Federal do Rio Grande do Norte e Lavid, da Universidade Federal da Paraíba. Como ambos os projetos foram construídos sobre a clássica arquitetura cliente-servidor, encontramos problemas quanto a escalabilidade de tais sistemas. Visando resolver tal problema, de modo a torna-los ambientes de larga escala, iniciamos pesquisas nesse sentido, objetivando encontrar alternativas ao modelo cliente-servidor adotado. Uma breve descrição desses sistemas que nos serviram como motivação é apresentada a seguir: *Hyperpresence* e Torcida Virtual.

1.1.1 Hyperpresence

O *Hyperpresence*, é um ambiente de realidade mista, desenvolvido pelos Laboratórios Associados Natalnet (DIMAp e DCA). A idéia é ter dois mundos simultâneos, o primeiro em um espaço real e o segundo em um espaço virtual, representando a mesma situação. A Figura 1 ilustra o modelo conceitual do sistema *Hyperpresence*. Em um primeiro momento (seta 1, na Figura 1) temos um fluxo unidirecional partindo do plano real e chegando ao virtual. Nesse caso, o plano real é formado por um conjunto de elementos reativos (robôs autônomos) e obstáculos. O plano virtual funciona como uma espécie de filtro de visualização responsável por exibir exatamente o que está acontecendo no plano real em forma tridimensional. No segundo momento (seta 2, na Figura 1), é proposta uma alternativa para prover simulação e interação dentro dessa plataforma já estabelecida no primeiro momento, ou seja, transferir o controle para o plano virtual, sendo assim, o plano real passa a simular os eventos gerados no plano virtual e vice-versa. Dessa forma, é possível inserir um obstáculo no plano virtual e

ele ser percebido pelo robô real que irá executar uma ação de desvio, por exemplo, mesmo que o obstáculo não exista fisicamente no mundo real, sendo entendido como um “obstáculo fantasma”.

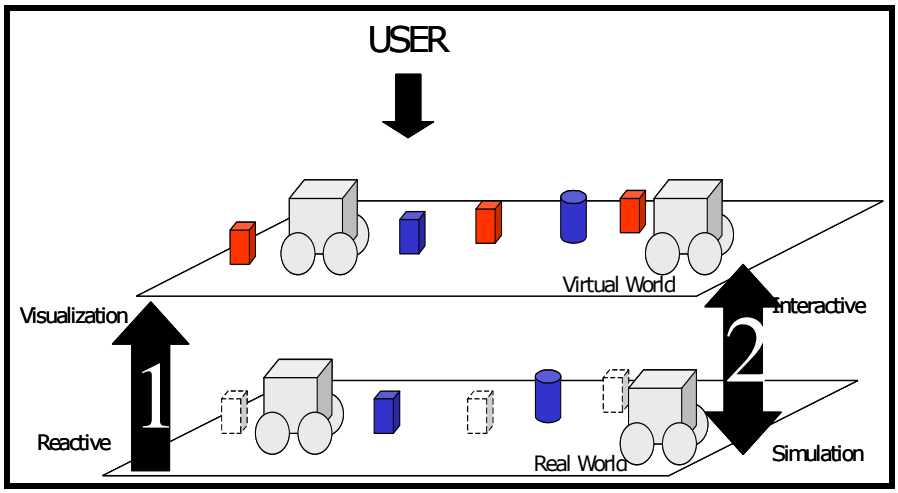

Figura 1 – Sistema Hyperpresence. (Tavares, 2003)

1.1.2 Torcida Virtual

O segundo sistema que motivou a realização deste trabalho foi o programa aplicação desenvolvido para Televisão Digital interativa (TVDI) chamado de Torcida Virtual. A Torcida Virtual provê ferramentas para que o telespectador, em um ambiente de televisão interativa possa participar, “junto” com outros telespectadores, da recepção e transmissão de eventos esportivos, como por exemplo, uma partida de futebol. A Torcida Virtual possibilita o compartilhamento com outros torcedores, tanto do espaço acústico quanto do espaço físico de um estádio virtual. O compartilhamento acústico é implementado através da mixagem do som capturado nos ambientes onde estão localizados os torcedores. Por exemplo, os sons que denotam a animação de um torcedor durante a comemoração de um gol podem ser escutados pelos outros torcedores que fazem parte do mesmo espaço compartilhado. A partir da captação dos sons do grupo é possível gerar “o som da torcida virtual”, através da mixagem do sons de cada indivíduo, o que pode inclusive ser transmitido para o local onde está acontecendo o evento esportivo, que será compartilhado com o som da torcida presencial.

1.2 Contribuições

Da experiência de desenvolvimento e uso dos sistemas citados (*Hyperpresence* e Torcida Virtual), percebemos a necessidade de tornar-los escaláveis, permitindo assim a participação de um grande número de usuário sem que haja perda na qualidade do serviço prestado. Que para o caso específico do Torcida Virtual consiste de uma necessidade crítica, já que em tais sistemas o número de usuário tende a ser grande.

Como a arquitetura até então utilizada (cliente-servidor) não supria tal necessidade, fomos motivados a buscar outras soluções que resolvessem tal problema. Assim, a proposta metodológica de trabalho é a concepção, projeto, implementação e testes de uma arquitetura para o desenvolvimento de ambientes multiusuário escaláveis.

A arquitetura resultante, o H-N2N, é a contribuição final do presente trabalho, tendo sido testada em simulação e em várias aplicações atualmente em funcionamento no Laboratório Natalnet. Ainda como contribuições, nesta tese, propomos técnicas e ferramentas eficientes para solucionar alguns dos principais problemas inerentes aos AVCs, tais como determinar agrupamentos de usuários baseados em vetores de características, problemas de distribuição de usuários entre os servidores ativos atendendo um AVC, problemas de perda de nós da hierarquia, entre outros.

1.3 Metodologia de Trabalho

A execução desse trabalho considerou o paradigma de engenharia de software denominado prototipagem, baseado na produção de protótipos que evoluíram rumo ao produto final (paradigma evolutivo). Como ferramenta de modelagem foi utilizado o JUDE (JUDE,2005) e a notação UML (BOOCH, 2000) devido à facilidade de atualização de código e documentação oferecida, assim como também diagramas ACME (Garlan, 1997), que dão uma visão mais alto nível da arquitetura do sistema.

As seguintes metas foram cumpridas durante o desenvolvimento desta pesquisa:

- Estudo do embasamento teórico necessário: (a) sistemas multiusuário; (b) sistemas multimídias; (c) ambientes virtuais colaborativos (d) frameworks (e) sistemas distribuídos
- Estudo de tecnologias relacionadas: Java, Eclipse, JUDE, ACME;
- Estudo de ferramentas de simulação de redes.
- Estudo de caso com desenvolvimento de aplicações multimídia colaborativas distribuídas.
- Desenvolvimento do protótipo inicial do H-N2N
- Execução de testes e coleta de dados do sistema e sua análise, visando realimentação no modelo de desenvolvimento.
- Introdução de melhorias no protótipo e, iterativamente, execução de novos testes e coleta de dados e análise, até atingir a evolução atual do sistema.

1.4 Estrutura da Tese

Esta tese está estruturada da seguinte forma:

a) o Capítulo 1 introduziu a proposta, contribuições e principais resultados conseguidos com a pesquisa;

b) o Capítulo 2 discute o estado da arte mostrando primeiramente aspectos conceituais relevantes ao tema de pesquisa desenvolvido, procurando elucidar definições e conceitos necessários ao entendimento do restante do trabalho e por fim apresenta um estudo sobre trabalhos relacionados;

c) o Capítulo 3 apresenta e discute a arquitetura H-N2N, através de uma descrição arquitetural de seus componentes;

d) no Capitulo 4, falamos sobre os detalhes de implementação do H-N2N;

e) temos, no Capítulo 5, a descrição dos experimentos e principais resultados obtidos com a arquitetura aqui proposta; e

f) finalmente, no Capítulo 6 apresentamos as conclusões da pesquisa.

Capítulo 2 – Estado da Arte

"Se você acha que a educação é cara,
tenha a coragem de experimentar a ignorância"
Derek Bok

Para o entendimento do trabalho aqui proposto é necessário discutir alguns conceitos fundamentais que serão apresentados nesse capítulo. Esses conceitos foram agrupados de acordo com as duas vertentes principais que motivaram este trabalho, são elas: Arquitetura de Sistemas Distribuídos e Ambientes Virtuais, discutidos na seção 2.1 e 2.2, onde explanaremos alguns trabalhos relacionados ao tema AVC.

2.1 Embasamento Teórico

É importante começarmos com a definição do que vem a ser um sistema distribuído. Várias definições podem ser encontradas na literatura, sendo que, dentre elas, a mais utilizada é a dada por Tanenbaum (2002):

> *"Um sistema distribuído é uma coleção de computadores independentes que aparentam para os usuários ser um único computador".*

Tanenbaum (2002) fala ainda dos dois principais aspectos relacionados a essa definição, que podem levar em conta tanto aspectos de software como de hardware e que, embora ambos sejam essenciais, os aspectos de software se destacam por serem os responsáveis pela criação da abstração onde as várias máquinas do sistema distribuído são vistas como se fossem somente uma pelo usuário.

A principal diferença entre sistemas com um processador e sistemas distribuídos está na comunicação inter-processos. Em sistemas com somente um processador, essa comunicação é realizada através de uma memória

compartilhada. Já os sistemas distribuídos realizam tal comunicação através de outras estratégias como os modelos cliente/servidor e P2P, apresentados a seguir.

2.1.1 Modelo Cliente/Servidor

Neste modelo de comunicação, o processamento ocorre de maneira distribuída, existindo basicamente dois componentes: o Cliente e o Servidor. O Cliente, como o próprio nome sugere, é um processo que requisita informações, dados ou serviços a um Servidor, que por sua vez trata de suprir tais necessidades. Ambos utilizando para isso um protocolo simples de pedido/resposta (*request/reply*), como ilustrado na Figura 2.

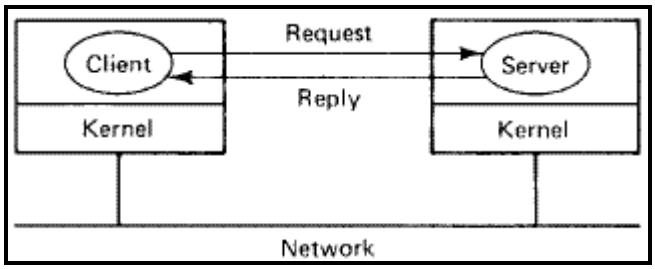

Figura 2 – Modelo Cliente/Servidor (Tanenbaum 2002)

Desse modo, as atribuições do servidor são diferentes das do cliente, havendo uma centralização das informações e tarefas por parte do servidor. Em um sentido mais amplo, a definição do modelo cliente-servidor aplicado a redes é dada por Rüdiger (2002) da seguinte maneira:

> *"Uma rede cliente-servidor possui uma arquitetura distribuída com um sistema de alta performance, o servidor, e vários clientes de menor performance. O servidor é uma unidade central de registro e também o único provedor de serviço e conteúdo. Um cliente somente faz requisições de conteúdo ou execução de serviços ao servidor, sem compartilhar nenhum de seus próprios recursos."*.

Exemplos de serviços que utilizam tal arquitetura são facilmente encontrados, como por exemplo, serviços de bate-papo, jogos *on-line*, HTTP e DNS, entre outros. Nesses serviços, temos mais de um cliente e possivelmente mais de um servidor também, obrigando a uma estruturação do sistema no que diz respeito a como os clientes e servidores devem se relacionar. Dessa estruturação,

novos modelos foram derivados, sem que perdessem a característica de serem sistemas cliente/servidor. Os modelos *Flat*, *Mirrored-Server* e Hierárquico são alguns exemplos (Tanenbaum 2002).

2.1.1.1 Modelo Flat

No modelo Flat(plano, horizontal) temos um servidor e vários clientes conectados a ele. A Figura 3 traz um exemplo de comunicação entre 4 clientes e um servidor utilizando o modelo *flat*.

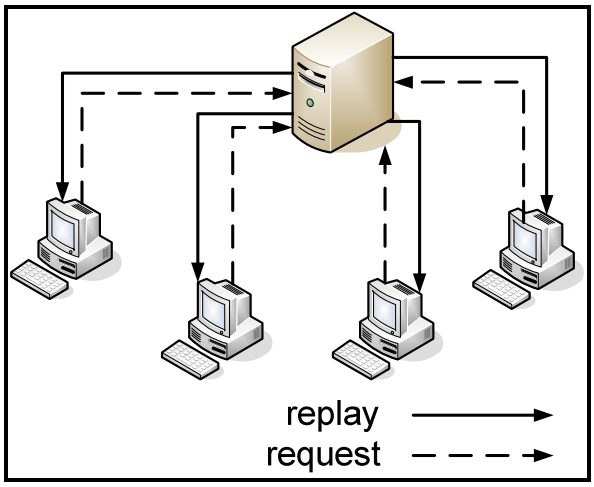

Figura 3 – Modelo Flat(Tanenbaum 2002)

Esse é o modelo mais simples, onde temos somente um servidor atendendo as requisições dos clientes. Tal modelo é de fácil implementação, por concentrar a maior parte do processamento e armazenamento no servidor, porém é necessário que se tenha disponível um computador de melhor performance para se tornar o servidor. E mesmo assim, dependendo do número de clientes, o servidor sempre será um "gargalo" no sistema.

2.1.1.2 Modelo Mirrored-Server

Já o modelo Servidor Espelhado (*Mirrored-Server*) acrescenta a comunicação servidor-servidor ao modelo flat, como podemos ver na Figura 4, objetivando com isso aumentar a sua escalabilidade, ou seja, número de clientes que o sistema possa suportar.

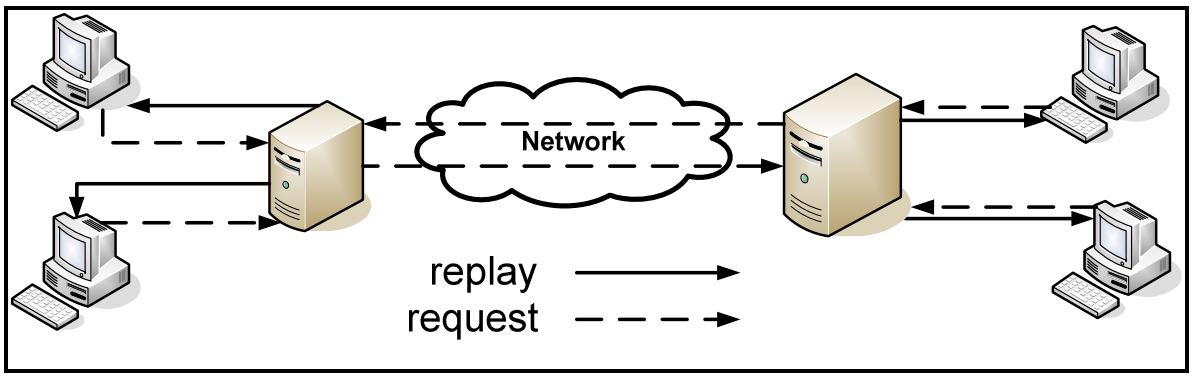

Figura 4 – Modelo Mirrored-Server(Tanenbaum 2002)

Nesse modelo, temos vários servidores conectados entre si através de uma rede privada, preferencialmente de alta velocidade, e os servidores são "espelhados", ou seja, replicando informações e dividindo a carga do sistema.

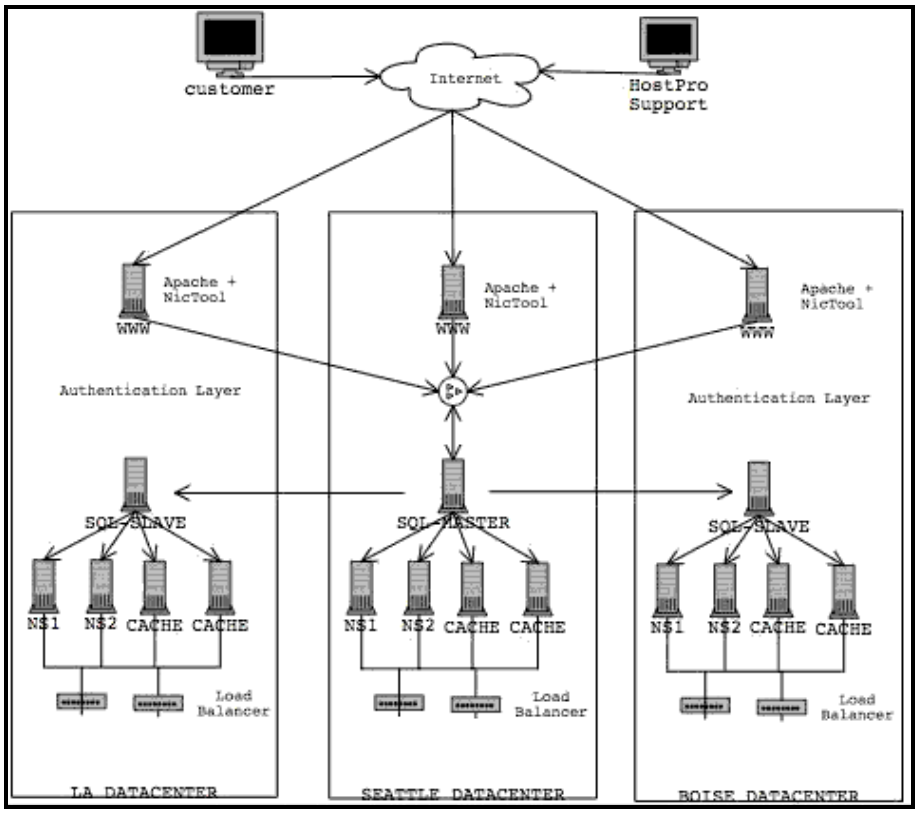

Figura 5 – Modelo Hierárquico (exemplo DNS)

2.1.1.3 Modelo Hierárquico

No modelo hierárquico temos também a comunicação entre servidores, porém ele é mais complexo que o modelo *Mirrored-Server*, pois o sistema fica estruturado

hierarquicamente em forma de árvore. O melhor exemplo desse modelo pode ser encontrado implementado no serviço DNS (Mockapetris, 1989), sendo ele ilustrado na Figura 5.

2.1.2 Modelo Peer-to-Peer

Kellerer (1998) define o modelo *Peer-to-Peer* da seguinte maneira:

> *" Uma arquitetura de rede distribuída pode ser chamada Peer-to-Peer (P-to-P, P2P, ...) se os participantes compartilharem parte de seus próprios recursos de hardware (poder de processamento, capacidade de armazenamento, banda de rede, impressoras, ...) . Esses recursos são necessários para prover os serviços e conteúdo oferecidos pela rede (ex. compartilhamento de arquivos ou espaços de trabalho para colaboração mútua). Os serviços e recursos são acessíveis por todos os pares sem necessidade de passar por nenhuma entidade intermediária. Os participantes dessa rede são tanto provedores de recursos (serviços e conteúdo) como demandadores desses mesmos recursos (conceito de Servent)".*
>
> Já Milojicic (2002), define o modelo Peer-to-Peer da seguinte maneira:
>
> *"Classe de sistemas e aplicações que empregam recursos distribuídos na execução de uma função crítica de uma maneira descentralizada".*

O modelo P2P abre mão da centralização das informações vitais ao sistema. É um modelo de comunicação onde as partes envolvidas são iguais, ou seja, cada uma têm as mesmas funcionalidades e direitos. Não existem clientes nem servidores, todos são nós(*peers*) na rede e tem a mesma função. Esse é o conceito básico de *peer-to-peer* (P2P), porém podemos encontrar variações a este modelo, como veremos a seguir.

2.1.2.1 Redes Puras

Baseado na definição P2P dada por Kellerer (1998), Rüdiger (2002) define redes Puras da seguinte maneira:

"Uma arquitetura de rede distribuída pode ser classificada como P2P 'pura', se em primeiro lugar for P2P conforme a definição anterior e em segundo lugar se qualquer entidade terminal da rede puder ser individualmente removida sem que a rede sofra qualquer perda em termos de 'serviço".

Ou seja, são redes onde todos os participantes tem os mesmo direitos e deveres, onde todos contribuem para o roteamento, fazendo tanto o papel de cliente quanto o de servidor. Um exemplo deste tipo de rede é o GNUtella (GNUtella, 2008). Desse modo os participantes podem se comunicar sem precisar de um servidor mediando a comunicação, como ilustrado na Figura 6, onde temos o modelo da arquitetura do GNUtella.

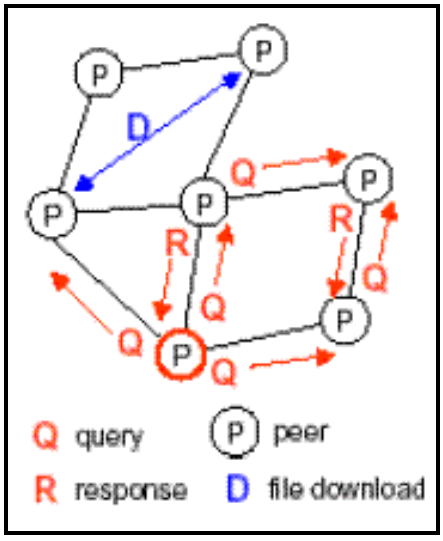

Figura 6 – P2P Pura (GNUtella)

2.1.2.2 Redes Híbridas

Outra definição é dada por Rüdiger (2002) ainda baseado na definição P2P de Kellerer:

"Uma arquitetura de rede distribuída deve ser classificada como P2P 'híbrida' se em primeiro lugar for P2P conforme a primeira definição e em segundo lugar tiver a necessidade de uma entidade central para prover parte dos serviços da rede".

Desse modo, podemos ter uma arquitetura onde uma arquitetura cliente-servidor emularia uma rede *peer-to-peer*, que é o caso do *AIM* (AIM, 2008). Outro

exemplo de rede híbrida são os *Brokers*, onde o servidor central retêm informações sobre os clientes e faz o papel de facilitador nas conexões entre os *peers*, um exemplo disto seria o *Napster* (Napster, 2008). Na Figura 7, temos o modelo híbrido com o qual o *Napster* funciona, o seu computador se recebendo informações de um computador central para poder se comunicar com outro computador na rede peer-to-peer.

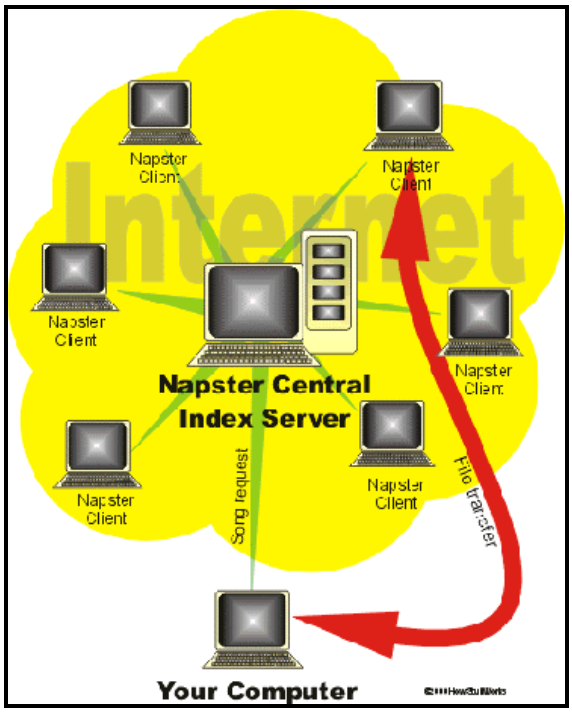

Figura 7 – Modelo Híbrido (Napster)

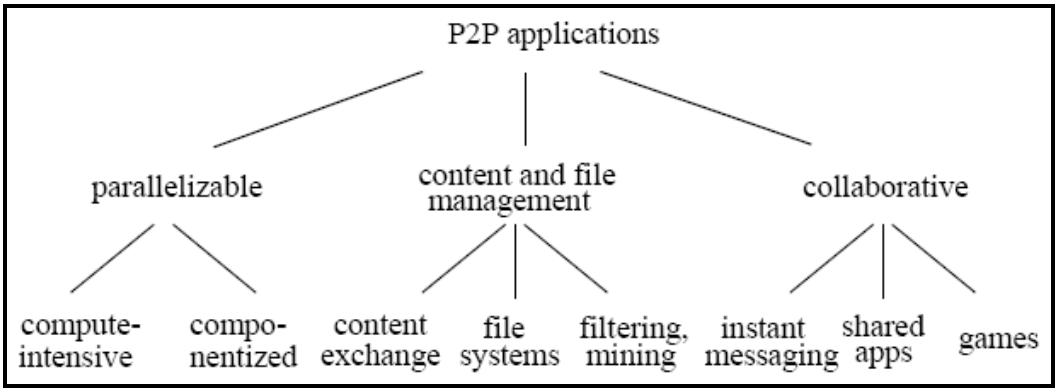

Figura 8 – Taxonomia Aplicações P2P(Milojicic, 2002)

2.1.2.3 Aplicações P2P

Podemos ainda separar as aplicações P2P em três categorias e 8 subcategorias, como mostrado na Figura 8 (Milojicic, 2002).

Neste trabalho, consideraremos somente a categoria aplicações colaborativas (*collaborative applications*), por pertencerem ao domínio de aplicações de nosso interesse, aqui discutidos, que incluem *chats*, ambientes virtuais e jogos.

Tais aplicações são normalmente baseadas em eventos, sendo que os elementos (*peers*) da rede formam grupos e começam a realizar tarefas. O tamanho do grupo pode variar de tamanho, podendo ser grande ou somente formado por dois indivíduos. Quando um evento ocorre, ele deve ser repassado para todos os outros participantes do grupo.

Podemos destacar como desafios técnicos inerentes a esses tipos de aplicações a tarefa de localização dos *peers* que se desejam comunicar e a garantia de que todos os *peers* devem ter a mesma visão da aplicação, o que implica em fazer com que todos recebam as mesmas mensagens, isso sem falar no requisitos de tempo-real necessários a esses sistemas.

2.1.3 Comunicação em Grupos

Uma vez definida que arquitetura será utilizada em um sistema distribuído, seja ela Cliente/Servidor, P2P ou alguma alternativa hibrida, é preciso definir como os elementos dessa arquitetura irão se comunicar. Não importa que arquitetura seja escolhida, todas elas implementam estratégias de comunicação em grupo, sendo necessário saber quais são essas estratégias e como elas influenciam um sistema distribuído. Segundo Tanenbaum (2002), um grupo nada mais é do que:

> *"...uma coleção de processos que agem juntos em algum sistema ou de alguma forma especifica".*

Em um sistema distribuído, o número de processos envolvidos numa comunicação pode variar desde uma simples troca de dados entre dois processos a até mesmo entre milhares. Desse modo, é necessário que haja uma coordenação entre eles e que uma comunicação em grupo seja implementada,

permitindo assim a criação de grupos de processos que cooperem para fornecer um serviço.

Basicamente, a implementação dessa comunicação pode ser feita através de *unicast*, *multicast* e *broadcast*:

- **UNICAST:** transmissão ponto a ponto, aqui o processo deve enviar a uma mensagem para cada destinatário.
- **BROADCAST:** a mensagem é enviada uma só vez para todos os processos, e depois são filtradas pelos processos interessados.
- **MULTICAST:** cada mensagem é enviada uma só vez somente para os integrantes do grupo.

Essas implementações são as mais comumente encontradas, porém, muitas outras implementações foram derivadas a partir das supracitadas como por exemplo: ANYCAST, MANYCAST e OVERCAST, detalhadas nas subseções seguintes.

2.1.3.1 UNICAST

Resumindo, o *unicast* seria uma comunicação ponto a ponto, entre um remetente e um destinatário. Na Figura 9, temos um exemplo de comunicação *unicast*, onde é preciso enviar três mensagens, uma para cada destinatário, a fim de disseminar a informação entre todos os clientes.

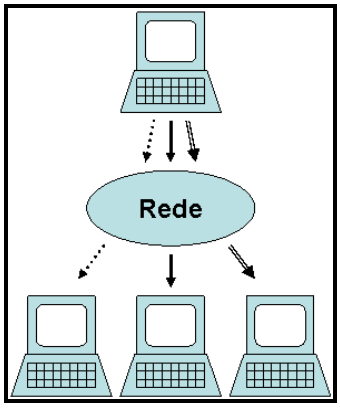

Figura 9 – Comunicação unicast

2.1.3.2 BROADCAST

No *broadcast*, as mensagens são enviadas para todos na rede, nesse método, ao invés de ser enviada uma mensagem para cada um dos usuários, somente uma única mensagem é enviada, sendo que todos "ouvem" essa mensagem, a Figura 10 ilustra esse método de comunicação.

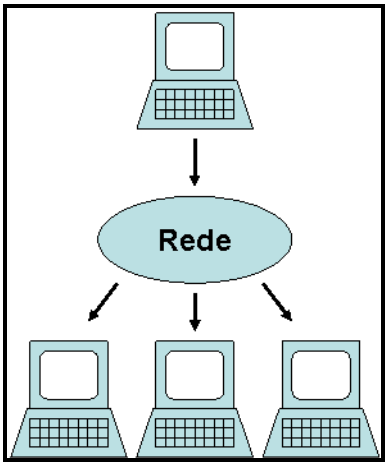

Figura 10 – Comunicação Broadcast

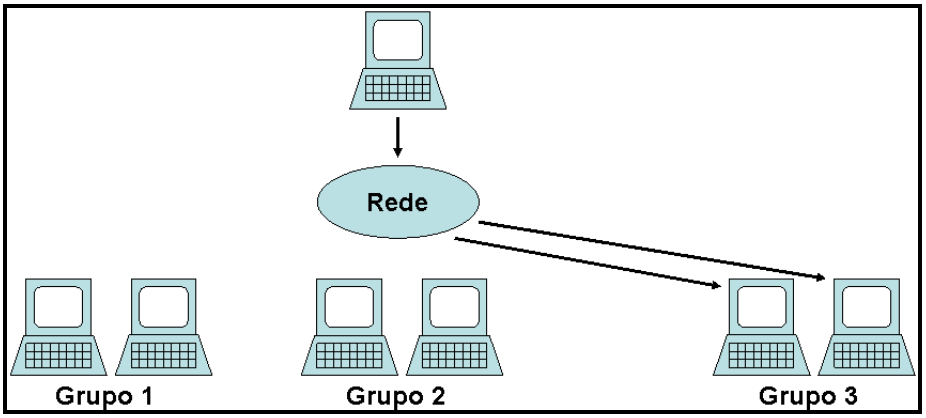

Figura 11 – Comunicação Multicast

2.1.3.3 MULTICAST

O *multicast*, ou *IP multicast*, mereceria destaque devido a sua melhor performance e rapidez na entrega das mensagens aos componentes do grupo, pois apenas uma mensagem é gerada para todos os pertencentes do grupo, economizando banda da rede, ver Figura 11. Porém, infelizmente essa tecnologia é inviável de

ser utilizada num ambiente Internet, pois não são todos o roteadores que a implementam.

2.1.3.4 ANYCAST

Originalmente definido na RFC 1546 (Partridge,1993), o termo *anycast* refere-se à comunicação onde um simples transmissor envia uma mensagem para um receptor (normalmente o mais próximo a ele) sendo esse receptor pertencente a um grupo.

2.1.3.5 MANYCAST

Segundo (Bhattachargee,1997), em uma comunicação onde temos receptores participando de um grupo e um transmissor enviando uma mensagem para alguns desses participantes, configura-se numa comunicação *manycast*, utilizando o envio de várias mensagens por *anycast* para atingir esse objetivo.

2.1.3.6 OVERCAST

Overcast é um sistema de multicast implementado no nível de aplicação, ao contrário do *IP multicast* que funciona a nível de rede (Jannotti,2000). Essa característica o torna mais viável para ambientes Internet, pois não é necessário que os roteadores implementem o *multicast*. Ele é implementado como uma rede *overlay*. Uma rede *overlay* consiste de um conjunto de nós colocados em lugares estratégicos sobre uma rede já existente, criando assim uma rede sobre outra já existente.

2.1.4 Ambientes Reais e Virtuais

Após essa explanação sobre sistemas distribuídos, apresentaremos nessa seção alguns conceitos relacionados a disciplina Realidade Virtual(RV), indispensáveis ao total entendimento do trabalho aqui proposto.

Segundo (Milgram,1994) dependendo do que está sendo exibido podemos dividir o ambiente que percebemos em 4 tipos básicos: ambientes reais, ambientes de realidade aumentada(RA), ambientes de virtualidade aumentada(VA) e ambientes virtuais.

2.1.4.1 Ambientes Reais

Ambientes Reais são ambientes onde somente existem objetos reais, vistos diretamente pela pessoa ou exibidos através de vídeos. Na Figura 12 temos um exemplo de ambiente real sendo visto através de vídeo.

Figura 12 – Exemplo de Ambiente real

Observe que no exemplo de ambiente real mostrado na Figura 12, todos os elementos exibidos devem ser elementos reais, o chão, as pessoas assistindo o jogo de capoeira assim como os capoeiristas.

Figura 13 – Exemplo de ambiente virtual

2.1.4.2 Ambientes Virtuais

Ambientes Virtuais são ambientes totalmente compostos por objetos virtuais, como exemplificado na Figura 13. O conceito de Ambiente Virtual foi introduzido há bastante tempo, por Ivan Sutherland (Sutherland,2002), quando em 1965 ele teve a idéia de inserir pessoas em ambientes sintéticos gerados por computador.

Outra definição importante foi dada por Steve Ellis (Burdea, 1996). Ele definiu o termo "virtualização" como sendo o processo pelo qual um espectador humano interpreta uma impressão sensorial moldada para ser um objeto estendido em um ambiente diferente daquele que existe fisicamente.

Ellis também dividiu o processo em três níveis:

- Espaço virtual - espaço onde será projetada a imagem virtual;
- Imagem virtual - percepção de profundidade dos objetos;
- Ambiente virtual - o usuário torna-se parte do mundo virtual.

Um outro conceito interessante, muito discutido atualmente, é o de comunidades virtuais. Este se baseia no conceito mais geral de comunidade e o termo virtual é acrescido para diferenciar sua natureza, ou seu campo de atuação, que, nesse caso, é o espaço virtual. Pode-se dizer que uma comunidade virtual é estabelecida em torno de um mundo virtual.

Sendo assim, podemos definir mundo virtual como sendo um espaço virtual compartilhado, visitado periodicamente por pessoas que possuem pelo menos algo em comum: atividades, metas, idioma especializado, ou uma ligação ética.

Dentre as origens dos mundos ou ambientes virtuais, pode-se citar:

- MUD's (Curtis, 1993);
- Habitat(Morningstar,1991);
- Ambientes de Jogos: Doom, Renderman, Myst;

O Habitat (Morningstar,1991) foi o primeiro mundo virtual no qual as pessoas tinham uma representação visual e tinham a capacidade de se comunicar e formar uma comunidade virtual.

A Figura 14 ilustra a interface de usuário do Habitat. O Habitat constitui um ponto inicial na consolidação do conceito de ambientes virtuais multiusuário.

Figura 14 – Interface de Usuário no Habitat

Hoje em dia, pode-se definir ambientes virtuais multiusuário como ambientes caracterizados por suportar espaços sintéticos tridimensionais que são compartilhados por múltiplos usuários, remotamente localizados.

O acesso a estes ambientes através da *WWW* (*Word Wide Web*) é importante, porque permite que uma ampla gama de aplicações, do entretenimento à educação e medicina, seja oferecida, através de uma interface bastante difundida como os navegadores da *WWW*. A maioria dos ambientes virtuais existentes na *WWW* foi concebida através de tecnologias 3D como *VRML* (*Virtual Reality Modeling Language*) e *X3D* (*Extensible 3D*) – esta em desenvolvimento pelo *Web3D Consortium*.

Apesar da *VRML* ser a linguagem mais utilizada na construção desses ambientes na *WWW*, outras tecnologias, tais como, *Java3D* e *MPEG-4*, têm surgido como alternativas poderosas. Estas tecnologias não são encaradas apenas como tecnologias que competem com a *VRML*, mas principalmente como tecnologias que complementam a *VRML*.

2.1.4.3 Ambientes de Realidade Aumentada

Ambientes de Realidade Aumentada são ambientes com predominância de objetos reais sendo exibidos através de vídeo juntamente com alguns objetos virtuais que fornecem informações extras ao usuário do ambiente. Ou então um ambiente com objetos reais sendo vistos diretamente pela pessoa juntamente com objetos virtuais.

A Figura 15 traz dois exemplos desse tipo de ambiente. No primeiro, informações textuais são adicionadas a visão do usuário, já no segundo temos objetos gráficos tridimensionais sendo utilizados para agregar informações as imagens do mundo real.

Figura 15 – Exemplo de ambientes de realidade aumentada

Figura 16 – Exemplo de Ambiente com Virtualidade Aumentada

2.1.4.4 Ambientes de Virtualidade Aumentada

Virtualidade Aumentada, são ambientes formados por objetos virtuais sendo exibidos juntamente com alguns objetos reais exibidos através de vídeos ou então diretamente pela visão da pessoa. Na Figura 16 podemos ver um exemplo desses ambientes, onde quase todo o ambiente é virtual, a não ser pelo pano de fundo(background) que é um vídeo. Os outros objetos são gráficos tridimensionais ou imagens estáticas.

2.1.4.5 Ambientes de Realidade Mista

Dentro da história da Realidade Virtual, enquanto sub-área de pesquisa da Computação Gráfica, o termo Realidade Virtual Mista referia-se à combinação da Realidade Virtual Imersiva, isto é, aquela em que são providos todos os meios e técnicas para que o usuário, usando todos os seus sentidos, sinta-se totalmente dentro do ambiente virtual, com a Realidade Virtual não Imersiva, aquela onde não ocorre esta sensação, como é o caso por exemplo quando se usa uma interface de visualização padrão (monitor), mesmo com noção de profundidade (visão estéreo), como interface de visualização do ambiente.

Seguindo o conceito mais atual (Milgram 1994), usado neste trabalho, adotamos o termo Ambientes de Realidade Mista para denominar o conjunto composto por ambientes de Realidade Aumentada e de Virtualidade Aumentada, ou seja, todos aqueles ambientes que não sejam puramente reais nem puramente virtuais estão dentro do contexto de ambientes de realidade mista. Na Figura 17, temos duas imagens que representam bem um ambiente de realidade mista. Nesse exemplo, temos tanto uma interface com características de RA quando uma com características de VA.

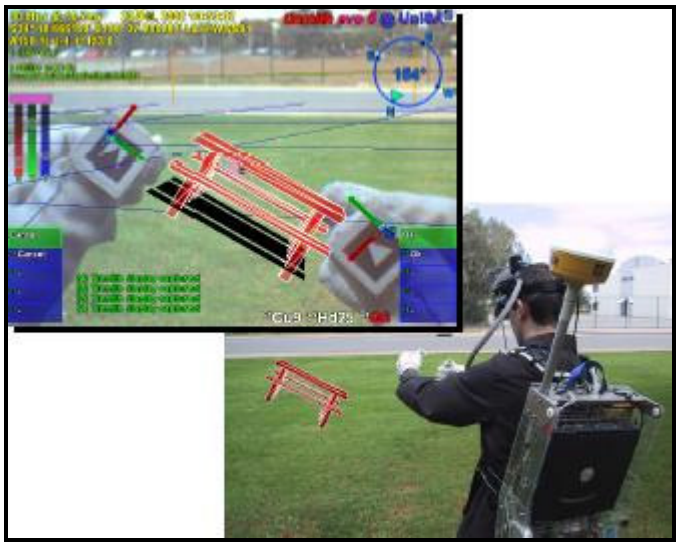

Figura 17 – Exemplo de Ambiente Misto

A Figura 18 relaciona os vários tipos de ambientes mostrando a transição entre eles, partindo da realidade (ambiente real) até a virtualidade (ambiente

virtual). Essa figura é uma simplificação do *Reality-Virtuality Continuum* introduzida por Milgram (1994).

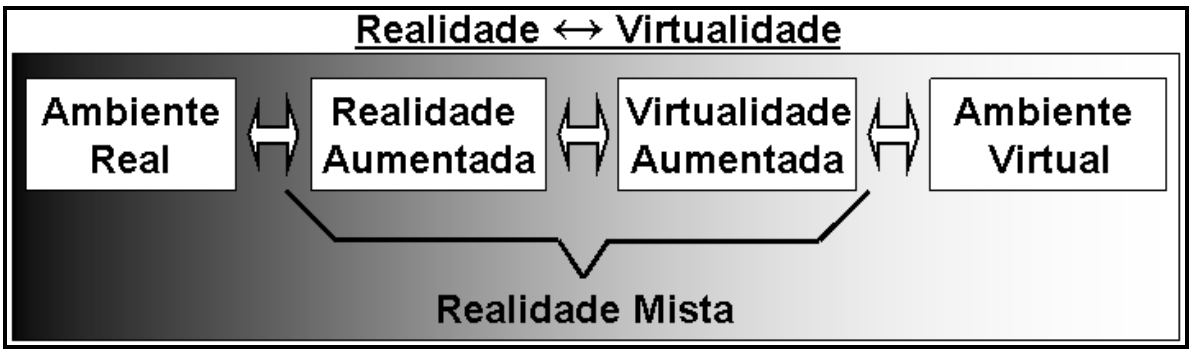

Figura 18 – Transição Realidade-Virtualidade

Na Figura 19 temos um bom exemplo dessa transição de realidades, onde em A temos o mundo real, em B a realidade aumentada e em C o mundo virtual.

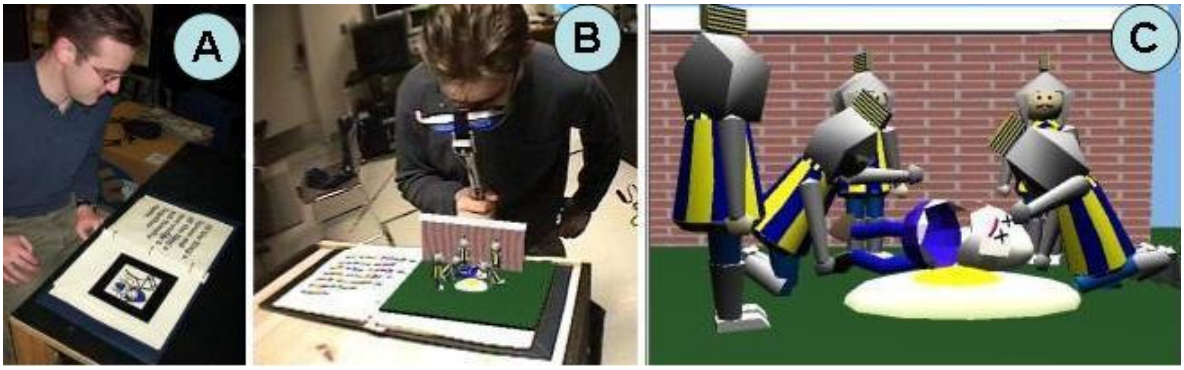

Figura 19 – Exemplo de transição de realidade

2.1.5 Representação de Usuários

No dia-a-dia as pessoas possuem representações, isto é, um artifício qualquer capaz de caracterizar um indivíduo em particular. A carteira de identidade é um bom exemplo, pois, possui um número único e uma representação visual (foto) capazes de distinguir uma determinada pessoa na sociedade. Isso também é possível em ambientes de RV através do uso de um avatar. O termo avatar é oriundo da mitologia hindu (avatãr). Significa a encarnação de um deus em um corpo mortal. Este termo foi utilizado pela primeira vez com uma semântica computacional por Chip Morningstar. Em termos de Realidade Virtual, um avatar é

uma representação visual ou incorporação do usuário, que é notada pelos outros usuários do ambiente virtual. O usuário, por sua vez, também verá os outros usuários como avatares.

Na Internet, podemos dizer que a utilização de avatares permite aos usuários a "encarnação" de diferentes facetas gráficas, tridimensionais ou não, definidas num ambiente virtual. Através de tecnologias como VRML, Java3D e X3D podemos utilizar qualquer objeto 3D simples ou objetos combinados para compor um avatar. Os humanóides, por exemplo, são composições tridimensionais complexas que se assemelham com formas de seres humanos. Outra opção é associar recursos 2D a um objeto 3D qualquer. A escolha do layout do avatar é uma decisão que irá refletir diretamente no desempenho da aplicação. Representações mais complexas demandam mais performance, enquanto que soluções mais simples ou híbridas podem produzir resultados satisfatórios com custos reduzidos. Exemplo de avatares podem ser vistos na Figura 20.

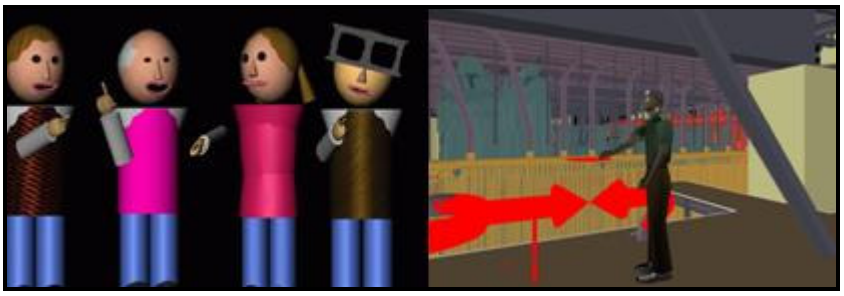

Figura 20 – (a) Avatares construídos por composição. (b) Humanóide.

2.1.6 Ambientes Virtuais Multiusuário

Porém, um avatar não é meramente uma representação visual. Para que um ambiente disponibilize essa funcionalidade, é necessário o suporte a serviços multiusuário, isto é, serviços capazes de controlar e manipular a interação de vários usuários em uma mesma aplicação de RV. Estes serviços são oferecidos através da implementação de um sistema distribuído que deve prover, basicamente, as seguintes funcionalidades:

- Autenticação de usuários;

- Sincronização de eventos entre os usuários no ambiente tridimensional;
- Monitoramento das ações dos usuários;
- Controle de objetos inseridos no ambiente virtual.

Dessa forma, é possível garantir aos usuários do ambiente 3D a capacidade de percepção, pois uma vez que um usuário possui uma representação visual, sua presença poderá ser percebida por outros usuários do mesmo ambiente. Outra vantagem é o envolvimento, pois o usuário é transportado para dentro da aplicação, onde pode se movimentar e interagir com o sistema através da sua própria representação.

2.2 Trabalhos Relacionados

Uma vez entendido os conceitos sobre sistemas distribuídos e ambientes virtuais, apresentamos as soluções existentes, relacionadas a implementação de Ambientes Virtuais Colaborativos, em especial aquelas preocupadas com o provimento de escalabilidade em tais sistemas, mostrando suas principais características e diferenças.

Várias arquiteturas são encontradas na literatura com o intuito de suportar sistemas multiusuário de larga escala, visando o gerenciamento do sistema como um todo, incluindo operações de inclusão de usuários, fornecimento de informação, dados e conexões, entre outras. A seguir faremos uma breve descrição das principais arquiteturas que tratam do problema da escalabilidade, tais como: NPSNET (Macedonia, 1994), RING(Funkhouser, 1995), MASSIVE (Greenhalgh, 1995), VELVET (Oliveira, 2003), VORONOI P2P (Hu, 2004) o DPM (Radenkovic, 2002) e a arquitetura distribuída de Assiotis (Assiotis, 2006).

2.2.1 NPSNET

Foi desenvolvido pelo Departamento de Ciências e Computação da Escola de Pós-graduação Naval em Monterey, na Califórnia em 1994. Atualmente se encontra na quarta versão, sendo que a primeira e a segunda se caracterizavam por serem baseadas em tecnologias *ethernet*, limitando suas aplicações a LANs

com poucas estações. Na seqüência, surgiu a versão NPSNET-Steath, e por último o NPSNET-IV que funciona através de *IP multicast*, com configuração dinâmica de grupos *multicasts* refletindo uma partição hexagonal (ver Figura 21) de um ambiente virtual. Esse modelo garante que somente 7 avatares sejam visíveis por vez, o que reduz o trafego e processamento nas estações. Cada vez que um avatar se move de um hexágono para outro, ele deixa um grupo *multicast* e entra em outro.

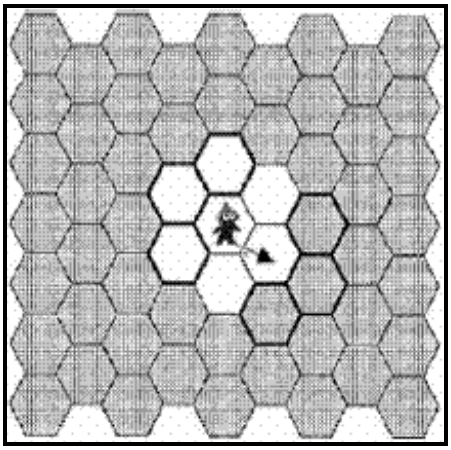

Figura 21 – Modelo de Hexágonos do NPSNET(Macedonia, 1994)

2.2.2 RING

O RING(Funkhouser, 1995) é um AVC que utiliza a arquitetura Cliente/Servidor para suportar interação visual em tempo-real entre um grande número de usuários. A característica chave desse sistema é que ele utiliza algoritmos de visibilidade objetivando reduzir o número de mensagens necessárias para manter consistente o estado entre as várias estações clientes distribuídas através da rede. Quando uma entidade gera mensagens de atualizações, essas mensagens só chegarão aos clientes aptos a recebê-las.

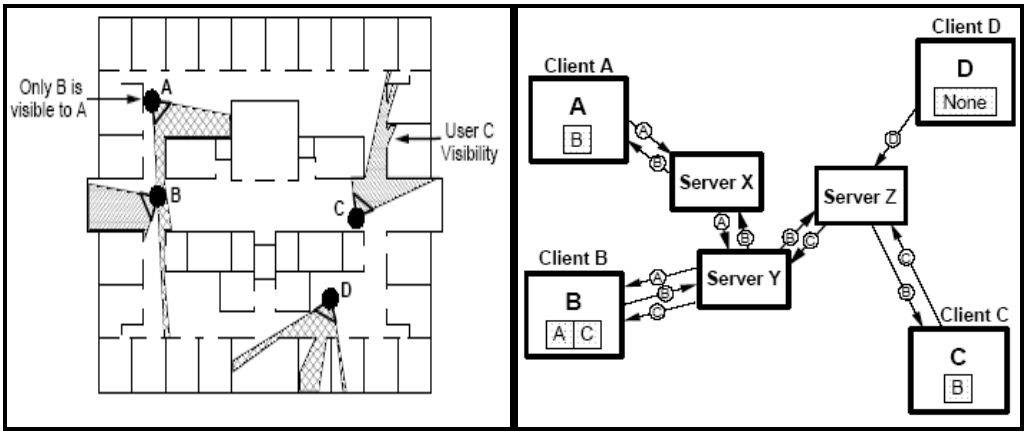

Figura 22 – Arquitetura do RING

Na Figura 22, parte esquerda, temos um mapa com 4 entidades, A, B, C e D representando clientes. O sistema RING, através de seus algoritmos de visibilidade detecta que somente B pode ser visto por A, com isso, as mensagens de atualização de B são somente enviadas para A. Analisando agora a parte direita da Figura 22, temos um diagrama onde podemos ver uma configuração da estrutura de clientes e servidores desse exemplo. Nesse exemplo vemos que o cliente A recebe somente mensagens de B. B recebe mensagens de A e de C, o cliente C por sua vez recebe mensagens de B, e D não recebe mensagens de ninguém. Todas essas mensagens são repassadas para os clientes através dos servidores X, Y e X, responsáveis pelo cálculo dos algoritmos de visibilidade.

2.2.3 MASSIVE

O primeiro protótipo foi desenvolvido em 1997 pelo Departamento de Ciências da Computação da Universidade de Nottingham, e atualmente ele se encontra na terceira versão. A principal contribuição deste modelo foi a introdução do conceito *Third-Party Objects*, que permite a representação de um grupo de objetos, indivíduos (*crowds*) como se fosse um só. As mensagens dos indivíduos do grupo só são escutadas individualmente se o usuário fizer parte da *crowd*. Na proposta MASSIVE-1, a comunicação é feita através de *unicast* e *peer-to-peer*, o que não permite um grande número de usuários conectados. Já no MASSIVE-2, adota-se o

multicast, e no MASSIVE-3 é utilizado *multicast* com arquitetura cliente-servidor para prover escalabilidade (Greenhalgh, 2000).

2.2.4 VELVET

O VELVET propõe uma arquitetura híbrida e adaptativa para resolver um problema encontrado nas soluções que utilizam o modelo de *locales*. A idéia de *locales* consiste em particionar o mundo virtual, de modo que os eventos gerados dentro de um *locale* sejam percebidos somente pelos habitantes desse *locale*. Porém, numa situação em que um grande número de usuários decida entrar em um mesmo *locale*, o fluxo de mensagens trocadas entre seus participantes comprometeriam aqueles participantes com menores recursos de processamento e de rede.

Para resolver este problema o VELVET define o AoI (Área de Interesse), que consiste em um círculo em volta do avatar demarcando o seu campo de visão, conforme é mostrado na Figura 23. O alcance do AoI vária de acordo com os recursos disponíveis por cada usuário assim como pelo número de objetos dentro de seus domínios, sempre procurando incluir um número máximo de objetos sem que haja perda na qualidade do serviço oferecido.

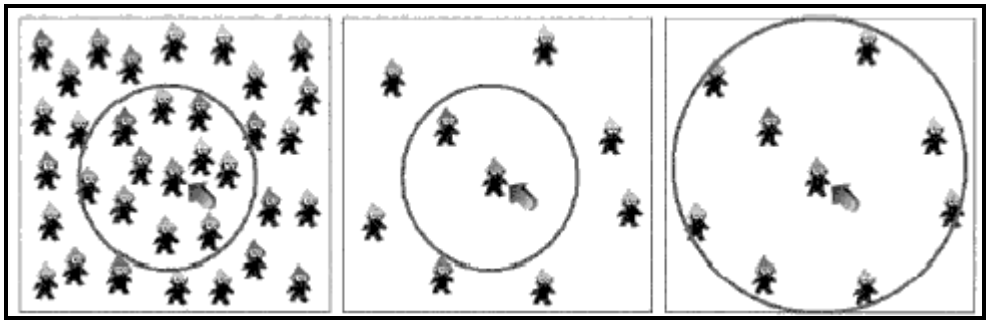

Figura 23 – Área de Interesse do VELVET

2.2.5 Voronoi Based P2P

Dos trabalhos encontrados na literatura, uma das propostas mais recentes cujo foco não está na arquitetura cliente-servidor nem na aplicação de grupos *multicast*, é o trabalho de Hu (2004), que, diferentemente do NPSNET-IV que utiliza hexágonos, baseia-se no uso do Diagrama de Voronoi (Guibas, 1985) como

ferramenta matemática para a determinação de vizinhança e no modelo de comunicação *peer-to-peer*. A noção de área de interesse (AoI) também é aplicada para reduzir a quantidade de mensagens trocadas entre os vizinhos. Desse modo cada usuário mantém um diagrama de voronoi que indica quais são seus vizinhos e com isso uma conexão direta com cada um deles é feita, ver Figura 24.

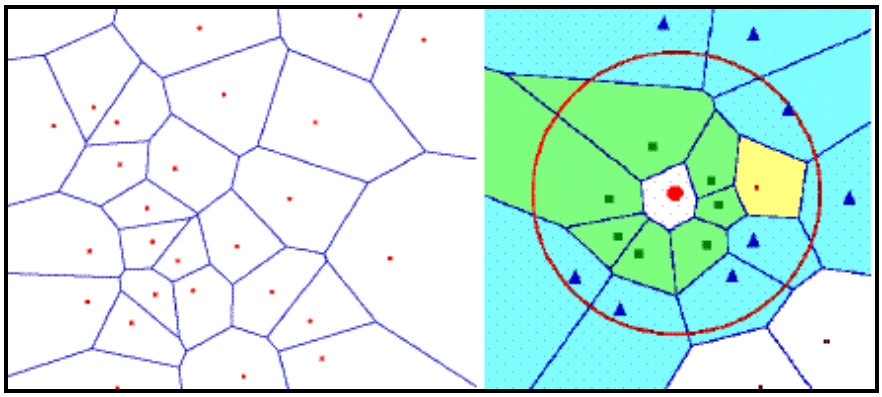

Figura 24 – Diagrama de Voronoi

2.2.6 DPM

A técnica DPM (*Distributed Partial Mixer*) é um serviço de áudio para AVCs projetado para suportar muitas pessoas falando simultaneamente e operar através de internet (Radenkovic, 2002). Este serviço realiza dinamicamente a adaptação do sistema, de acordo com variações no número de usuários e congestionamentos na rede. Porém, os autores do trabalho citado não dão uma certeza do quão escalável tal serviço é, comentando apenas que o sistema pode suportar dezenas ou ate centenas de usuários falando simultaneamente, mas sem um número preciso.

Ao se analisar situações onde existam várias pessoas se comunicando ao mesmo tempo, alguns padrões podem ser encontrados. Por exemplo, uma pergunta verbal pode ser seguida de uma única resposta, de períodos de silêncio, ou ainda de respostas simultâneas, com todos falando ao mesmo tempo. Outro fator observado é a importância dos eventos que não estão associados à comunicação verbal mas que são essenciais em tais ambientes. Eventos como a duração do silêncio e outros eventos ocorridos no ambiente físico onde o usuário se encontra são exemplos disso. Baseado nessas analises, Radenkovic (2002)

utiliza métodos que não restringem os canais de áudio. Ou seja, é sugerido como serviço ideal aquele que realize a transmissão contínua do áudio ou no máximo utilize a supressão do silêncio.

Em uma arquitetura de mixagem centralizada, todo o fluxo de áudio é enviado para um misturador central (doravante mixador, aportuguesado do inglês *mixer*) que soma múltiplos fluxos e depois redistribui esse fluxo para cada cliente. A idéia de realizar uma mixagem especifica para cada cliente foi uma proposta do sistema SPLINE (Waters, 1997). Os autores citam que tais métodos realizam uma mixagem excessiva, uma vez que mesmo em momentos onde haja largura de banda suficiente a mixagem é realizada.

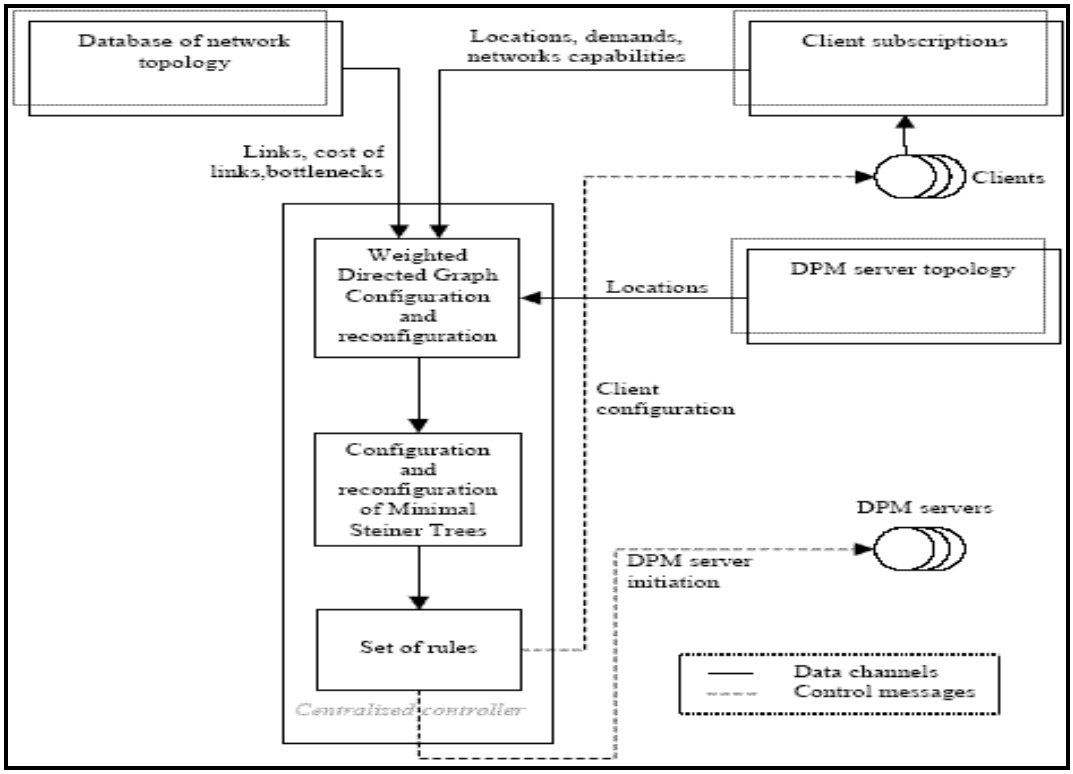

Figura 25 – Serviço DPM estático centralizado

A Mixagem Parcialmente Distribuída (DPM) requer que alguns componentes, os Mixadores Parciais, existam na rede, sejam eles servidores dedicados ou clientes. Os clientes se comunicam através desses componentes, seja enviando seu áudio ou recebendo o áudio dos outros clientes. Ao realizar a mixagem parcial do áudio, apenas um subconjunto do fluxo de áudio será

escolhido para ser misturado. Em vez de produzir um fluxo único em todos os casos, poderá ser produzido várias combinações dependendo da situação.

A principal contribuição do trabalho citado (Waters, 1997) é considerar a potencialidade do uso do DPM na internet. É proposto o uso de árvores compartilhadas (*Shared Tree*) de servidores DPM.

Dois modelos de cenários são descritos na aplicação do DPM, o Centralizado e o auto-organizável. No modelo centralizado os PM(Partial Mixing) são pré-estabelecidos, existem um conhecimento prévio da rede, conforme mostrado na Figura 25. Já no modelo auto-organizável, a medida que for necessário os clientes se tornarão PM e ficaram organizados em uma arvore *multicast*, mostrado na Figura 26.

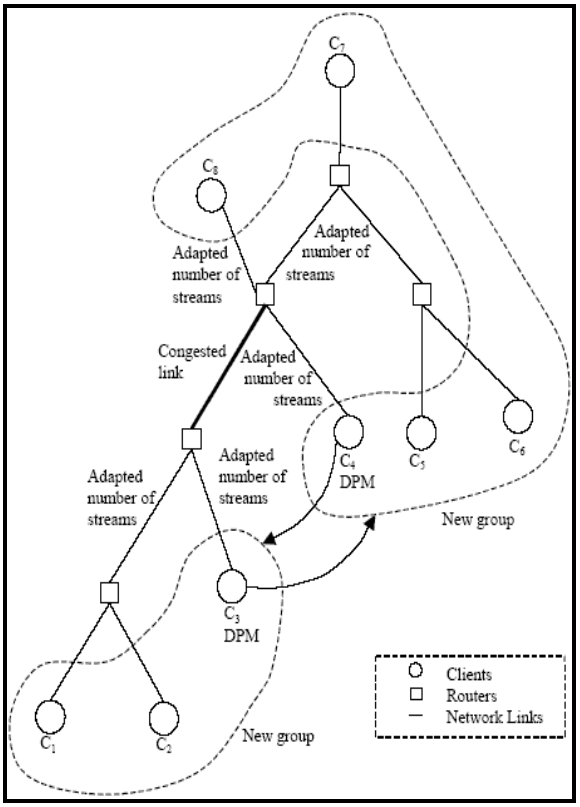

Figura 26 – Serviço DPM auto-organizável

2.2.7 CAN

Content-Addressable Network (CAN), estrutura de objetos distribuídos que mapeia as chaves em nós. Baseado em um espaço de coordenadas cartesianas (d-

Dimensional), o qual é dividido dinamicamente entre todos os nós do sistema em algum ponto de espaço. Quanto mais dimensões existirem, menor fica o espaço dos caminhos dos roteamentos.

A chave de um par é mapeada sobre um ponto no espaço usando uma função *hash* uniforme. Esse par é armazenado ao nó que possui a parte da coordenada no espaço onde o ponto se encontra (conhecido como Zona). Cada nó tem uma tabela de roteamento com os endereços IP e a zona virtual de coordenadas de cada vizinho.

Quando um novo nó/ponto entra na rede, tenta encontrar um nó existente, a zona deste nó é dividida em duas partes, uma destas partes é atribuído ao novo nó e a outra parte é mantida pelo nó já existente. Após a tabela de roteamento dos vizinhos é atualizada, ver Figura 27.

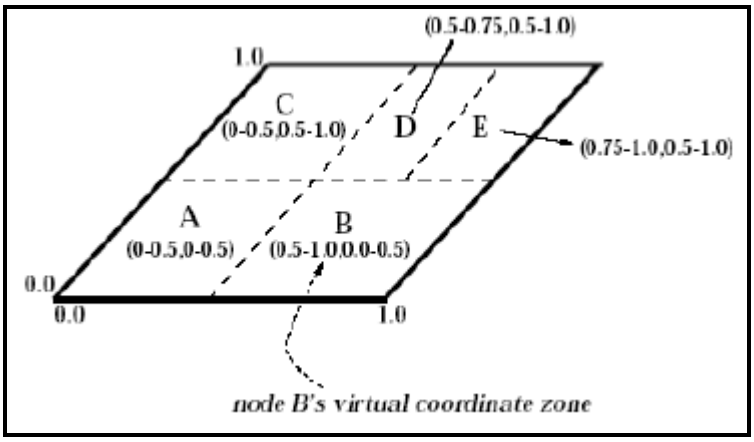

Figura 27 – CAN, espaço 2D com 5 nós.

2.2.8 Arquitetura Distribuída de Assiotis

Mario Assiotis e Velin Tzanov (Assiotis, 2006), ambos do MIT, desenvolveram uma arquitetura para AVC voltada para jogos. Mais especificamente, construído para MMORPG (Massive Multiplayer Online Role-Playing Game), tal solução apresenta técnicas e algoritmos que reduzem os requisitos de banda passante para ambos servidores e clientes, apresentando soluções para problemas de falha de servidores e permitindo a comunicação entre clientes em diferentes servidores.

Ao contrario de soluções P2P, Assiotis (2006) especifica uma arquitetura formada por vários servidores interligados através de uma rede de alta velocidade.

O mundo virtual é dividido em micro células e cada servidor fica responsável por uma ou mais dessas células. Para implementar tolerância a falhas, são usados servidores *backup*, um para cada servidor operacional. Esses servidores de backup são cópias dos servidores operacionais (principais), que em caso de falha assumem o seu lugar.

A passagem de eventos de um servidor para outro é feita baseada em um parâmetro R (raio) que todo evento possui. Caso o raio do evento ultrapasse as barreiras da micro célula de um servidor e esteja entrando em uma micro célula sob a responsabilidade de outro servidor esse evento será repassado.

Um problema dessa arquitetura é que o número de usuários que podem entrar numa micro célula do mundo virtual fica limitado ao fazer com que um servidor seja o responsável por essa micro célula. Ou seja, caso algum evento faça com que todos os usuários do mundo virtual se desloquem para o mesmo canto, o servidor responsável pela micro célula poderá não suportar sozinho essa concentração de usuários.

2.3 Discussões e Generalidades

Os conceitos sobre sistemas distribuídos e realidade virtual vistos nas seções 2.1 e 2.2 são fundamentais para o entendimento do problema abordado no presente trabalho.

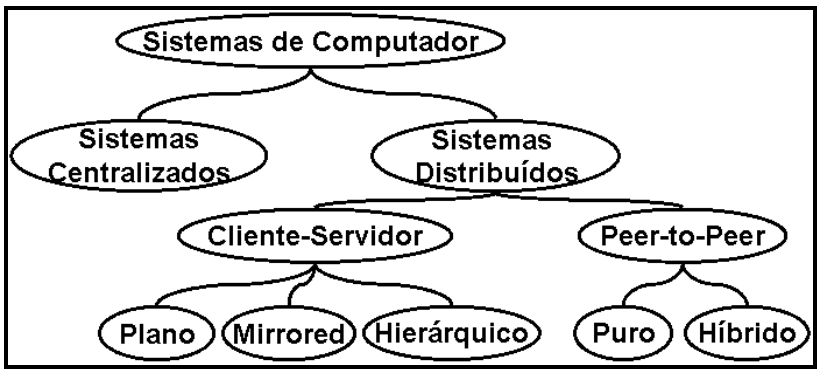

Figura 28 – Taxonomia de sistemas de computador

A Figura 28 resume a taxonomia simplificada, incluindo possíveis modelos de sistemas distribuídos, como por exemplo o modelo cliente-servidor e o modelo *peer-to-peer*, e ainda incluindo derivações de tais modelos como por exemplo o

modelo hierárquico e o modelo híbrido. A comunicação em grupo, que pode ser implementada de diferentes formas, é essencial quando um sistema distribuído se torna multiusuário, bem como o conceito de *avatar*, que é essencial para implementar a percepção em AVC. Assim, é possível a criação de ambientes visivelmente populados, facilitando a interação e colaboração entre seus usuários. Assim, a integração dos conceitos que foram apresentados neste Capítulo é essencial para se obter um AVC, que nada mais é do que um sistema distribuído que implementa ambientes virtuais ou de realidade mista.

Aproveitamos aqui para deixar claro que a nossa arquitetura é transparente quanto a qual técnica de realidade será utilizada para representação do ambiente, seja ela RA, VA ou Virtual. Nossa preocupação está em tornar tais ambientes massivos, independente da realidade adotada.

Várias comparações entre os AVCs já foram realizadas focando aspectos como visualização (Capin, 1999), distribuição de dados, processos e de participantes (Singhal, 1999), e comunicação em rede (Macedonia, 1997).

Com relação à comunicação em rede, o uso do modelo cliente-servidor tem atendido a demandas simples, quando o número de usuários é razoavelmente pequeno. Isto é, este número pode crescer até um ponto em que o servidor e a rede ainda consigam suportar bem o processamento e a troca de informações necessárias. Um problema surge no uso deste modelo para ambientes virtuais colaborativos de larga escala (Oliveira,1999; Pullen, 1997), quando a quantidade de usuários cresce em demasia. Nesse caso, o alto custo associado ao aumento da capacidade de processamento e da conexão de rede do computador servidor é o grande problema desta abordagem. Além do que, mesmo desconsiderando o custo, a tecnologia impõe limites máximos para o aumento de capacidade do servidor.

Como vimos, devido à alta escalabilidade desejada em AVCs e requisitos inerentes aos sistemas interativos distribuídos, soluções baseadas no modelo peer-to-peer (P2P) também estão sendo propostas na literatura (Kawahara, 2003; Hu, 2004; Knutsson, 2004). Entretanto, conforme observado por Morillo (2003), soluções P2P ainda demandam uma alta complexidade de desenvolvimento e

gerenciamento, além de possuir problemas relacionados a segurança, interoperabilidade e controle da rede.

A perda na qualidade do serviço em AVCs que consigam suportar um elevado número de usuários conectados ao mesmo tempo nesses ambientes está diretamente relacionada com o grande volume de mensagens trocadas entre seus usuários. De acordo com Shirmohammadi (2001), podemos classificar as mensagens que transitam num AVC nos seguintes tipos de dados:

- Dados de tempo real (vídeo/áudio) – dados multimídia.
- Dados de descrição de Objeto/Cena – descrição do avatar e do cenário virtual.
- Dados de *Groupware* - mensagens de comunicação de texto.
- Dados de Controle – entrada/saída de usuários, colaboração.

Assim, o desenvolvimento de mecanismos de comunicação e percepção em AVCs deve considerar, principalmente, esquemas eficientes para o tratamento e troca de mensagens, isto é, identificar e tratar as mensagens geradas pelo ambiente virtual e seus usuários de acordo com as funcionalidades oferecidas pela aplicação.

Outros trabalhos apontam alguns outros requisitos básicos que devem ser considerados em tais aplicações. Por exemplo, Park (1999) apresenta um estudo sobre o retardo permitido para a troca de mensagens de colaboração. Esses problemas são facilmente resolvidos para um pequeno número de usuários, devido à evolução das tecnologias de redes de computadores e poder de processamento das máquinas atuais. Porém, quando o número de usuários cresce, a qualidade de serviço, que está diretamente relacionada com a satisfação do cliente, cai. Existem algumas propostas que visam resolver esses problemas como os trabalhos de Greenhalgh (1999) e o de Oliveira (2003), que fazem isso reduzindo o número de clientes visíveis num determinado momento.

Como vimos, todos os modelos de arquiteturas aqui descritos resolvem o problema de escalabilidade encontrado em AVC, realizando para isso filtragens das mensagens entre seus usuários. Desse modo, abrindo mão de exibir todos os eventos pertencentes ao ambiente virtual a um usuário, seja particionando o

espaço em *locales*, micro células, utilizando o conceito de vizinhança ou mesmo implementando o conceito de área de interesse (AoI).

Essas soluções funcionam, mas trazem consigo um problema. Em um dado momento um cliente somente pode receber mensagens de um pequeno grupo de usuário do qual ele faz parte. Ou seja, os clientes ficam ilhados em seus grupos, alheios a acontecimentos que estejam ocorrendo nos grupos vizinhos. Acontecimentos esses que podem ser de grande importância para os clientes de tais sistemas.

Por exemplo, imagine a situação onde uma pequena cidade é vizinha a um vulcão. Nos modelos aqui apresentados, o vulcão poderia não fazer parte do grupo onde os clientes da pequena cidade estão, desse modo, numa erupção, eles só seriam avisados quando a lava já estivesse invadindo a cidade e destruindo suas casas. Podemos citar outro exemplo onde várias pessoas estão em um mesmo local, formando uma multidão, como num show, e uma delas dispara um tiro com uma arma de fogo. Com os atuais sistemas, o tiro só seria percebido pelas pessoas que estivessem perto (no mesmo grupo) de quem fez o disparo, porém, num ambiente real praticamente todos escutariam aquele disparo.

Em situações como essas precisamos de uma solução tal que promova escalabilidade e ao mesmo tempo permita que clientes em grupos diferentes, porém virtualmente no mesmo local, possam trocar informações sobre certos acontecimentos, ou seja, soluções que resolvam o problema do **"Tiro na multidão"**.

A nossa proposta, se aplicada ao exemplo anterior do vulcão, pode fazer com que, mesmo que o vulcão esteja fazendo parte de outro grupo, ele gere uma mensagem de alta prioridade, relacionada a sua erupção, de modo que ela possa chegar aos habitantes da pequena cidade vizinha e também que eventos de alta prioridade, quando gerados dentro de um local com uma multidão de usuários, possam ser percebidos por todos os que estiverem nesse local, como no caso da torcida virtual (Tavares, 2004), onde todos os usuários estão em um estádio.

Capítulo 3 – A Arquitetura H-N2N

"O Homem teme o Tempo, e ainda o tempo teme as Pirâmides"

Provérbio Árabe

A arquitetura apresentada no presente trabalho, denominada de H-N2N, possibilita a construção de ambientes virtuais e de realidade mista colaborativos que precisem suportar um elevado número de usuários conectados ao mesmo tempo, sem que haja perda na qualidade de serviço (QoS). A idéia principal consiste em separar os clientes (indivíduos) em grupos e organizar esses grupos de maneira hierárquica, como ilustrado na Figura 29.

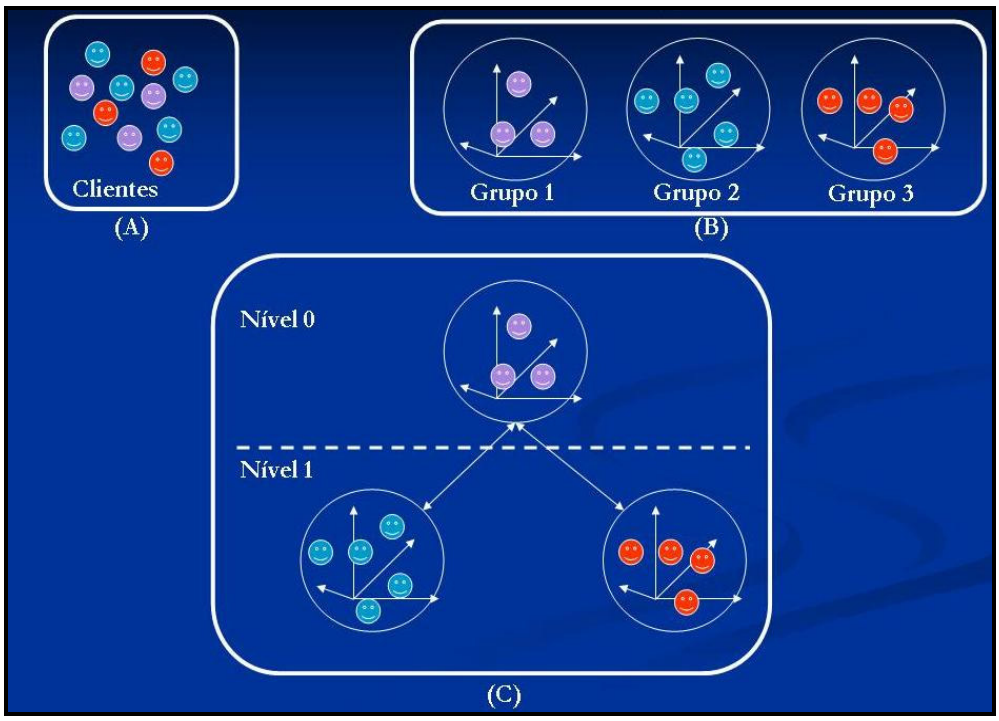

Figura 29 – (A) Clientes, (B) Grupos de Clientes, (C) Organização Hierárquica.

O H-N2N é então uma arquitetura cliente-servidor hierárquica, onde, para cada grupo de clientes, existe um servidor à sua disposição. Na configuração

inicial, existe somente um servidor, representando o primeiro grupo de usuários (Grupo 1). Este servidor (S0) encontra-se no nível 0 de hierarquia do sistema. À medida que mais e mais clientes se conectam ao servidor S0, haverá um momento em que este ficará sobrecarregado, dependendo de suas limitações de hardware, fazendo com que a qualidade do sistema fique comprometida.

Nesse momento, uma função de agrupamento é calculada e avaliada e um novo servidor é criado, representando o Grupo 2, para ajudar a suportar os usuários. Este servidor (S1), recém criado, será posicionado no nível 1, que se encontra logo abaixo do nível 0 e terá uma conexão com S0. Periodicamente os clientes poderão também serem reagrupados de forma a dividirem-se entre os servidores representados pelos grupos 1 e 2. Dois aspectos importantes a serem tratados nessa arquitetura dizem respeito a como será realizado o agrupamento/reagrupamento dos clientes e como esses clientes trocarão mensagens.

3.1 Funções de Agrupamento/Reagrupamento

O agrupamento e reagrupamento dos clientes é feito levando-se em consideração diversos fatores como:

- Capacidade de processamento dos servidores;
- Capacidade de processamento dos clientes;
- Características da rede (banda, atraso, *jitter*, etc);
- Objetivos comuns dos clientes dentro do AVC;

Características, como afinidades, interesse na troca de informação (comunicação) são também utilizadas para definir os indivíduos que formam um grupo e, em níveis mais altos na hierarquia, para definir agrupamentos.

Uma função recebe esses fatores como parâmetros de entrada. E quando houver um evento indicando a necessidade de agrupamento ou reagrupamento, seja ele um evento temporal, evento de entrada de cliente ou disparado através de alguma análise do trafego entre os nós, o servidor ativa essa função para que ela verifique a necessidade de rearranjar os clientes em novos grupos.

A função que determina os componentes dos novos grupos pode envolver o uso de algoritmos de agrupamento simples, encontrados na literatura, como o SOM (Kohonen, 2001), técnicas de segmentação (Zhang, 1997), PCA (Raychaudhuri, 2000), ou mesmo o uso de teoria de agentes (Hyacinth, 1996). Isto foge ao escopo desta tese, mas está sendo estudado em paralelo, em um outro trabalho.

No presente trabalho, estamos utilizando um algoritmo simples, sem aprendizado, como descrito a seguir. Seja *0* (zero) o nível do único servidor, no estado inicial do ambiente (apenas um servidor e nenhum usuário conectado a ele). Seus filhos, ou clientes, ou usuários (ou indivíduos), os quais nele se conectam, são também de nível *0*. À medida que um outro servidor for criado, este passa a ser de nível *1*, bem como seus filhos são clientes de nível *1* e assim por diante. A função de agrupamento a ser especificada, C_α, envolve então retirar alguns elementos do servidor atual (nível *n*) e repassá-los ao novo servidor (nível *n+1*), de uma maneira eficiente. Para isso, seja G_n um grupo de clientes conectados em um servidor do nível *n*. Seja G_{n+1} um grupo de clientes conectados a um servidor no nível *n+1*. Seja I_n um indivíduo no nível *n* (isto é, conectado a um servidor no nível *n*). Note que In pode ser cliente se considerarmos que está conectado ao nível *n* ou servidor se considerarmos que pode ter conexões (filhos) do nível *n+1*. Podemos especificar a operação C_α como sendo a combinação de duas operações, $C_{\alpha1}$ e $C_{\alpha2}$, da seguinte forma:

- $C_\alpha = C_{\alpha1} + C_{\alpha2}$, onde
  - $C_{\alpha1}$: $G'_n \subset G_n \rightarrow G_{n+1}$
    - Subgrupo de indivíduos do nível *n* é transferido para um grupo de indivíduos do nível *n+1*.
  - $C_{\alpha2}$: $G_n \rightarrow I_n$
    - Grupo no nível *n* é colapsado em indivíduo do mesmo nível *n*.

Do mesmo modo, existirá uma função de desagrupamento (***D***) que poderá ser ativada quando ocorrer a saída ou desconexão de clientes do sistema. Esta função tem o objetivo de evitar o desperdício de recursos que pode ocorrer ao se manter servidores com poucos clientes:

- $\boldsymbol{D}: G_n \rightarrow G'_{n-1} \subset G_{n-1}$
  - Grupo de nível *n* se torna subgrupo do nível *n-1*.

3.2 Funções de Filtragem (Junção/Seleção) de Eventos

Como vimos anteriormente, várias mensagens podem ser trocadas dentro de um AVC. Ao agruparmos os clientes, cada grupo contendo seu próprio servidor, temos uma arquitetura que facilita a troca de mensagens entre clientes no mesmo grupo, mas que à primeira vista impediria que mensagens fossem trocadas entre clientes de grupos diferentes. Na arquitetura aqui proposta, clientes em grupos diferentes podem trocar mensagens, as quais, embora eventualmente filtradas, permitirão o envio da informação original ou de parte dela. Para que isso ocorra, quando houver um agrupamento em um nível (por exemplo: nível 0), um novo grupo será formado em um nível abaixo (nível 1) para conter os novos clientes. Nesse caso, o grupo criado passa a ser tratado no modelo como um individuo em seu grupo pai. Assim, os grupos do nível *n* são tratados como indivíduos nos grupos do nível *n-1*.

A Figura 30 ilustra o conceito anterior, onde indivíduos de um grupo representam na realidade grupos de indivíduos do nível inferior na hierarquia. Na figura, um dos filhos do servidor A, no nível 0, indivíduo neste nível, é um grupo no nível abaixo (nível 1). No caso o grupo do nível 1 é então associado ao servidor B que gerencia o grupo de indivíduos a ele conectados. Ou seja, na Figura 30, temos um indivíduo no nível 0 conectado no Servidor A que é uma representação dos indivíduos que estão no grupo controlado pelo servidor B.

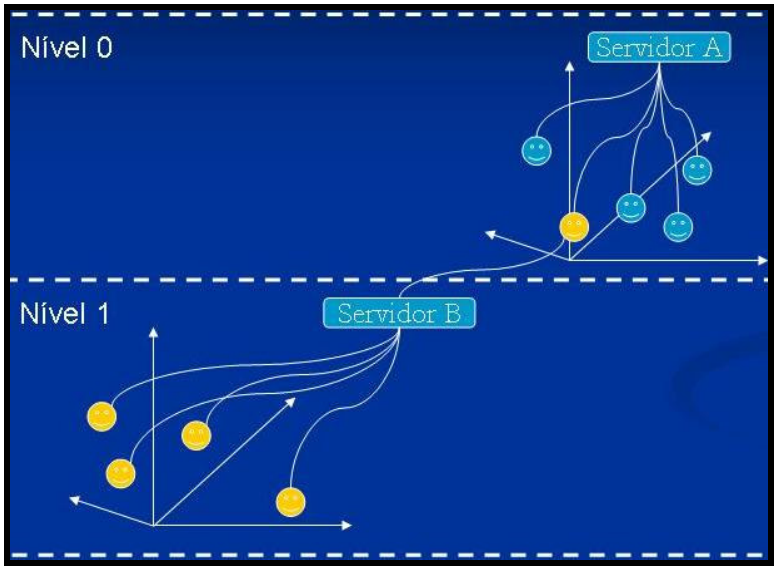

Figura 30 – Grupo como Indivíduo de Grupo Superior

No modelo H-N2N, o indivíduo no nível 0 (acima) representa o grupo de indivíduos do nível 1 (abaixo) e será responsável pelo envio, tanto de mensagens do nível de cima para o de baixo quanto pelo inverso. Porém, em AVCs de grande porte não é viável repassar todas as mensagens entre os níveis, pois, se isto ocorrer, o número de mensagens seria elevado e o sistema retornaria ao problema de perda de qualidade devido à necessidade de tratar esse grande volume de mensagens.

Note que algum tipo de manipulação para reduzir o volume de informação se faz necessário, pelo menos para o fluxo em direção à raiz da árvore. Para resolver isso, aplicamos técnicas de filtragem (seleção e junção) nas mensagens que serão trocadas entre os níveis. Com a seleção, reduzimos o número de mensagens ao selecionarmos quais delas serão repassadas. Com a técnica de junção (mixagem), juntamos várias mensagens numa só (Ex: transformar vários fluxos de áudio em um só).

Chamaremos essas mensagens trocadas entre os clientes de eventos, pois as mensagens são nada mais do que informações a respeito de algum evento que tenha ocorrido no sistema. Os servidores aplicarão funções de filtragem ou mixagem de acordo com o evento que tenha ocorrido. As funções relacionadas ao tratamento dos eventos gerados no sistema podem ser geradas entre os

diferentes níveis e dentro de um mesmo nível. De acordo com estas origens, elas podem ser definidas da seguinte forma:

a) Filtros entre os diferentes níveis

- **gα: EN → EN+1:** Função de filtragem de eventos do nível superior para um inferior.
- **gα: EN+1 → EN**: Função de filtragem de eventos do nível inferior para um superior.

b) Filtros dentro do mesmo nível

- **iα: GN → iN** : Função de filtragem aplicada entre as mensagens geradas pelo grupo e repassadas para um indivíduo deste mesmo grupo.
- **iα: iN → GN** : Função de filtragem aplicada entre as mensagens geradas por um indivíduo e repassadas para o grupo.

3.3 Modelo de propagação de eventos

Um quesito importante em tais sistemas diz respeito a como os eventos serão propagados entre os clientes. A aplicação de filtros resolve o problema da redução do fluxo de informações que poderiam transformar alguns dos nós de nossa hierarquia em "gargalos" do sistema. Porém, uma estratégia deve ser traçada a fim de responder a algumas perguntas, como por exemplo, até onde um evento será percebido dentro de um ambiente, quem perceberá tais eventos e como eventos são passados de um ambiente para outro ?

Nenhum dos trabalhos encontrados na literatura resolve o problema do tiro na multidão. Eles não repassam eventos entre seus ambientes, principalmente quando vários servidores estão representando um único ambiente virtual. Como no exemplo de uma sala lotada, onde temos vários servidores representando a mesma sala e os diversos clientes conectados ficam ilhados em seus servidores, estando aptos a trocar mensagens somente com os clientes que estejam no mesmo servidor que ele.

Propomos uma solução baseada na localização do usuário dentro do ambiente virtual, na proximidade entre cada ambiente e na arquitetura hierárquica

do H-N2N para resolver tal problema. Utilizamos para isso uma matriz de localização e um grafo ponderado para interligar em tais ambientes. Nessa matriz, os nós servidores são mantidos informados das posições de seus clientes dentro do ambiente assim como das proximidades entre os outros servidores.

Na Figura 31 temos um exemplo de um ambiente (uma casa), dividido em 7 partes ou salas (S0,S1,S2...S6). Nas salas S1 e S4 temos em ambos, apenas um avatar conectado (avatar A e avatar D, respectivamente), em S0,S2 e S5 nenhum avatar, em S3 dois avatares (B e C) e na S6 temos quatro (E, F, G e H).

Suponhamos agora que o avatar A detonasse dentro da S1 uma bomba com explosivo suficiente para destruir um estádio de futebol. Esse evento iria ser enviado para o servidor responsável pela S1 que por sua vez decidiria se esse evento seria ou não um evento relevante a ponto de ser repassado para um nível acima. Nesse caso, todas as salas devem receber o evento da explosão. Porém nem todos os eventos devem ser entregues a todas as salas ou a todos usuários do ambiente como um todo. Alguns eventos com prioridades baixas e intermediarias devem ser percebidos somente por quem está nas proximidades do que aconteceu.

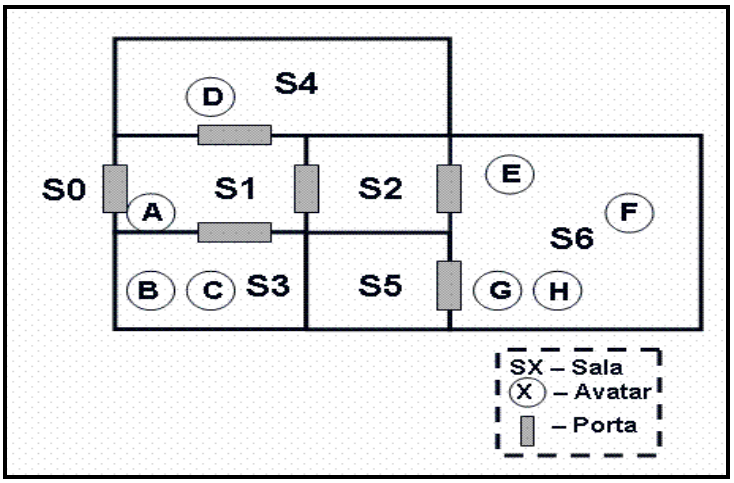

Figura 31 – Exemplo de Ambiente

Desse modo, precisamos descobrir a vizinhança entre as salas, para que ao ser gerado um evento, os servidores responsáveis possam saber para quem essa mensagem deve ou não ser repassada. Por exemplo, se o avatar A batesse na porta que leva à sala S4, um evento seria gerado somente para o servidor S4.

Baseado em uma descrição do mundo virtual, dada através de uma matriz (ver Figura 32) como mapeamento do ambiente, podemos usar arquivos *xml, vrml,* ou qualquer outra forma que especifique o mundo virtual, desde que possibilite a criação de um grafo que represente as ligações entre salas próximas, onde cada ligação tem um peso que diz o quão é fácil ou difícil que uma mensagem atravesse essa ligação, ver Figura 33 e Figura 34. A essa estrutura, chamamos de **grafo de dispersão**. Nesse exemplo as salas que possuem portas ligadas, o peso atribuído é de apenas 1, enquanto que as unidas fisicamente, mas que não possuem porta interligando, o peso é 5.

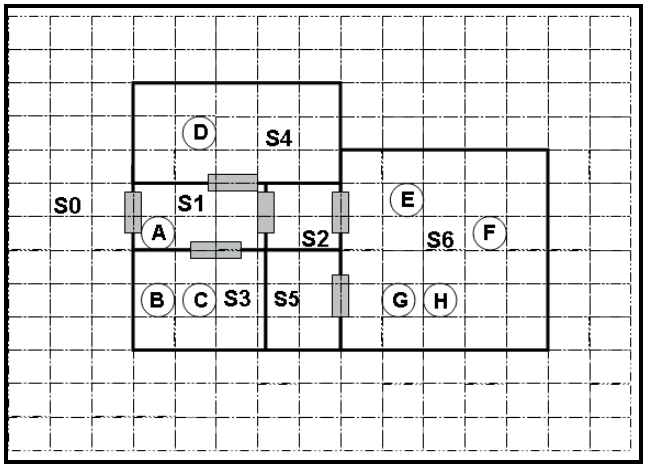

Figura 32 – Matriz armazenando as posições dos clientes

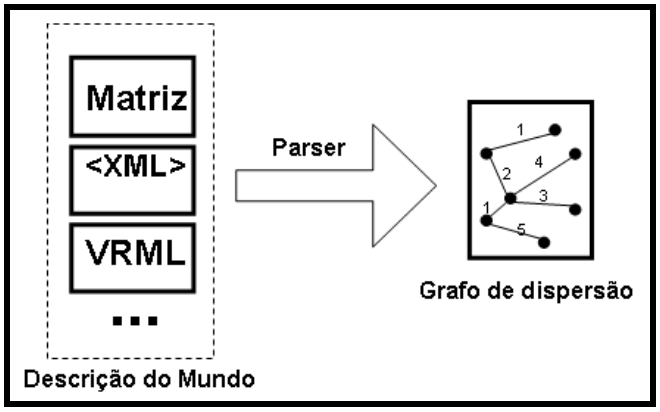

Figura 33 – Geração do Grafo de dispersão

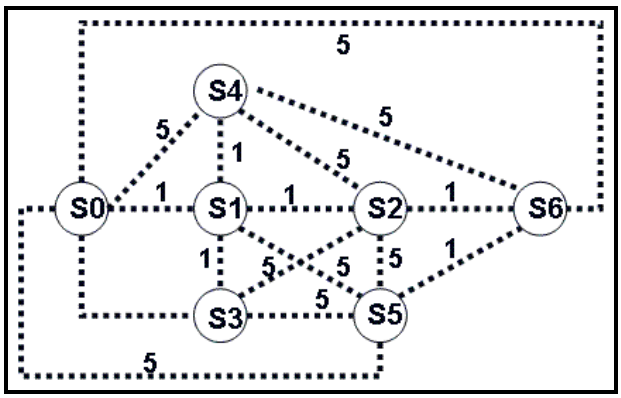

Figura 34 – Grafo de Dispersão interligando as salas

Essa solução transforma o H-N2N em uma arquitetura hierárquica que pode ser exibida em um diagrama em três dimensões. Decidimos organizá-la em forma piramidal, já que teremos um servidor inicial, que contém a cópia inicial do mundo, e conectado a ele estarão os clientes, que por sua vez se transformarão em servidores e com isso assumirão a responsabilidade de atualização de uma parte do mundo, como de uma sala, por exemplo. Quando cada sala possuir um servidor responsável, poderemos ter uma arquitetura parecida com a da Figura 35, onde as bolas representam os elementos H-N2N(clientes e/ou servidores).

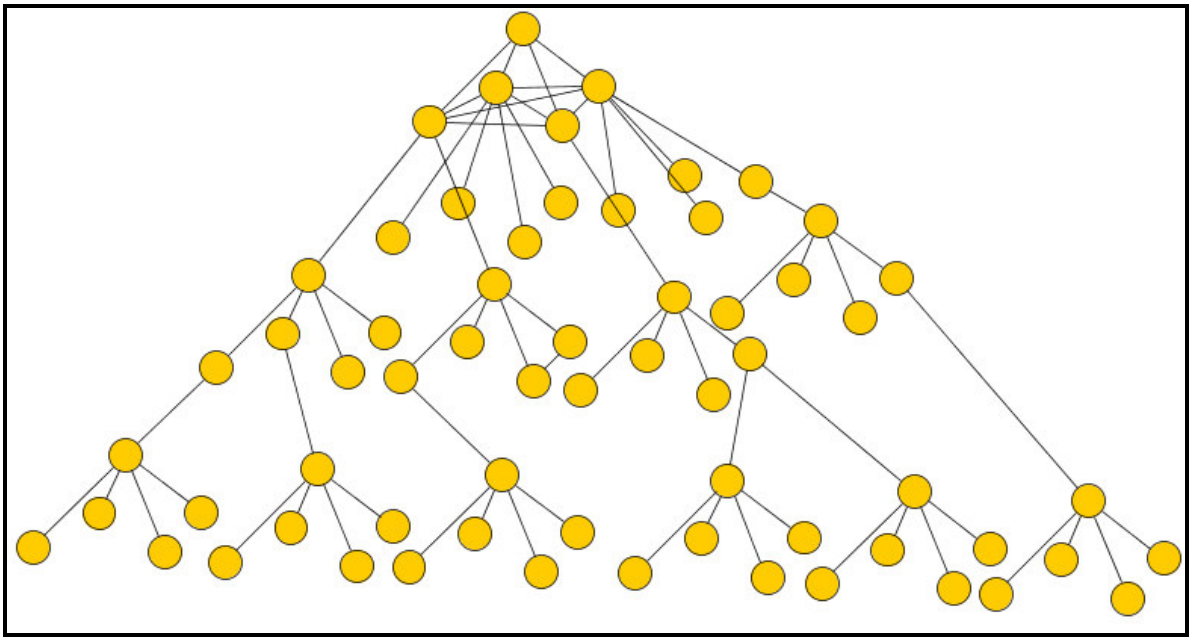

Figura 35 – Arquitetura Piramidal do H-N2N

Os primeiros filhos, ligados ao nó raiz, seriam o servidores dos grupos existentes. Eles estariam interligados, para deixar o sistema mais robusto. Caso o nó raiz caia o filho mais apto será eleito o sucessor e substituirá o nó pai (raiz).

Desse modo, organizamos os eventos em cinco tipos principais, baseados em seu grau de dispersão dentro do ambiente. São eles:

- **Tipo I (Intra-Grupo):** Eventos enviados para um subconjunto de clientes do grupo onde o evento foi gerado;
- **Tipo II (Grupo):** Eventos que serão repassados para todos os clientes dentro do grupo onde o evento foi gerado;
- **Tipo III (Sub-Árvore):** Eventos que serão repassados para toda a sub-arvore de grupos ao qual o grupo que gerou o evento pertence;
- **Tipo IV (Sub-Árvore e Vizinhos):** Idem tipo III além de os eventos serem repassados para os grupos vizinhos a este grupo, baseado no grafo de dispersão;
- **Tipo V (Árvore):** Eventos serão enviados para todos os grupos da árvore H-N2N.

Dependendo da aplicação, podem ser criado níveis extras de propagação de eventos, baseados em necessidades intrínsecas da aplicação. No capítulo 4, expomos os detalhes da implementação do grafo de dispersão.

3.4 Componentes da arquitetura H-N2N

Os principais componentes pertencentes à arquitetura do H-N2N são: ApplicationServer, GroupServer, SlaveServer e UserClient. As funções de cada componente são descritas a seguir:

- **ApplicationServer:** Componente responsável pela espera das conexões dos clientes e criação dos grupos de clientes, ele é o servidor principal.
- **UserClient:** Componente que representa o cliente usuário, fazendo o pedido de conexão ao ApplicationServer, e enviando as mensagens do usuário para o servidor.

- **GroupServer:** Esse componente contém a lista dos SlaveServer que representam os clientes conectados. Ele é o responsável por gerenciar as trocas de mensagens entre os clientes (UserClient) de um grupo em particular. Ele também responde pelo processamento das operações de filtragem e junção. O GroupServer contém informações sobre seus irmãos mais próximos.
- **SlaveServer:** Mantém uma comunicação com o cliente. Existe um para cada cliente. Auxilia o GroupServer na tarefa de envio e recebimento de mensagens dos clientes (UserClient).

Maiores detalhes sobre esses e os demais componentes do H-N2N serão vistos na seção 4.1.3.

Na Figura 36, podemos ver um exemplo de possível configuração. O sistema possui 37 clientes conectados (UC), sendo que 8 desses clientes estão fazendo também o papel de servidores (GS). Cada cliente possui um SlaveServer (SS) associado.

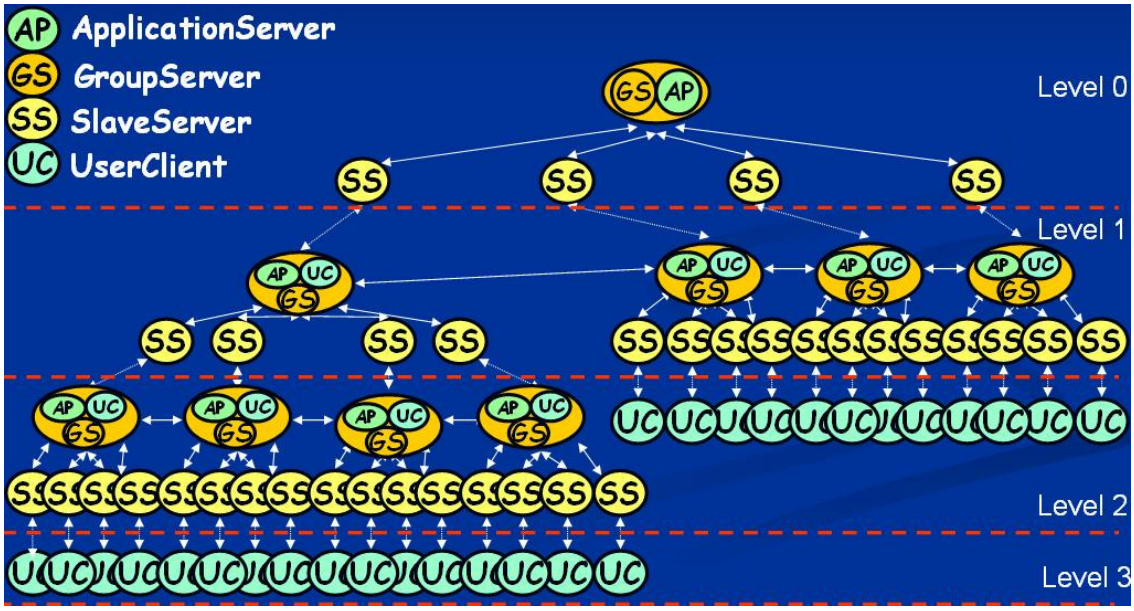

Figura 36 – Componentes do H-N2N numa configuração exemplo

Inicialmente o ApplicationServer é o único que contém toda a especificação do mundo, mundo esse que dependendo da aplicação pode também ser modelado hierarquicamente, como mostrado na Figura 37.

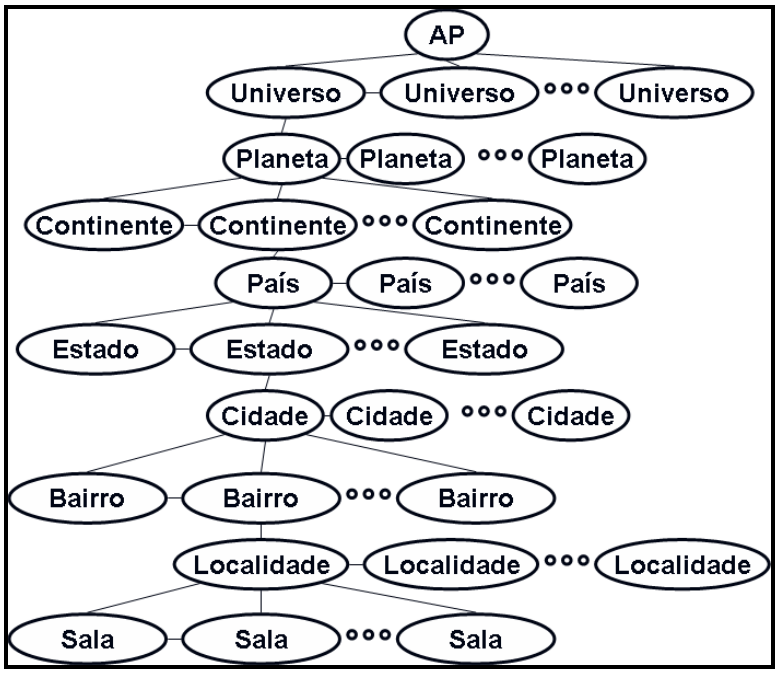

Figura 37 – Mundo Hierarquizado

O ApplicationServer instancia toda essa estrutura e fica a espera de clientes que possam vir a dividir responsabilidades, ajudando-o na tarefa de manter o mundo. Cada cliente que chega é candidato a ser um novo servidor que possa vir a se agregar a rede do H-N2N.

Características como escalabilidade, armazenamento, replicação, robustez são essências para tais sistemas. Desse modo, a pirâmide H-N2N divide sua arquitetura em três camadas, como mostra a Figura 38.

- Camada 1: Responsável pelo entrada no sistema e replicação dos servidores (Robustez);
- Camada 2: Responsável pelo armazenamento e consistência do mundo (Consistência);
- Camada 3: Responsável pelas interações entre os usuários (Escalabilidade);

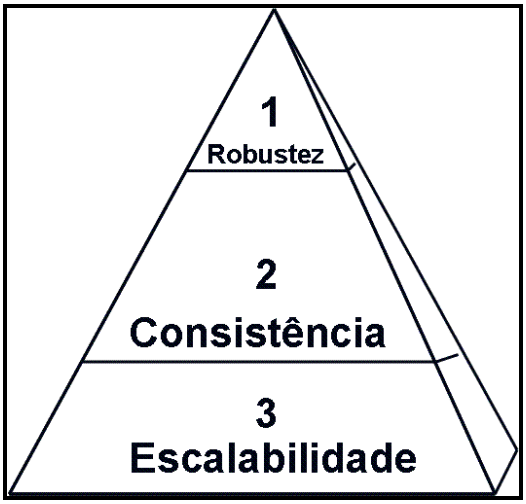

Figura 38 – Camadas da Pirâmide H-N2N

O H-N2N foi projetado para ambientes massivos, em que os clientes podem se transformar em servidores e vice-versa, mas, obviamente, sua arquitetura também permite a criação de aplicações simples provendo serviço para poucos clientes. Ou seja, embora ele especifique três camadas com serviços diferentes a serem preenchidas por servidores, um único servidor poderá prover o serviço por completo. Porém, na medida que os clientes forem acessando o sistema, chegará um momento que ele passará responsabilidades para esses clientes que se tornarão servidores H-N2N.

Inicialmente, tais servidores preencherão a camada 3 da pirâmide, e com isso sanando o problema da escalabilidade. O sistema pode aprender (utilizando *logs* do sistema, por exemplo), sabendo a um dado momento quais clientes são robustos e estáveis a ponto de fazerem parte da camada 1, deixando o sistema mais robusto, ou da camada 2, tornando o sistema mais consistente.

Ou seja, o H-N2N é flexível o suficiente para funcionar tanto em uma aplicação simples, como por exemplo executando uma aplicação de teleconferência numa tele aula particular, quanto para suportar uma aplicação mais massiva, como por exemplo a transmissão de um jogo de futebol através da aplicação torcida virtual (Tavares, 2004), como mostrado na Figura 39. Nossa arquitetura prevê ainda a inscrição de servidores que podem entrar para compor a

rede H-N2N, sem participar primeiramente como clientes, o que os levariam a fazer parte das camadas 1 e 2 antecipadamente, melhorando ainda mais o desempenho da aplicação.

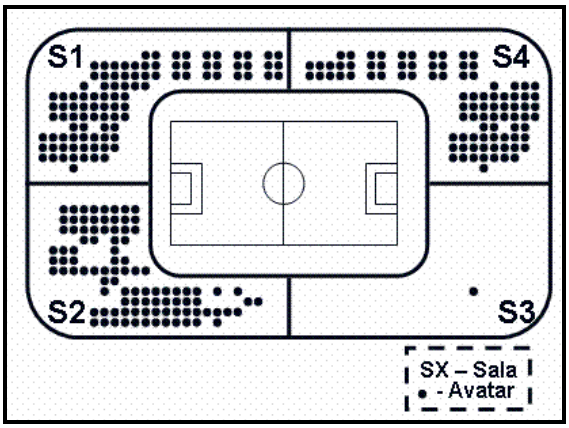

Figura 39 – Exemplo de Ambiente Massivo (Torcida Virtual)

A aplicação Torcida Virtual (ToV) (Tavares, 2004) possui a característica de concentrar uma grande quantidade de usuários em um mesmo ambiente virtual. Dependendo do jogo transmitido, a audiência, que reflete no número de usuários conectados, pode ser de milhões de pessoas, como em uma final de copa do mundo. Nesse caso, todos os usuários estariam virtualmente em um mesmo espaço, o estádio virtual. No caso do espaço físico real do estádio, este possui limitações físicas. Na ToV há espaço para todos os torcedores.

3.5 Robustez da arquitetura

Uma vez definida a estrutura básica da arquitetura, onde tratamos da entrada de usuários, organização dos clientes em grupos, filtragem, junção e propagação dos eventos entre os usuários, discutimos agora a saída de usuários (seja natural ou forçada) e como o sistema deverá se comportar nesses casos. O procedimento a ser tomado na saída de usuários do sistema depende da maneira como ela ocorre, podendo ser: normal ou forçada.

3.5.1 Saída Normal

Uma saída normal configura-se quando um usuário deseja sair do sistema e para isso ele realiza o logout, avisando para os demais sua decisão (através de mensagens de saída). As ações a serem realizadas neste caso dependem da função e localização do usuário dentro da árvore do sistema. Descrevemos abaixo as possíveis situações e soluções:

- **Saída de um nó filho:** como neste caso a única ligação existente com o nó que está saindo é com o seu pai, o que ocorre é simplesmente a retirada desse filho da lista do pai.
- **Saída de um nó pai:** quando isso ocorre, o pai elegerá seu descendente (filho, neto...) mais apto, ou seja, com mais recursos disponíveis para assumir o seu lugar, e, ao sair, todos os filhos órfãos se conectaram a esse novo pai. Esse procedimento também vale para o nó raiz.

3.5.2 Saída Forçada

Quando um nó sai da árvore do sistema de maneira abrupta, sem enviar as mensagens normais de saída do sistema isso se configura uma saída forçada. Assim como na saída normal, as ações a serem realizadas neste caso dependem da função e localização do nó que está saindo dentro da árvore do sistema. Abaixo, temos as possíveis situações e as soluções:

- **Falha em um nó filho:** idem Saída Normal.
- **Falha no servidor principal (nó raiz):** quando isso ocorre, os filhos não terão mais um pai para se conectar, então eles elegerão o filho mais apto, ou seja, com mais recursos para assumir o lugar do nó raiz, e com isso todos os filhos órfãos se conectaram a ele. Para que isso funcione corretamente, todos os filhos de nível 0 devem conhecer seus irmãos.
- **Falha em um nó pai:**
    - Estratégia 1: quando um nó possui vários filhos e o seu funcionamento é encerrado de maneira abrupta, seus filhos,

ao perceberem a sua falta, enviarão novamente um pedido de conexão à raiz do sistema.

- Estratégia 2: idem Falha no servidor principal.

3.6 Protocolo de Comunicação

Como citado anteriormente as mensagens que transitam num AVC podem ser classificadas nos seguintes tipos: Controle, Comunicação, Descrição Objeto/Cena e Dados Tempo Real. Para efetivar a troca de mensagens, adotamos um protocolo para prover a comunicação entre os componentes do H-N2N.

- **Controle:** Entrada/Saída de Usuários
  - MSG 1: Pedido de Entrada
  - MSG 2: Resposta Positiva do Pedido
  - MSG 5: Servidor Ocupado + Listas dos Nós Filhos
  - MSG 6: Ping para os Clientes.
  - MSG 7: Retorno do Ping + Capacidade. (Pong).
  - MSG 11: Aviso de Saída
  - MSG 12: Busca por Filho Disponível
- **Comunicação:** Mensagens textuais
  - MSG 8: Mensagens de Texto
- **Descrição Objeto/Cena:** Mudanças no ambiente
  - MSG 3: Movimento, Alteração de Ambiente (proveniente dos clientes);
  - MSG 4: Movimento, Alteração de Ambiente (proveniente dos servidores);
- **Dados Tempo Real:** Áudio/Vídeo
  - MSG 9: Mensagens de Áudio:
  - MSG 10: Mensagens de Vídeo

Para exemplificarmos os passos necessários a uma conexão no H-N2N, utilizaremos uma rede com três roteadores (0,1,2) e nove máquinas normais, ver Figura 40, em uma situação onde o servidor pode aceitar novas conexões.

Na Figura 40-A, o Servidor é inicializado no nó 7, e testa sua capacidade (que é de 4 clientes) e depois fica esperando conexões. Na Figura 40-B, o cliente testa sua capacidade (que é 3), e envia uma mensagem ao servidor com o pedido de entrada no sistema (MSG1). Na Figura 40-C, o servidor recebe a MSG1, verifica se está livre, como ele não tem nenhum cliente, o nó 3 é adicionado a sua lista de clientes, e por último o servidor envia uma mensagem informando o nó 3 da aceitação (MSG2). Na Figura 40-D, o nó 3 recebe MSG2, e começa a enviar periodicamente mensagens de atualização de sua posição no mundo (MSG3) e o nó 7 começa a atualizar o 3 com mensagens de atualização do mundo (MSG4).

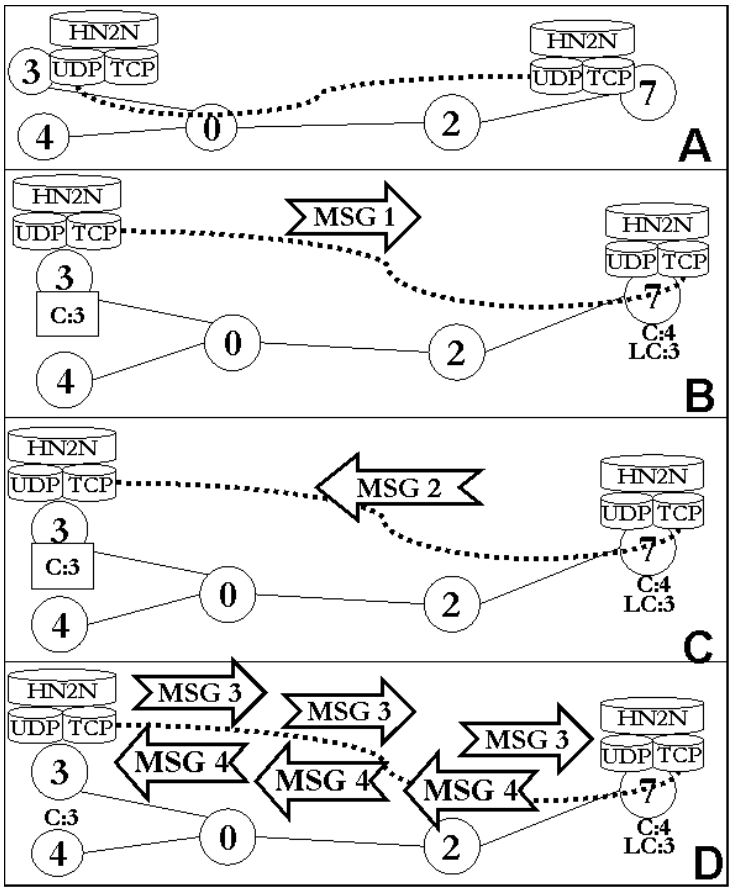

Figura 40 – Conexão de um cliente

A Figura 41 utiliza da mesma rede do exemplo anterior só que explica como o sistema resolve o problema quando o servidor chega ao seu limite de clientes. Na Figura 41-A, temos 4 clientes conectados ao servidor (nó 7). Na Figura 41-B, um quinto cliente tenta se conectar ao nó 7, enviando uma MSG1. Na Figura 41-

C, o servidor percebe que já esta no seu limite e nega o pedido de conexão, mas envia uma MSG5 contendo a listas de seus filhos para que esse novo cliente possa se conectar a algum deles. Na Figura 41-D o cliente(10) envia, para cada cliente da lista recebida na MSG5, uma MSG6. Essa mensagem tem o objetivo de colher informações a respeito dos filhos e com isso escolher qual o nó mais propício à sua conexão. Figura 41-E, os clientes que receberam um MSG6 retornam uma MSG7 com informações como sua capacidade de receber conexões assim como o *delay* com o cliente. Figura 41-F, o cliente analisa as MSG7 recebidas e percebe que o filho 5 é o mais apto a se tornar seu servidor, e com isso envia uma MSG1 retornando ao caso anterior da Figura 40. A Figura 41-G, mostra a possível configuração no caso dos clientes 8, 9 e 11 também se conectarem. Todos eles ficaram sendo clientes de 5, que se mostrou ser a melhor escolha. A Figura 41-H mostra a árvore de conexões do sistema no momento que somente 3, 4, 5, 6, 7, 10 e 11 estão participando.

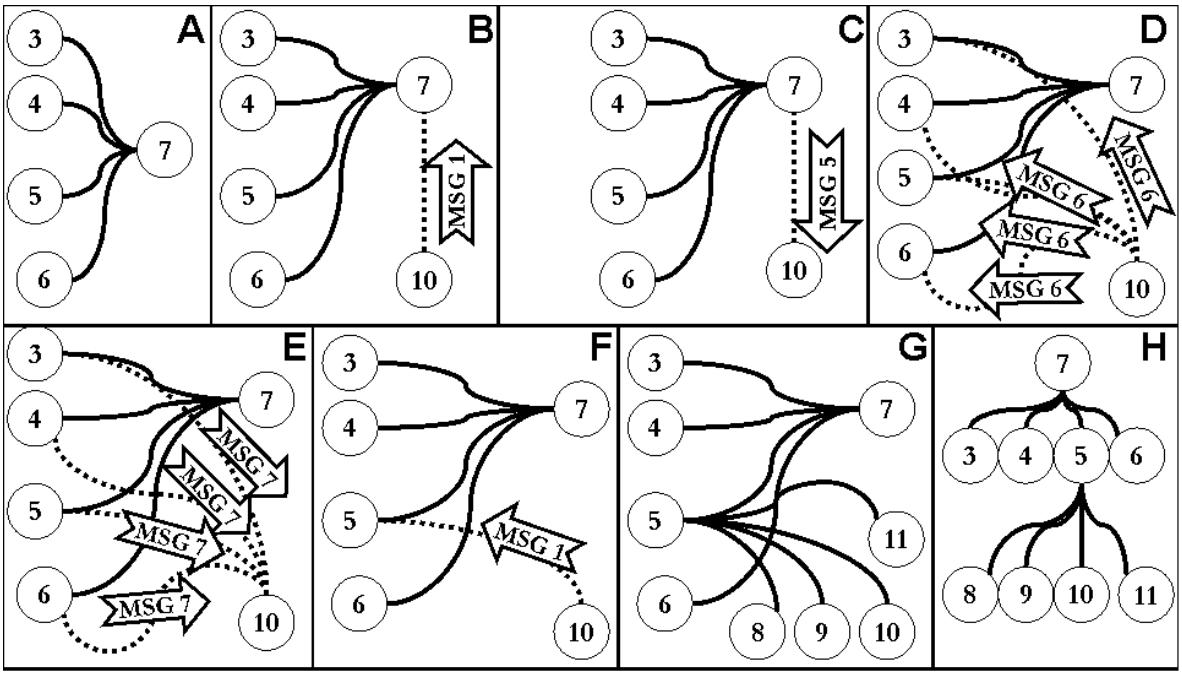

Figura 41 – Conexão quando o servidor está lotado

Capítulo 4 – Implementação do H-N2N

"Faça o que pode, com o que tem, onde estiver"

Roosevelt

A dinamicidade do mundo moderno exige que as aplicações sejam desenvolvidas o mais rápido possível. Um simples atraso na construção de algumas aplicações pode significar o fracasso de um sistema. Por isso, o uso de *frameworks* (Roberts, 1997) é indicado como uma boa estratégia para enfrentar o problema de reduzir o tempo de desenvolvimento. A utilização de *frameworks* permite acelerar o ciclo de desenvolvimento e melhorar a qualidade das aplicações.

Em nosso caso, o desenvolvimento de ambientes virtuais colaborativos massivos (AVCM) é considerado não trivial, pois são aplicações nas quais muitas vezes estão envolvidas diversas mídias, que demandam necessidades especiais de processamento e distribuição pela rede. No presente trabalho, considerando técnicas que possibilitem facilitar o processo de criação dos sistemas acima mencionados, optamos pela utilização de *frameworks*.

Framework é essencialmente uma técnica de reuso orientada a objetos (Fayad, 1999). O desenvolvimento baseado em *framework* utiliza programas e/ou aplicações semi-prontas, classes abstratas, através das quais novas aplicações do mesmo domínio podem ser construídas. Sob a ótica do reuso, a utilização de *frameworks* atinge três dimensões: reuso de análise, de projeto e de código (Biggerstaff, 1989).

O presente trabalho originou-se da necessidade de generalizar a aplicabilidade de três sistemas produzidos nos Laboratórios Natalnet/Lavid: ICSpace (Tavares, 2001), Vixnu e Torcida Virtual (Tavares, 2004). O ICSpace é um espaço cultural virtual, na Internet, que permite que vários usuários, visitantes ou autores, circulem pelos ambientes virtuais do mesmo e apreciem o acervo multimídia exposto, formado por mídias digitais como imagens, sons e vídeos. O Vixnu foi originado de uma extensão do conceito de espaço virtual do ICSpace,

para permitir a percepção e a comunicação entre os seus visitantes. Como visto no capítulo 1, a Torcida Virtual (ToV) é uma aplicação distribuída para plataforma de TV Digital que provê ferramentas para que o telespectador possa assistir e participar, "junto" com outros telespectadores, da recepção de eventos esportivos, como por exemplo, uma partida de futebol. Nela, é permitido o compartilhamento de áudio com os demais participantes da aplicação (torcedores).

Encontramos na literatura dois *frameworks*, relacionados com o tema do trabalho aqui proposto: o Amphibian (Bittencourt, 2003) e o COSMOS (Darlagiannis, 2000). Estes *frameworks* foram desenvolvidos com base no conceito de aplicações colaborativas distribuídas. Outros trabalhos foram encontrados na literatura relacionados a ambientes massivos, porém não utilizam o conceito de *frameworks* (Greenhalch, 1999; Oliveira, 2003).

O Amphibian é um *framework* desenvolvido para a criação de máquinas de jogos (*game engines*) multi-plataformas. O objetivo do Amphibian é permitir a construção de máquinas de jogos, tanto para jogos simples quanto complexos, que executem em computadores pessoais, computadores de mão (*handhelds*) e telefones celulares. Com a divisão em componentes provida pelo Amphibian, espera-se que seja possível reutilizar objetos lógicos de um jogo, além dos objetos estruturais, como por exemplo os visualizadores, a arquitetura cliente-servidor e os mecanismos de persistência. O intento é que o desenvolvedor crie a lógica de determinado jogo e que esta possa ser reutilizada para criar uma versão para computadores pessoais, *handhelds* e/ou telefones celulares.

Certamente, cada plataforma terá implementações específicas, sendo, entretanto, a lógica do jogo mantida. Desta forma, para criar uma máquina de jogos, bastaria escolher os componentes principais, a lógica do jogo, e efetuar a criação dos recursos multimídia. O nome atual do Amphibian é JFrog (Bittencourt, 2008).

O COSMOS (Sistema Colaborativo baseado em Objetos e *streams* MPEG-4) é um *framework* orientado a objetos que aumenta a velocidade de desenvolvimento de aplicações colaborativas. O COSMOS oferece uma base para

construção de novas aplicações colaborativas, com componentes extensíveis, modulares e reusáveis.

O Amphibian/JFrog e o COSMOS foram escolhidos para estudo por se tratarem de *frameworks* utilizados para a construção de AVCs. Após uma análise de ambos *frameworks*, tiramos as seguintes conclusões: o Amphibian não é massivo, por isso não satisfaz nossas necessidades. O Cosmos utiliza IP multicast (RFC 2588), o que inviabiliza aplicações voltadas para Internet, pois o multicast não é implementado em todas as redes que compõem a internet atualmente.

Embora existam várias soluções que tratem o problema de lidar com AVC Massivos, não foi encontrado nenhum que a disponibilize na forma de um framework que facilite o desenvolvimento de tais aplicações. Sendo assim, propomos, na próxima seção, uma maneira elegante, rápida e flexível de construir ambientes virtuais colaborativos massivos (o *framework* H-N2N), que pode agilizar o processo de criação e manutenção de AVCM dentro dos mais variados ambientes: LANs, ambientes WEB, ou ate mesmo em Televisão Digital Interativa. Usamos *multicast* a nível de aplicação para a troca de mensagens.

4.1 O Framework H-N2N

No desenvolvimento do projeto do H-N2N, procuramos ser o mais genérico e abrangente possível, visando facilitar o desenvolvimento de diversas aplicações. Porém, é natural que o *framework* H-N2N possua inicialmente características que nos leva a classificá-lo como sendo do tipo *whitebox* (Fayad, 1999), pois ele necessita de intervenção programática do desenvolvedor para a geração de suas aplicações. Por ser um *framework,* o H-N2N tem intrinsecamente a característica de ser evolutivo, Dificilmente, o projeto de um *framework* é dado por encerrado. Se isto acontece, sinaliza que ele não está mais sendo utilizado, ou que, por razões das mais diversas, foi descontinuado (Fayad, 1999).

4.1.1 Desenvolvimento do Framework

Para a construção do H-N2N, utilizamos uma metodologia de construção de *frameworks*, criada por Johnson (JOHNSON, 1993), intitulada “Projeto Dirigido Por

Exemplo", que divide o ciclo de desenvolvimento nas seguintes fases: análise do domínio de aplicações, projeto da hierarquia de classes e validação do *framework*.

4.1.2 Análise do Domínio de Aplicações

A análise do domínio de aplicações é a primeira etapa do processo de desenvolvimento de um *framework* (Johnson, 1993). Nela, o desenvolvedor precisa analisar no mínimo quatro aplicações que apresentem características comuns entre si, ou seja, que estejam enquadradas em um mesmo domínio e, a partir dessa análise, chegar aos pontos comuns deste domínio de aplicações.

Como visto acima, neste trabalho, as aplicações escolhidas foram: Vixnu, Torcida Virtual além de uma aplicação de Chat (Bate-Papo) e Enquete (Perguntas e respostas). Estas aplicações foram escolhidas por já terem sido idealizadas e implementadas pelo Laboratório Natalnet/Lavid, o que viabiliza a realização de uma melhor análise de suas características. Após compararmos cada uma destas aplicações entre si, identificamos a área de interseção entre elas. O gráfico da Figura 42 sumariza estas aplicações, bem como suas inter-relações, baseando-se na teoria dos conjuntos. A área no centro representa soluções em termos de design e implementação que se repetem. Esse é o ponto por onde iniciaremos a construção do framework H-N2N.

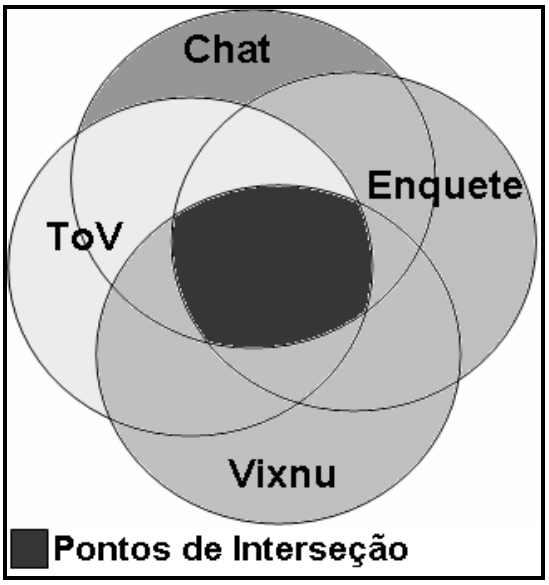

Figura 42 – Pontos de Interseção (Vixnu, Enquete, ToV e Chat)

Para cada aplicação escolhida, existe uma parte dela que pode ser reutilizada tanto para a construção destas aplicações quanto para qualquer outra que compartilhe das mesmas características. Após esta análise inicial, alguns pontos comuns foram encontrados:

- Todas as aplicações utilizam a arquitetura cliente/servidor;
- O número de usuários pode variar, desde dezenas até a milhares ou mesmo a milhões de usuários (no caso de TV);
- No servidor existem processos que tratam separadamente as requisições de cada cliente (Servidores Escravos, como será visto adiante);
- Os clientes são organizados em grupos, cada cliente pertence a um gupo;
- Um cliente pode sair de um grupo e entrar em outro, se necessário;
- Todas as aplicações possuem uma conexão de controle/dados (TCP), e possivelmente outras de dados (TCP, UDP, RTP, etc);
- As aplicações suportam vários tipos de mensagens, possivelmente com conteúdos multimídia;
- Existe a troca de mensagens entre um cliente e outro cliente, ou entre um cliente um grupo de clientes;
- As mensagens podem ser processadas tanto na chegada ao servidor, quanto ao cliente;
- Os clientes podem conter várias interfaces gráficas diferentes; ou seja, as mensagens não são dependentes do modo como o usuário irá visualizá-las;

Analisando os pontos em comum, podemos identificar que as aplicações acima podem ser classificadas como virtuais colaborativas, usando ambientes massivos, sendo este o domínio definido para o presente trabalho.

4.1.3 Projeto da Hierarquia de classes

O próximo passo é a concepção de uma estrutura de classes abstratas para serem reutilizadas na construção, não só das aplicações discutidas acima, mas também de outras aplicações pertencentes ao domínio definido. Na solução proposta neste trabalho, definimos os seguintes componentes comuns a todas as aplicações:

- **ApplicationServer**

É o responsável por receber conexões dos clientes e por realizar o gerenciamento de troca de clientes de um grupo para outro. Mantém uma lista com todos os grupos de clientes.(já comentado na seção 3.4).

• **SlaveServer**

É o responsável pelo processamento das mensagens de um cliente específico; para cada cliente. Existe um SlaveServer associado. (já comentado na seção 3.4)

• **GroupServer**

Como o próprio nome diz, é o responsável pela formação de grupos; contém a lista dos SlaveServers que se comunicam com os clientes pertencentes a este grupo. (já comentado na seção 3.4)

• **LinkObserver**

Componente responsável pelo processamento das mensagens. Existem instâncias deste componente tanto do lado cliente quanto do lado servidor, podendo ele ser acoplado ao lado servidor (SlaveServer, GroupServer) realizando algum processamento das mensagens recebidas, ou ao lado cliente (UserClient, descrito abaixo), tratando neste as mensagens enviadas pelo servidor.

• **CommunicationModule**

Encarrega-se tanto de fazer a entrega quanto de receber as mensagens em ambos os lados. Existem instâncias deste componente tanto no lado cliente quanto no lado servidor. O CommunicationModule é responsável pela comunicação entre os clientes e o servidor.

• **CommunicationLink**

É o processo responsável por enviar, receber e armazenar as mensagens. Pode ser implementado de várias maneiras como, por exemplo, através de um socket, utilizando UDP ou TCP, por exemplo.

• **UserClient**

O UserClient é um “modelo” do cliente. Ele contém um módulo de comunicação CommunicationModule, e também implementações do padrão MVC. (já comentado na seção 3.4)

• **MVC (Model-View-Controller)**

Cada UserClient pode conter uma ou várias implementações do padrão MVC(Glenn, 1988), onde a parte relativa ao *Controller* receberá tanto eventos do usuário, quanto mensagens que chegaram pelo comunicador (CommunicationModule).

Os componentes que fazem parte do *framework* foram idealizados com base nas características encontradas na seção 4.1.2. A seguir, é apresentada a implementação do *framework*. Optamos por implementar o *framework* H-N2N usando a linguagem de programação Java. Foram criados cinco pacotes para a estruturação do *framework*, de forma a dividir suas classes de acordo com suas funções: *server, client, process, net* e *util*.

Basicamente, o pacotes *server*, *client* e *process* dependem do pacote *net*, utilizado para a comunicação entre os subsistemas cliente e servidor. O pacote *process* realiza processamento das mensagens que chegam pelas classes do pacote *net*. Nessa atual versão o H-N2N conta com 36 classes e 5 interfaces e 1919 linhas de código, onde as principais classes definidas para cada um dos pacotes do *framework* são:

- ***br.natalnet.hn2n.server:****ApplicationServer, GroupServer, SlaveServer, GroupCreator, SlaveCreator e Portal.*
- ***br.natalnet.hn2n.client:*** *UserClient.*
- ***br.natalnet.hn2n.net:****Message, MConnectUDP, MDataUDP, MIdentification, MCreateServer, MHelp, MTrySon, MResource, Package, CommunicationModule, CommunicationLink, LinkTCP, e LinkUDP.*
- ***br.natalnet.hn2n.process:*** *LinkObserver.*
- ***br.natalnet.hn2n.util:*** *MessageQueue, Mutex, PoolMessage e PooledMessage*.

A Figura 43 apresenta o diagrama de classes da parte relativa ao servidor, que inclui as classes *ApplicationServer, GroupServer* e *SlaveServer*. As funções da maioria dessas classes já foram discutidas na definição dos componentes, acima. Além dessas, existem duas classes novas: *GroupCreator* e *SlaveCreator*. Estas classes utilizam o padrão Fábricas de Objetos (*Factory Method*) (Gamma, 1995).

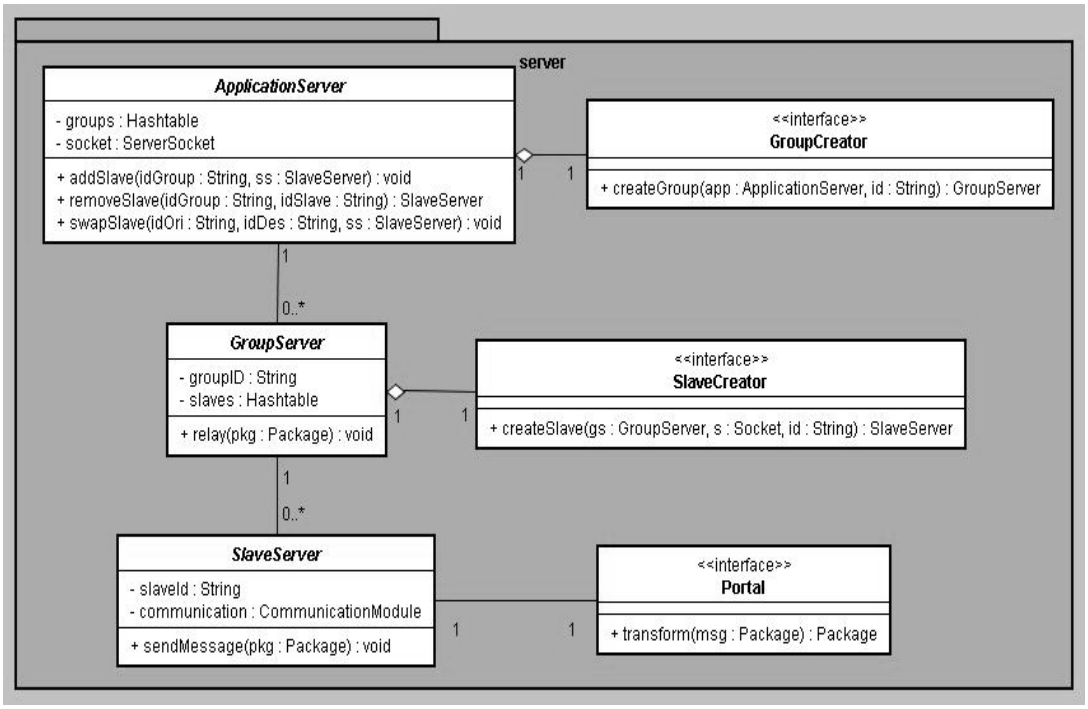

Figura 43 – Diagrama de Classes do package br.lavid.hn2n.server.

Como podemos ver na Figura 43, a classe abstrata ApplicationServer contém dois atributos: um criador de grupos (GroupCreator), uma tabela hash para armazenar os vários GroupServers que serão criados pelo criador de grupos, e um serverSocket utilizado para receber as conexões dos clientes. A classe abstrata GroupServer possui como atributos: um criador de escravos (SlaveCreator), uma tabela *hash* para armazenar referências para SlaveServers que serão criados além de um identificador. A classe abstrata SlaveServer contém um CommunicationModule como atributo, assim como um identificador, e uma referência para o GroupServer que ele está armazenado.

O Componente Portal é utilizado para realizar transformações das mensagens que vão seguir para os clientes. Através deste componente, é possível que clientes utilizando diferentes interfaces (3D, 2D ou 1D), sejam eles PCs, Celulares ou *SetopBox,* interajam de maneira transparente em um mesmo ambiente, provendo assim interdimensionalidade ao sistema (Burlamaqui, 2006a). Este conceito foi implementado seguindo o diagrama da Figura 4, segundo o padrão de projeto Strategy (GAMMA, 1995).

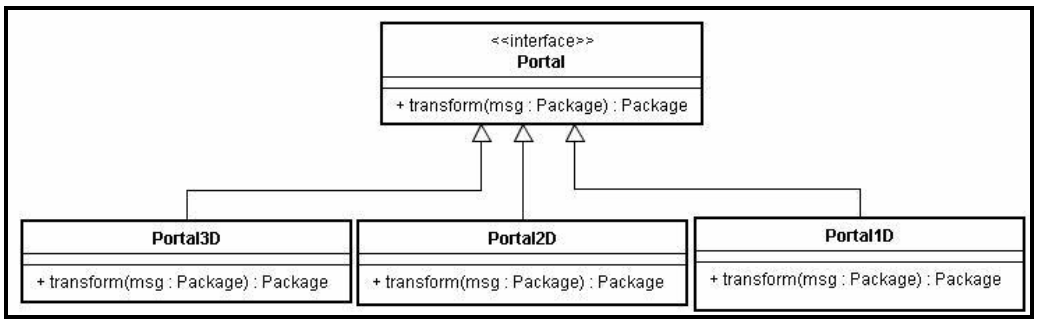

Figura 44 – Diagrama de classes do PORTAL

Existem ainda métodos na classe ApplicationServer que são responsáveis pela adição, remoção e transferência de um SlaveServer de um GroupServer para outro.

A Figura 45 apresenta o diagrama de classes do componente de comunicação, que foi organizado dentro do pacote br.natalnet.hn2n.net.

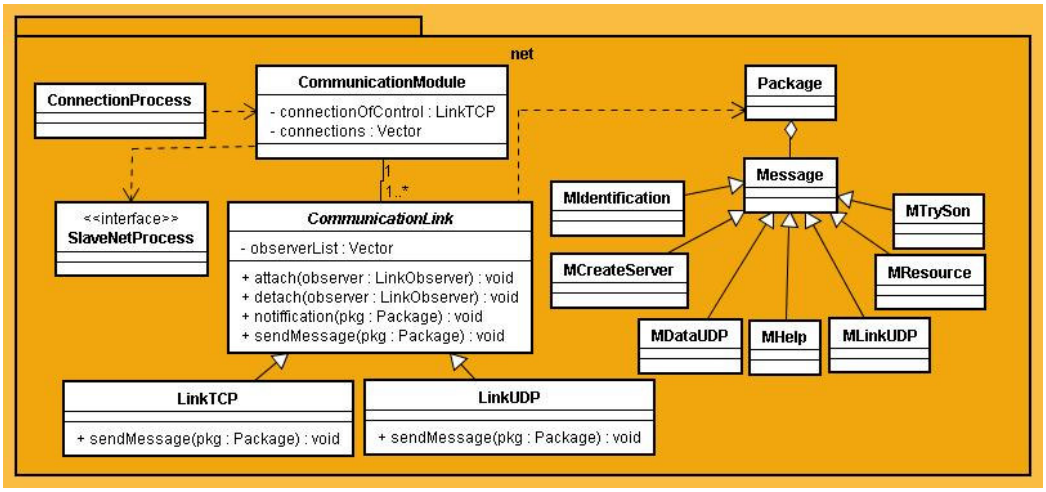

Figura 45 – Diagrama de Classes do package br.lavid.hn2n.net

O CommunicationModule é o componente responsável pela comunicação entre os lados cliente e servidor. Há uma instância dele para cada cliente (junto ao UserClient), e outra correspondente no lado servidor (junto a cada SlaveServer). Ele é implementado utilizando-se uma classe que contém dois atributos:

- Uma conexão de controle (subclasse de CommunicationLink), responsável pela troca confiável de mensagens de controle (valor padrão LinkTCP), como pedido de abertura de novas conexões ou mesmo mensagens de controle, referentes a aplicações especificas.

- Um vetor de conexões (subclasse de CommunicationLink), que serão utilizadas pelas aplicações para trafegar o fluxo de dados, podendo ser utilizado qualquer protocolo. Implementamos usando os protocolos UDP (com LinkUDP) e TCP (LinkTCP).

As mensagens enviadas pelas conexões são sempre instâncias da classe Package, que contém uma instância de uma subclasse da classe Message e um número de identificação. Para exemplificar o funcionamento do sistema na transmissão de uma mensagem, suponha um cliente e um servidor em execução. Como para cada cliente existe um CommunicationModule e para cada cliente existe um SlaveServer que, por sua vez, também possui um CommunicationModule, teremos dois CommunicationModule (um no lado cliente e outro no lado servidor) trocando mensagens. Essas mensagens são encapsuladas dentro de objetos do tipo Package (que contêm objetos do tipo Message). Assim, podemos criar um Package e enviá-lo tanto do cliente para o servidor como no sentido inverso, utilizando para isso o método sendMessage() que recebe como parâmetro um objeto do tipo Package.

A classe ConnectionProcess é utilizada para o processamento do pedido de abertura de novas conexões assim como para o tratamento de outras mensagens de controle do sistema. Ou seja, como ela implementa a interface LinkObserver, se algum Package chegar contendo alguma mensagem de pedido de abertura de conexão, ela criará uma nova conexão atendendo o pedido e a adicionará no vetor de conexões do CommunicationModule. A MLinkUDP é um exemplo de mensagem de pedido de abertura de um link UDP. MDataUDP é um exemplo de mensagem de dados sobre um link UDP (ambas vistas na Figura 5).

4.2 Padrões no Framework

Alguns dos padrões de projeto usados na construção do framework H-N2N são: Observer, Factory Method, Message Queue e Pool Allocation.

4.2.1 Padrão Observer

A Figura 46 mostra o diagrama de classes exemplificando o padrão Observer no H-N2N. A classe abstrata CommunicationLink age como observável. Ela possui

duas subclasses: LinkTCP e LinkUDP que são os observáveis concretos, uma vez que são implementações da classe CommunicationLink. A Figura 46 inclui ainda uma interface LinkObserver e exemplos de suas subclasses: MixerProcess, ConnectionProcess e ChatProcess. As três implementam a interface LinkObserver que é a classe observadora.

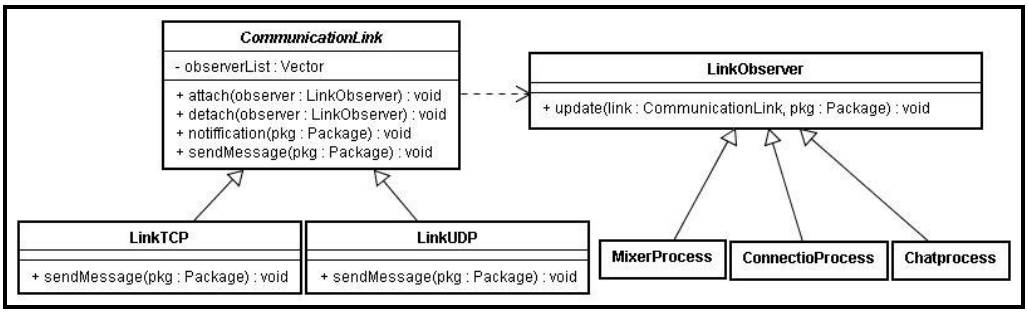

Figura 46 – Padrão Observer no H-N2N

As subclasses de CommunicationLink adicionam uma referência para todos os objetos que as estão observando. Sendo assim, sempre que houver uma modificação em seu estado, todos os seus observadores serão notificados.

4.2.2 Padrão Factory Method

Na Figura 47 , podemos ver um exemplo do uso do padrão *Factory Method* no H-N2N. Como não é possível prever como serão os servidores de grupo criados pelos usuários do *framework*, a interface *GroupCreator* permite que o programador defina uma classe que cria seus próprios servidores de grupo. Na Figura 47, é mostrada uma classe CriadorDeAmbientes, que implementa a interface *GroupCreator*, e que define como criar um servidor de ambiente.

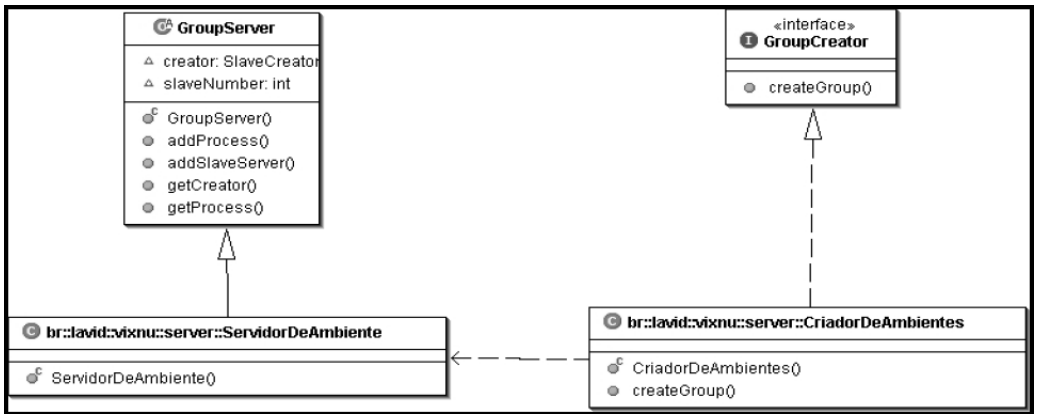

Figura 47 – Padrão Factory Method no H-N2N

4.2.3 Padrão Message Queuing

Dentro do pacote Util, temos as classes *MessageQueue* e *Mutex*. Elas implementam o padrão *Message Queuing*, ou seja, um fila de mensagens com acesso controlado através da implementação de um semáforo, aqui representado pela classe *Mutex*. Na Figura 48, podemos ver o diagrama de classes que mostra este padrão. A classe *BufferProcess* utiliza esta fila de mensagens para armazenar as mensagens que ele recebe.

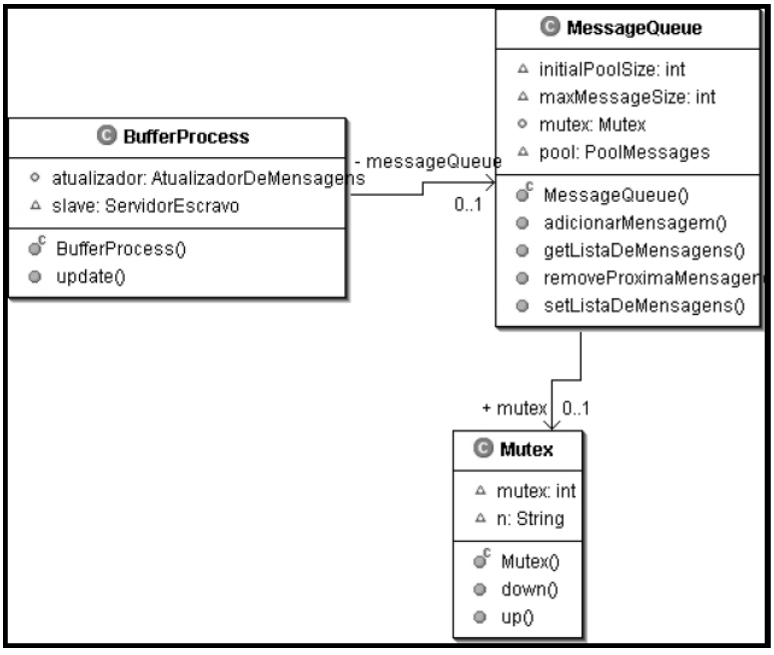

Figura 48 – Padrão Message Queuing no H-N2N

As classes mostradas na Figura 48 podem ser utilizadas quando se deseja

armazenar as mensagens recebidas por uma conexão. A classe *BufferProcess* é um exemplo de classe responsável pelo armazenamento das mensagens recebidas por uma conexão. Ela funciona como um buffer, necessário para que, por exemplo, numa transmissão de dados multimídia como áudio ou vídeo, mensagens não sejam perdidas. Desse modo, há um processo com a tarefa única de armazenar as mensagens recebidas, de modo que outros processos que realizam a exibição ou outro tipo de processamento possam retirar as mensagens armazenadas nesta fila sem se preocupar se houve ou não perda de dados.

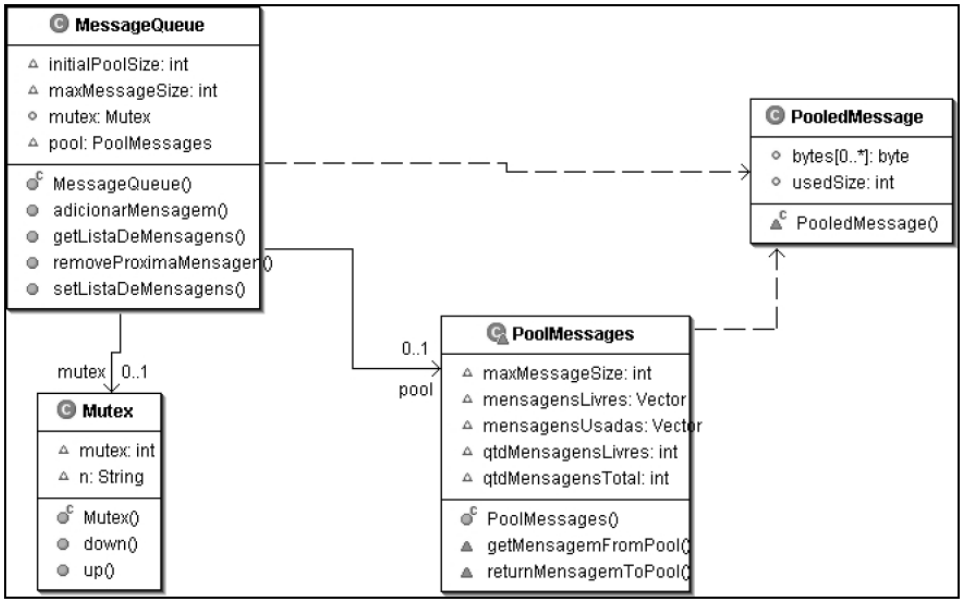

Figura 49 – Padrão Pool Allocation no H-N2N

4.2.4 Padrão Pool Allocation

Outro padrão também utilizado neste *framework* é o *Pool Allocation*. Como podemos ver na Figura 49, a classe *MessageQueue* utiliza um conjunto de mensagens para evitar a criação desnecessária de novas mensagens. Para isso ela acessa o conjunto (*PoolMessages*), retira uma mensagem que esteja livre (*PooledMessage*) e, assim que essa mensagem não for mais necessária, ela é devolvida ao conjunto. Notamos que aqui trabalhamos com dois padrões ao mesmo tempo, o *Pool Allocation* e o *Message Queuing*. Este padrão ajuda a diminuir o indeterminismo do tempo necessário para a criação de novos objetos do tipo *PooledMessage*. Isto se deve ao fato do conjunto de objetos que serão

utilizados ter sido criado na inicialização do sistema. Esse tempo é muito importante num sistema que necessita atender a requisitos de tempo-real.

4.3 Grafo de Dispersão

O DispersionGraph (grafo de dispersão) é o componente responsável por informar ao sistema como os ambientes estão interligados dentro do mundo virtual. Quando um evento é gerado, é atribuído um número a ele informando o seu grau de dispersão. Ele então, baseado na topologia do grafo de dispersão e nos pesos associados à suas arestas informa para que outros servidores essa informação deverá ser repassada.

```
<mundo>
        <sala nome="0">
            <entrada nome="1" peso="1"/>
            <entrada nome="4" peso="5"/>
            <entrada nome="3" peso="5"/>
        </sala>
        <sala nome="1">
            <entrada nome="4" peso="1"/>
            <entrada nome="0" peso="1"/>
            <entrada nome="3" peso="1"/>
            <entrada nome="2" peso="1"/>
        </sala>
        <sala nome="3">
            <entrada nome="0" peso="5"/>
            <entrada nome="1" peso="1"/>
            <entrada nome="2" peso="5"/>
            <entrada nome="5" peso="5"/>
        </sala>
        <sala nome="4">
            <entrada nome="0" peso="5"/>
            <entrada nome="1" peso="1"/>
            <entrada nome="2" peso="5"/>
            <entrada nome="6" peso="5"/>
        </sala>
        <sala nome="2">
            <entrada nome="4" peso="5"/>
            <entrada nome="1" peso="1"/>
            <entrada nome="2" peso="5"/>
            <entrada nome="5" peso="5"/>
            <entrada nome="6" peso="1"/>
        </sala>
        <sala nome="5">
            <entrada nome="3" peso="5"/>
            <entrada nome="1" peso="5"/>
            <entrada nome="2" peso="5"/>
            <entrada nome="6" peso="1"/>
        </sala>
        <sala nome="6">
            <entrada nome="4" peso="5"/>
            <entrada nome="2" peso="1"/>
            <entrada nome="5" peso="1"/>
        </sala>
</mundo>
```

Figura 50 – Descrição da vizinhança em XML

Em nossa implementação, esse grafo de dispersão é alimentado através de um arquivo XML que descreve que salas (grupos de usuários que são mantidos por um servidor) estão próximas umas das outras e se, embora próximas, um evento trafegaria facilmente de um servidor para o outro.

Esse arquivo XML é construído baseado na modelagem do ambiente virtual. Na Figura 50, temos um exemplo do XML que utilizamos na geração do grafo de vizinhança para o ambiente mostrado na Figura 31.

Um outro componente, um parser (XmlToDisGraphParser), lê o xml e alimenta o DispersionGraph. Uma vez atualizado, o DispersionGraph pode ser utilizado pelo ConnectionProcess para decidir se uma evento (mensagem) deve ou não ser repassada para outros servidores.

4.4 Validação do Framework

Dando prosseguimento à técnica **Projeto Dirigido por Exemplo**, chega-se à fase de testes. Nesta fase deve-se aplicar o *framework* na reconstrução das aplicações analisadas, a fim de validar e testar se ele realmente aumenta a velocidade de desenvolvimento e consegue apoiar a reconstrução total das antigas aplicações utilizando para isso as classes pertencentes ao *framework*.

4.4.1 Re-implementação

Durante a re-implementação das aplicações utilizadas como exemplos deste domínio de aplicações, pudemos observar o impacto obtido ao utilizarmos o H-N2N. Como será visto à frente, nos experimentos realizados, graças ao reuso propiciado pelo *framework* H-N2N, o código ficou menor, bem mais organizado e, além de tudo, o desenvolvimento dessas aplicações se tornou uma tarefa bem mais rápida do que se fôssemos realizá-la sem o seu uso. Com isso, podemos dizer que o H-N2N conseguiu cumprir com o objetivo para qual ele foi criado, servindo como ferramenta de reuso não só de código como também de projeto e design de aplicações multimídias colaborativas distribuídas.

O *framework* H-N2N possui atualmente cerca de duas mil linhas de código, divididas entre 41 classes/interfaces, distribuídas em 5 pacotes. Ou seja, ele

possui um código ainda pequeno, deixando claro que a sua maior contribuição no quesito reuso se concentra na análise e projeto. Contudo devemos notar que apesar de só serem duas mil linhas de código, serão duas mil linhas a menos para o desenvolvedor testar e manter.

Por exemplo, na implementação bate-papo textual (chat) utilizando o H-N2N, gastamos apenas 1358 linhas, se não o utilizássemos esse número duplicaria, ou seja esforço dobrado para manutenção. A Tabela 1 contém mais detalhes sobre a quantidade de linhas de código reutilizado pelas aplicações.

Tabela 1 - Reuso de implementação promovido pelo H-N2N

	Sem o H-N2N	Com o H-N2N
Enquete	2115 linhas de código	893 linhas de código
Chat Text	2951 linhas de código	1358 linhas de código
Chat Áudio	5012 linhas de código	1541 linhas de código
Torcida Virtual	6037 linhas de código	2365 linhas de código
Vixnu	8765 linhas de código	4334 linhas de código

É importante salientar que o H-N2N é um *framework* que, por não tratar de aspectos relativos a renderização, captura e exibição de áudio/vídeo, trabalha em conjunto com outras tecnologias, como por exemplo: awt(Java,2006), swing(Java,2006), swt (Java,2006) para a construção da interface gráfica, JMF ,JavaSound e JavaSpeech (Java, 2006) para captura e exibição de áudio e vídeo, Java3D (Java, 2006) para renderização de gráficos 3D, entre outras tecnologias. O seu foco está na troca das mensagens e gerenciamento de grupos. Dessa forma, temos um *framework* flexível que o torna apto a ser utilizado no desenvolvimento uma grande variedade de aplicações.

Observando a Figura 51, fica bem mais fácil entender como essas tecnologias trabalham juntas com o H-N2N. Começando da base da figura, temos a camada **net**, responsável pelo envio das mensagens pela rede. Depois que uma mensagem chega, ela sobe para a camada **process**. Uma vez que tenhamos a mensagem já processada, podemos armazená-la em *buffers* existentes na

camada **util**. Então, a camada **server** e **client** terão acesso à mensagem. Seu conteúdo poderá ser exibido usando uma camada externa ao H-N2N que contém todas às tecnologias comentadas no parágrafo anterior. Deixamos aqui claro que o H-N2N não inclui outros *frameworks* em sua implementação, mas os prevê na sua utilização.

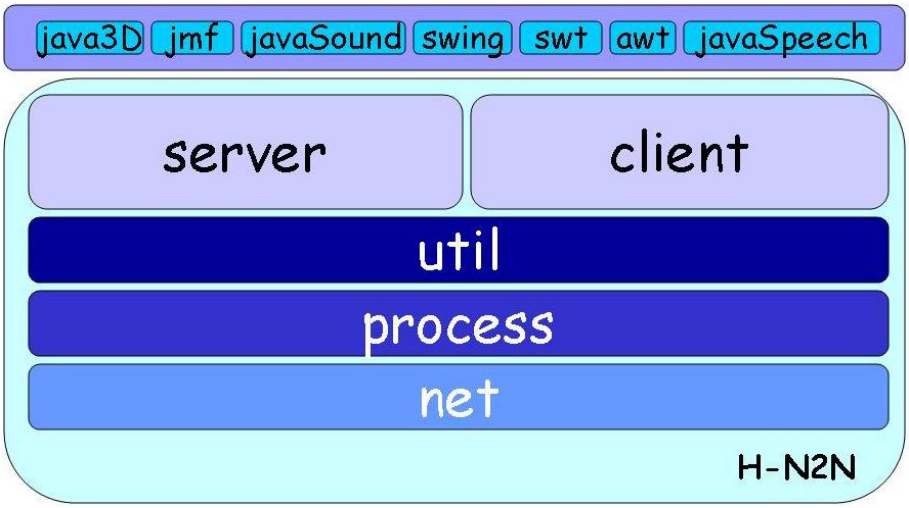

Figura 51 – Camadas internas do H-N2N

4.4.2 Aprendizagem do Framework

Uma vez definido, implementado e testado, um *framework* não estará completo sem a sua documentação. Criado com a perspectiva de aumentar a produtividade e qualidade, em função da reutilização promovida, os usuários e desenvolvedores de aplicações no domínio coberto pelo *framework* necessitam entender sua estrutura, de modo a cumprir os requisitos da aplicação a ser desenvolvida. Pensando neste objetivo, definimos alguns aspectos relacionados à documentação e aprendizagem do *framework* H-N2N.

A documentação do H-N2N foi preparada seguindo o modelo proposto por Johnson (Johnson, 1992). Segundo este modelo, a documentação deve ser dividida em três níveis de profundidade, de modo a facilitar a extração das informações necessárias aos propósitos dos usuários:

- Nível 1: saber que tipos de aplicações podem ser desenvolvidas;
- Nível 2: ser capaz de desenvolver aplicações elementares sob o *framework*;

- Nível 3: conhecer em detalhes o projeto do *framework*;

Mais detalhes a respeito da documentação podem ser encontrados em sua página Internet (H-N2N, Documentação). Visando verificar a facilidade de uso do H-N2N, inserimos o mesmo em duas aulas na disciplina Projeto de Sistemas Web e Conteúdo Digital, ministrada para alunos do Curso de Especialização (Pós-Graduação Lato Sensu) da Universidade Potiguar - Tecnologia e Desenvolvimento de Software Orientado a Objetos para Web. Cada aula teve duração de 4 horas e todos os 20 alunos foram capazes de entrar no Nível 2 de entendimento do H-N2N, ao desenvolverem uma aplicação de bate-papo. Neste estudo de caso, todos os alunos já sabiam programar na linguagem Java.

4.5 Comparação do H-N2N

Nesta seção, faremos uma comparação do H-N2N com os *frameworks AMPHIBIAN* e *COSMOS*, descrevendo suas principais vantagens e desvantagens. Uma vez que o AMPHIBIAN é destinado à criação de jogos, seu domínio de aplicações tende a ser menor do que o domínio do *COSMOS* e do H-N2N. Ele ainda está em fase de construção e, embora especifique em sua arquitetura um módulo para comunicação que seria o responsável pela comunicação cliente/servidor, este ainda não está implementado, nem existem detalhes de como será implementado. Como uma das vantagens do AMPHIBIAN, podemos citar a disponibilidade de muita documentação, inclusive códigos fontes. O *COSMOS* é um *framework* baseado em MPEG-4, e o seu funcionamento depende desse padrão. Não é um *framework* de código aberto e existe pouca documentação a seu respeito. Com relação à comunicação, ele utiliza conexões *IP multicast*. Em um ambiente Internet, nem todos os roteadores usam este padrão, podendo inviabilizar algumas aplicações.

Tanto o COSMOS quanto o AMPHIBIAN enfatizam muito o aspecto da renderização, tanto é que incluem muitas classes para esta tarefa. No H-N2N, a preocupação maior é com a distribuição e processamento das mensagens e

gerenciamento de grupos de clientes. Portanto, fica livre ao usuário poder escolher a melhor forma de exibição dessas mensagens.

Na Tabela 2, apresentamos um resumo das principais características de cada um dos *frameworks* aqui discutidos.

Tabela 2 - Comparação entre os *Frameworks*

	AMPHIBIAN	COSMOS	H-N2N
Domínio	Jogos multiplataforma	Sistemas virtuais colaborativos multi-usuários	Sistemas virtuais colaborativos multi-usuários
Comunicação	Sockets TCP	Sockets UDP	Sockets TCP,UDP
Serviço	Unicast	IP Multicast	Multicast a nível de aplicação
Multimídia	Através do JMF	Através do JMF	Através do JMF
Conceitos de groupware	NÃO	SIM	SIM
Inderdimensionalidade	NÃO	NÃO	SIM
Massivo	NÃO	NÃO	SIM

Observando a Tabela 2, percebe-se que os *frameworks* possuem muitas características comuns, mas diferem em alguns aspectos. O Amphibian e o Cosmos são mais específicos, possuindo um domínio menor de aplicações, enquanto o H-N2N é mais geral, uma vez que jogos e ambientes virtuais podem fazer parte de um subconjunto dos sistemas colaborativos distribuídos.

Capítulo 5 – Experimentos e Resultados

"Erros são, no final das contas, fundamentos da verdade. Se um homem não sabe o que uma coisa é, já é um avanço do conhecimento saber o que ela não é."

Carl Jung, psicólogo austríaco

Objetivando a validação da arquitetura, utilizamos nossa implementação do H-N2N em algumas aplicações, apresentadas a seguir, como estudo de caso. O primeiro estudo de caso consiste em um sistema onde robôs e pessoas interagem em um museu (NAC-UFRN), o segundo caso (GIGAVR) trata-se de um sistema onde podemos controlar um robô que transmite um *stream* de vídeo do ambiente em que ele se encontra, tudo isso através da internet. E por último, apresentamos o que chamamos de aplicações interperceptivas (Azevedo, 2006).

Ainda com o objetivo de validação da arquitetura, implementamos o H-N2N na camada de aplicação do simulador de rede J-SIM (J-Sim, 2005) e realizamos também experimentos emulando uma rede utilizando o H-N2N.

5.1 NAC-UFRN

O Conviv'art é uma galeria gerenciada pelo NAC-UFRN (Núcleo de Cultura e Arte da Universidade Federal do Rio Grande do Norte). Trabalhos de artistas como pinturas, esculturas, fotografias entre outros manifestações culturais e artísticas são frequentemente expostas nele. Como um experimento, colocamos um robô guia dentro desta galeria e fizemos uma cópia 3D dela, totalmente detalhada, incluindo os trabalhos expostos.

Neste experimento, há pessoas visitando a galeria fisicamente e também remotamente, através de nosso ambiente virtual colaborativo. Desse modo, nós criamos um ambiente de realidade mista onde o robô guia pode interagir com os

usuários realmente presentes como também com os usuários virtuais do sistema conectados através da web.

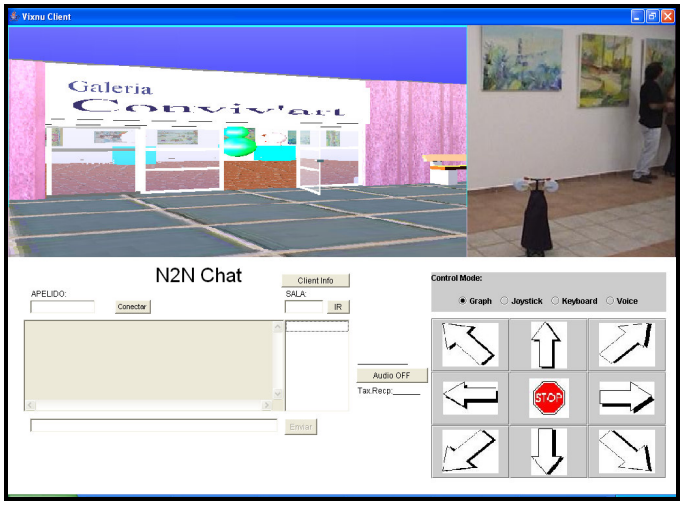

Figura 52 – Vixnu interface

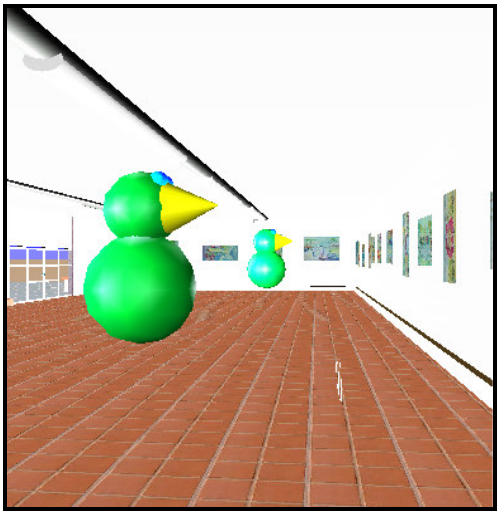

Figura 53 – Avatar escolhido por um visitante virtual

A Figura 52 mostra a versão final da interface do cliente Vixnu (reimplementado utilizando a arquitetura H-N2N) para este experimento, incluindo várias ferramentas como *chat* com mensagens de texto, botões de controle remoto, comunicação com voz, transmissão ao vivo de vídeo e uma versão virtual do ambiente. Figura 53 mostra uma caso ilustrativo onde avatares são escolhidos para os visitantes virtuais no museu. Figura 54 mostra o robô no ambiente real.

Figura 54 – Pequeno robô em um ambiente real.

Nesse experimento, nós identificamos três principais tipos de elementos participantes: Visitantes Virtuais (VV), virtualmente presentes na galeria, Visitantes Reais (VR), presentes fisicamente, e o robô (RB). Então, utilizando tal sistema, nós pode-se observar a interação entre todos os participantes (veja Tabela 3).

Interação	Meio
VV com VV	Chat, áudio e vídeo.
VV com VR	Encarnando o Robô.
VV com RB	Voz, Data Glove, Joystick, etc.
VR com RV	Voz, gestos, etc.
VR com VV	Encarnado o Robô
VR com RB	Reconhecimento de Voz
RB com VV	Voz, texto, áudio, vídeo
RB com VR	Voz sintetizada
RB com RB	Mensagens Web, síntese de voz, e reconhecimento.

Tabela 3 – Resumo dos meio de interação entre os elementos participantes

Note que VV podem se comunicarem entre eles através de troca de mensagens multimídia, e com os VR através da encarnação do robô. Os usuário podem mover e falar com se fosse o RB. Os VV podem se comunicar com o RB, através de comandos dados através de *joystick*, comandos de voz, teclados ou mesmo outros dispositivos de entrada em RV como luvas.

Os RB podem interagir com os VV do mesmo modo que com os VR, através de síntese e reconhecimento de voz, e podem usar isso também para interagir com outros RBs presentes, ou através de mensagens de texto enviadas pela web, no caso do outro RB estar em outro ambiente. Os VR podem se comunicar através dos meios comuns entre eles, com os RB usando comandos de voz e com os VV solicitando para o RB enviar mensagens de áudio, fazendo com quem que os VV escutem o VR.

5.2 GIGAVR

O objetivo dessa aplicação é usar um sistema de captura de vídeo panorâmico e estereoscópico e transmitir vários fluxos de vídeo (*streaming*) fornecendo um campo visual de 360 graus para um sítio remoto, através da rede GIGA. A idéia é colocar o sistema de captura na plataforma robótica que ira navegar em um ambiente real, na Universidade Federal do Rio de Janeiro (UFRJ). O vídeo enviado da UFRJ é processado na USP (Universidade de São Paulo) em um *cluster* de computadores que processa imagens para serem visualizadas na caverna digital lá existente. Além disso, o vídeo panorâmico deve ser exibido na caverna da USP, assim como versões reduzidas dele são transmitidas para salas de visualização localizadas em outras universidades, a UFRN e a UFPB, na região nordeste do Brasil. Interações como o robô que irá carregar as câmeras podem ser conseguidas através de luvas (*Data Gloves*), *Joysticks*, mouse e teclado, localizados em salas de visualização estéreo e na caverna digital da USP. Com isso, o robô pode ser controlado remotamente pelos usuários em algum desses ambientes.

Este tipo de aplicação permite que usuários utilizem dispositivos remotos, para interagirem e terem a sensação de imersão usando um ambiente virtual imersivo, com transmissão de vídeo real.

Na especificação do sistema GIGAVR como um todo, utilizamos diagramas ACME, identificando 7 componentes principais, conforme mostrado na Figura 55, descritos a seguir.

Vixnu: Subsistema composto do cliente e servidor multiusuário.

Robot_Plataform: Robô + Aglomerado Gráfico + RobotControler.

Robô: Utilizado para representação e movimentação.

Aglomerado Gráfico: (12 câmeras, Barebones e placas wireless), acoplados ao robô.

RobotControler: Software de controle do robô.

Cluster: Subsistema responsável pela rasterização dos fluxos de vídeos.

Video_Server I: Subsistema responsável pela transmissão do vídeo bruto(vindo das 12 câmeras) para o cluster.

Video_Server II: Subsistema responsável pela transmissão do vídeo rasterizado do cluster para os clientes do vixnu.

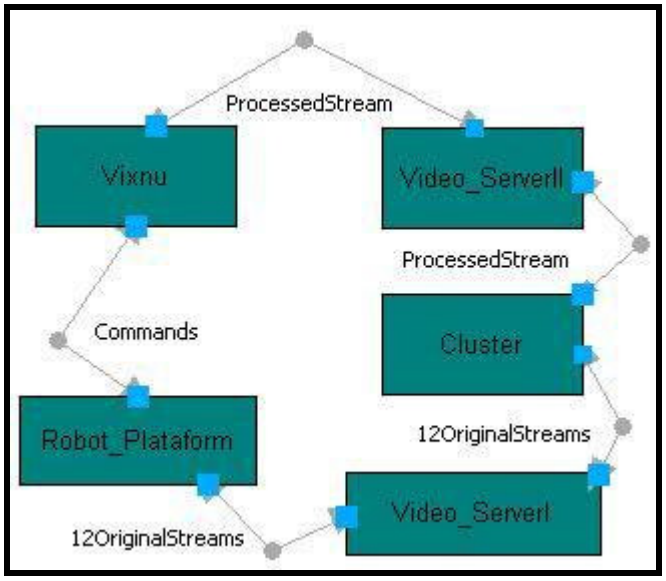

Figura 55 – Modelagem do Sistema GIGAVR

Interligando estes componentes temos cinco conectores:

Robot_Plataform-12OriginalStream-Video_ServerI: O Video-ServerI recebe o fluxo bruto(gerado pelas 12 câmeras) e o transmite**.**

Vídeo_ServerI-12OriginalStream-Cluster: O Cluster recebe esse fluxo original(bruto) e realiza o seu processamento, transformando os 12 fluxo num só panorâmico.

Cluster-ProcessedStream-Video_ServerII: O Vídeo_serverII recebe o fluxo processado desse cluster e o transmite para o clientes vixnu.

Video_ServerII-ProcessedStream-Vixnu: O cliente vixnu recebe o fluxo do Vídeo_ServerII

Vixnu-Commands-Robot_Controler: O vixnu envia comandos para o Robot_Plataform

5.2.1 Subsistema Multiusuário

Uma vez definido o sistema inteiro, resta especificar o subsistema multiusuário. Os componente presentes em sua especificação, mostrados na Figura 56, são:

Vixnu_Server: Parte servidora, recebe os pedidos de conexões dos clientes, provê os serviços de comunicação, percepção e controle do agente robótico.

Vixnu_UserClient: Cliente usuário, realiza funções como movimentar-se pelo ambiente, controle do robô, comunicação entre os outros usuários, etc.

Vixnu_RobotClient: Cliente robô, esse componente realiza a interface entre o sistema vixnu e o Robot_Controler. Utilizado para encaminhas os comandos vindo dos usuários ao robô.

Vídeo_ServerII: Subsistema responsável pela transmissão do vídeo rasterizado(vídeo 360 do cluster para os clientes do vixnu.

Robot_Controler: Subsistema controlador do robô.

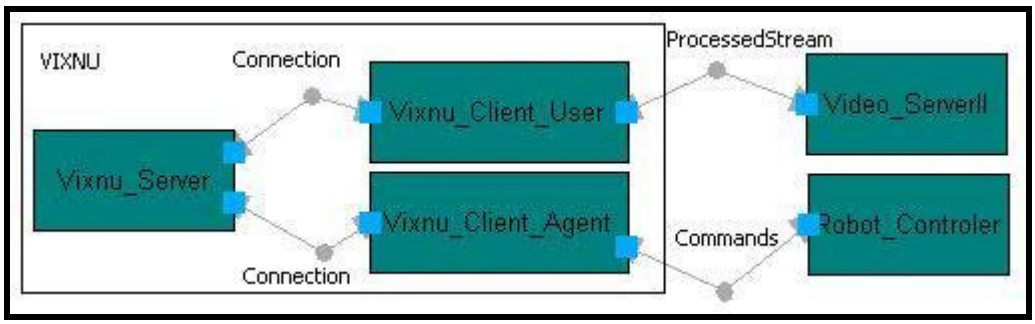

Figura 56 – Modelagem em ACME do subsistema Vixnu

Interligando estes componentes temos os seguintes conectores:

Connection: Representa conexões entre os clientes e os servidores, pelas quais as mensagens de controle e comunicação trafegam.

ProcessedStream: Fluxo de vídeo processado no Cluster e enviado pelo servidor de vídeo para os clientes remotos.

Commands: Comandos dados pelos usuários ao robô. Mensagens contendo ordens para controlar o robô remotamente.

5.3 Aplicações Interperceptivas

Ainda fazendo uso do *framework* H-N2N, foram concebidas aplicações onde usuários com diferentes necessidades ou recursos podem participar do AVC, seja colaborando, comunicando, percebendo ou interagindo, tudo isso de um maneira transparente. Sendo assim, usuário com hardwares diferentes, conectados por redes diferentes podem compartilhar o mesmo ambiente virtual. Isso é possível graças a arquitetura interperceptiva (Azevedo, 2006), mostrada na Figura 57. Esta arquitetura provê diferentes interfaces, que são customizadas de acordo com as necessidades e recursos disponíveis de cada usuário.

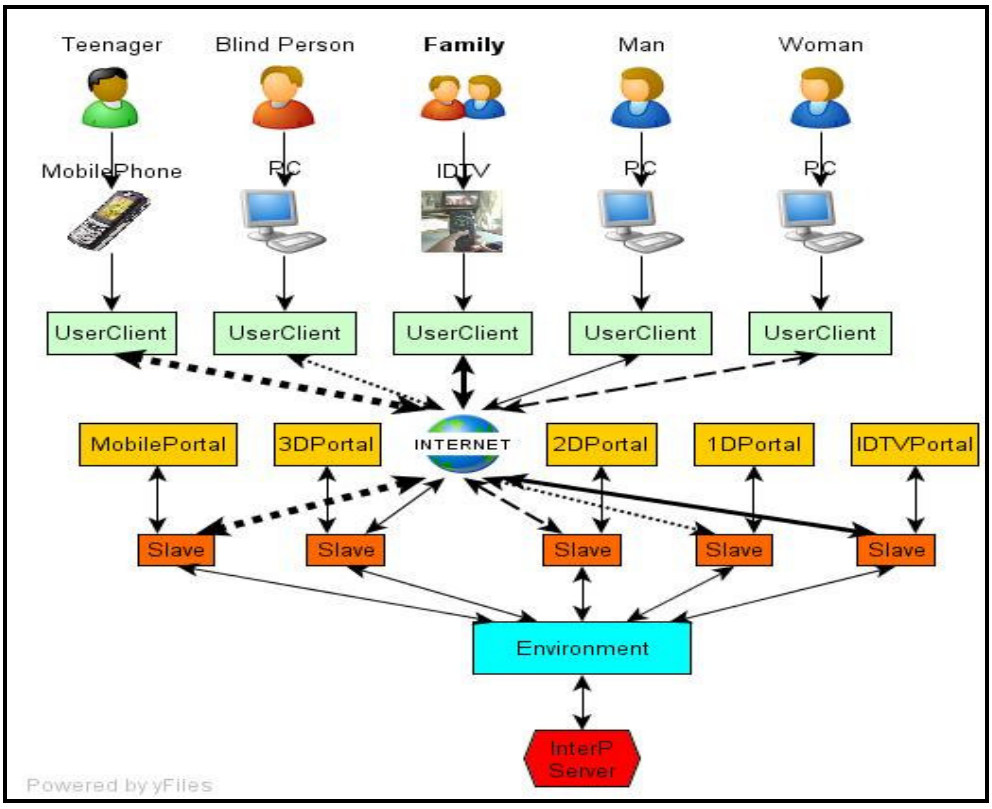

Figura 57 – Arquitetura Interperceptiva

A Figura 57 fornece uma idéia melhor do modelo de interpercepção. Note que no topo da figura, temos usuários diferentes: adolescente (*teenager),* pessoa com necessidades visuais *(blind person),* família *(family),* homem *(man)* e mulher *(woman*). Esses usuários estão utilizando dispositivos diferentes, como celular, PCs com diferentes configurações de hardware e até mesmo uma Setopbox (dispositivo para TV Digital Interativa), e também conectados por redes diferentes, sejam elas conexões discadas, banda larga, etc. Porém graças à essa arquitetura, que conta com um componente chamado Portal, todos podem interagir de maneira transparente, como se todos fossem iguais.

Compreendemos que aplicações que utilizem essa arquitetura formem um novo domínio de aplicações, que chamamos **aplicações interperceptivas**. Algumas dessas aplicações desenvolvidas utilizando o H-N2N, como o *Festival Garagem Virtual* e o *FutebolDeBotao*.

Figura 58 – Telas da aplicação Garagem Virtual. Interfaces(3D,2D e 1D)

5.3.1 Festival Garagem Virtual

O festival Garagem Virtual consiste de um ambiente virtual colaborativo onde usuários podem expor suas músicas em um ambiente virtual, mediante cadastro, e os visitantes podem votar e escolher as melhores músicas do ambiente. Funciona

como um campeonato entre bandas amadoras, onde são os visitantes que devem eleger o vencedor. O diferencial desse sistema está no fato do usuário poder escolher, dentre três tipos de interfaces diferentes, a que mais se encaixa com seu perfil. Como podemos ver na Figura 58, temos a interface tridimensional para usuários munidos de computadores com melhor poder de processamento gráfico, a interface 2D para quem possui menos recurso computacional e a interface textual, para quem possui pouquíssimo recurso.

5.3.2 Futebol Botão Virtual

Uma outra aplicação idealizada, tendo como base a arquitetura interperceptiva do H-N2N, foi o Futebol de Botão Virtual. Essa aplicação consiste de um jogo multiusuário, onde os jogadores podem assumir o controle de "botões", que representam os jogadores em um campo de futebol. Inicialmente implementado em um versão em três dimensões, hoje contamos com a versão 2D criada para que o jogo possa ser utilizado também por jogadores com máquinas com baixo poder de processamento, como também permitir a utilização de usuário em um programa de TV Digital Interativa (já que ate o momento o padrão brasileiro de TVDI ainda não definiu nenhuma API 3D em sua especificação). A Figura 59 dá uma melhor idéia desta aplicação. Nela, podemos ver a duas interfaces possíveis nesse ambiente, o que o torna um jogo interperceptivo.

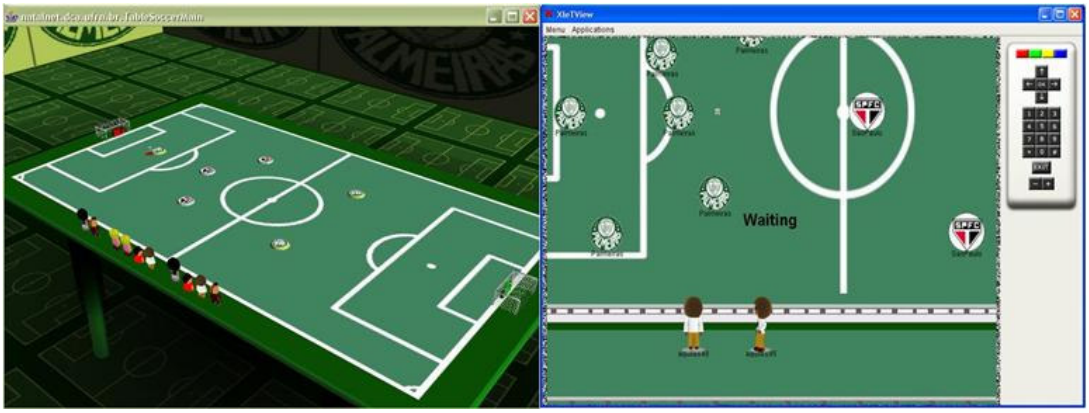

Figura 59 – Telas Futebol Botão Virtual, Interface 3D e 2D.

Assim como as demais aplicações aqui apresentadas, o H-N2N foi utilizado para prover toda troca de mensagens e organização em grupos requeridas pelas

aplicações. Acelerando o desenvolvimento, uma vez que o desenvolvedor somente precisar definir as mensagens a serem trocadas e o processamento associado a elas, assim como a interface gráfica das aplicações.

5.4 Simulação do H-N2N

Na simulação utilizamos o J-SIM, escolhido por :

- ter bom desempenho, quando comparado com os demais do mercado (J-Sim, 2005);
- utilizar a linguagem Java, promovendo uma maior velocidade de desenvolvimento ao possibilitar a reutilização do código na construção de futuras aplicações e por ser multi-plataforma.

Simulamos um AVC considerado massivo, com 100 participantes, distribuídos na topologia mostrada na Figura 60. Note que temos 6 roteadores (0,1,2,12,13,14) e os demais são estações normais.

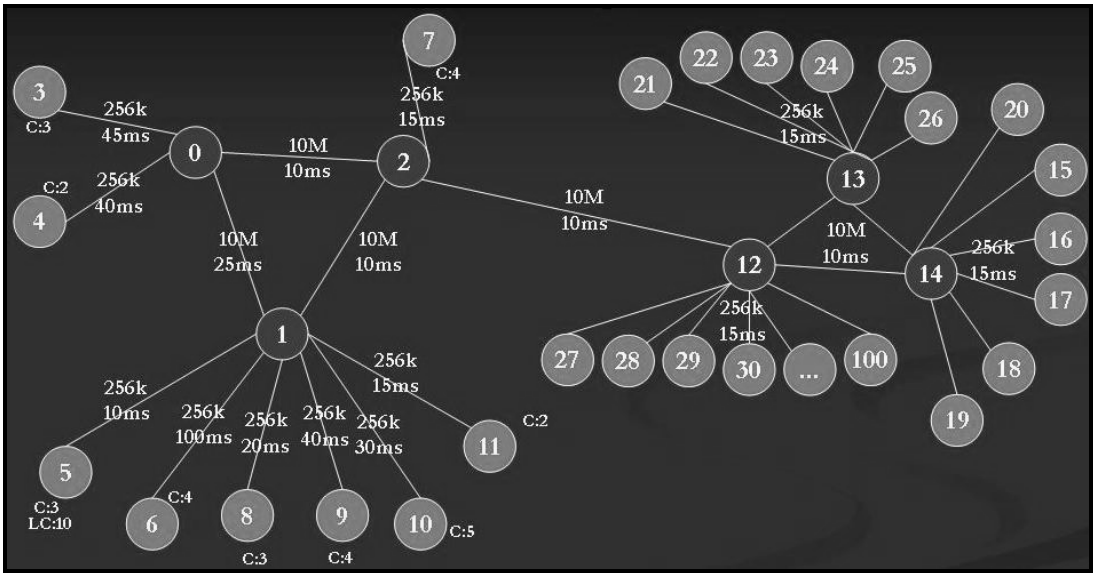

Figura 60 – Topologia utilizada na Simulação

5.4.1 Resultados da simulação

O servidor H-N2N foi adicionado no nó 7 e os clientes nos restantes. Colocamos clientes entrando no sistema tanto sequencialmente quanto simultaneamente. A Figura 61 mostra o gráfico contendo o desempenho (*throughput*) no nó 7 ao longo

do tempo. Podemos ver que no decorrer da simulação ele permanece constante, mesmo com a entrada de vários usuários. Isto mostra a eficiência da arquitetura quanto à escalabilidade. O pico no inicio do gráfico é decorrente das várias entradas simultâneas no sistema logo no inicio da simulação. Esse comportamento se repete para os demais nós servidores na simulação.

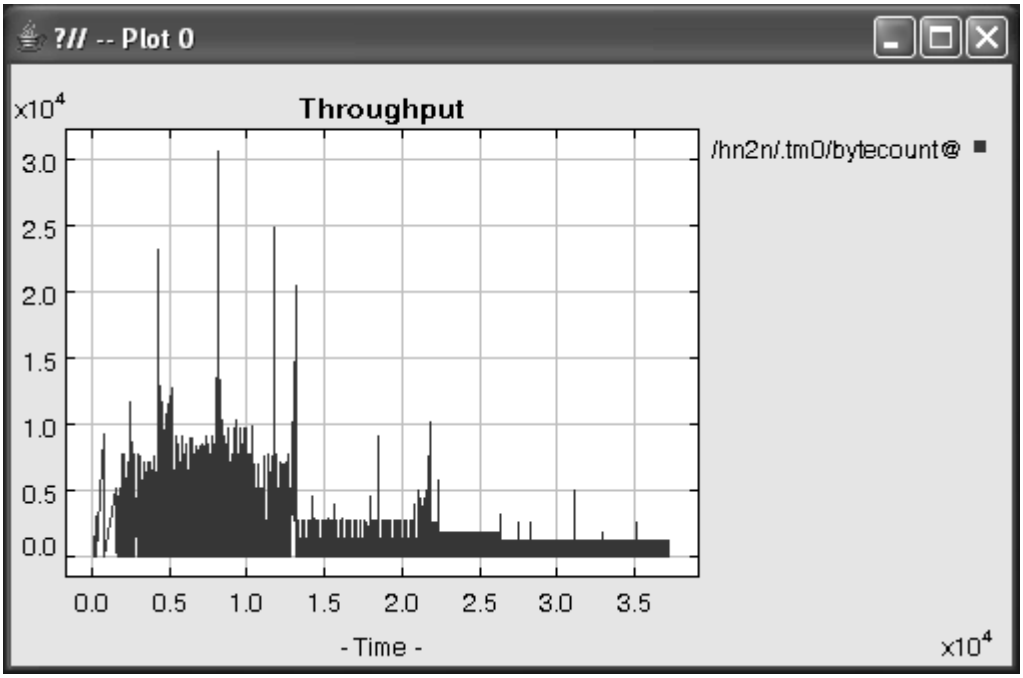

Figura 61 –*Throughput* no nó 7 (root H-N2N)

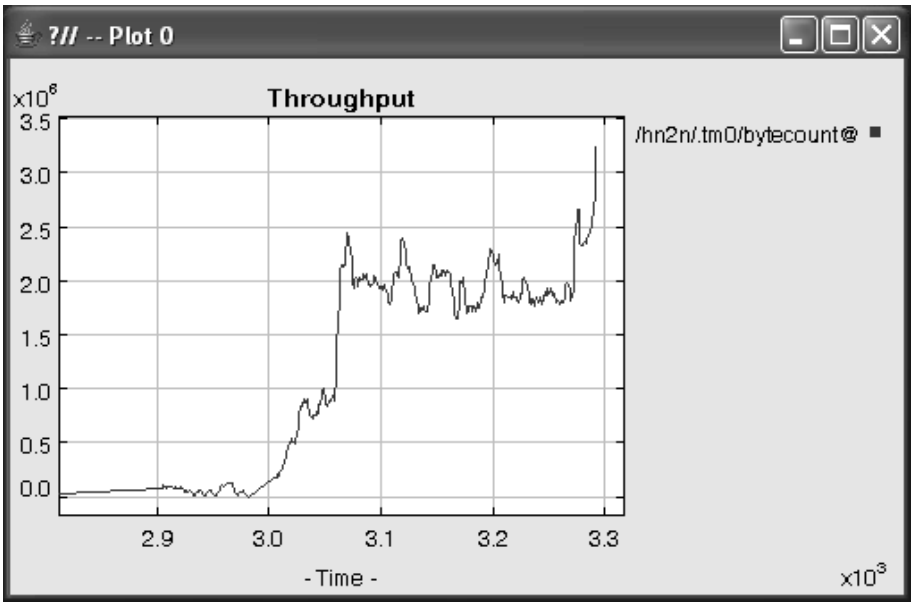

Figura 62 –*Throughput* no Servidor Normal

Simulamos também uma arquitetura cliente-servidor *ad-hoc*, onde todos os clientes se conectam a um servidor único. O gráfico da Figura 62 mostra aumento do fluxo de mensagens a medida que os clientes chegam. Podemos notar que ao utilizarmos o H-N2N o gráfico se estabiliza não importando quantos clientes se conectem, o que não acontece com a arquitetura *ad-hoc* acima.

Os resultados acima, embora que pareçam previsíveis, mostram que, ao dividirmos os clientes em vários servidores, o *throughput* é dividido e cada servidor somente lidará com o que ele é capaz de lidar. Percebemos que um problema surge no que se refere a entrada de múltiplos clientes simultaneamente. Para resolver isso, podemos utilizar servidores dedicados somente para a entrada no sistema, como também utilizar DNS dinâmicos (RFC2136, 1997) para mudar o endereço do servidor disponível para a conexão. Resolver tal problema é importante, principalmente para aplicações onde a entrada de usuários tende a ocorrer ao mesmo tempo, como por exemplo em aplicações para TV Digital. Embora tenhamos proposto algumas soluções para esse problema, ele não é o foco de nosso trabalho e não entraremos em detalhes de como implementar servidores dedicados à conexão ou aplicação de DNS Dinâmicos ou outras técnicas.

Após colhermos os resultados supracitados no simulador J-SIM, decidimos continuar os testes em um ambiente mais real e não mais em um simulador. Apresentamos a seguir os resultados com uma emulação do H-N2N. Com a emulação, poderíamos utilizar os mesmos componentes de software que seriam utilizados nas aplicações em ambientes reais em testes envolvendo máquinas reais, servidores, *firewalls*, NATs, DNS, diferentes redes, etc. Os códigos utilizados nessa simulação podem ser encontrados na página do h-n2n (http://hn2n.wikidot.com/).

5.5 Emulação do H-N2N

Visando uma melhor validação do *framework*, foram realizados testes emulando um ambiente real. Os testes foram executados em uma LAN (rede local), utilizando para isto 6 computadores na geração de clientes e servidores H-N2N.

Esta emulação envolveu a implementação de vários outros componentes que viabilizassem este experimento.

5.5.1 Configuração da Emulação

Os testes foram realizados utilizando as máquinas cujas características principais estão descritas na Tabela 4.

Máquina	IP	Processador	Memória RAM
Laptop1	10.13.100.11	AMD 1,8 GHz	512 RAM
PC1	Thing: 10.13.99.208	DELL, Pentium 4 HT3.0GHz	1GB RAM
PC2	Batman:10.13.99.197	Pentium 4 3.0GHz	1GB RAM
PC3	DareDevil:10.13.99.195	Celerom 2.53 GHz	256 MB RAM
PC4	lablm20:10.13.99.212	Pentium 4 3.0GHz	1GB RAM
PC5	vampira:10.13.99.221	Pentium 4 3.0GHz	1GB RAM

Tabela 4 - Especificações das Máquinas

Visando emular o trafego gerado por aplicações executando sob o H-N2N, implementamos três tipos de tráfegos, cujos detalhes são mostrados na Tabela 5.

Nome	Carga
Trafego1	0Kbps. (clientes não geram trafego constante, somente mensagens de controle e algumas mensagens de aplicação, quando requisitadas)
Trafego2	~5Kbps
Trafego3	~2Kbps

Tabela 5 - Trafego gerado por cada cliente

Alem das máquinas e do tráfego gerado pela aplicação, temos que considerar também o trafego interno do *framework*, aquele gerado por suas mensagens de controle. Sendo assim, são mostradas na Tabela 6 as mensagens de controle utilizada pelo H-N2N.

Nome	Protocolo	Tamanho
MCLIENTREADY	TCP	306 bytes
MIDENTIFICATION	TCP	362 bytes
MCREATEGROUP	TCP	356 bytes
MRESOURCE	TCP	370 bytes
MCREATESERVER	TCP	349 bytes
MSERVERCREATED	TCP	350 bytes
MTRYSON	TCP	372 bytes
MIDENTIFICATION	TCP	362 bytes
MCREATEGROUP	TCP	356 bytes
MRESOURCE	TCP	370 bytes

Tabela 6 – Mensagens H-N2N de controle, trocadas nos experimentos

Para essa emulação foi implementada uma aplicação simples que realiza um comando *ping* entre os clientes. Esta aplicação foi executada para medirmos o retardo, assim como para realizarmos testes de conectividade ente as partes. Foram criadas mensagens de aplicação, tanto TCP quanto UDP para podermos avaliar o desempenho desses dois protocolos em nosso sistema. Na Tabela 7 temos detalhes dessas mensagens.

Nome	Protocolo	Tamanho
MPING	TCP	296 bytes
MPONG	TCP	345 bytes
MPING	UDP	15 bytes
MPONG	UDP	35 bytes

Tabela 7 - Mensagens da aplicação TestPing, trocadas nos experimentos

5.5.2 Gerenciamento dos Experimentos

Para realizarmos os experimentos nessa fase de emulação, foram criados alguns componentes de software para facilitar o gerenciamento do experimento entre as várias máquinas envolvidas. Para isso foram criados mais quatro pacotes de software associados ao H-N2N:

- **br.akls.hn2n.tester** : Responsável pela implementação de clientes e servidores de uma aplicação criada para testar o tráfego nos nós da rede e para testar o retardo na troca de mensagens. O cliente funciona como um robô (*bot*), simulando um cliente do sistema.
- **br.akls.hn2n.tester.generator** : responsável pela criação de vários clientes, BotGenerator.
- **br.akls.hn2n.tester.manager** : Sua principal classe é a BotManager, responsável por gerenciar os geradores de bots(BotGenerators).
- **br.akls.hn2n.tester.net:** Definição das mensagens utilizadas durante os testes.
- **br.akls.hn2n.tester.statistics**: Este pacote é responsável pelo provimento de componentes para o monitoramento da rede, no que diz respeito ao chegada e saída de pacotes de uma entidade(cliente ou servidora).

Cada máquina participante da emulação executa um gerador de clientes (*generator*), que por sua vez se conecta ao gerente (*manager*) de testes. Desse modo, podemos controlar todo o experimento a partir de uma única máquina.

O *BotManager* contém uma lista dos *BotsGenerators* disponíveis. Após selecionar um dos *generators,* é exibido a lista dos clientes (*bots*) criados por aquele *generator*. Ao selecionar um cliente é possível enviar comandos para que ele possa executá-los. Como por exemplo, pedir para que um cliente envie uma mensagem de *ping* para os outros clientes. A interface do *BotManager* pode ser vista na Figura 63.

5.5.3 Experimento I - Conectando ao Servidor

O objetivo deste experimento é observar o tráfego gerado no servidor por vários clientes conectando ao mesmo tempo. Como tráfego foi usado o **Tráfego1.** As máquinas participantes deste experimento são:

- Laptop1: executando H-N2N Server

- PC1: Executando 50 clientes tentando se conectar.

- PC2: Executando 50 clientes tentando se conectar.

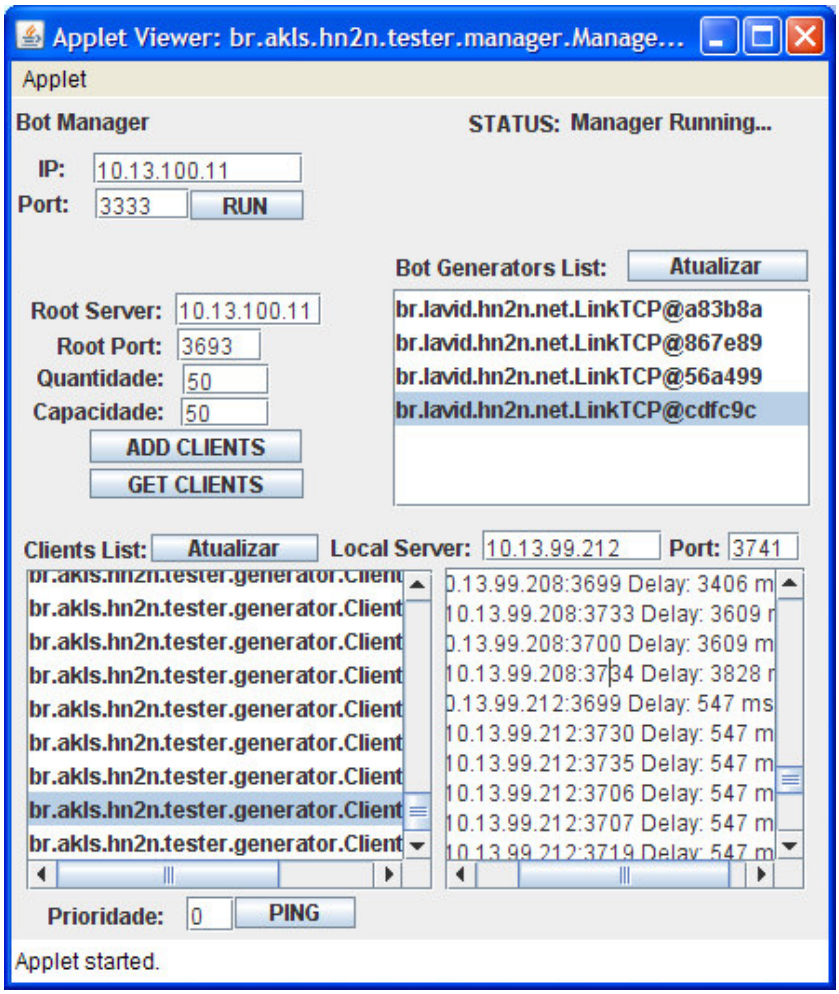

Figura 63 – Interface do *BotManager*.

Neste experimento, foram colocados 50 clientes conectando ao servidor principal. Metade destes (25 clientes) estavam no PC1, a outra metade no PC2 e eles tentaram a conexão ao mesmo tempo. Com isso foi gerado no servidor um trafego de mensagens de controle com uma média de 11/12 Kbps como fluxo de entrada/saída, respectivamente, e máximo de 19/22 Kbps, conforme ilustrado na Figura 64. As linhas no gráfico representam o tráfego de entrada e de saída. Para cada cliente entrando o servidor recebeu uma mensagem de controle. Depois desse momento, o servidor atingiu o seu máximo. Em seguida, recebeu mais outros 50 clientes (25 de cada PC), no servidor. O mesmo procedimento é realizado gerando o trafego mostrado na Figura 65. Para decidir quem é o cliente mais apto, uma mensagem é enviada para seus “filhos” que por sua vez respondem ao chamado. O servidor elege o filho apto, envia uma mensagem

informando que ele foi escolhido, e o filho escolhido, por sua vez, informa ao pai quando ele estiver pronto, ou seja, quando terminar de se tornar servidor, para poder receber novos pedidos de conexão.

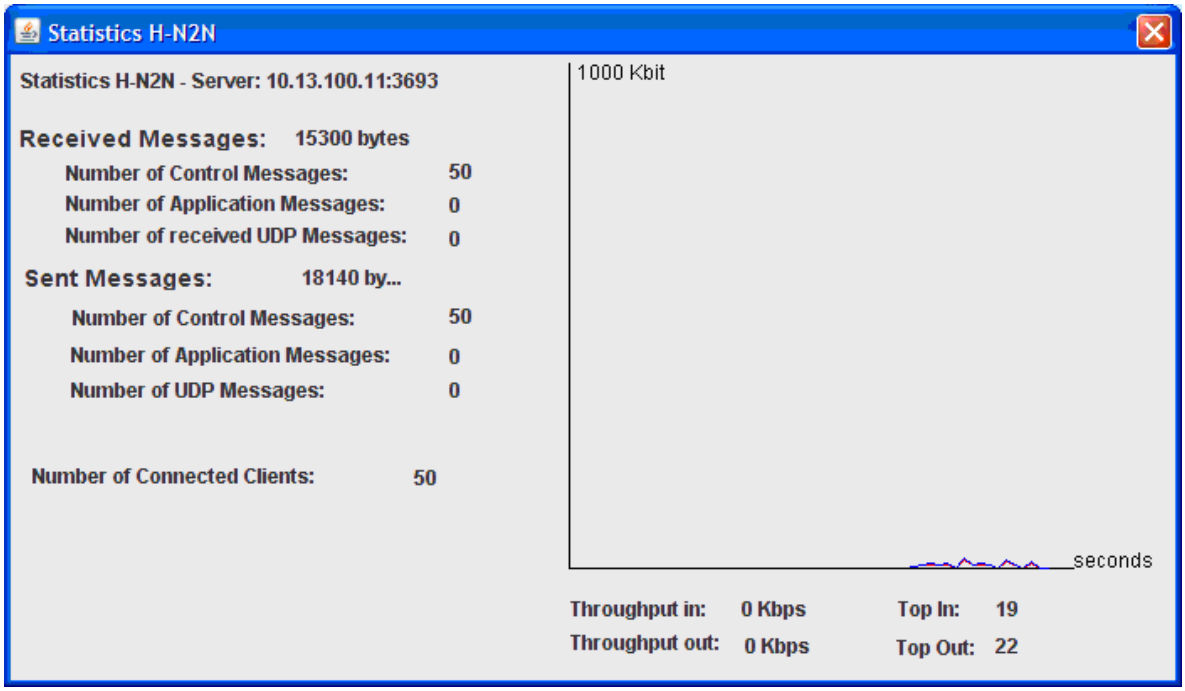

Figura 64 – Tentativa de conexão simultânea, 50 clientes.

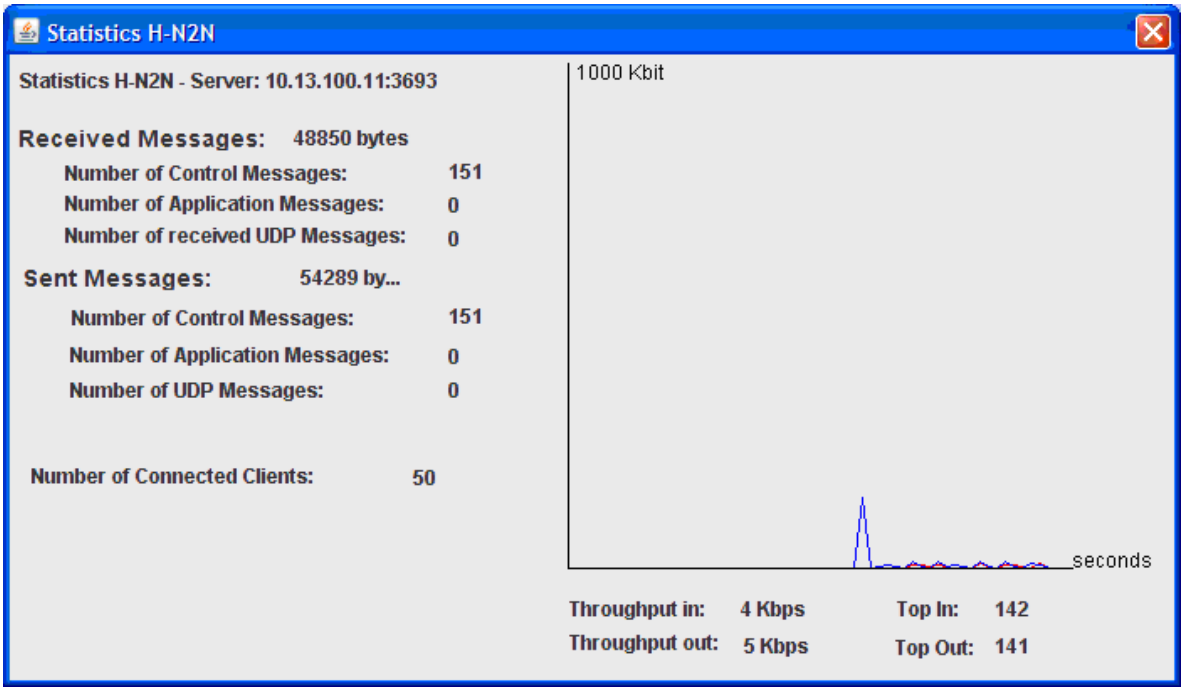

Figura 65 – Máximo atingido, mais 100 clientes se conectam.

Podemos notar um pico no gráfico da Figura 65, devido à necessidade de tentar descobrir qual dos clientes conectados é o mais apto a se tornar servidor. O restante do trafego gerado posteriormente é causado pelo recebimento de pedidos de conexão e envio do endereço do servidor do nível abaixo para os novos clientes que estão chegando. Note que o número de clientes conectados (*Number of Connected Clients*) continua 50 embora tenhamos 100 clientes conectados ao sistema. Os 50 últimos a entrarem foram conectados a outro servidor de um nível abaixo.

Realizamos mais um teste, cujos resultados são mostrados na Figura 66, para ver como o servidor se comportaria caso um número maior de clientes desejasse se conectar. No caso, 200 clientes foram inseridos.

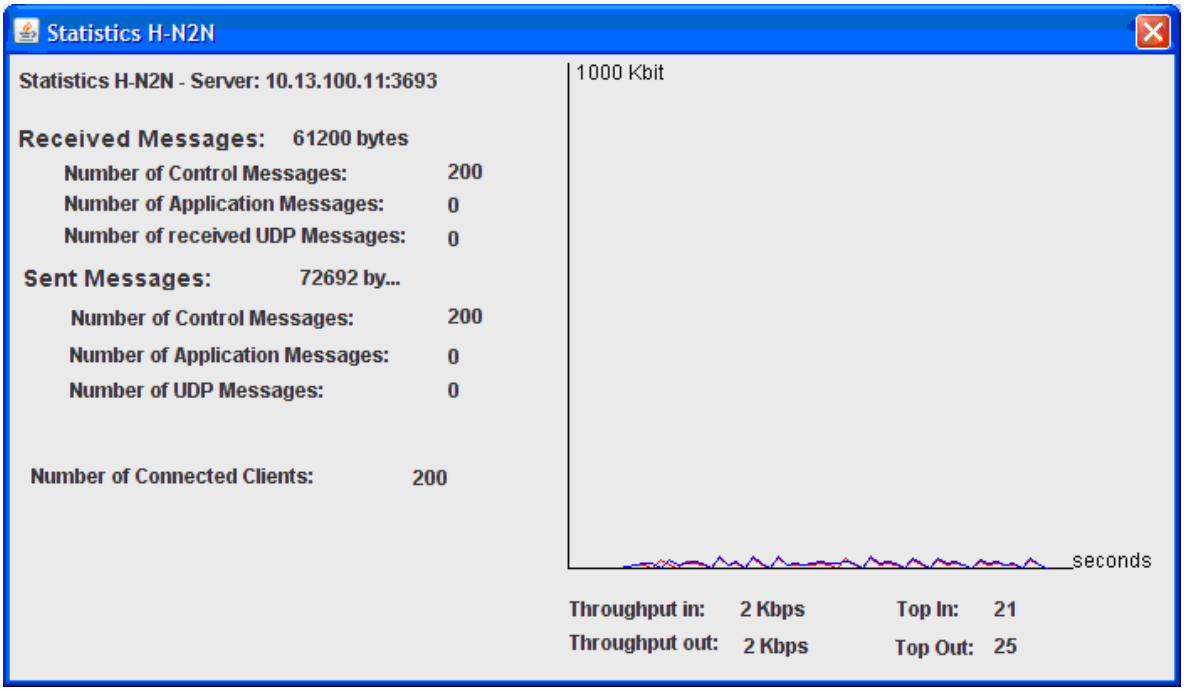

Figura 66 – Trafego gerado por 200 clientes se conectando simultaneamente

Observamos que a taxa de recebimento e envio não sofre grandes alterações. Com 50 clientes temos 19/22 Kbps, trafego entrado e saindo respectivamente. E com 200 temos 21/25 Kbps. Esse tráfego é referente somente a troca de mensagens de controle necessárias ao estabelecimento das conexões.

5.5.4 Experimento II – Divisão de carga entre níveis

O objetivo é observar o tráfego gerado nos servidores entre diferentes níveis da hierarquia H-N2N. Nesse experimento, ao contrario do passado, os clientes geram um tráfego de aproximadamente 2 Kbps, simulando o envio de mensagens de uma aplicação. O tráfego é de aproximadamente **2Kbps** por cliente. Participaram deste experimento as seguintes máquinas:

- **Laptop1**: executando H-N2N Server
- **PC1**: Executando 10 clientes tentando se conectar.
- **PC2**: Executando 10 clientes tentando se conectar.
- **PC3**: Executando 10 clientes tentando se conectar.

O experimento conta com 30 clientes. Uma máquina é o servidor principal e as outras três contendo 10 clientes cada. Para uma melhor visualização da configuração do experimento, veja a Figura 67.

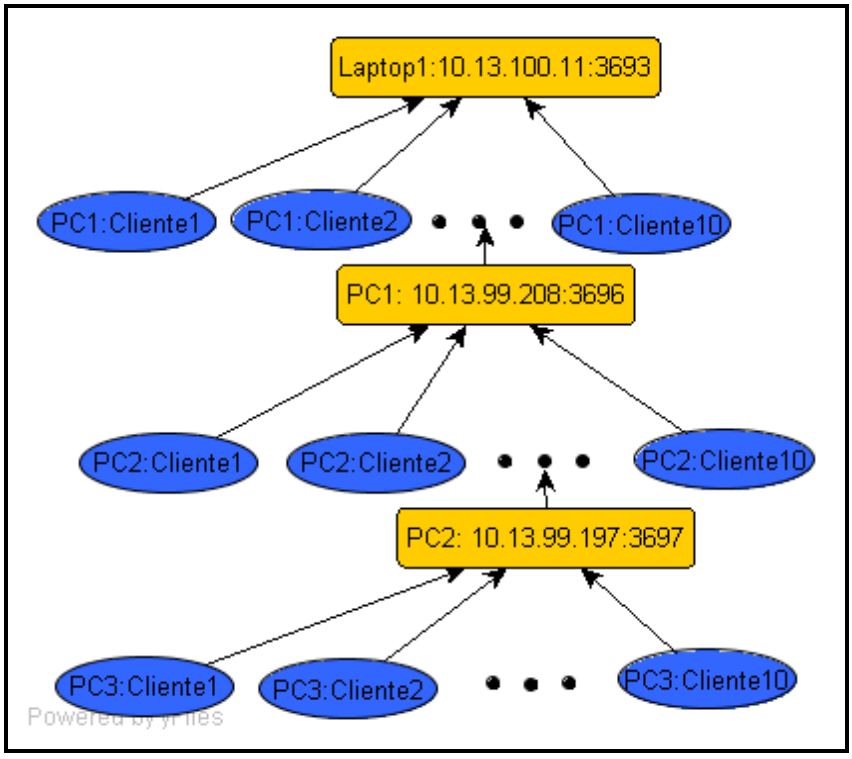

Figura 67 – Arvore H-N2N para o experimento II

A Figura 68 mostra que o tráfego continua o mesmo depois das conexões. O sistema divide os 30 clientes entre 3 servidores. O tráfego no servidor principal, executado no **Laptop1** também pode ser visto na Figura 68.

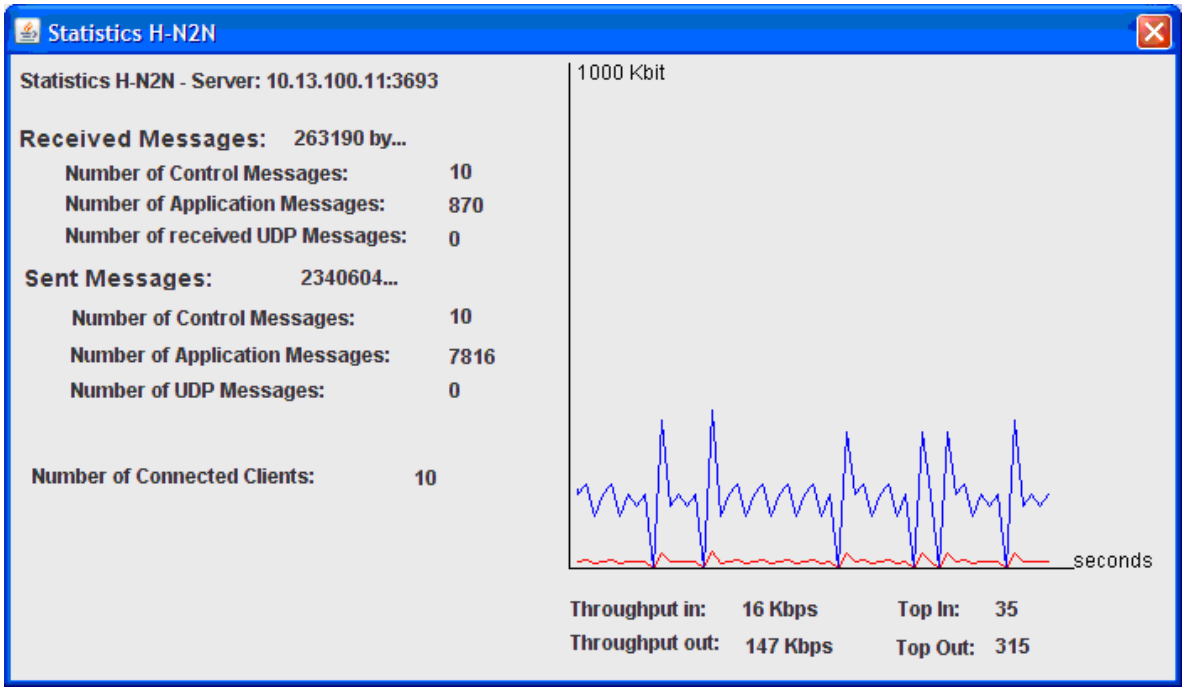

Figura 68 – Trafego do servidor em Laptop1, 10 clientes conectados.

A seguir, mais 10 clientes entram no sistema, vindos de PC2. Um cliente que estava executando em PC1 se transforma em servidor. O tráfego gerado pode ser visto na Figura 69.

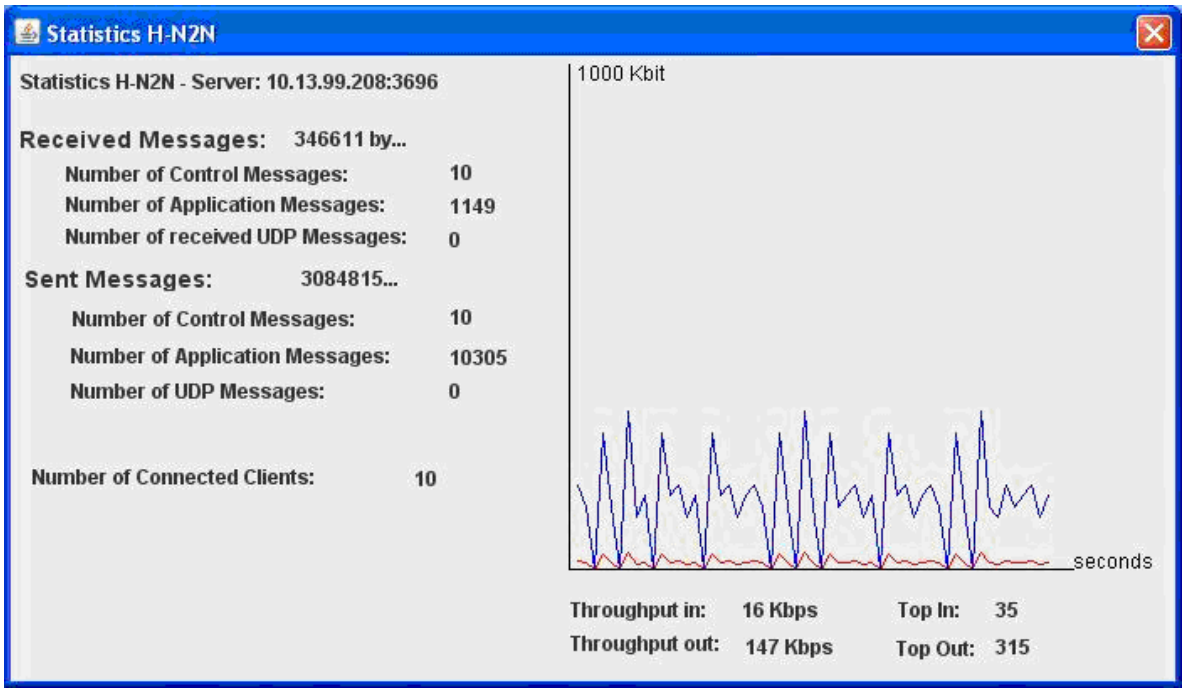

Figura 69 - Tráfego do servidor em PC1, 10 clientes conectados.

Mais uma vez, são adicionados mais 10 clientes, totalizando 30 clientes no sistema. Dessa vez o PC3 é o responsável pela criação dos 10 novos clientes e um cliente do PC2 é o escolhido para tornar-se servidor. Analogamente o gráfico do trafego gerando nele pode ser visto na Figura 70.

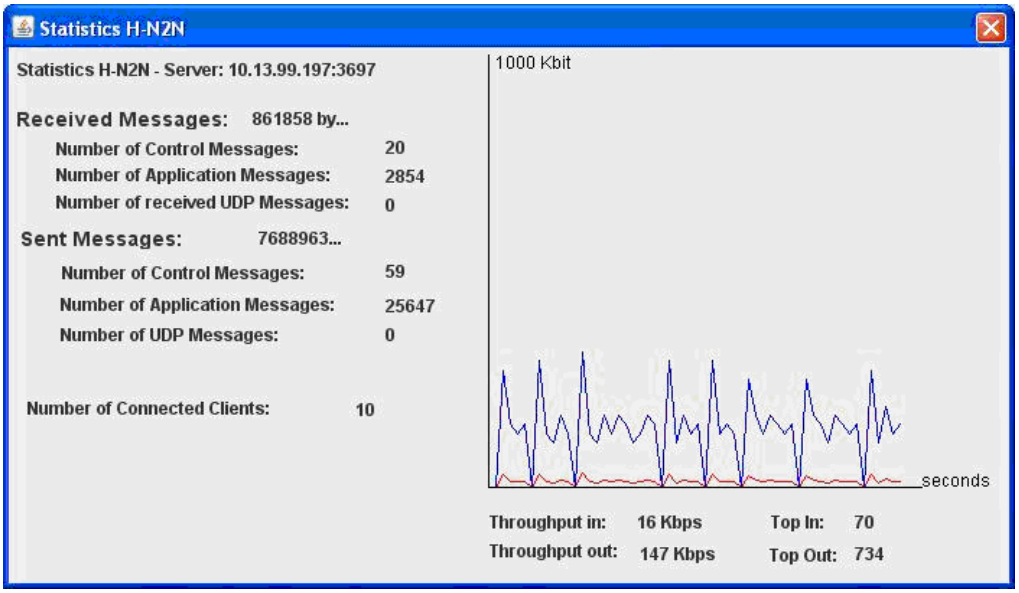

Figura 70 - Tráfego do servidor em PC2, 10 clientes conectados

Caso esses 30 clientes fossem conectar-se a um único servidor, o trafego gerado seria o mostrado na Figura 71, retirado de um outro experimento onde executamos 30 clientes em PC3 e todos eles se conectaram ao servidor que estava em Laptop1.

Através da comparação entre os três gráficos (Figura 68, Figura 69 e Figura 70) com o da Figura 71, podemos notar que com a divisão do sistema a quantidade de mensagens a serem tratadas por cada servidor é reduzida. No próximo exemplo testaremos o envio de mensagens entre clientes de servidores distintos.

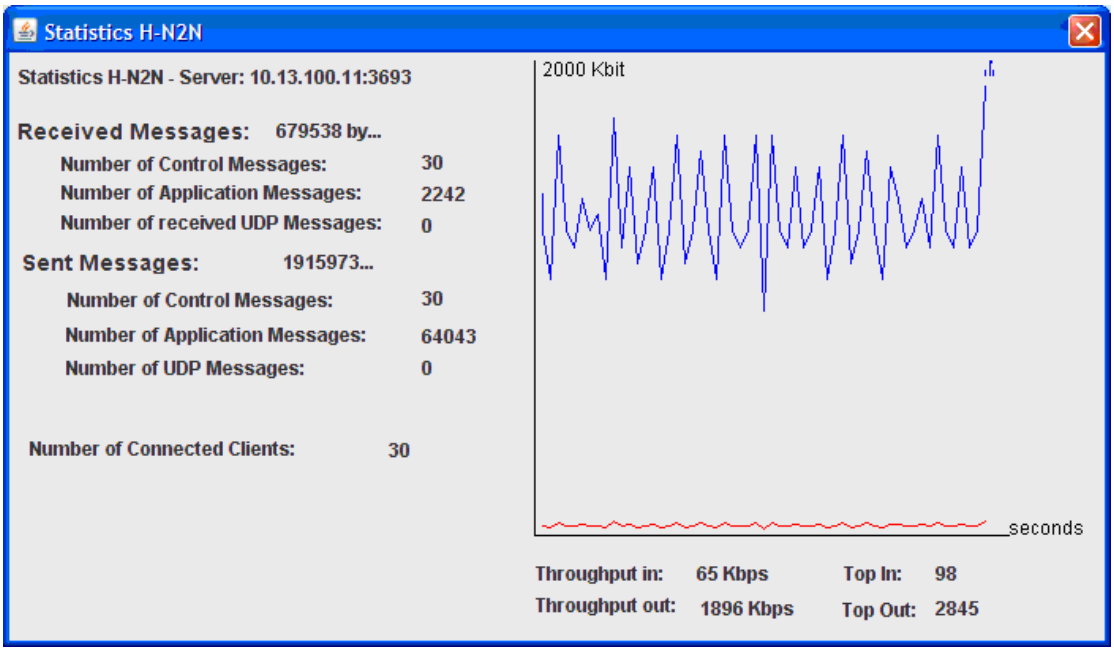

Figura 71 - Tráfego do servidor em Laptop1, 30 clientes conectados

5.5.5 Experimento III – Delay entre níveis

O objetivo é calcular o atraso gerado na troca de mensagens entre os clientes de mesmo nível e níveis diferentes dentro do H-N2N. Testando tanto mensagens UDP quanto TCP. Na primeira configuração, usando **Tráfego1**, os participantes são:

- **Laptop1**: Executando H-N2N Server.
- **PC1**: Executando 2 clientes tentando se conectar.
- **PC2**: Executando 2 clientes tentando se conectar.

- **PC3**: Executando 2 clientes tentando se conectar.
- **PC4**: Executando 2 clientes tentando se conectar.

Além do objetivo de descobrir o *atraso* entre nós que estão em níveis diferentes, objetivamos também testar se o modelo de propagação de eventos está funcionando corretamente. Para isso desenvolvemos uma simples aplicação que envia mensagens (***PING***) pedindo para que quando as outras máquinas recebam tal mensagem elas respondam com uma outra mensagem (***PONG***). Assim que a resposta chega, a máquina que fez o pedido calcula o atraso que a mensagem demorou para ir ao outro cliente, ser processada por ele, ser re-enviada e processada ao chegar de volta. As mensagens criadas foram a ***MPING*** e a ***MPONG***, descritas no inicio desta seção. A configuração do experimento pode ser vista na Figura 72.

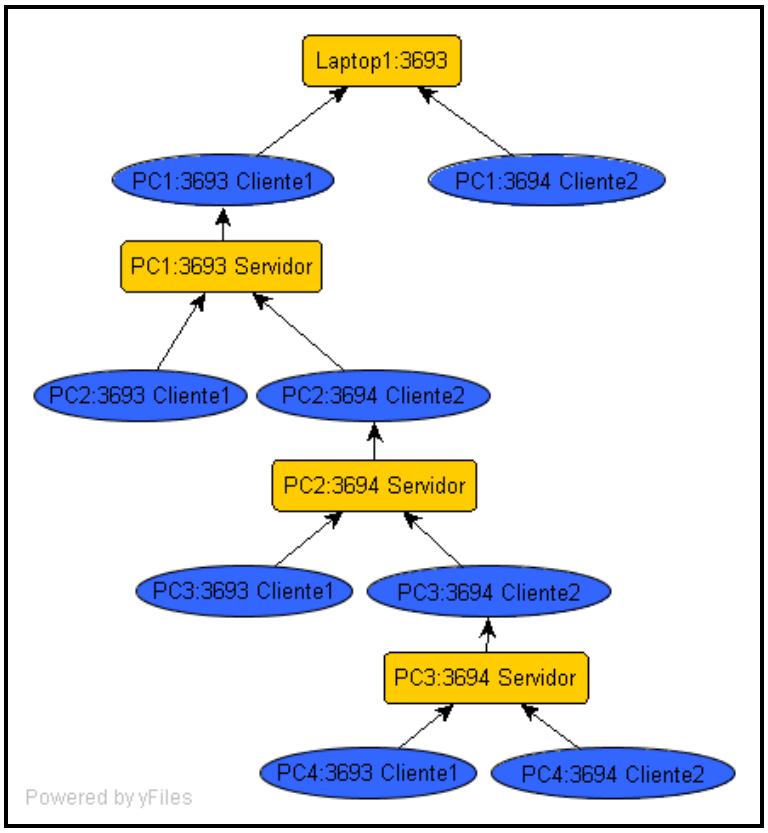

Figura 72 – Configuração da arvore H-N2N para o experimento III

As mensagens foram encapsuladas em objetos *Package* do pacote *net* do H-N2N, e enviadas por um *LinkTCP*. Os testes foram realizados em cada um dos

clientes, onde eles enviaram dois tipos de *ping*. O primeiro, *PING* 0, era recebido somente pelos clientes que estavam no seu mesmo grupo. Já o *PING* 3, era um mensagem com prioridade alta e por isso era recebido por todos os clientes da árvore. Os arquivos de *log* gerados durante este experimento podem ser encontrados no Apêndice A, procurando por Experimento III, Configuração 1.

Usamos os dois tipos de protocolos, TCP e UDP, cujos resultados são comentados a seguir.

Mensagens TCP: O experimento foi realizado enviado-se duas mensagens TCPs diferentes, uma do tipo PING 0 e outra do tipo PING 3. A PING 0 é enviada somente parar usuários do mesmo nível, e a PING 3 é feita enviada para todos os usuários. As mensagens PING 0 foram enviadas oito vezes (de cada um dos participantes). E a mensagem PING 3 foi enviada 16 vezes (duas vezes por participante). Como podemos ver nos arquivos de *log* gerados, para o PING 0 a média de atraso entre a mensagem chegar no destino e retornar foi de 600 ms, e para o PING 3 o atraso médio foi 670 para o mesmo nível, 800 para o segundo nível e 1000 para o terceiro nível. Ou seja, em média para cada nível a mais que é gerado no H-N2N é adicionado um atraso de 200 ms.

Mensagens UDP: Para as mensagens PING 0 e PING 3 sob UDP o atraso gerado para o mesmo nível foi 0 ms, e para cada nível extra, em média, foi adicionado um atraso de 5 ms.

Na segunda configuração, usando **Tráfego1**, foi realizado um experimento similar ao descrito na configuração 1, só que com 200 clientes, sendo 50 em cada PC. Os participantes são:

- Laptop1: executando servidor H-N2N.

- PC1: Executando 50 clientes.

- PC2: Executando 50 clientes.

- PC3: Executando 50 clientes.

- PC4: Executando 50 clientes.

Os logs gerados durante o experimento podem ser encontrados no Apêndice A, procurando por Experimento III, configuração 3.

Como temos 50 clientes por servidor, o atraso aumentou consideravelmente se comparado a configuração I (que possui somente 10 clientes por servidor), chegando a 859 ms para mensagens TCP no mesmo nível. E para uma mensagem chegar e voltar no terceiro nível o *delay* gerado foi 3828 ms. Com mensagens UDP no mesmo nível o *delay* máximo foi 485 ms, e para o terceiro nível foi de 2625ms.

Baseado nesses resultados, percebemos que os servidores H-N2N devem ser configurados levando-se em conta qual aplicação esta sendo executada. Cada aplicação possui requisitos mínimos que devem ser compridos para garantir a qualidade do serviço. O atraso mínimo é um desses requisitos que devem ser garantidos. Como vimos nos experimentos, vários fatores influenciam o atraso, tais como:

- O aumento no número de clientes por servidor;
- Quantidades de níveis de servidores;
- Protocolo utilizado.

Outros fatores como a rede utilizada e o hardware dos participantes, entre outros, também devem ser considerados.

Nossa arquitetura funciona muito bem para ambientes massivos que não tenham grandes requisitos temporais, como jogos de estratégia, jogos que utilizam turnos, entre outros.

Caso a aplicação necessite de baixo delay, e o sistema precisa trabalhar com um número elevado de usuários é necessário que a raiz(root) e os primeiro filhos trabalhem como servidores dedicados, em uma rede de alta velocidade, como na arquitetura Ring e na de Assitotis, para reduzir o tempo na entrega das mensagens. Os clientes prioritários(os que pagam pelo serviço, por exemplo) ficariam mais próximos desses servidores, o que garantiria a qualidade do serviço, e os clientes secundários(clientes não pagantes) ficariam no final da árvore e embora recebendo mensagens com um *delay* maior, não ficariam excluídos do sistema.

Capítulo 6 – Conclusões e Perspectivas

"O único lugar onde sucesso vem antes do trabalho é no dicionário"

Albert Einstein

Neste trabalho, propomos uma nova abordagem para resolver o problema de Ambientes Colaborativos de Larga Escala, baseada no modelo Cliente-Servidor Hierárquico. Introduzimos maneiras eficientes de solucionar problemas que possam aparecer, tais como, a definição de modelos e operadores para determinar agrupamentos de usuários. Tais operadores são necessários no caso de entrada e saída de usuários no sistema. Propomos um modelo de dispersão de eventos, ainda não tratado nos atuais sistemas massivos, ao definirmos prioridades e aplicação de filtragem aos eventos. Propomos também soluções visando obter uma maior robustez da arquitetura e validamos o modelo através de simulações e emulações de rede, o que nos possibilitou realizar testes com número elevado de usuários. O modelo esta sendo utilizado em um estudo de caso real, a torcida virtual (TAVARES, 2004). O modelo também foi testado em uma aplicação que implementa um museu virtual (**NAC-UFRN**), disponível dentro do projeto **ICSpace** (Tavares, 2001) assim como também aplicado aos projetos **GIGAVR** e **GTVR**(Grupo de Trabalho Visualização Remota), descritos anteriormente, em que um usuário poderá tele-operar uma plataforma robótica móvel utilizando um ambiente virtual colaborativo como interface através da rede GIGA, assim como em aplicações **interperceptivas**, como é o caso do projeto **Garagem Virtual** e **Futebol de Botão**.

6.1 Contribuições

A contribuição principal deste trabalho é o *Framework*, H-N2N em si, que foi descrito no texto através de exemplos de aplicações, bem como mostrados seus

detalhes de implementação. Os testes e experimentos comprovam a aplicabilidade do H-N2N e as várias publicações conseguidas mostram a originalidade e mérito científico e acadêmico deste trabalho (Aranibar, 2006; Azevedo, 2006; Bezerra, 2005; Burlamaqui, 2005a; Burlamaqui, 2005b; Burlamaqui, 2005c; Burlamaqui, 2006a; Burlamaqui, 2006b; Burlamaqui 2006c; Dantas, 2005a; Dantas, 2005b; Dantas, 2006; Melo, 2006; Melo, 2007). Várias outras contribuições de caráter teórico e prático podem ser ainda enumeradas como resultados deste trabalho, sendo discutidas a seguir.

6.1.1 Contribuições teóricas

Este trabalho é uma boa fonte de consulta para o estudo sobre técnicas de reuso de software, como por exemplo padrões de projeto, contendo não só a descrição de alguns padrões como também exemplos de sua utilização, e *frameworks*, pois fornece também uma boa fundamentação teórica e serve como um bom exemplo dos passos a serem seguidos na construção de novos *frameworks*. Além do material aqui contido, são oferecidos muitas referências indispensáveis a quem deseje se aprofundar no assunto.

Outra contribuição deste trabalho está relacionada à análise de algumas aplicações desenvolvidas no âmbito do laboratório Natalnet. Depois de notarmos uma certa repetição nas soluções dadas ao construirmos novas aplicações, conseguimos capturar e comparar a essência de cada uma delas, e com isso encontramos seus pontos comuns, definindo com isso um **domínio** em que essas aplicações se enquadram. Desse modo, ao idealizarmos novas aplicações, temos parâmetros e características que tornem possível a classificação ou não dessa nova aplicação neste domínio.

A partir dessa análise, foi projetado um *framework*, onde foi aplicado padrões de projetos: tanto relacionados a criação de objetos (*Factory Method*), como padrões comportamentais (*Observer*), padrões com características de concorrência (*Message Queuing*) e de alocação de memória (*Pool Alocation*).

O *framework* foi projetado para promover o reuso em todos os três níveis do desenvolvimento, de modo que os desenvolvedores poderão tirar todos os

benefícios de quem utiliza esta ferramenta de reuso, ou seja, maximização no reaproveitamento de design e minimização na codificação.

O H-N2N é a única arquitetura dentre as encontradas na literatura que consegue lidar com o problema do "tiro na multidão". Por conseguir prover várias cópias de um mesmo ambiente que se encontra lotado e mesmo assim possibilitar que certas mensagens possam chegar a todos os clientes desses ambientes. A arquitetura hierárquica do H-N2N juntamente com a sua solução do grafo de dispersão fazem do H-N2N único.

6.1.2 Contribuições práticas

A contribuição prática mais importante consiste na implementação do *framework* H-N2N. Além de implementa-lo inteiramente, utilizando a linguagem Java, fizemos a sua validação, ou seja, o teste do *framework*. Dando por finalizada a primeira etapa na construção de um *framework*.

Outra contribuição deixada por este trabalho consistiu na produção de ferramentas construídas sobre o *framework* H-N2N, ferramentas estas que além de servirem como exemplos na construção de novas aplicações sobre o *framework*, são ferramentas operacionais que pode desde já serem reutilizadas, graças a flexibilidade promovida pelo H-N2N (reflexo da utilização do padrão MVC em sua arquitetura).

Por fim, temos a construção de um *website* (http://hn2n.wikitod.com), onde publicamos todos os resultados, documentação, exemplos de aplicações, códigos fontes do *framework*, ou seja, tudo relacionado ao H-N2N, com o objetivo de trocar informações e manter sempre sua arquitetura e códigos em evolução através da troca de experiências de sua utilização.

6.2 Perspectivas

Muitos são os planos para o futuro do H-N2N. De imediato, uma vez que sua implementação já está concluída e validada, temos a tarefa de sua manutenção. Esta manutenção consiste em dar suporte a seus usuários, através da atualização da documentação, disponibilização de aplicações exemplo, e também através da

análise das novas aplicações que surgem e que mostrem a necessidade de realizar pequenas alterações que venham a melhorar a solução dada pela versão atual do H-N2N, promovendo assim a sua evolução.

Outra idéia já mencionada anteriormente é a de, com o tempo e evolução do *framework*, fazer com que o H-N2N perca a característica de ser um *Whitebox*. Ou seja, com a evolução de suas aplicações, poderemos incluir cada vez mais códigos que possam ser reutilizados na construção de novas aplicações. A idéia é que ele não perca também totalmente sua flexibilidade, desse modo ele passará a ser um misto entre *framework Whitebox* e *Blackbox*, conhecido na literatura como *Graybox*.

Podemos ter uma idéia parcial do que o H-N2N poderá se tornar baseado nos projetos que já o utilizam. Atualmente o H-N2N foi ou está sendo utilizado em vários projetos de pesquisa , como:

- Projeto ICSPACE - Um Espaço Cultural Aberto na Internet, na implementação do Servidor multiusuário VIXNU, com o propósito de prover percepção e comunicação entre os usuário do sistema *web* ICSPACE.
- Projeto HITV - Desenvolvimento de Hardware e Software para TV de Alta Definição, na construção do programa de TV Interativa Torcida Virtual. Comprovado a sua eficiência na construção deste tipo de sistema, ele poderá vir a ser utilizado em vários outros programas de TV Interativa.
- Projeto GIGAVR, na implementação de uma extensão do Servidor multiusuário VIXNU. O objetivo deste projeto e montar uma estrutura composta de CAVERNAS DIGITAIS (Zuffo, 2004) espalhadas pelo Brasil, e interligadas pela rede GIGA (RNP, 2004). O H-N2N pode ser utilizado na construção de um sistema que promova o compartilhamento, percepção e comunicação entre os usuários das cavernas.
- Projeto GT-VR: o objetivo do GT de Visualização Remota é desenvolver e disponibilizar a comunidade um sistema multiusuário que aprimore as formas de interação existentes nos atuais ambientes de realidade mista (incluindo qualquer sistema manipulativo) tornando-os bem mais colaborativos e acessíveis pela comunidade.

- Além de projetos de pesquisa, algumas trabalhos acadêmicos incluem o H-N2N no seu desenvolvimento, tal como dissertação de mestrado em andamento, utilizando o H-N2N para o estudo de AVC interperceptivos.
- Projeto GT-MV: Sistema para Construção e Manutenção de Museus Virtuais 3D e 2D. O objetivo é projetar um sistema para construção de ambientes virtuais na forma de museus onde os participantes da rede RNP possam disponibilizar seus acervos de obras na rede de uma maneira fácil. Este sistema quando pronto deverá funcionar como um serviço disponível diariamente e de fácil manipulação por qualquer usuário da rede. O sistema ainda fornecerá ferramentas para comunicação entre todos os visitantes que estão acessando o museu on-line ao mesmo tempo. Através dessas ferramentas será permitido aos visitantes virtuais compartilhar suas experiências e sensações de visitar aquele museu.

Referências Bibliográficas

(AIM, 2008) The official Web site of AOL Instant Messenger http://www.aim.com/; Ultimo acesso Janeiro de 2008.

(Araújo, 2001) ARAÚJO, A. S., TAVARES, T. A., SOUZA FILHO, G. L., O Desenvolvimento de um Centro Cultural Virtual na Internet, In: News Generation Boletim bimestral sobre tecnologia de redes produzido e publicado pela RNP - Vol. 1, Num. 4. ISSN 1518 5974, 2001.

(Aranibar, 2006) ARANIBAR, Dennis Barrios ; BURLAMAQUI, Aquiles Medeiros Filgueira ; GURGEL, Viviane ; SANTOS, Marcela ; ARAUJO, Gianna ; CESAR, Valber; GONÇALVES, Luiz Marcos . Technological Inclusion Using Robots. In: Encontro de Robótica Inteligente, 2006, Campo Grande, 2006

(Assiotis, 2006) Assiotis, Mario; Tzanov, Velin; A Distributed Architecture for MMORPG, In The 5th workshop on Network & System Support for Games 2006 – NETGAMES 2006.

(Azevedo, 2006) AZEVEDO, Samuel; BURLAMAQUI, Aquiles M F ; DANTAS, Rummenigge R ; SCHNEIDER, Claudio A.; GOMES, Rafael Beserra ; MELO, Julio César ; XAVIER, Josivan ; GONÇALVES, Luiz M G . Interperception on Shared Virtual Environments. In: IEEE International Conference on Virtual Environments, Human-Computer Interfaces, and Measurement Systems,VECIMS, 2006, La Corunha, 2006.

(Bezerra, 2005) BEZERRA, João Paulo ; BURLAMAQUI, Aquiles Medeiros Filgueira ;Gonçalves, Luiz M. Controle Imersivo Colaborativo de

Uma Plataforma Robótica Móvel Via Internet. In: Workshop de Teses e Dissertações, SIBGRAPI, 2005, NATAL, 2005.

(Bhattachargee,1997) S. Bhattachargee, M. Ammar, E. Zegura, N. Shah, and Z. Fei. Application layer anycasting. In Proc IEEE Infocom'97, 1997.

(Bittencourt, 2003) BITTENCOURT, João R.; GIRAFFA, Lucia M.M.; SANTOS, Rafael C. Criando Jogos Computadorizados Multiplataforma com Amphibian. In: II Congresso Internacional de Tecnologia e Inovação em Jogos para Computadores, Curitiba: GameNet/CTIS, 2003.

(Bittencourt, 2008) BITTENCOURT, João R.; GIRAFFA, Lucia M.M.; SANTOS; Home Page do JFrog; http://www.inf.pucrs.br/~ jricardo/jfrog/index.html. Ultimo acesso 03/01/2008.

(Booch, 2000) BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar; UML Guia do Usuário; Editora Campus 2000.

(Bondi, 2000) Bondi, André B.; 'Characteristics of scalability and their impact on performance', Proceedings of the 2nd international workshop on Software and performance, Ottawa, Ontário, Canadá, 2000, ISBN 1-58113-195-X, pagina 195 - 203

(Burdea, 1996) BURDEA, Grigore C. Force and Touch Feedback for Virtual Reality. Ed. WileyProfessional Computing, 1996. Printed in United States of América.

(Burlamaqui, 2002) BURLAMARQUI, Aquiles; TAVARES, Tatiana Aires e SOUZA FILHO, Guido Lemos de. Vixnu - Um Servidor Multi-usuário com Suporte a Comunicação em Ambientes Virtuais Colaborativos. In: VIII SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E HIPERMÍDIA - SBMIDIA, 2002, Fortaleza, CE, Brasil. Anais do VIII Simpósio Brasileiro de Sistemas Multimídia e Hipermídia. 2002.

(Burlamaqui, 2005a) BURLAMAQUI, Aquiles Medeiros Filgueira ; SOUZA, Anderson A S ; BEZERRA, João Paulo A ; DANTAS, Rummenigge R ; SERRANO, Carlos G R ; SOUZA FILHO, Guido Lemos ; OLIVEIRA, Jauvane ; GONÇALVES, Luiz M G . A Framework for Collaborative Interaction of People and Robots in the Web. In: 3rd Latin American Web Congress,LA Web, IEEE, 2005, Buenos Aires, 2005. p. 179-182.

(Burlamaqui, 2005b) BURLAMAQUI, Aquiles Medeiros Filgueira ; OLIVEIRA, Jauvane ; SOUZA FILHO, Guido Lemos ; GONÇALVES, Luiz M G . H-N2N - Uma solução escalável para ambientes virtuais colaborativos massivos. In: IV Workshop Técnico-Cientifico do DIMAp, 2005, Natal, 2005

(Burlamaqui, 2005c) BURLAMAQUI, Aquiles Medeiros Filgueira ; SOUZA FILHO, Guido Lemos ; OLIVEIRA, Jauvane ; GONÇALVES, Luiz M G . H-N2N - Uma solução escalável para ambientes virtuais colaborativos massivos. In: Workshop de Tese e Dissertações, Webmidia, 2005, Poços de Caldas, MG, 2005

(Burlamaqui, 2006a) BURLAMAQUI, Aquiles Medeiros Filgueira ; DANTAS, Rummenigge R ; SCHNEIDER, Claudio A. ; XAVIER, Josivan ; AZEVEDO, Samuel ; MELO, Julio César ; GOMES, Rafael Beserra ; GONÇALVES, Luiz M G . Desenvolvimento de ambientes multi-usuários interdimensionais. In: Simpósio Realidade Virtual, SVR, 2006, Belém, PA. Proceedings of SVR 2006, 2006.

(Burlamaqui, 2006b) BURLAMAQUI, Aquiles Medeiros Filgueira ; OLIVEIRA, Marlos Marques ; GONÇALVES, Luiz M G ; SOUZA FILHO, Guido Lemos ; OLIVEIRA, Jauvane . A Scalable Hierarchical Architecture for large scale multi-user virtual environments. In: IEEE International Conference on Virtual Environments, Human-

Computer Interfaces, and Measurement Systems,VECIMS, 2006, La Corunha, 2006.

(Burlamaqui, 2006c) BURLAMAQUI, Aquiles Medeiros Filgueira ; SOUZA, Anderson A S ; BEZERRA, João Paulo A ; DANTAS, Rummenigge R ; SCHNEIDER, Claudio A. ; XAVIER, Josivan ; GONÇALVES, Luiz M G . A Multimedia Framework for Collaborative Interaction of People and Robots Through the Web. In: IEEE International Conference on Virtual Environments, Human-Computer Interfaces, and Measurement Systems,VECIMS, 2006, La Corunha. Proceedings of VECIMS2006, 2006.

(Capin, 1998) CAPIN, T. K., PANDZIC, I. S., THALMANN, N. M, THALMANN D., Realistic Avatars and Autonomous Virtual Humans, Virtual Worlds in the Internet (R.Earnshaw and J. Vince, eds) IEEE Computer Society Press, pp.157-174, 1998.

(Capin, 1999) CAPIN, T., A Taxonomy of Networked Virtual Environments, in Avatars in Networked Virtual Environments, D. Thalmann and I. Pandzic Editors, 1999, John wiley & Sons, pp.15-58.

(Curtis, 1993) CURTIS, Pavel and NICHOLS, David A. MUDs Grow Up: Social Virtual Reality in the Real World (1993) COMPCON. < http://citeseer.nj.nec.com/curtis93muds.html>

(Darlagiannis, 2000) V. Darlagiannis, N. D. Georganas, “Virtual Collaboration and Media Sharing using COSMOS”, 4th world multiconference on Circuits, Systems, Communications & Computers, July 2000

(Dantas, 2005a) DANTAS, Rummenigge R ; BURLAMAQUI, Aquiles Medeiros Filgueira ; BURLAMAQUI, Aquiles M F ; GONÇALVES, Luiz M G . Um Componente Gráfico Portável para o Desenvolvimento de Ambientes Virtuais Multi-Usuários. In: Workshop de Iniciação Científica, SIBGRAPI, 2005, Natal, 2005.

(Dantas, 2005b) DANTAS, Rummenigge R ; BURLAMAQUI, Aquiles Medeiros Filgueira ; SCHNEIDER, Claudio A. ; XAVIER, Josivan ; BURLAMAQUI, Aquiles M F ; GONÇALVES, Luiz Marcos . Um Componente Gráfico para o Desenvolvimento de Ambientes Virtuais Massivos com Java3D. In: Workshop de Iniciação Cientifica, WebMedia 2005, 2005, Poços de Caldas, 2005.

(Dantas, 2006) DANTAS, Rummenigge; BURLAMAQUI, Aquiles Medeiros Filgueira ; BURLAMAQUI, Aquiles M F ; GONÇALVES, Luiz Marcos . Providing Acessibilty on Shared Virtual Environments. In: WebMedia 2006 - Workshop de Teses e Dissertações, 2006, Natal. WebMedia 2006 - Workshop de Teses e Dissertações, 2006. p. 58-62.

(Ellis, 1994) ELLIS, S. R., What are Virtual Environments?, IEEE Computer Graphics & Applications, 14(1):17-22, Jan. 1994.

(Fayad, 99) FAYAD, Mohamed C.; SCHMIDT, Douglas C.; JOHNSON, Ralph E. Building Application Frameworks – Object-Oriented foundations of frameworks design. Estados Unidos: Wiley, 1999

(Funkhouser, 1995) Funkhouser, Thomas A. RING: A Client-Server System for Multi-User Virtual Environments, ACM SIGGRAPH Special Issue on 1995 Symposium on Interactive 3D Graphics, (Monterey, CA), 1995, 85-92.

(GAMMA, 95) Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John; Padrôes de Projeto, Soluções Reutilizáveis de Software Orientado a Objetos. Bookman.

(Garlan, 1997) GARLAN, David; MONROE, Robert & WILE, David. Acme: an Architecture Interchange Language. Janeiro-1997.

(Glenn , 1988) Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80.

Journal of Object-Oriented Programming, 1(3):26-49, Agosto/Setembro 1988.

(Gnutella, 2008) Site oficial GNUtella; http://www.gnutella.com/; Ultimo acesso Janeiro de 2008.

(Greenhalgh, 2000) C. Greenhalgh, J. Purbrick, and D. Snowdon, "Inside Massive-3:Flexible support for data consistency and world structuring," in Proceedings of the Third International Conference on Collaborative Virtual Environments, (San Francisco, California), September 2000.

(Greenhalgh, 1995) GREENHALGH, C., BENFORD, S., MASSIVE, A Collaborative Virtual Environment for Teleconferencing, in ACM Transactions on Human- Computer Interaction (Special Issue on Virtual Reality Software and Technology), Vol. 2, No. 3, 1995.

(Greenhalgh, 1999) GREENHALGH, C., Large Scale Collaborative Virtual Environments, 1999: Springer-Verlag, London-UK, 229p.

(Guibas, 1985) L. Guibas, J. Stolfi. Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams. ACM Trans. on Graphics, Vol. 4, No. 2, Pages 74-123, April 1985.

(Hagsand, 1997) HAGSAND, O., Interactive Multiuser VEs in the DIVE System, in IEEE Multimedia, Vol. 4, No.1, Jan-Mar, 1997.

(Halsall, 2001) Halsall, F. Multimedia Communications: applications, networks, protocols and standards. Addison Wesley, 2001.

(Hu, 2004) HU S., LIAO, G., Scalable Peer-to-Peer Networked Virtual Environment, Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04: Network and system support for games, Portland, Oregon, USA, 2004, pp.129-133.

(Hyacinth, 1996) HYACINTH S. N., Software Agents: An Overview, Knowledge Engineering Review, Vol. 11, No 3, Cambridge University Press, 1996, pp. 205-244.

(Jannotti, 2000) J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O. Jr. Overcast: Reliable multicasting with an overlay network. In Proceedings of USENIX Symposium on Operating Systems Design and Implementation, Oct. 2000.

(J-Sim, 2005) J-Sim Home Page - http://www.j-sim.org/ (29/06/2005)

(Johnson, 1992) JONHSON, R, Documenting Frameworks Using Patterns. *Proceedings of OOPSLA* 1992, pp. 63-76, Vancouver, BC , Outubro 1992.

(Johnson, 1993) JOHNSON,R,E. How to design frameworks. 1993. Disponível por FTP anômino em st.es.cs.uiuc.edu.(dez. 98)

(JUDE, 2005) UML Modeling Tool – JUDE - http://objectclub.esm.co.jp/Jude/jude-e.html, Ultimo acesso 04/12/2005.

(Kawahara, 2003) KAWAHARA, Y., MORIKAWA, H., AOYAMA, T., A Peer-to-Peer Message Exchange Scheme for Large Scale Networked Virtual Environments, In proceedings of 1st European Across Grids Conference, Volume 2970 of Springer LNCS, pp.182-189, Santiago de Compostela, Spain, February, 2003.

(Kellerer, 1998) W. Kellerer,Dienstarchitekturen in der Telekommunikation - Evolution, Methoden und Vergleich, Techinical Report TUM-LKN-TR-9801, 1998.

(Knutsson, 2004) KNUTSSON, B., LU, H., XU, W., HOPKINS, B., Peer-to-Peer Support for Massively Multiplayer Games, INFOCOM 2004, March 2004, Hong Kong, China.

(Kohonen, 2001) KOHONEN, T., Self-Organizing Maps, 3rd Ed. Springer-Verlag, Berlin Heidelberg New York , 2001.

(Louvre, 2004) Site do Museu do Louvre - http://www.louvre.fr/louvrea.htm (01/12/2004)

(Macedonia, 1997) MACEDONIA, M. R., ZYDA, M. J., A Taxonomy for Networked Virtual Environments, IEEE Multimedia, Vol.4, No.1, January-March 1997, pp. 48-56.

(Macedonia, 1994) MACEDONIA, M. R., ZYDA, M. J., PRATT, D. R., BARHAM, P. T., ZESWITZ, S., NPSNET: A Network Software Architecture for Large Scale Virtual Environments, Presence, Vol. 3, No. 4, Fall , 1994, pp.265-287.

(Manninent, 2001) MANNINEN, T., Virtual Team Interactions in Networked Multimedia Games - Case: “Counter Strike”- Multiplayer 3D Action Game, In Proceedings of PRESENCE2001 Conference, May 21-23, Philadelphia, USA, 2001.

(Mchicago, 2004) Site do Museu de Arte Contemporânea de Chicago - http://www.mcachicago.org/ (01/12/2004)

(Melo, 2006) MELO, Julio César ; SCHNEIDER, Claudio A. ; AZEVEDO, Samuel ; SOUZA FILHO, Guido Lemos ; DANTAS, Rummenigge R ; BURLAMAQUI, Aquiles Medeiros Filgueira ; GONÇALVES, Luiz M G . Percepcom2D, uma abordagem 2D para ambientes Virtuais Colaborativos Multi-Usuário. In: WebMedia 2006 - Workshop de trabalhos de Iniciação Cientifica, 2006, Natal. WebMedia 2006 - Workshop de trabalhos de Iniciação Cientifica, 2006. p. 118-120.

(Melo, 2007) MELO, Julio; DANTAS, Rummenigge R; BURLAMAQUI, Aquiles ; SCHNEIDER, Claudio A. ; XAVIER, Josivan ; AZEVEDO, Samuel ; GONÇALVES, Luiz M G . Interperceptive games on SBGAMES 2007

(Milgram, 1994) Milgram, P., et al. Augmented reality: a class of displays on the reality-virtuality continuum. in SPIE Volume 2351:Telemanipulator and Telepresence Technologies. 1994.

(Milojicic, 2002) D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins and Z. Xu, "Peer to Peer Computing," HP Technical Report, HPL-2002-57.

(Mockapetris, 1989) MOCKAPETRIS, P. 1989. DNS Encoding of Network Names and Other Types. Internet Request For Comment 1101 Network Information Center, SRI International, Menlo Park, California.

(Morillo, 2003) MORILLO, P., FERNÁNDEZ, M., PELECHANO, N., A Grid Representation for Distributed Virtual Environments, In proceedings of 1st European Across Grids Conference, Volume 2970 of Springer LNCS, pp.182-189, Santiago de Compostela, Spain, February, 2003.

(Morningstar, 1991) Morningstar and R.F. Farmer. The lessons of Lucasfilm's habitat. In M. Benedikt, editor, Cyberspace: First Steps, pages 273--302. MIT Press, 1991.

(Napster, 2008) Site Oficial; http://www.napster.com/; Ultimo acesso Janeiro de 2008;

(Oliveira, 2003) OLIVEIRA, J. C., GEORGANAS, N. D., VELVET: An Adaptive Hybrid Architecture for VEry Large Virtual EnvironmenTs, Presence: Teleoperators and Virtual Environments, Volume 12 , Issue 6, 2003, pp.555-580.

(Oliveira, 1999) OLIVEIRA, M. A. M. S., TODESCO, G., ARAUJO, R. B., The Limitations of Interactive Multiuser 3D Enviroment in the WWW, In proceedings of the 10th International Workshop on Database & Expert Systems Applications, IEEE Conference, Florence-Italy, September, 1999.

(Park, 1999) PARK, K. S., KENYON, R. V., Effects of Network Characteristics on Human Performance in a Collaborative Virtual Environment, In:

IEEE International Conference on Virtual Reality (VR '99), Houston, Texas, March 1999.

(Partridge, 1993) C. Partridge, T. Mendez, and W. Milliken. RFC 1546: Host anycasting service, November 1993. Status: INFORMATIONAL. 184.

(Pullen, 1997) PULLEN, J. M. et al., Limitations of Internet Protocol Suite for Distributed Simulation in the Large Multicast Environments, Working Internet Draft <draft-ietf-lsma-limitations-01.txt>, March, 1997.

(Pygoya, 2004) Site do Museu Pygoya - http://www.lastplace.com (01/12/2004).

(Radenkovic, 2002) Radenkovic, Milena; Greenhalgh, Chris; Benford, Steve; Deployment Issues for Multi-User Audio Support in CVEs; 2002, ACM VRST'02

(Raychaudhuri, 2000) RAYCHAUDHURI, S. et al., Principal components analysis to summarize microarray experiments: application to sporulation time series, Pac. Symp. Biocomput, 2000, 455–466.

(RFC2136, 1997) RFC 2136 - Dynamic Updates in the Domain Name System (DNS UPDATE); http://tools.ietf.org/html/rfc2136

(Rüdiger, 2002) Rüdiger Schollmeier, A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architetures and Applications, IEEE Internet Computing, July, 2002.

(Shaw, 1993) SHAW, C., et al., Decoupled Simulation in Virtual Reality with the MRToolkit, ACM Transactions on Information Systems, Vol. 11, No. 3, July 1993, pp. 287-317.

(Shirmohammadi, 2001) SHIRMOHAMMADI, S., GEORGANAS, N. D., An End-to-End Communication Architecture for Collaborative Virtual Environments, Computer Networks: The International Journal of

Computer and Telecommunications Networking, Volume 35, Issue 203, February, 2001, pp.351-367.

(Singh, 1994) SINGH, G., et al., BrickNet: A Software Toolkit for Networked-Based Virtual Worlds, in Presence Teleoperators and Virtual Environments, Vol. 3, No.1,Winter, 1994.

(Singhal, 1999) SINGHAL, S., MACEDONIA, M. R., Networked Virtual Environments: Design and Implementation, 1999: Addison-Wesley, 331p.

(Sutherland, 2002) Web Site artmuseum http://www.artmuseum.net/w2vr/timeline/Sutherland.html, Ultimo acesso 2002.

(Tanenbaum, 2002) Tanenbaum, A.S. Distributed Systems: Principals and Practice. Prentice-Hall International, 2002.

(Tavares, 2001) TAVARES, T. A., ARAÚJO, A. S., SOUZA FILHO, G. L., ICSpace – The Artists place on the Net, In: AMT´2001, 2001, Hong Kong. Springer-Verlag Lectures notes in Computer Science. Hong Kong: Springer-Verlag Press, 2001. v. 2252

(Tavares, 2003) TAVARES, Douglas M; BURLAMAQUI, Aquiles M. F; DIAS, Anfranserai M; MONTEIRO, Meika I.; ANTUNES, Vivivane A.; TAVARES, Tatiana A.; LIMA, Carlos M. de; GOLÇALVES, Luiz M e SOUZA FILHO, Guido L; “HYPERPRESENCE - An Application Environment for Control of Multi-User Agents in Mixed Reality Spaces” In: 36th Simulation Simposium 2003, Orlando, Florida.1080-241X/03 $17.00 2003 IEEE.

(Tavares, 2004) TAVARES T., et all. Sharing Virtual Acoustic Spaces over Interactive TV Programs - Presenting "Virtual Cheering" Application. Em: The 2004 IEEE International Conference on Multimedia and Expo-ICME´2004.

(Waters, 1997) Waters, R., Anderson, D., Barrus, J., Brogan, D., Casey, M., McKeown, S., Nitta, T., Sterns, I. & Yerazunis, W., Diamond Park

and Spline: a social virtual reality system with 3D animation, spoken interaction, and runtime extendibility, PRESENCE, 6 (4), 461-481, 1997, MIT.

(Zhang, 1997) ZHANG, Y. C., Evaluation and Comparison of Different Segmentation Algorithms, Pattern Recognition Letters 18,1997, pp. 963-974.

(Zyda, 1996) Zyda, Michael J. Networking Virtual Environments, California, 1996.

Apêndice A – Log dos experimentos

Nas tabelas abaixo podemos ver como cada um dos clientes se comportou durante os experimentos realizados. A notação das mensagens utiliza o seguinte formato:

- id_do_cliente_destino:ip_do_cliente_destino:porta_do_cliente_destino:delay_em_milissegundos

Ex: - 1:10.13.99.208:3694 Delay: 657 ms

Experimento III

Configuração 1

Log dos testes usando o protocolo TCP:

Cliente 10.13.99.208:3693		
PING 0	**PING 3 (resultado de duas mensagens)**	
- 1:10.13.99.208:3694 Delay: 657 ms	**- 1:10.13.99.197:3694 Delay: 328 ms** **- 0:10.13.99.197:3693 Delay: 328 ms** **- 1:10.13.99.195:3694 Delay: 500 ms** **- 1:10.13.99.208:3694 Delay: 500 ms** **- 0:10.13.99.195:3693 Delay: 500 ms** **- 0:10.13.99.212:3693 Delay: 984 ms** **- 1:10.13.99.212:3694 Delay: 984 ms**	**- 1:10.13.99.197:3694 Delay: 328 ms** **- 0:10.13.99.197:3693 Delay: 328 ms** **- 1:10.13.99.208:3694 Delay: 547 ms** **- 1:10.13.99.195:3694 Delay: 547 ms** **- 0:10.13.99.195:3693 Delay: 547 ms** **- 0:10.13.99.212:3693 Delay: 984 ms** **- 1:10.13.99.212:3694 Delay: 984 ms**

Cliente 10.13.99.208:3694		
PING 0	**PING 3 (resultado de duas mensagens)**	
- 0:10.13.99.208:3693 Delay: 656 ms	**- 0:10.13.99.208:3693 Delay: 656 ms** **- 0:10.13.99.197:3693 Delay: 875 ms** **- 1:10.13.99.197:3694 Delay: 875 ms** **- 1:10.13.99.195:3694 Delay: 1313 ms** **- 0:10.13.99.195:3693 Delay: 1313 ms** **- 0:10.13.99.212:3693 Delay: 1531 ms** **- 1:10.13.99.212:3694 Delay: 1531 ms**	**- 0:10.13.99.208:3693 Delay: 656 ms** **- 1:10.13.99.197:3694 Delay: 875 ms** **- 0:10.13.99.197:3693 Delay: 875 ms** **- 1:10.13.99.195:3694 Delay: 1312 ms** **- 0:10.13.99.195:3693 Delay: 1312 ms** **- 0:10.13.99.212:3693 Delay: 1312 ms** **- 1:10.13.99.212:3694 Delay: 1312 ms**

Cliente 10.13.99.197:3693	
PING 0	**PING 3 (resultado de duas mensagens)**

- 1:10.13.99.197:3694 Delay: 657 ms	- 0:10.13.99.208:3693 Delay: 328 ms	- 0:10.13.99.208:3693 Delay: 218 ms
	- 1:10.13.99.197:3694 Delay: 562 ms	- 1:10.13.99.197:3694 Delay: 437 ms
	- 1:10.13.99.195:3694 Delay: 781 ms	- 0:10.13.99.195:3693 Delay: 656 ms
	- 0:10.13.99.195:3693 Delay: 781 ms	- 1:10.13.99.195:3694 Delay: 656 ms
	- 1:10.13.99.208:3694 Delay: 781 ms	- 1:10.13.99.208:3694 Delay: 656 ms
	- 0:10.13.99.212:3693 Delay: 984 ms	- 0:10.13.99.212:3693 Delay: 1093 ms
	- 1:10.13.99.212:3694 Delay: 984 ms	- 1:10.13.99.212:3694 Delay: 1093 ms

Cliente 10.13.99.197:3694		
PING 0	PING 3 (resultado de duas mensagens)	
- 0:10.13.99.197:3693 Delay: 657 ms	- 1:10.13.99.195:3694 Delay: 329 ms	- 1:10.13.99.195:3694 Delay: 328 ms
	- 0:10.13.99.208:3693 Delay: 329 ms	- 0:10.13.99.195:3693 Delay: 328 ms
	- 0:10.13.99.195:3693 Delay: 329 ms	- 0:10.13.99.208:3693 Delay: 328 ms
	- 0:10.13.99.212:3693 Delay: 657 ms	- 0:10.13.99.197:3693 Delay: 656 ms
	- 0:10.13.99.197:3693 Delay: 657 ms	- 0:10.13.99.212:3693 Delay: 656 ms
	- 1:10.13.99.212:3694 Delay: 657 ms	- 1:10.13.99.212:3694 Delay: 656 ms
	- 1:10.13.99.208:3694 Delay: 704 ms	- 1:10.13.99.208:3694 Delay: 703 ms

Cliente 10.13.99.195:3693		
PING 0	PING 3 (resultado de duas mensagens)	
- 1:10.13.99.195:3694 Delay: 657 ms	- 1:10.13.99.197:3694 Delay: 328 ms	- 1:10.13.99.197:3694 Delay: 328 ms
	- 0:10.13.99.208:3693 Delay: 547 ms	- 1:10.13.99.195:3694 Delay: 547 ms
	- 1:10.13.99.195:3694 Delay: 547 ms	- 0:10.13.99.208:3693 Delay: 547 ms
	- 0:10.13.99.212:3693 Delay: 766 ms	- 1:10.13.99.212:3694 Delay: 984 ms
	- 0:10.13.99.197:3693 Delay: 766 ms	- 0:10.13.99.197:3693 Delay: 984 ms
	- 1:10.13.99.212:3694 Delay: 766 ms	- 0:10.13.99.212:3693 Delay: 984 ms
	- 1:10.13.99.208:3694 Delay: 1203 ms	- 1:10.13.99.208:3694 Delay: 1203 ms

Cliente 10.13.99.195:3694		
PING 0	PING 3 (resultado de duas mensagens)	
- 0:10.13.99.195:3693 Delay: 656 ms	- 1:10.13.99.212:3694 Delay: 328 ms	- 0:10.13.99.212:3693 Delay: 328 ms
	- 1:10.13.99.197:3694 Delay: 328 ms	- 1:10.13.99.212:3694 Delay: 328 ms
	- 0:10.13.99.212:3693 Delay: 328 ms	- 1:10.13.99.197:3694 Delay: 328 ms
	- 0:10.13.99.195:3693 Delay: 656 ms	- 0:10.13.99.208:3693 Delay: 656 ms
	- 0:10.13.99.208:3693 Delay: 656 ms	- 0:10.13.99.195:3693 Delay: 656 ms
	- 0:10.13.99.197:3693 Delay: 875 ms	- 0:10.13.99.197:3693 Delay: 875 ms
	- 1:10.13.99.208:3694 Delay: 1093 ms	- 1:10.13.99.208:3694 Delay: 1094 ms

Cliente 10.13.99.212:3693	
PING 0	PING 3 (resultado de duas mensagens)

- 0:10.13.99.212:3693 Delay: 656 ms	- 1:10.13.99.195:3694 Delay: 328 ms - 1:10.13.99.197:3694 Delay: 563 ms - 1:10.13.99.212:3694 Delay: 563 ms - 0:10.13.99.208:3693 Delay: 1000 ms - 0:10.13.99.195:3693 Delay: 1000 ms - 0:10.13.99.197:3693 Delay: 1000 ms - 1:10.13.99.208:3694 Delay: 1641 ms	- 1:10.13.99.195:3694 Delay: 328 ms - 1:10.13.99.212:3694 Delay: 562 ms - 1:10.13.99.197:3694 Delay: 562 ms - 0:10.13.99.208:3693 Delay: 1000 ms - 0:10.13.99.195:3693 Delay: 1000 ms - 0:10.13.99.197:3693 Delay: 1093 ms - 1:10.13.99.208:3694 Delay: 1312 ms

Cliente 10.13.99.212:3694		
PING 0	PING 3 (resultado de duas mensagens)	
- 0:10.13.99.212:3693 Delay: 656 ms	- 1:10.13.99.195:3694 Delay: 328 ms - 1:10.13.99.197:3694 Delay: 546 ms - 0:10.13.99.212:3693 Delay: 546 ms - 0:10.13.99.208:3693 Delay: 765 ms - 0:10.13.99.195:3693 Delay: 984 ms - 0:10.13.99.197:3693 Delay: 1203 ms - 1:10.13.99.208:3694 Delay: 1421 ms	- 1:10.13.99.195:3694 Delay: 328 ms - 1:10.13.99.197:3694 Delay: 547 ms - 0:10.13.99.212:3693 Delay: 547 ms - 0:10.13.99.208:3693 Delay: 766 ms - 0:10.13.99.195:3693 Delay: 984 ms - 0:10.13.99.197:3693 Delay: 1203 ms - 1:10.13.99.208:3694 Delay: 1422 ms

Mesmo experimento usando o protocolo UDP

Cliente 10.13.99.208:3693		
PING 0	PING 3 (resultado de duas mensagens)	
- 1:10.13.99.208:3694 Delay: 0 ms	- 1:10.13.99.197:3694 Delay: 0 ms - 0:10.13.99.197:3693 Delay: 0 ms - 1:10.13.99.208:3694 Delay: 0 ms - 1:10.13.99.195:3694 Delay: 0 ms - 0:10.13.99.221:3693 Delay: 16 ms - 0:10.13.99.195:3693 Delay: 16 ms - 1:10.13.99.221:3695 Delay: 16 ms	- 1:10.13.99.197:3694 Delay: 0 ms - 0:10.13.99.197:3693 Delay: 0 ms - 1:10.13.99.208:3694 Delay: 0 ms - 1:10.13.99.195:3694 Delay: 0 ms - 0:10.13.99.221:3693 Delay: 0 ms - 0:10.13.99.195:3693 Delay: 15 ms - 1:10.13.99.221:3695 Delay: 15 ms

Cliente 10.13.99.208:3694		
PING 0	PING 3 (resultado de duas mensagens)	
- 0:10.13.99.208:3693 Delay: 0 ms	- 0:10.13.99.208:3693 Delay: 0 ms - 0:10.13.99.197:3693 Delay: 0 ms - 1:10.13.99.197:3694 Delay: 0 ms - 0:10.13.99.195:3693 Delay: 16 ms - 1:10.13.99.195:3694 Delay: 16 ms - 0:10.13.99.221:3693 Delay: 32 ms - 1:10.13.99.221:3695 Delay: 32 ms	- 0:10.13.99.208:3693 Delay: 0 ms - 1:10.13.99.197:3694 Delay: 0 ms - 0:10.13.99.197:3693 Delay: 0 ms - 1:10.13.99.195:3694 Delay: 15 ms - 0:10.13.99.221:3693 Delay: 15 ms - 1:10.13.99.221:3695 Delay: 15 ms - 0:10.13.99.195:3693 Delay: 15 ms

Cliente 10.13.99.197:3693		
PING 0	**PING 3 (resultado de duas mensagens)**	
- 1:10.13.99.197:3694 Delay: 0 ms	- 0:10.13.99.208:3693 Delay: 0 ms	- 0:10.13.99.208:3693 Delay: 0 ms
	- 1:10.13.99.197:3694 Delay: 0 ms	- 1:10.13.99.197:3694 Delay: 0 ms
	- 1:10.13.99.195:3694 Delay: 0 ms	- 1:10.13.99.195:3694 Delay: 0 ms
	- 1:10.13.99.208:3694 Delay: 0 ms	- 1:10.13.99.208:3694 Delay: 0 ms
	- 0:10.13.99.221:3693 Delay: 15 ms	- 1:10.13.99.221:3695 Delay: 0 ms
	- 0:10.13.99.195:3693 Delay: 15 ms	- 0:10.13.99.221:3693 Delay: 0 ms
	- 1:10.13.99.221:3695 Delay: 15 m	- 0:10.13.99.195:3693 Delay: 15 ms

Cliente 10.13.99.197:3694		
PING 0	**PING 3 (resultado de duas mensagens)**	
- 0:10.13.99.197:3693 Delay: 0 ms	- 0:10.13.99.208:3693 Delay: 0 ms	- 0:10.13.99.208:3693 Delay: 0 ms
	- 0:10.13.99.197:3693 Delay: 0 ms	- 0:10.13.99.195:3693 Delay: 0 ms
	- 0:10.13.99.195:3693 Delay: 0 ms	- 0:10.13.99.197:3693 Delay: 0 ms
	- 1:10.13.99.195:3694 Delay: 0 ms	- 1:10.13.99.195:3694 Delay: 0 ms
	- 1:10.13.99.208:3694 Delay: 0 ms	- 1:10.13.99.208:3694 Delay: 0 ms
	- 1:10.13.99.221:3695 Delay: 0 ms	- 1:10.13.99.221:3695 Delay: 0 ms
	- 0:10.13.99.221:3693 Delay: 16 ms	- 0:10.13.99.221:3693 Delay: 0 ms

Cliente 10.13.99.195:3693		
PING 0	**PING 3 (resultado de duas mensagens)**	
- 1:10.13.99.195:3694 Delay: 0 ms	- 1:10.13.99.197:3694 Delay: 0 ms	- 1:10.13.99.197:3694 Delay: 0 ms
	- 0:10.13.99.208:3693 Delay: 0 ms	- 1:10.13.99.195:3694 Delay: 0 ms
	- 0:10.13.99.197:3693 Delay: 0 ms	- 0:10.13.99.208:3693 Delay: 0 ms
	- 1:10.13.99.195:3694 Delay: 16 ms	- 0:10.13.99.197:3693 Delay: 0 ms
	- 1:10.13.99.221:3695 Delay: 16 ms	- 1:10.13.99.208:3694 Delay: 16 ms
	- 0:10.13.99.221:3693 Delay: 16 ms	- 1:10.13.99.221:3695 Delay: 16 ms
	- 1:10.13.99.208:3694 Delay: 16 ms	- 0:10.13.99.221:3693 Delay: 16 ms
Cliente 10.13.99.195:3694		
PING 0	**PING 3 (resultado de duas mensagens)**	
- 0:10.13.99.195:3693 Delay: 0 ms	- 0:10.13.99.221:3693 Delay: 0 ms	- 1:10.13.99.197:3694 Delay: 0 ms
	- 1:10.13.99.221:3695 Delay: 0 ms	- 0:10.13.99.208:3693 Delay: 0 ms
	- 1:10.13.99.197:3694 Delay: 0 ms	- 0:10.13.99.195:3693 Delay: 0 ms
	- 0:10.13.99.208:3693 Delay: 0 ms	- 0:10.13.99.197:3693 Delay: 0 ms
	- 0:10.13.99.195:3693 Delay: 0 ms	- 1:10.13.99.221:3695 Delay: 0 ms
	- 0:10.13.99.197:3693 Delay: 0 ms	- 1:10.13.99.208:3694 Delay: 16 ms
	- 1:10.13.99.208:3694 Delay: 16 ms	- 0:10.13.99.221:3693 Delay: 16 ms

Cliente 10.13.99.221:3693	
PING 0	**PING 3 (resultado de duas mensagens)**

- 1:10.13.99.221:3695 Delay: 0 ms	- 1:10.13.99.195:3694 Delay: 0 ms	- 1:10.13.99.195:3694 Delay: 0 ms
	- 1:10.13.99.221:3695 Delay: 0 ms	- 1:10.13.99.197:3694 Delay: 0 ms
	- 1:10.13.99.197:3694 Delay: 0 ms	- 1:10.13.99.221:3695 Delay: 0 ms
	- 0:10.13.99.195:3693 Delay: 0 ms	- 0:10.13.99.195:3693 Delay: 0 ms
	- 0:10.13.99.208:3693 Delay: 16 ms	- 0:10.13.99.208:3693 Delay: 0 ms
	- 0:10.13.99.197:3693 Delay: 16 ms	- 0:10.13.99.197:3693 Delay: 0 ms
	- 1:10.13.99.208:3694 Delay: 16 ms	- 1:10.13.99.208:3694 Delay: 16 ms

Cliente 10.13.99.221:3695		
PING 0	PING 3 (resultado de duas mensagens)	
- 0:10.13.99.221:3693 Delay: 0 ms	- 1:10.13.99.195:3694 Delay: 0 ms	- 1:10.13.99.195:3694 Delay: 0 ms
	- 0:10.13.99.221:3693 Delay: 0 ms	- 0:10.13.99.221:3693 Delay: 0 ms
	- 1:10.13.99.197:3694 Delay: 0 ms	- 1:10.13.99.197:3694 Delay: 0 ms
	- 0:10.13.99.195:3693 Delay: 0 ms	- 0:10.13.99.208:3693 Delay: 0 ms
	- 0:10.13.99.208:3693 Delay: 0 ms	- 0:10.13.99.197:3693 Delay: 0 ms
	- 0:10.13.99.197:3693 Delay: 0 ms	- 1:10.13.99.208:3694 Delay: 16 ms
	- 1:10.13.99.208:3694 Delay: 0 ms	- 0:10.13.99.195:3693 Delay: 16 ms

Configuração 2

Mensagem TCP

Cliente 10.13.99.212:3741		
PING 0	PING 3 (resultado de uma mensagem)	
- 6:10.13.99.212:3699 Delay: 547 ms	- 40:10.13.99.195:3733 Delay: 328 ms	- 5:10.13.99.197:3698 Delay: 2344 ms
- 37:10.13.99.212:3730 Delay: 547 ms	- 44:10.13.99.212:3737 Delay: 547 ms	- 26:10.13.99.197:3719 Delay: 2344 ms
- 42:10.13.99.212:3735 Delay: 547 ms	- 38:10.13.99.212:3731 Delay: 547 ms	- 41:10.13.99.195:3734 Delay: 2344 ms
- 13:10.13.99.212:3706 Delay: 547 ms	- 9:10.13.99.212:3702 Delay: 547 ms	- 44:10.13.99.195:3737 Delay: 2344 ms
- 14:10.13.99.212:3707 Delay: 547 ms	- 31:10.13.99.212:3724 Delay: 547 ms	- 14:10.13.99.197:3707 Delay: 2344 ms
- 26:10.13.99.212:3719 Delay: 547 ms	- 39:10.13.99.212:3732 Delay: 547 ms	- 43:10.13.99.195:3736 Delay: 2344 ms
- 10:10.13.99.212:3703 Delay: 547 ms	- 7:10.13.99.212:3700 Delay: 547 ms	- 18:10.13.99.197:3711 Delay: 2344 ms
- 16:10.13.99.212:3709 Delay: 547 ms	- 19:10.13.99.212:3712 Delay: 547 ms	- 11:10.13.99.195:3704 Delay: 2344 ms
- 23:10.13.99.212:3716 Delay: 547 ms	- 17:10.13.99.212:3710 Delay: 547 ms	- 36:10.13.99.195:3729 Delay: 2344 ms
- 18:10.13.99.212:3711 Delay: 547 ms	- 32:10.13.99.212:3725 Delay: 547 ms	- 0:10.13.99.195:3693 Delay: 2344 ms
- 34:10.13.99.212:3727 Delay: 547 ms	- 36:10.13.99.212:3729 Delay: 547 ms	- 42:10.13.99.195:3735 Delay: 2344 ms
- 11:10.13.99.212:3704 Delay: 547 ms	- 40:10.13.99.212:3733 Delay: 547 ms	- 38:10.13.99.195:3731 Delay: 2344 ms
- 0:10.13.99.212:3693 Delay: 547 ms	- 24:10.13.99.212:3717 Delay: 547 ms	- 12:10.13.99.195:3705 Delay: 2344 ms
- 32:10.13.99.212:3725 Delay: 547 ms	- 28:10.13.99.212:3721 Delay: 547 ms	- 28:10.13.99.195:3721 Delay: 2344 ms
- 15:10.13.99.212:3708 Delay: 547 ms	- 18:10.13.99.212:3711 Delay: 547 ms	- 15:10.13.99.197:3708 Delay: 2344 ms
- 8:10.13.99.212:3701 Delay: 547 ms	- 21:10.13.99.212:3714 Delay: 687 ms	- 42:10.13.99.197:3735 Delay: 2344 ms
- 20:10.13.99.212:3713 Delay: 703 ms	- 25:10.13.99.212:3718 Delay: 687 ms	- 11:10.13.99.197:3704 Delay: 2344 ms
- 21:10.13.99.212:3714 Delay: 703 ms	- 29:10.13.99.212:3722 Delay: 687 ms	- 48:10.13.99.195:3741 Delay: 2344 ms
- 5:10.13.99.212:3698 Delay: 703 ms	- 5:10.13.99.212:3698 Delay: 687 ms	- 47:10.13.99.197:3740 Delay: 2344 ms
- 49:10.13.99.212:3742 Delay: 703 ms	- 13:10.13.99.212:3706 Delay: 891 ms	- 46:10.13.99.197:3739 Delay: 2484 ms
- 22:10.13.99.212:3715 Delay: 703 ms	- 41:10.13.99.212:3734 Delay: 891 ms	- 19:10.13.99.197:3712 Delay: 2484 ms
- 30:10.13.99.212:3723 Delay: 703 ms	- 45:10.13.99.212:3738 Delay: 891 ms	- 38:10.13.99.197:3731 Delay: 2484 ms

- 31:10.13.99.212:3724 Delay: 703 ms	- 22:10.13.99.212:3715 Delay: 891 ms	- 35:10.13.99.197:3728 Delay: 2484 ms
- 43:10.13.99.212:3736 Delay: 703 ms	- 46:10.13.99.212:3739 Delay: 891 ms	- 3:10.13.99.197:3696 Delay: 2484 ms
- 36:10.13.99.212:3729 Delay: 703 ms	- 14:10.13.99.212:3707 Delay: 891 ms	- 15:10.13.99.195:3708 Delay: 2484 ms
- 39:10.13.99.212:3732 Delay: 703 ms	- 2:10.13.99.212:3695 Delay: 906 ms	- 30:10.13.99.197:3723 Delay: 2484 ms
- 12:10.13.99.212:3705 Delay: 703 ms	- 30:10.13.99.212:3723 Delay: 1109 ms	- 32:10.13.99.197:3725 Delay: 2484 ms
- 35:10.13.99.212:3728 Delay: 703 ms	- 27:10.13.99.212:3720 Delay: 1109 ms	- 10:10.13.99.195:3703 Delay: 2484 ms
- 40:10.13.99.212:3733 Delay: 703 ms	- 42:10.13.99.212:3735 Delay: 1109 ms	- 46:10.13.99.195:3739 Delay: 2484 ms
- 47:10.13.99.212:3740 Delay: 703 ms	- 10:10.13.99.212:3703 Delay: 1109 ms	- 14:10.13.99.195:3707 Delay: 2484 ms
- 28:10.13.99.212:3721 Delay: 703 ms	- 15:10.13.99.212:3708 Delay: 1109 ms	- 2:10.13.99.195:3695 Delay: 2484 ms
- 9:10.13.99.212:3702 Delay: 703 ms	- 11:10.13.99.212:3704 Delay: 1109 ms	- 45:10.13.99.195:3738 Delay: 2484 ms
- 46:10.13.99.212:3739 Delay: 703 ms	- 35:10.13.99.212:3728 Delay: 1109 ms	- 3:10.13.99.195:3696 Delay: 2484 ms
- 2:10.13.99.212:3695 Delay: 703 ms	- 1:10.13.99.212:3694 Delay: 1109 ms	- 5:10.13.99.195:3698 Delay: 2484 ms
- 38:10.13.99.212:3731 Delay: 703 ms	- 8:10.13.99.212:3701 Delay: 1109 ms	- 16:10.13.99.195:3709 Delay: 2484 ms
- 33:10.13.99.212:3726 Delay: 703 ms	- 12:10.13.99.212:3705 Delay: 1109 ms	- 47:10.13.99.195:3740 Delay: 2484 ms
- 44:10.13.99.212:3737 Delay: 719 ms	- 16:10.13.99.212:3709 Delay: 1281 ms	- 34:10.13.99.195:3727 Delay: 2484 ms
- 3:10.13.99.212:3696 Delay: 719 ms	- 4:10.13.99.212:3697 Delay: 1281 ms	- 39:10.13.99.195:3732 Delay: 2500 ms
- 4:10.13.99.212:3697 Delay: 719 ms	- 0:10.13.99.212:3693 Delay: 1297 ms	- 4:10.13.99.195:3697 Delay: 2500 ms
- 7:10.13.99.212:3700 Delay: 719 ms	- 37:10.13.99.212:3730 Delay: 1297 ms	- 1:10.13.99.195:3694 Delay: 2500 ms
- 19:10.13.99.212:3712 Delay: 719 ms	- 33:10.13.99.212:3726 Delay: 1297 ms	- 13:10.13.99.195:3706 Delay: 2500 ms
- 24:10.13.99.212:3717 Delay: 719 ms	- 47:10.13.99.212:3740 Delay: 1297 ms	- 4:10.13.99.197:3697 Delay: 2500 ms
- 27:10.13.99.212:3720 Delay: 719 ms	- 3:10.13.99.212:3696 Delay: 1297 ms	- 33:10.13.99.197:3726 Delay: 2500 ms
- 25:10.13.99.212:3718 Delay: 719 ms	- 43:10.13.99.212:3736 Delay: 1297 ms	- 31:10.13.99.197:3724 Delay: 2500 ms
- 29:10.13.99.212:3722 Delay: 719 ms	- 26:10.13.99.212:3719 Delay: 1297 ms	- 39:10.13.99.197:3732 Delay: 2500 ms
- 17:10.13.99.212:3710 Delay: 719 ms	- 37:10.13.99.197:3730 Delay: 1297 ms	- 43:10.13.99.197:3736 Delay: 2500 ms
- 45:10.13.99.212:3738 Delay: 719 ms	- 20:10.13.99.212:3713 Delay: 1297 ms	- 44:10.13.99.197:3737 Delay: 2734 ms
- 1:10.13.99.212:3694 Delay: 719 ms	- 19:10.13.99.208:3712 Delay: 1297 ms	- 34:10.13.99.197:3727 Delay: 2734 ms
- 41:10.13.99.212:3734 Delay: 859 ms	- 29:10.13.99.195:3722 Delay: 1297 ms	- 10:10.13.99.197:3703 Delay: 2734 ms
	- 49:10.13.99.212:3742 Delay: 1297 ms	- 0:10.13.99.197:3693 Delay: 2734 ms
	- 6:10.13.99.212:3699 Delay: 1297 ms	- 29:10.13.99.208:3722 Delay: 2734 ms
	- 34:10.13.99.212:3727 Delay: 1297 ms	- 22:10.13.99.208:3715 Delay: 2734 ms
	- 23:10.13.99.212:3716 Delay: 1297 ms	- 9:10.13.99.208:3702 Delay: 2734 ms
	- 27:10.13.99.195:3720 Delay: 1297 ms	- 25:10.13.99.208:3718 Delay: 2734 ms
	- 26:10.13.99.195:3719 Delay: 1297 ms	- 24:10.13.99.208:3717 Delay: 2734 ms
	- 23:10.13.99.195:3716 Delay: 1297 ms	- 17:10.13.99.208:3710 Delay: 2734 ms
	- 24:10.13.99.195:3717 Delay: 1297 ms	- 28:10.13.99.208:3721 Delay: 2734 ms
	- 25:10.13.99.195:3718 Delay: 1297 ms	- 26:10.13.99.208:3719 Delay: 2734 ms
	- 21:10.13.99.195:3714 Delay: 1297 ms	- 27:10.13.99.208:3720 Delay: 2734 ms
	- 22:10.13.99.195:3715 Delay: 1297 ms	- 46:10.13.99.208:3739 Delay: 2734 ms
	- 19:10.13.99.195:3712 Delay: 1297 ms	- 16:10.13.99.208:3709 Delay: 2734 ms
	- 17:10.13.99.195:3710 Delay: 1297 ms	- 8:10.13.99.208:3701 Delay: 2734 ms
	- 49:10.13.99.195:3742 Delay: 1297 ms	- 12:10.13.99.208:3705 Delay: 2734 ms
	- 6:10.13.99.195:3699 Delay: 1297 ms	- 48:10.13.99.208:3741 Delay: 2734 ms
	- 7:10.13.99.195:3700 Delay: 1297 ms	- 23:10.13.99.208:3716 Delay: 2734 ms
	- 18:10.13.99.195:3711 Delay: 1875 ms	- 49:10.13.99.208:3742 Delay: 2734 ms
	- 8:10.13.99.195:3701 Delay: 1875 ms	- 18:10.13.99.208:3711 Delay: 2734 ms
	- 20:10.13.99.195:3713 Delay: 1875 ms	- 1:10.13.99.208:3694 Delay: 2734 ms
	- 9:10.13.99.195:3702 Delay: 2094 ms	- 0:10.13.99.208:3693 Delay: 2734 ms

	- 33:10.13.99.195:3726 Delay: 2094 ms	- 10:10.13.99.208:3703 Delay: 2734 ms
	- 35:10.13.99.195:3728 Delay: 2094 ms	- 11:10.13.99.208:3704 Delay: 2734 ms
	- 31:10.13.99.195:3724 Delay: 2094 ms	- 32:10.13.99.208:3725 Delay: 2734 ms
	- 8:10.13.99.197:3701 Delay: 2094 ms	- 35:10.13.99.208:3728 Delay: 2734 ms
	- 6:10.13.99.197:3699 Delay: 2094 ms	- 36:10.13.99.208:3729 Delay: 2859 ms
	- 48:10.13.99.197:3741 Delay: 2094 ms	- 2:10.13.99.208:3695 Delay: 2953 ms
	- 27:10.13.99.197:3720 Delay: 2094 ms	- 43:10.13.99.208:3736 Delay: 2953 ms
	- 24:10.13.99.197:3717 Delay: 2094 ms	- 42:10.13.99.208:3735 Delay: 2953 ms
	- 28:10.13.99.197:3721 Delay: 2094 ms	- 3:10.13.99.208:3696 Delay: 2953 ms
	- 20:10.13.99.197:3713 Delay: 2094 ms	- 47:10.13.99.208:3740 Delay: 2953 ms
	- 2:10.13.99.197:3695 Delay: 2109 ms	- 20:10.13.99.208:3713 Delay: 2969 ms
	- 21:10.13.99.197:3714 Delay: 2109 ms	- 14:10.13.99.208:3707 Delay: 2969 ms
	- 9:10.13.99.197:3702 Delay: 2109 ms	- 38:10.13.99.208:3731 Delay: 2969 ms
	- 16:10.13.99.197:3709 Delay: 2109 ms	- 39:10.13.99.208:3732 Delay: 2969 ms
	- 36:10.13.99.197:3729 Delay: 2109 ms	- 13:10.13.99.208:3706 Delay: 3172 ms
	- 29:10.13.99.197:3722 Delay: 2109 ms	- 33:10.13.99.208:3726 Delay: 3172 ms
	- 17:10.13.99.197:3710 Delay: 2109 ms	- 15:10.13.99.208:3708 Delay: 3187 ms
	- 25:10.13.99.197:3718 Delay: 2109 ms	- 4:10.13.99.208:3697 Delay: 3187 ms
	- 1:10.13.99.197:3694 Delay: 2109 ms	- 44:10.13.99.208:3737 Delay: 3187 ms
	- 22:10.13.99.197:3715 Delay: 2109 ms	- 5:10.13.99.208:3698 Delay: 3187 ms
	- 30:10.13.99.195:3723 Delay: 2109 ms	- 21:10.13.99.208:3714 Delay: 3187 ms
	- 49:10.13.99.197:3742 Delay: 2109 ms	- 34:10.13.99.208:3727 Delay: 3187 ms
	- 37:10.13.99.195:3730 Delay: 2109 ms	- 37:10.13.99.208:3730 Delay: 3391 ms
	- 32:10.13.99.195:3725 Delay: 2109 ms	- 45:10.13.99.208:3738 Delay: 3391 ms
	- 45:10.13.99.197:3738 Delay: 2109 ms	- 31:10.13.99.208:3724 Delay: 3406 ms
	- 13:10.13.99.197:3706 Delay: 2344 ms	- 30:10.13.99.208:3723 Delay: 3406 ms
	- 23:10.13.99.197:3716 Delay: 2344 ms	- 6:10.13.99.208:3699 Delay: 3406 ms
	- 7:10.13.99.197:3700 Delay: 2344 ms	- 40:10.13.99.208:3733 Delay: 3609 ms
	- 40:10.13.99.197:3733 Delay: 2344 ms	- 7:10.13.99.208:3700 Delay: 3609 ms
	- 41:10.13.99.197:3734 Delay: 2344 ms	- 41:10.13.99.208:3734 Delay: 3828 ms
	- 12:10.13.99.197:3705 Delay: 2344 ms	

Mensagens UDP

Cliente 10.13.99.221		
Cliente 10.13.99.221:3742 PING 0	Cliente 10.13.99.221:3742 PING 3 (resultado de uma mensagem)	
- 0:10.13.99.221:3693 Delay: 16 ms	- 30:10.13.99.195:3723 Delay: 15 ms	- 34:10.13.99.195:3727 Delay: 1984 ms
- 29:10.13.99.221:3722 Delay: 31 ms	- 0:10.13.99.221:3693 Delay: 15 ms	- 33:10.13.99.195:3726 Delay: 2000 ms
- 28:10.13.99.221:3721 Delay: 31 ms	- 29:10.13.99.221:3722 Delay: 125 ms	- 0:10.13.99.195:3693 Delay: 2000 ms
- 27:10.13.99.221:3720 Delay: 47 ms	- 28:10.13.99.221:3721 Delay: 125 ms	- 45:10.13.99.195:3738 Delay: 2046 ms
- 26:10.13.99.221:3719 Delay: 63 ms	- 27:10.13.99.221:3720 Delay: 125 ms	- 44:10.13.99.195:3737 Delay: 2062 ms
- 25:10.13.99.221:3718 Delay: 78 ms	- 26:10.13.99.221:3719 Delay: 125 ms	- 1:10.13.99.195:3694 Delay: 2062 ms
- 24:10.13.99.221:3717 Delay: 94 ms	- 25:10.13.99.221:3718 Delay: 140 ms	- 43:10.13.99.195:3736 Delay: 2125 ms
- 23:10.13.99.221:3716 Delay: 110 ms	- 24:10.13.99.221:3717 Delay: 140 ms	- 42:10.13.99.195:3735 Delay: 2140 ms
- 22:10.13.99.221:3715 Delay: 110 ms	- 23:10.13.99.221:3716 Delay: 140 ms	- 46:10.13.99.195:3739 Delay: 2156 ms
- 21:10.13.99.221:3714 Delay: 125 ms	- 22:10.13.99.221:3715 Delay: 140 ms	- 48:10.13.99.195:3741 Delay: 2156 ms
- 20:10.13.99.221:3713 Delay: 141 ms	- 21:10.13.99.221:3714 Delay: 156 ms	- 16:10.13.99.195:3709 Delay: 2156 ms

- 9:10.13.99.221:3702 Delay: 156 ms	- 20:10.13.99.221:3713 Delay: 156 ms	- 5:10.13.99.195:3698 Delay: 2156 ms
- 8:10.13.99.221:3701 Delay: 156 ms	- 9:10.13.99.221:3702 Delay: 156 ms	- 12:10.13.99.195:3705 Delay: 2171 ms
- 19:10.13.99.221:3712 Delay: 172 ms	- 19:10.13.99.221:3712 Delay: 171 ms	- 31:10.13.99.195:3724 Delay: 2171 ms
- 7:10.13.99.221:3700 Delay: 188 ms	- 8:10.13.99.221:3701 Delay: 171 ms	- 17:10.13.99.197:3710 Delay: 2171 ms
- 18:10.13.99.221:3711 Delay: 188 ms	- 7:10.13.99.221:3700 Delay: 171 ms	- 4:10.13.99.195:3697 Delay: 2171 ms
- 6:10.13.99.221:3699 Delay: 203 ms	- 18:10.13.99.221:3711 Delay: 171 ms	- 18:10.13.99.197:3711 Delay: 2187 ms
- 17:10.13.99.221:3710 Delay: 219 ms	- 6:10.13.99.221:3699 Delay: 187 ms	- 6:10.13.99.197:3699 Delay: 2187 ms
- 48:10.13.99.221:3741 Delay: 219 ms	- 17:10.13.99.221:3710 Delay: 187 ms	- 7:10.13.99.197:3700 Delay: 2187 ms
- 5:10.13.99.221:3698 Delay: 235 ms	- 48:10.13.99.221:3741 Delay: 187 ms	- 4:10.13.99.197:3697 Delay: 2187 ms
- 16:10.13.99.221:3709 Delay: 250 ms	- 5:10.13.99.221:3698 Delay: 203 ms	- 27:10.13.99.197:3720 Delay: 2203 ms
- 47:10.13.99.221:3740 Delay: 266 ms	- 16:10.13.99.221:3709 Delay: 203 ms	- 5:10.13.99.197:3698 Delay: 2203 ms
- 4:10.13.99.221:3697 Delay: 266 ms	- 47:10.13.99.221:3740 Delay: 203 ms	- 49:10.13.99.197:3742 Delay: 2203 ms
- 15:10.13.99.221:3708 Delay: 297 ms	- 4:10.13.99.221:3697 Delay: 203 ms	- 26:10.13.99.197:3719 Delay: 2218 ms
- 46:10.13.99.221:3739 Delay: 313 ms	- 15:10.13.99.221:3708 Delay: 218 ms	- 21:10.13.99.197:3714 Delay: 2218 ms
- 3:10.13.99.221:3696 Delay: 328 ms	- 3:10.13.99.221:3696 Delay: 218 ms	- 24:10.13.99.195:3717 Delay: 2218 ms
- 14:10.13.99.221:3707 Delay: 328 ms	- 46:10.13.99.221:3739 Delay: 218 ms	- 15:10.13.99.197:3708 Delay: 2218 ms
- 45:10.13.99.221:3738 Delay: 344 ms	- 14:10.13.99.221:3707 Delay: 218 ms	- 27:10.13.99.195:3720 Delay: 2234 ms
- 2:10.13.99.221:3695 Delay: 360 ms	- 45:10.13.99.221:3738 Delay: 234 ms	- 30:10.13.99.197:3723 Delay: 2234 ms
- 13:10.13.99.221:3706 Delay: 360 ms	- 2:10.13.99.221:3695 Delay: 234 ms	- 31:10.13.99.197:3724 Delay: 2234 ms
- 44:10.13.99.221:3737 Delay: 375 ms	- 13:10.13.99.221:3706 Delay: 234 ms	- 32:10.13.99.197:3725 Delay: 2250 ms
- 12:10.13.99.221:3705 Delay: 391 ms	- 44:10.13.99.221:3737 Delay: 250 ms	- 35:10.13.99.197:3728 Delay: 2250 ms
- 1:10.13.99.221:3694 Delay: 391 ms	- 12:10.13.99.221:3705 Delay: 250 ms	- 16:10.13.99.197:3709 Delay: 2265 ms
- 43:10.13.99.221:3736 Delay: 391 ms	- 1:10.13.99.221:3694 Delay: 250 ms	- 34:10.13.99.197:3727 Delay: 2265 ms
- 11:10.13.99.221:3704 Delay: 406 ms	- 11:10.13.99.221:3704 Delay: 265 ms	- 36:10.13.99.197:3729 Delay: 2281 ms
- 10:10.13.99.221:3703 Delay: 406 ms	- 42:10.13.99.221:3735 Delay: 265 ms	- 11:10.13.99.197:3704 Delay: 2281 ms
- 42:10.13.99.221:3735 Delay: 406 ms	- 43:10.13.99.221:3736 Delay: 265 ms	- 39:10.13.99.197:3732 Delay: 2296 ms
- 41:10.13.99.221:3734 Delay: 422 ms	- 10:10.13.99.221:3703 Delay: 265 ms	- 38:10.13.99.197:3731 Delay: 2296 ms
- 39:10.13.99.221:3732 Delay: 422 ms	- 40:10.13.99.221:3733 Delay: 281 ms	- 40:10.13.99.197:3733 Delay: 2328 ms
- 40:10.13.99.221:3733 Delay: 422 ms	- 39:10.13.99.221:3732 Delay: 281 ms	- 10:10.13.99.197:3703 Delay: 2328 ms
- 38:10.13.99.221:3731 Delay: 438 ms	- 38:10.13.99.221:3731 Delay: 296 ms	- 49:10.13.99.195:3742 Delay: 2328 ms
- 37:10.13.99.221:3730 Delay: 438 ms	- 37:10.13.99.221:3730 Delay: 296 ms	- 3:10.13.99.197:3696 Delay: 2359 ms
- 36:10.13.99.221:3729 Delay: 438 ms	- 36:10.13.99.221:3729 Delay: 296 ms	- 0:10.13.99.197:3693 Delay: 2359 ms
- 35:10.13.99.221:3728 Delay: 453 ms	- 35:10.13.99.221:3728 Delay: 312 ms	- 33:10.13.99.197:3726 Delay: 2375 ms
- 33:10.13.99.221:3726 Delay: 453 ms	- 34:10.13.99.221:3727 Delay: 312 ms	- 18:10.13.99.195:3711 Delay: 2406 ms
- 34:10.13.99.221:3727 Delay: 469 ms	- 33:10.13.99.221:3726 Delay: 312 ms	- 19:10.13.99.195:3712 Delay: 2406 ms
- 32:10.13.99.221:3725 Delay: 469 ms	- 32:10.13.99.221:3725 Delay: 312 ms	- 46:10.13.99.197:3739 Delay: 2421 ms
- 30:10.13.99.221:3723 Delay: 469 ms	- 31:10.13.99.221:3724 Delay: 328 ms	- 14:10.13.99.197:3707 Delay: 2421 ms
- 31:10.13.99.221:3724 Delay: 485 ms	- 30:10.13.99.221:3723 Delay: 500 ms	- 8:10.13.99.195:3701 Delay: 2421 ms
	- 41:10.13.99.221:3734 Delay: 500 ms	- 9:10.13.99.195:3702 Delay: 2437 ms
	- 47:10.13.99.197:3740 Delay: 515 ms	- 48:10.13.99.197:3741 Delay: 2437 ms
	- 9:10.13.99.197:3702 Delay: 718 ms	- 12:10.13.99.197:3705 Delay: 2453 ms
	- 24:10.13.99.197:3717 Delay: 718 ms	- 2:10.13.99.197:3695 Delay: 2453 ms
	- 8:10.13.99.197:3701 Delay: 718 ms	- 44:10.13.99.197:3737 Delay: 2453 ms
	- 29:10.13.99.195:3722 Delay: 734 ms	- 13:10.13.99.197:3706 Delay: 2468 ms
	- 11:10.13.99.195:3704 Delay: 734 ms	- 3:10.13.99.195:3696 Delay: 2484 ms
	- 2:10.13.99.195:3695 Delay: 734 ms	- 38:10.13.99.195:3731 Delay: 2546 ms
	- 25:10.13.99.195:3718 Delay: 921 ms	- 37:10.13.99.195:3730 Delay: 2546 ms

	- 20:10.13.99.197:3713 Delay: 937 ms **- 17:10.13.99.195:3710 Delay: 937 ms** **- 7:10.13.99.195:3700 Delay: 1218 ms** **- 28:10.13.99.197:3721 Delay: 1218 ms** **- 26:10.13.99.195:3719 Delay: 1265 ms** **- 23:10.13.99.195:3716 Delay: 1328 ms** **- 20:10.13.99.195:3713 Delay: 1328 ms** **- 21:10.13.99.195:3714 Delay: 1343 ms** **- 14:10.13.99.195:3707 Delay: 1656 ms** **- 14:10.13.99.208:3707 Delay: 1656 ms** **- 28:10.13.99.195:3721 Delay: 1671 ms** **- 22:10.13.99.195:3715 Delay: 1687 ms** **- 41:10.13.99.195:3734 Delay: 1687 ms** **- 39:10.13.99.195:3732 Delay: 1703 ms** **- 15:10.13.99.195:3708 Delay: 1703 ms** **- 40:10.13.99.195:3733 Delay: 1984 ms** **- 36:10.13.99.195:3729 Delay: 1984 ms**	**- 35:10.13.99.195:3728 Delay: 2546 ms** **- 13:10.13.99.195:3706 Delay: 2562 ms** **- 10:10.13.99.195:3703 Delay: 2578 ms** **- 47:10.13.99.195:3740 Delay: 2578 ms** **- 32:10.13.99.195:3725 Delay: 2578 ms** **- 29:10.13.99.197:3722 Delay: 2593 ms** **- 41:10.13.99.197:3734 Delay: 2593 ms** **- 42:10.13.99.197:3735 Delay: 2593 ms** **- 43:10.13.99.197:3736 Delay: 2609 ms** **- 37:10.13.99.197:3730 Delay: 2609 ms** **- 45:10.13.99.197:3738 Delay: 2625 ms** **- 6:10.13.99.195:3699 Delay: 2625 ms** **- 1:10.13.99.197:3694 Delay: 2625 ms**

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo