Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Computação
Programa de Pós-Graduação em Ciência da Computação

Uma Solução Peer-to-Peer para a Implementação de Jogos Multiusuário Baseada no Padrão Emergente MPEG-4 MU

Marcelo Martins Laffranchi

São Carlos - SP
Agosto – 2003

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

L163sp

Laffranchi, Marcelo Martins.
Uma solução peer-to-peer para a implantação de jogos multiusuário baseada no padrão emergente MPEG-4 MU / Marcelo Martins Laffranchi. -- São Carlos : UFSCar, 2004.
108 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2004.

1. Análise e projeto de sistemas. 2. Peer-to-peer. 3. MPEG-4. 4. MPEG-4 MU. 5. Ambientes virtuais multiusuário.I. Título.

CDD: 004.21 (20ª)

"Jamais considere seus estudos como uma obrigação, mas como uma oportunidade invejável para aprender a conhecer a influência libertadora da beleza do reino do espírito, para seu próprio prazer pessoal e para proveito da comunidade à qual seu futuro trabalho pertencer."

Albert Einstein

Dedico este trabalho:

A Deus, que soube me orientar nos momentos mais difíceis da minha vida.

Aos meus queridos pais, José Francisco e Maria Silvia, pelos esforços contínuos e por terem sempre me apoiado.

A minha noiva, Cristiane, que sempre esteve ao meu lado me apoiando e que sempre estará.

Aos meus irmãos, Sergio e Murilo, que também sempre estiveram juntos, presenciando este momento importante em minha vida.

Aos meus avós, Afonso e Angelina, Rudá e Eunice (in memorian) a quem sempre estimei.

AGRADECIMENTOS

À minha orientadora, Profa. Dra. Regina Borges de Araujo, pela dedicação e paciência durante todo o desenvolvimento deste trabalho.

Aos meus amigos, Will, Fernando, Takeda, Alexandre e Goiano e a todos os que me apoiaram na trajetória desse trabalho.

Aos alunos de graduação, Rafael Rabelo, Raphael Cortez e Rodrigo Gimenez Ribeiro que sempre me apoiaram e colaboraram no desenvolvimento deste trabalho.

Ao grupo LRVNet, Simone, Christmas, Freire, Urso, Matheus, Taciana, Gislaine, Goiano, Fernando e o Richard.

A todos os funcionários do departamento de computação, Dona Vera, Dona Ofélia, Cristina Trevelin e Miriam, pelo carinho obtido e paciência.

Ao pessoal da lista de discussão do *Limewire*, *Greg Bildson*, *Sam Berlim*, *Philippe Verdy* e a todos que com muita paciência e dedicação me auxiliaram neste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

API	Application Programming Interface
3D	Tridimensional
AV3DM	Ambiente Virtual 3D Multiusuário
BIFS	Binary Format for Scene
CPU	Central Processor Unit
DAI	DMIF Application Interface
DMIF	Delivery Multimedia Integration Framework
DNS	Domain Name System
DPI	DMIF Plug in Interface
ESD	Elementary Stream Descriptor
gNet	Gnutella Networks
GWebCaches	Gnutella Web Caches
http	Hyper Text Transfer Protocol
ICQ	"I seek you"
ID	Identificator
IP	Internet Protocol
IPv4	IP versão 4
J2SE	Java 2 Standard Edition
JXTA	"Justapose"
LAN	Local Area Network
LRVnet	Laboratório de Realidade Virtual em Rede
MAI	Multiuser Application Interface
MBK	Mutech Bookkeeper
MCM	Mutech Channel Manager
MMH	MUTech Message Handler
MPEG	Moving Picture Experts Group
MPEG-4 MU	MPEG-4 Multiuser
MPEG-J	MPEG-4 Java
MSC	Mutech Session Controller
MU	Multiusuário
MUD	Multiuser Dungeons
NPC	Non Personal Character
NPM	Non Persistent Multiplayer
OCI	Object Content Information
OD	Object Descriptor
P2P	Peer-to-peer
PC	Personal Computer
PC	Personal Character
PDA	Personal Digital Assitant
PM	Persistent Multiplayer
QoS	Quality of Service
RPG	Role Playing Games
SDM	System Decoder Model
SETI	Search for Extraterrestrial Intelligence
SL	Sync Layer
SO	Sistema Operacional
SPMD	Single Process Multiple Data
TCP	Transmission Control Protocol

TLS	Transport Layer Security
TTL	Time To Live
VRML	Virtual Reality Modeling Language
WAN	Wide Área Network
WG11	Work Group 11
WWW	World Wide Web
XML	Extensible Markup Language

Resumo

Este trabalho descreve a implementação de uma estrutura de suporte a jogos virtuais 3D em rede, baseada no padrão emergente MPEG-4 multiusuário em uma rede *Gnutella peer-to-peer* híbrida. Esta solução minimiza as desvantagens das soluções híbridas existentes, que são baseadas em *proxies*, as quais têm que ser re-configuradas sempre que uma nova aplicação surge na rede. Para isso, o código que implementa a rede *Gnutella* foi modificado de modo a incluir um serviço de busca de jogos e de sessões ativas. Dois componentes definidos e especificados pelo padrão emergente MPEG-4 MU foram implementados e integrados à rede *Gnutella* para controle de sessão de jogos e atualização das cenas. Quando um nodo designado como controlador sai, outro deve assumir de forma rápida e contínua. Esses, entre outros desafios na implementação de jogos multiusuário como aplicações *peer-to-peer*, serão discutidos neste trabalho, juntamente com a integração das tecnologias *Gnutella* e MPEG-4 MU. A avaliação desta implementação nos permitiu chegar a algumas conclusões sobre a adequação dessas redes no suporte a aplicações que exigem colaboração contínua, como é o caso de um jogo 3D em que múltiplos participantes alteram a cena constantemente e também a viabilidade de se implementar um controlador de sessão em um dos nodos da rede.

Abstract

This work describes the implementation of a support structure to 3D virtual networked games, based on the emergent standard multiuser MPEG-4 in a Gnutella hybrid peer-to-peer network. This solution minimizes the disadvantages of the existent hybrid solutions, that they are based on proxies, which have to be re-configured whenever a new application appears in the net. For that, the code that implements the Gnutella network it was modified from way to include a service of search of games and of active sessions. Two defined components and specified by the emergent standard MPEG-4 MU were implemented and integrated into the Gnutella network for games session control and updating of the scenes. When a node designated as controller leaves, another should assume in a fast and continuous way. Those, among other challenges in the implementation of multiuser games as peer-to-peer applications, they will be discussed in this work, together with the integration of the technologies Gnutella and MPEG-4 MU. The evaluation of this implementation allowed to conclude the some topics about the adaptation of those networks in the support to applications that demand continuous collaboration, as it is the case of a 3D game in that multiples participant constantly alter the scene and also the viability of implementing a session controller in one of the nodes of the network.

1. Introdução

A integração da tecnologia de computação com a de comunicações deve constituir a base para as novas aplicações que estão surgindo. Dentro de algum tempo, qualquer indivíduo poderá ter acesso a essas aplicações através da comunicação sem fio e com fio, utilizando dispositivos como PC's e portáteis que vão dos PDA's aos telefones celulares de terceira geração.

Novas tecnologias de comunicação criarão um "canal" para prover acesso aos serviços e aplicações baseados na Internet.

Ocorrerá, também, o crescimento dos serviços e aplicações distribuídas, em que todos compartilham informações e estados através da rede. Tais tipos de aplicações são conhecidos como *peer-to-peer* **[HER03].**

A tecnologia *peer-to-peer* não é tão nova. O crescimento no número de usuários com conexões de alta velocidade vem fazendo do *peer-to-peer* um dos tópicos mais discutidos no setor da tecnologia da informação **[SZO02]**.

Por sua vez, os jogos multiusuário estão emergindo como uma forma de aplicação extremamente popular. Para esse tipo de aplicação, o modelo de comunicação cliente-servidor é o mais utilizado comercialmente, tanto para jogos persistentes com milhares de usuários, como é o caso dos *Massively Multiplayer Online Role Playing games* (dentre eles, *ExerQuest, Ultima Online, e Asheron´s Call*), quanto para jogos com número de usuários limitados (dentre eles, *Quake*). Além de oferecer gerenciamento mais simples e proteção contra fraudes, o modelo cliente-servidor oferece, ainda, uma forma mais simples de cobrança. Entretanto, o modelo centrado em um servidor está sujeito a sofrer gargalos,

aumentando, assim, o atraso no acesso ao servidor, podendo chegar à negação de acesso aos usuários participantes. Em contrapartida, modelo de comunicação *peer-to-*

peer possibilita que dois ou mais nodos, em uma mesma rede colaborem entre si, de igual-para-igual, sem a necessidade de uma coordenação central. Desta forma, problemas em um nodo não comprometem a aplicação, que continua a rodar mesmo sem o nodo comprometido. Mais ainda, as soluções *peer-to-peer* reduzem o problema de congestionamento inerente ao modelo cliente-servidor, uma vez que não há um ponto central no sistema, reduzindo assim a latência da rede e, conseqüentemente, diminuindo o tempo de resposta do usuário participante. Entretanto, são soluções distribuídas e, portanto, mais complexas para gerenciar. Exemplos de implementação de jogos em rede como aplicações *peer-to-peer* incluem *MIMAZE* **[DIOT99]**, *MASSIVE* **[MASS95]** e *DIVE* **[DIV96]**.

Soluções híbridas combinam as vantagens dos modelos distribuídos e centralizados. *Mauve* e colegas **[MAU02]** descrevem uma solução híbrida baseada em *proxies* que são utilizados como extensão de um servidor central, possuindo algumas das funcionalidades desse servidor e localizados mais próximos das máquinas dos jogadores. Uma das maiores dificuldades com esta solução é que os *proxies* precisam ser configurados toda vez que surge um novo jogo. Assim, os *proxies* têm que ser capazes de suportar várias arquiteturas de jogos diferentes ao mesmo tempo.

O que se pretende nesta pesquisa é descrever a implementação e avaliação de uma solução de suporte a jogos multiusuário em uma rede *peer-to-peer* híbrida. Esta solução minimiza a desvantagem das soluções híbridas baseadas em *proxies* que têm que ser re-configurados sempre que uma nova aplicação é disponibilizada na rede. Na solução implementada neste trabalho, novos jogos são disponibilizados espontaneamente por diferentes nodos da rede. A rede aqui considerada é a rede *Gnutella* **[GNU03]**, acessada através do software cliente *Limewire* **[LIME03]**. Sua escolha como solução para compartilhamento de jogos multiusuário, se justifica pelos seguintes motivos: a sua natureza aberta e descentralizada, o que a torna mais resistente a falhas de software e ataques maliciosos; simplicidade do protocolo básico, que facilita o seu uso em experimentos diversos; e a existência de uma comunidade de desenvolvimento muito ativa. A versão utilizada neste estudo contém nodos especiais, denominados *Ultrapeers*, com maior potencial de processamento e de largura de banda. Esses nodos são utilizados como pontos de controle de sessões de jogos e o software cliente *Limewire*, que suportaria apenas a busca de arquivos de áudio e vídeo, se estende para acomodar a busca de jogos pela rede. Implementou-se também uma extensão dos componentes de controle de sessão e atualização de cenas,

definidos pelo MPEG-4 multiusuário, um padrão emergente de suporte MPEG-4 para múltiplos usuários em ambientes virtuais. O MPEG-4 é um padrão da ISO/IEC para codificar e entregar diferentes formatos de mídia em uma grande variedade de redes e plataformas computacionais e os resultados preliminares mostram a viabilidade da implementação de jogos multiusuário em ambientes de rede *peer-to-peer* híbridas.

Baseado então nessas tecnologias, este trabalho contribuirá com novos padrões e, assim, disseminará as aplicações de AV3DMs em dispositivos conectados à rede sem fio.

Sua estrutura se constitui primeiramente por uma revisão sobre a arquitetura *peer-to-peer* que envolve, entre outros itens, a rede *Gnutella*, utilizada nesse projeto e uma revisão sobre jogos multiusuário, como aplicações *peer-to-peer*. Incluiu-se também, uma revisão sobre o padrão MPEG-4 e seu emergente MPEG-4 MU e, em seguida, a solução *peer-to-peer* e a integração das tecnologias utilizadas foram discutidas, terminando com uma análise dos resultados obtidos e a conclusão.

1.1. Objetivos

Alguns dos os objetivos deste trabalho incluem:

- Implementar uma estrutura de suporte a jogos virtuais 3D em rede, baseada no padrão emergente MPEG-4 MU em uma rede *Gnutella*;
- Contribuir com o processo de padronização do modelo de comunicação *peer-to-peer* para o Padrão emergente MPEG-4;
- Integrar o padrão emergente MPEG-4 MU na rede *peer-to-peer Gnutella*;
- Avaliar a latência obtida para a atualização de dados e transferência de controle em ambientes de jogos multiusuário;
- Colaborar com a comunidade *Gnutella*, através das alterações aplicadas ao *Limewire* o software cliente *Gnutella* utilizado nesse projeto;
- Abrir o espaço para a implementação de novas aplicações nas mais diversas áreas, como, por exemplo, educação, saúde e entretenimento, além de aumentar a escala de acesso a estas aplicações.

1.2. Motivação

O surgimento das novas tecnologias, de padrões que suportam a codificação, transmissão e apresentação de aplicações multimídia complexas, e o uso da tecnologia *peer-to-peer* têm motivado, no mundo todo, a exploração de novas aplicações e novos serviços.

Assim sendo, a possibilidade da integração do Padrão MPEG-4 MU com uma rede *peer-to-peer*, para aplicações multiusuário e a também, dar continuidade a um trabalho de mestrado na implementação e especificação do modelo *peer-to-peer* do padrão emergente MPEG-4 MU, foram aspectos incentivadores para essa pesquisa. A utilização dessa rede para os jogos multiusuário, também se apresentou como um grande desafio, devido a aspectos de segurança, manutenção de consistência, entre outros.

2. O Modelo de Comunicação *Peer-to-Peer*

O *Peer-to-peer* (P2P) é um modelo de comunicação que possibilita dois ou mais pares a colaborarem, espontaneamente, em uma rede de pares (máquinas consideradas em nível de igualdade), usando informações apropriadas e um sistema de comunicação sem a necessidade de uma coordenação central. O *Peer-to-peer* tem o potencial de acelerar os processos de comunicação, explorar recursos inativos (ociosos) e facilitar a troca de informações recentemente criadas e distribuídas **[SCHO03]**.

Cada computador participante, em um modelo de comunicação P2P, é chamado de par (*peer*), indicando que os participantes interagem de igual-para-igual. Esses pares podem apresentar algumas regras como, por exemplo, quando acessam informações, eles são clientes; quando servem informações a outros clientes, eles são servidores; e quando encaminham informações para outros, eles são roteadores **[KUB03]**.

O modelo de comunicação *peer-to-peer* pode ser classificado de duas formas: o descentralizado e o híbrido. À primeira vista, tende-se a imaginar o modelo P2P como aquele em que os pares participantes entram em contato direto um com o outro, sem a participação de nenhum intermediário. Porém, de acordo com *Zai* e colegas **[ZAI03],** isso não é sempre verdade, e algumas das aplicações P2P mais famosas são classificadas como *peer-to-peer* híbridas. Vejamos a seguir, mais detalhadamente as duas definições.

O modelo *peer-to-peer* descentralizado, também chamado de P2P puro, de acordo com **[SCHOL02]**, trabalha sem a ajuda de nenhum tipo de servidor ou repositório central de informações. Todos os pares são auto-suficientes e podem assumir, simultaneamente, as funções de cliente e servidor, como pode ser visto na figura 1 **[ZAI03]**.

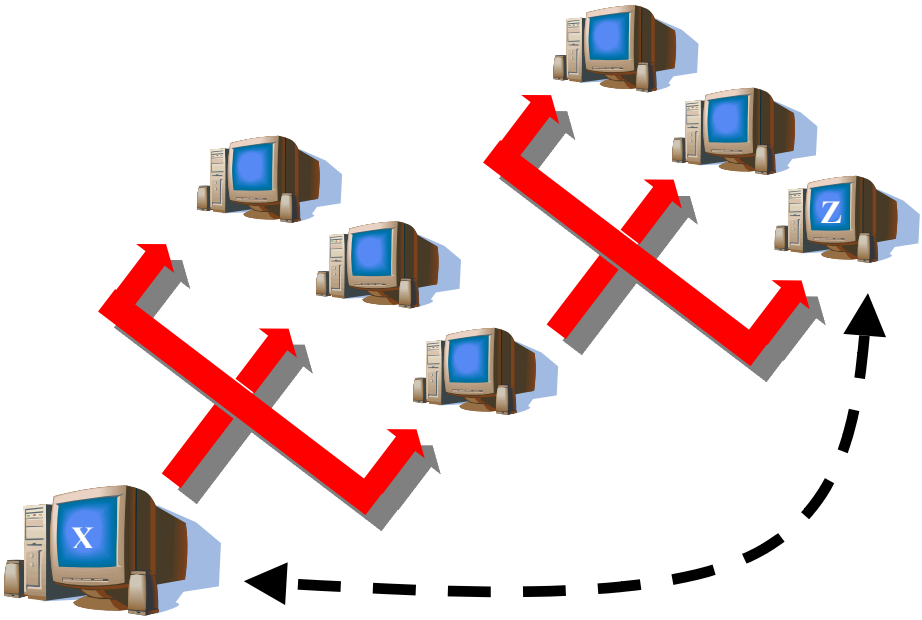


Figura 1- Modelo Descentralizado

Neste modelo, os pares efetuam solicitações aos pares vizinhos que estão ao seu alcance, no caso, "X" sai procurando nas máquinas com as quais tem uma conexão. Por sua vez, estes se conectam a outros pares, até que o recurso seja localizado, no caso, em "Z". Quando isto acontece, é passado um apontador para o par inicial, que estabelece uma conexão direta com o par onde o recurso foi encontrado, podendo assim fazer a transação sem intermediários.

O *peer-to-peer* descentralizado, devido a sua complexidade, é raramente usado conforme a definição acima. É muito mais freqüente a implementação de servidores para tarefas específicas e com pouco *overhead*, o que se trata do *peer-to-peer* híbrido **[ZAI03]**.

No modelo híbrido, são usados servidores para tarefas como autenticação de usuários, serviços de diretório e mapeamento de recursos disponíveis. A idéia básica é que os pares possam contatar algum servidor para iniciar a transação, ou para alguns dos serviços disponíveis na rede. O modelo híbrido é mostrado na figura 2.

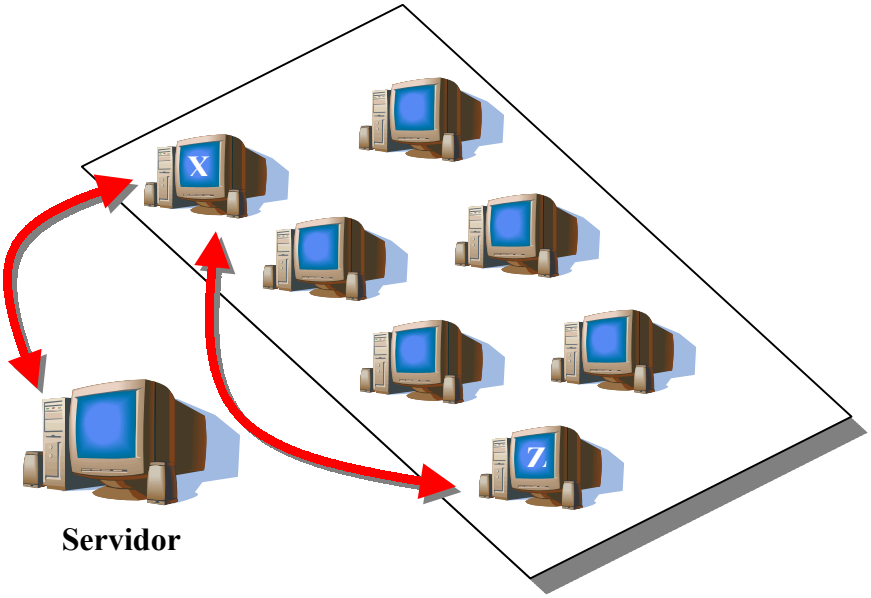


Figura 2 - Modelo Híbrido

Como mostra a figura 2, "X" contata o servidor para saber quem tem um determinado arquivo ou informação. Em seguida, o servidor devolve ao par inicial alguma informação pertinente que permita a ele conectar-se diretamente com outro par, "Z" e assim efetuar a transação **[ZAI03]**.

O modelo híbrido é usado na grande maioria dos sistemas P2P. No caso do *Napster*, o primeiro sistema de compartilhamento de arquivos que causou furor entre usuários do mundo todo e grandes dores de cabeça à indústria fonográfica, por exemplo, um servidor central armazena todos as contas de usuários e as listas de arquivos compartilhados. Ao efetuar-se a busca por um arquivo, o servidor central consulta a sua base e aponta um par que possui este arquivo. Daí em diante, a comunicação e transferência é processada diretamente, par a par **[ZAI03]**.

Existem hoje várias aplicações P2P, as quais podem ser divididas em grupos de acordo com suas funções **[CIG02]**. Serão vistos a seguir os diferentes tipos existentes, suas características e tecnologias.

2.1. Tipos de Aplicações Peer-to-Peer existentes

Milojicic e colegas **[MIL02]** descrevem quatro grandes tipos de aplicações *peer-to-peer*, que são:

- Computação Distribuída
- Compartilhamento de Arquivos
- Colaboração *peer-to-peer*
- Plataformas *peer-to-peer*

Esses tipos de aplicação *peer-to-peer* serão descritos a seguir , incluindo um estudo de caso de cada tipo de aplicação.

2.1.1. Computação Distribuída

Computação em *Grid* é uma coleção de recursos heterogêneos e distribuídos, possibilitando que sejam utilizados em grupo para executar aplicações de larga escala. Um software servidor divide as tarefas em pequenas partes as quais são divididas entre os PCs que estão conectados ao sistema. Os PCs executam as tarefas associadas apenas quando estão ociosos e, depois de prontas, enviam-nas de volta ao servidor. Podem ser chamados de sistemas orientados a pares (*peers*).

Uma questão comumente apresentada é se os sistemas de computação distribuída, no caso *Grid*, são realmente *peer-to-peer*. O argumento que contesta essa afirmação é que um servidor central é necessário para controlar os recursos, os pares não operam como servidores e não ocorre comunicação entre os pares. O que ocorre é que uma parte significante do sistema é executada nos pares, com alto anonimato. *Milojicic* e colegas, assim como **[CIG02]** consideram as aplicações distribuídas como sendo aplicações *peer-to-peer*.

Uma aplicação em *Grid* trabalha da seguinte forma: o problema a ser resolvido é dividido em pequenas partes independentes. O processamento de cada uma dessas partes é

feito num PC ou nodo de forma individual e os resultados são coletados num servidor central. Esse servidor central é responsável pela distribuição de partes de trabalho para os PCs espalhados pela Internet. Cada um dos PCs registrados é equipado com um software cliente. O software cliente tira vantagem dos períodos ociosos de um PC para executar algum processamento solicitado pelo servidor. Depois que o processamento termina, o resultado é enviado de volta ao servidor e um novo trabalho é passado ao cliente.

Uma das maiores limitações da computação distribuída é que ela requer trabalhos que possam ser divididos em pequenas partes independentes e que não necessitem de comunicação entre pares. As aplicações atuais consistem de aplicações *Single Process Multiple Data* , ou seja, problemas em que um dado trabalho possa ser executado em muitos conjuntos de entrada de dados diferentes. Isso inclui, na maioria das vezes, aplicações como simulações. Dentre as várias aplicações existentes, pode ser citado o Projeto do Genoma Humano **[GEN01]**, Pesquisas sobre o Câncer **[UD03]**, SETI@Home **[SETI03]**, entre outras, citadas em **[MIL02]**.

O projeto SETI é uma das aplicações mais conhecidas. É uma coleção de projetos de pesquisas científicas voltados para a descoberta de civilizações extraterrestres. O SETI@home analisa emissões de rádio recebidas do espaço e coletadas pelo gigante telescópio *Arecibo*, usando o poder de processamento de milhões de PCs ociosos na Internet. A Arquitetura pode ser vista na figura 3 **[AND02]**.

O projeto usa dois componentes principais: o servidor de dados e o software cliente. O cliente funciona como um protetor de tela para diversas plataformas tais como *Windows*, *Macintosh*, *Linux*, *Solaris*, e HP-UX. Esse cliente processa as tarefas enviadas pelo servidor e depois as devolve processadas.

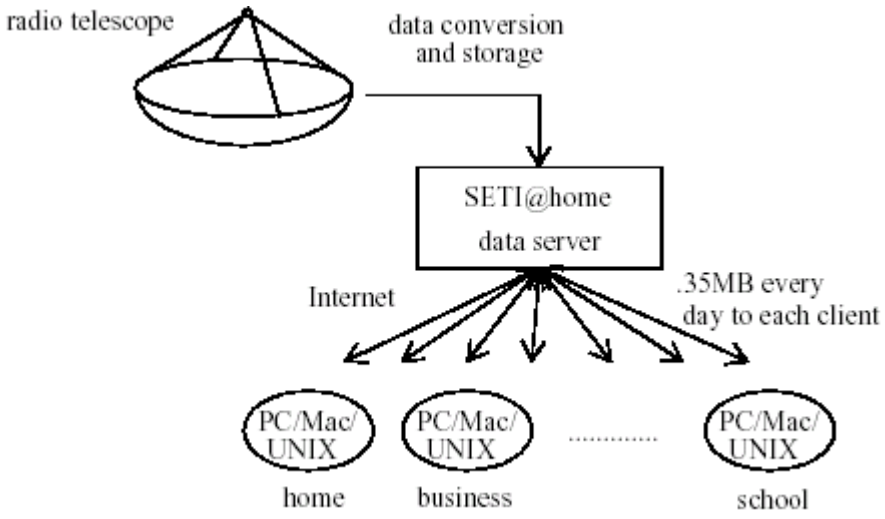


Figura 3 - Arquitetura SETI@home

O SETI apresenta como principais características:

- escalabilidade: o projeto SETI conta com seus servidores para distribuir as tarefas para cada par participante e depois recuperar os resultados usando diversas conexões, permitindo, assim, o trabalho e a conexão de milhões de computadores;

- anonimato: os usuários não ficam no anonimato, pois recebem créditos pelas tarefas que eles fazem. No *site* do projeto, são encontradas informações de usuários, como por exemplo, quais os que mais contribuíram com o projeto;

- baixo custo: tem como principal responsável o processamento distribuído;

- baixa conectividade: os usuários apenas precisam estar conectados para enviar e receber tarefas;

- segurança: o SETI oferece dois tipos de segurança: a proteção do usuário contra o sistema e a proteção do sistema contra os usuários. Os protocolos são protegidos e impedidos de executar qualquer programa e as conexões são feitas apenas para enviar e receber dados, evitando,

assim, problemas devido a conexão;

- transparência: o SETI pode ficar em execução sem que o usuário perceba que seu computador está sendo usado;

- tolerância a falhas: os servidores aguardam até três respostas iguais da tarefa que enviou, para não perder nenhum dos dados processados.

Essas e outras características podem ser encontradas e descritas em **[MIL02]**.

2.1.2. Compartilhamento de Arquivos

O compartilhamento de arquivos é uma aplicação típica P2P. A idéia é direta: substituir o disco rígido local de um computador por pontos de expansão de armazenamento ao longo da Internet. Assim, um computador interage com vários pares da rede para obter ou doar informações. *Napster* e *Gnutella* são as aplicações que têm sido muito usadas pelos usuários da Internet para evitar limitações de largura de banda que transformam a transferência de grandes arquivos em algo inaceitável **[MIL02]**.

As principais questões relacionadas a aplicações de compartilhamento de arquivos envolvem consumo de largura de banda, segurança e capacidades de busca. De acordo com **[MIL02]** existem hoje três principais modelos que são: modelo centralizado, como é o caso do *Napster*, o modelo de busca por *flooding*, como é o caso do *Gnutella*, e o modelo de roteamento de documento, como é o caso do *FreeNet*. Todas as aplicações de compartilhamento de arquivos podem ser classificadas dentro de um desses três modelos citados, embora existam variações, como por exemplo o *KazaA*, o qual permite uma melhor escalabilidade de sua aplicação e menos estresse na rede através de um componente, o *Supernode*. Veremos então o funcionamento desses três modelos.

Napster

De acordo com **[OPEN]**, *Napster* é um protocolo para compartilhamento de arquivos entre os usuários. Com o *Napster*, os arquivos ficam na máquina do cliente, nunca passando por um servidor. O servidor oferece apenas a habilidade de procurar arquivos em particular e iniciar a transferência direta entre os clientes. Além do mais, *chats* estão

disponíveis nos clientes *Napster*, além de um decodificador de vídeo e áudio.

No *Napster,* todos os participantes têm a mesma capacidade de comunicação e cada participante pode iniciar uma sessão de comunicação. O *Napster* usa o modelo centralizado, pois o sistema central guarda os diretórios dos arquivos compartilhados pelos usuários. O servidor recebe as consultas à sua base de dados e retorna os dados que permitam ao usuário fazer a conexão direta (geralmente usando HTTP) com o usuário que está compartilhando o arquivo. A figura 4 mostra o funcionamento da rede *Napster*.

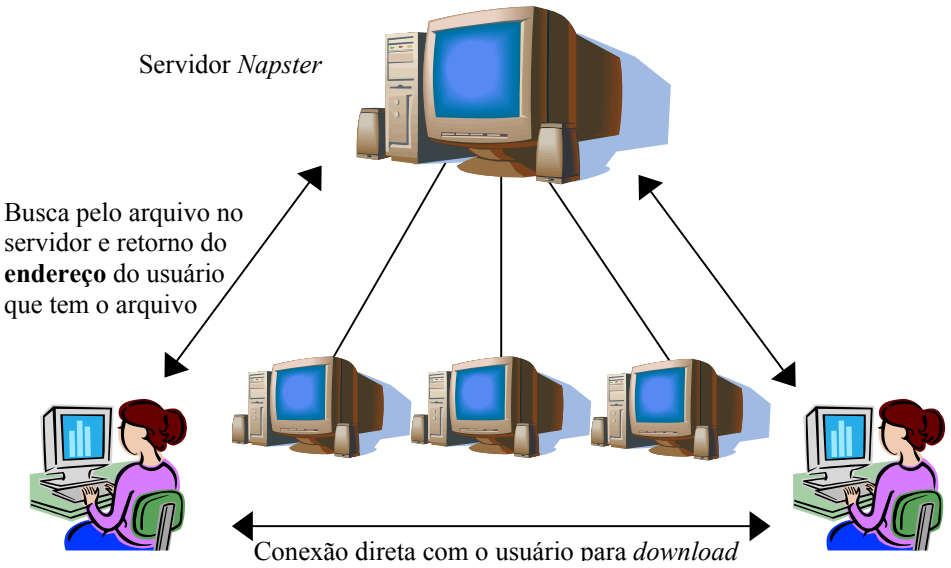


Figura 4 - Rede *Napster*

O modelo *peer-to-peer* usado pelo *Napster* é baseado no uso de um servidor central que direciona o trafego das informações entre usuários registrados. O servidor central mantém diretórios com informação dos arquivos dos usuários da aplicação.

Esses diretórios são atualizados sempre que um usuário se conecta ao servidor *Napster*. Cada vez que o usuário faz um pedido, ou procura por algum arquivo em particular, o servidor central cria uma lista de arquivos referentes ao arquivo buscado pelo usuário, comparando o pedido com o banco de dados dos arquivos que todos os usuários que estão conectados naquele momento possuem. O *download* do arquivo é feito diretamente de usuário para usuário (*peer-to-peer*).

Uma vantagem do modelo usado pelo *Napster* é a maneira rápida e eficiente com que o servidor acha os arquivos pedidos pelo usuário. Como a base do servidor é

constantemente atualizada, arquivos recém encontrados pelos usuários são imediatamente disponibilizados para o *download* de outros. Outra vantagem é o fato de todos os usuários serem obrigatoriamente registrados no servidor. Assim, a lista da resposta dos pedidos contém sempre usuários que estão conectados no momento da busca.

Porém, como uma de suas desvantagens, enquanto um servidor central garante uma busca rápida e eficiente, o sistema também possui um único ponto de entrada. Nesse caso, pode haver gargalo na rede, ou então a perda de conexão, se o servidor, ou servidores, se tornarem incapacitados. Além disso, o modelo pode fornecer informações desatualizadas por um período de tempo, uma vez a atualização é periódica e pode deixar de fornecer o estado mais atual quando, por exemplo, da saída de um usuário da rede.

Gnutella

Ao contrário do *Napster*, o *Gnutella* não usava, em sua concepção inicial, um servidor central para rastrear todos os arquivos dos usuários. Para compartilhar arquivos usando um cliente *Gnutella*, por exemplo, o *Limewire*, o terminal de um usuário (par) utiliza um computador na rede, chamado aqui de "A". Todos os pares podem ser chamados de "*servent*", pois o programa atua como uma combinação de servidor e cliente. O computador "A" então, se conectará a um computador "B" na rede, e avisará que está procurando um determinado arquivo. O computador "B", por sua vez, comunicará a todos os computadores com quem ele está conectado sobre o "A". Embora o alcance dessa rede possa ser infinito, ele é , na verdade, limitado por um mecanismo denominado TTL, que nada mais é do que o número de computadores que a busca atingirá. A maioria dos computadores (pares) rejeitarão qualquer mensagem na rede que tiver um TTL muito alto **[APL03]**.

Uma vez anunciado para os vários membros da rede, "A" pode, agora, procurar pelo arquivo desejado, nos diretórios dos outros pares que são compartilhados. A solicitação será enviada a todos os pares da rede começando com "B", depois "C", "D", "E", "F", que enviarão para os computadores aos quais estão conectados, e assim por diante. Se, por acaso, o computador "D", tiver o arquivo desejado por "A", ele transmite as informações (nome, tamanho, etc...) de volta, percorrendo todo o caminho que o par "A" fez, e uma lista também é entregue a "A" com todos os arquivos que se encaixam ao perfil do arquivo desejado. O par

"A" então poderá fazer uma conexão direta com "D" e será capaz de fazer o *download* diretamente do par "D". Esse esquema pode ser visto na figura 5**[APL03]**.

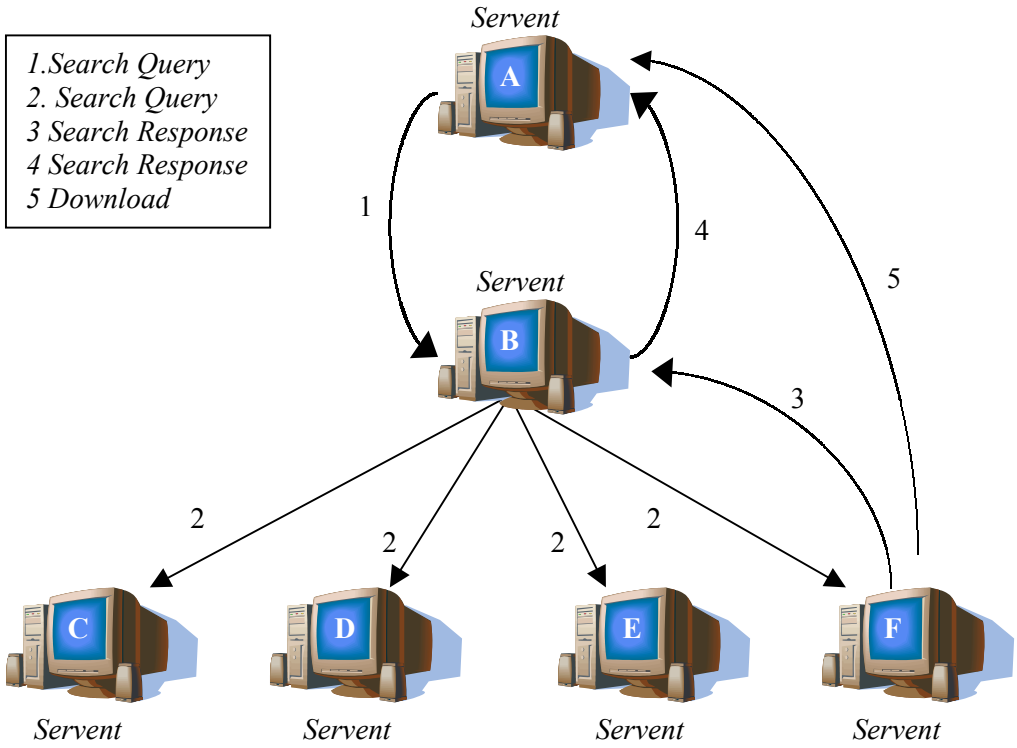


Figura 5 – Rede *Gnutella*

A rede *Gnutella* tem um número significativo de vantagens sobre os outros modelos *peer-to-peer*. A rede é descentralizada e, logo, mais robusta que o modelo centralizado porque elimina a dependência em servidores que são pontos nos quais facilmente podem existir falhas.

A rede pode fazer a procura de qualquer tipo de arquivo (de receitas, a fotos, a bibliotecas JAVA). Com um software cliente na rede *Gnutella*, um usuário também tem o poder de alcançar cada computador conectado à Internet, enquanto os melhores métodos de busca só cobrem 20% dos sítios disponíveis **[APL03]**.

Mensagens são transmitidas pela rede *Gnutella* de uma forma descentralizada, caso um ou vários computadores se desconectarem da rede, as solicitações continuarão a ser enviadas de nodo a nodo.

Enquanto o *Napster* tinha apenas um software como Cliente, o *Gnutella*

apresenta uma variedade, dentre os quais: *BearShare*, *Gnucleus*, *KazaA*, *LimeWire*, *Morpheus*, *WinMX*, *XoloX*.

Atualmente, a rede *Gnutella* e seus softwares clientes sofreram modificações para oferecer melhorias a seus usuários. Assim como no *KazaA*, o qual será comentado a seguir, a rede *Gnutella* implementou os *Ultrapeers*. Veremos essa modificação mais adiante.

KazaA

O *Kazaa Media Desktop* é mais uma aplicação para compartilhamento de arquivos, usando o modelo *peer-to-peer*. Isso significa que os usuários individuais se conectam uns aos outros, diretamente, sem a necessidade de um servidor central para gerenciamento. O *Kazaa* utiliza a tecnologia *FastTrack* que será apresentada logo abaixo no texto **[AIT]**.

O *FastTrack* incorporou a natureza descentralizada do *Gnutella* e adicionou um elemento chamado "*supernode*". Os *supernodes* agem como servidores de indexação temporários e ajudam a suportar a estabilidade da rede. Esses *supernodes* ficam fora do controle de qualquer empresa. Particularmente, essa implementação está incorporada dentro do software cliente, o qual é o *KaZaA*. Se um usuário estiver rodando uma conexão Internet rápida, através de um computador de alto poder de processamento, este usuário tem chances de ser um *supernode*.

Se um computador estiver funcionando como um *supernode,* outros usuários na sua vizinhança (de pares) irão, automaticamente, fazer um *upload* para essa máquina, contendo uma pequena lista dos arquivos que eles estão compartilhando e, sempre que possível usando o mesmo provedor de Internet. Quando eles quiserem fazer uma busca, a solicitação de busca é enviada para esse *supernode* que, no caso, é o mais próximo. Encontrado o arquivo, o *download* é feito diretamente com a máquina que compartilha o arquivo, e não a do *supernode*, de onde a busca foi iniciada. Caso haja perda de conexão, a busca vai para um *supernode* mais próximo, uma vez que não existe um, mas sim muitos *supernodes*.

Um *supernode* melhora o desempenho da rede e diminui a latência de rede dos

usuários. Outros computadores se conectam ao *supernode* que passa a ser o responsável pelas buscas. Um *supernode* comunica-se com outros *supernodes* na Internet para ajudar nas buscas. A quantidade de CPU que pode ser usada por um *supernode* é limitada em 10% do poder total de CPU disponível - e um usuário nem será notificado que o seu computador está funcionando como um *supernode*. Caso o usuário não queira isso, basta desabilitar a opção "*Do not function as a supernode*" ou "Não funcionar como um *supernode*".

Uma das vantagens de se usar *supernodes*, é a escalabilidade, uma vez que os estes estão espalhados por toda Internet. Embora descentralizado como o *Gnutella*, o *FastTrack* torna a rede estável, não importando a quantidade de usuários conectados.

FreeNet

A rede *FreeNet* para compartilhamento de arquivos *peer-to-peer* é completamente descentralizada e distribuída, significando que não há um servidor central e que todas as computações e interações acontecem entre os pares. No *Freenet*, todas as conexões para a rede são iguais. Ao iniciar a aplicação, os clientes conectam-se randomicamente a qualquer cliente, gerando, assim, uma topologia espalhada e desorganizada.

Essa aplicação tem como principais exigências que seus dados sejam identificados, usando uma única chave numérica, e que os nós estejam dispostos a armazenar as chaves um para o outro, facilitando, assim a busca de valores. Esses valores podem ser itens de dados (bloco de arquivos), ou então ponteiros para onde os dados estão atualmente armazenados. **[BAL03]**

Para publicar um arquivo sob um único nome particular, o publicador deverá converter o nome em uma chave numérica usando uma função *hash* ordinária, SHA-1. O Publicador deverá, então, enviar o arquivo para ser armazenado no par responsável pela chave. Um usuário, desejando acessar aquele arquivo, deverá obter seu nome, convertê-lo numa chave, e solicitar ao par que responder, uma cópia do arquivo **[BAL03]**.

Qualquer par que receba uma consulta por um identificador (*key*) "s", por exemplo, deve ser capaz de encaminhar a consulta para outros pares que possam conter o ID

"s". Essa regra garantirá que a consulta chegue ao par que contém o dado.

No *Freenet* **[CLAR00]** , as consultas são encaminhadas de par para par até que o objeto desejado seja encontrado, baseado nas tabelas de roteamento. Para oferecer anonimato, o *Freenet* evita associar um documento com qualquer servidor previsível, ou formar uma topologia previsível entre os servidores. Como resultado, documentos incomuns podem simplesmente desaparecer do sistema, uma vez que nenhum servidor tem a responsabilidade de manter réplicas. Além disso, uma busca pode freqüentemente precisar visitar uma grande quantidade de pares no sistema e nenhuma garantia disso é possível.

As aplicações de compartilhamento de arquivos podem apresentar as seguintes características:

- disponibilidade: a informação pode ser acessada 24 horas por dia, 7 dias por semana;
- durabilidade: a informação colocada no sistema pode ficar por lá, virtualmente, para sempre;
- controle de acesso: a informação é protegida. O controle de acesso inclui privacidade (entidades não autorizadas não podem ler informações) e integridade na escrita (entidades não autorizadas não podem mudar informações);
- autenticidade: a autenticidade dos documentos é preservada;
- negação de serviço (*Denial-of-service*): é muito difícil para um adversário, comprometer a disponibilidade dos arquivos;
- os recursos podem ser compartilhados, mas não são sincronizados, ou seja, enquanto uma pessoa está executando um arquivo na sua máquina, outra pode estar executando o mesmo arquivo numa máquina diferente, sem a necessidade de sincronismo;
- outras características podem ser vistas em *Milojicic* **[MIL02]**.

2.1.3. Colaboração peer-to-peer

Aplicações colaborativas *peer-to-peer* têm como principal objetivo permitir a colaboração como aplicação entre os usuários. Essas aplicações abrangem *chats* e mensagens instantâneas, jogos e aplicações compartilhadas que podem ser usadas comercialmente e até mesmo por um grupo de amigos em suas casas.

As aplicações colaborativas são baseadas em eventos. Os pares formam grupos e iniciam uma determinada tarefa. O grupo pode incluir apenas dois pares colaborando diretamente, ou um grupo maior. Quando uma mudança ocorre e um par, um evento é gerado e enviado ao resto do grupo. Na camada de aplicação, cada interface de um par é atualizada de acordo com cada aplicação.

Para as aplicações que envolvem colaboração, o *Groove* **[GRO]** é um exemplo de aplicação que habilita espaços compartilhados para o trabalho colaborativo de um grupo de pares. Ela oferece, além do espaço virtual, as ferramentas para interação de grupos, incluindo compartilhamento de conteúdo, comunicação e atividades em comum em tempo real. Cada grupo possui seu espaço compartilhado, replicado em todos os computadores membros. Quando um dos membros adiciona algo naquele espaço, a mudança é refletida em todas as máquinas. A aplicação oferece uma variedade de serviços - segurança, armazenamento, sincronização e conectividade. Utiliza um modelo híbrido, em que o suporte a colaboração é orientado a pontos e as funções de gerenciamento do sistema são centralizadas.

O *Groove* apresenta algumas características que são:

- segurança: manipulação automática de gerenciamento de chave pública/privada, criptografia e autenticação;
- armazenamento de objetos XML: permite a desconexão da rede;
- sincronização: permite a sincronização de múltiplas réplicas locais e remotas do espaço compartilhado ao qual o usuário pertence;
- conexão dos pares para suportar administração transparente, tais como endereços IP, seleção de largura de banda e *firewalls;*
- anonimato: o *Groove* não foi designado para proteger usuários do

anonimato. Pelo contrário, os desenvolvedores viram que uma colaboração/comunicação não poderia acontecer, a menos que os usuários se conhecessem;

- tolerância a falhas: o *Groove* suporta tolerância à falha através da interação de *transceivers* como parte de seu protocolo de sincronização. *Transceivers* retem cópias dos eventos, até que eles sejam finalizados.

Essas e outras características acima podem ser encontradas em **[MIL02]** e também em **[GROO01]**. Apesar de *Groove* ser interessante no suporte a ambientes virtuais colaborativos, nos quais os jogos poderiam, de certa maneira, se enquadrar, o *Groove* é uma solução proprietária que exige a instalação de software proprietário da Microsoft.

2.1.4. Plataforma peer-to-peer

Os sistemas operacionais estão se tornando cada vez menos relevantes como ambientes para aplicações. Soluções, tais como *Java Virtual Machines*, ou *Web browsers* e servidores são os ambientes dominantes que são dos interesses dos usuários bem como dos desenvolvedores de aplicações. Considerando-se isso, é provável que futuros sistemas em crescimento dependerão de algum outro tipo de plataforma que será um denominador comum para os usuários e serviços conectados a *Web,* ou numa rede *ad-hoc* **[MIL02]**.

As plataformas *peer-to-peer,* atualmente disponíveis, oferecem suporte para os seguintes componentes primários P2P: descoberta, comunicação, segurança, e agregação de recursos. Elas têm uma dependência de SO, mesmo que mínima. A maioria das aplicações P2P ou estão rodando em *Linux* ou então são baseadas no *Windows*.

Dentre as plataformas *peer-to-peer*, uma que se destaca é a plataforma JXTA. Trata-se de um conjunto de protocolos abertos e generalizados P2P que permitem a conexão de qualquer dispositivo na rede - de telefones celulares, PDAs e PCs - para se comunicarem e colaborarem como pares. Os protocolos JXTA são independentes de qualquer linguagem de programação, e múltiplas implementações existem para diferentes ambientes. O JXTA é baseado no software *Java*^{TM} 2 *Plataform*, *J2SE* **[JXTA]**.

Um dos objetivos do projeto JXTA é oferecer uma plataforma com funções básicas necessárias para uma rede P2P. Além do mais, a tecnologia JXTA busca superar as

faltas potenciais de muitos sistemas P2P existentes, como se vê a seguir:

- interoperabilidade: O JXTA é projetado para localizar e se comunicar com qualquer par da rede;
- independência de plataforma: O JXTA é projetado para ser independente de linguagens de programação, protocolos de transporte e plataformas de desenvolvimento;
- ubiqüidade: O JXTA é projetado para ser acessível a qualquer dispositivo e não apenas PCs ou plataformas específicas.

O JXTA padroniza a forma como os pares descobrem uns aos outros, permitindo, assim, como permite que esses pares se auto-organizem em grupos e, também, anunciem e descubram serviços de rede. A forma de comunicação entre os pares, bem como o monitoramento uns dos outros, são, igualmente, padronizados pela plataforma **[JXTA]**

O JXTA foi feito para ser independente de linguagem de programação e protocolos de transporte. Os seus protocolos podem ser implementados em Java, C/C++, Perl e muitas outras linguagens. Podem ser implementados sobre TCP/IP, HTTP, *Bluetooth*, entre outros protocolos de transporte.

Dentre as várias características, descritas em **[MIL02]**, podemos citar algumas:

- descentralização: A base original do JXTA era totalmente descentralizada. Não havia intenção de se usar DNS ou qualquer outro servidor de nome centralizado, embora as aplicações JXTA possam usá-lo. Porém, para fazer descoberta descobertas na rede, houve a necessidade de um ponto de encontro, para publicar os anúncios dos pares na rede. Esses pontos de encontro (*rendezvous*) são os únicos aspectos centralizados do JXTA;
- escalabilidade: O projeto JXTA tem uma arquitetura em camadas, que consistem em três blocos: JXTA core (para gerenciamento do grupo de pares), serviços JXTA (tais como busca e indexação), e aplicações JXTA (JXTA Shell). Essa arquitetura em camadas permite ao JXTA

incorporar novos protocolos e serviços para suportar um número crescente de usuários;

- anonimato: O anonimato pode ser construído no topo da arquitetura JXTA como um serviço e os protocolos JXTA não impedem o usuário de fazer isso;
- auto organização: O JXTA é uma plataforma auto-organizável;
- tolerância a falhas: A tolerância a falhas é construída dentro do mecanismo de descoberta. O JXTA é capaz de reiniciar e recuperar conexões e serviços;
- segurança: O JXTA suporta implementações da camada de transporte segura (TLS) com autenticação de clientes e servidores;
- transparência: todas as tarefas são transparentes para os usuários. Apresentam, porém, baixa transparência.

Para finalizar esta etapa da pesquisa, na qual discutimos os tipos de aplicações *peer-to-peer*, mostraremos uma tabela das aplicações P2P com as principais características de cada uma delas.

Tabela 1 - Comparação entre as tecnologias *peer-to-peer*

Aplicações P2P	**Modelo**	**Escalabilidade**	**Anonimato**	**Desempenho**	**Segurança**	**Transpa-rência.**	**Tolerância a Falhas**
seti@home	Client/server	Milhares	Médio	Alta velocidade	Proprietária	Alta	*Timed Checkpoint*
Groove	P2P Híbrido	25 espaços compart.	Não suportado mas possível	Médio	Autenticação Criptografia	Alta	Mensagens a serem enviadas
Gnutella	P2P descentralizado	Milhares	Baixa	Baixa	Não endereçável	Média	conclui *download*
JXTA	P2P Descentralizado	Tem potencial para alta escala	Pode ser criado	N/A	criptografia	Baixa	Reinicia e recupera conexões

2.2. Desafios do modelo Peer-to-Peer

Além do desafio inerente para se ter aplicações comunicando-se de igual para igual, outros desafios podem ser identificados no modelo P2P, tais como são segurança, interoperabilidade, controle de acesso, integridade dos dados, sincronização de uma aplicação entre duas ou mais máquinas (pares), além do grande desafio de se ter controle e dados distribuídos numa rede P2P. Vejamos as descrições desses desafios:

- segurança: devido à ampla quantidade de ameaças conhecidas nos sistemas de rede, mecanismos de segurança são extremamente necessários. Como o P2P está tornando-se interessante para o uso comercial, técnicas e métodos para autenticação, autorização, disponibilidade, integridade dos dados e confiança devem ser integradas ao P2P;

- interoperabilidade: atualmente, as aplicações utilizam protocolos e interfaces específicos. Como resultado, a interoperabilidade que se estende além de uma simples aplicação ou rede é rara nas atuais redes P2P. O desafio é encurtar o tempo de desenvolvimento e habilitar as aplicações a serem implementadas facilmente nos sistemas existentes;

- controle de acesso: outro desafio é o controle do acesso ao sistema, permitindo ou não acesso de usuários às informações;

- integridade dos dados: caso os dados estejam distribuídos entre os vários pares existentes na rede, deve haver uma preocupação com a integridade desses dados, que podem ou não depender uns dos outros para formar alguma informação;

- sincronização: um dos grandes desafios dos sistemas P2P é a sincronização dos dados, uma vez que vários pares atuando em aplicações multiusuário podem acessar um mesmo dado num memo intervalo de tempo;

- distribuição dos dados: a distribuição dos dados pode ser feita de várias formas, entre elas a replicação (uma alteração de estado é passada para

as réplicas) e particionamento (a aplicação é dividida em várias partes e cada máquina fica encarregada de uma);

- distribuição do controle: O controle em redes P2P é necessário e pode ser aplicado num servidor, como por exemplo, em *Napster*, *ICQ* entre outras aplicações de rede P2P híbridas, ou, então, pode ser aplicado nos próprios clientes, eliminando a presença de um servidor, como em redes P2P puras. O desafio é, principalmente, implementar-se o controle em redes P2P puras, ou seja, como administrar sem ter um servidor, como eleger uma máquina para que essa passe a ter o controle, e se essa máquina se desconectar da rede, para quem é passado o controle. Todas essas questões serão analisadas neste projeto.

2.3. Vantagens e desvantagens do Peer-to-Peer

O modelo de comunicação *Peer-to-peer* apresenta algumas vantagens relacionadas ao modelo cliente/servidor descritas a seguir:

- a falha num cliente afeta apenas a participação dos usuários ligado neste cliente, podendo ou não afetar a operação do sistema como um todo (depende da topologia);
- elimina o gargalo, uma vez que as atividades estão distribuídas e não centralizadas;
- dependendo da topologia, a distância entre os clientes pode ser reduzida;
- é escalável, podendo suportar um número crescente de clientes, mas pode apresentar problemas de engarrafamentos na rede;
- não apresenta barreiras para se criar um sistema P2P, uma vez que ele não requer nenhum arranjo administrativo ou financeiro, ao contrário do cliente/servidor;
- oferecem uma forma de agregar e fazer uso de enormes recursos

computacionais e de armazenamento através dos computadores pela Internet;

- a natureza descentralizada e distribuída do sistema P2P lhes dá o potencial de ser tolerante a falhas e/ou ataques intencionais.

Esse modelo, assim como o cliente/servidor, apresenta, porém, desvantagens, a saber:

- maior complexidade do software; custos mais altos de desenvolvimento e menor desempenho em cada cliente, devido à distribuição da coordenação das atividades entre os clientes;
- cada cliente tem que saber os endereços dos outros clientes, para que possa se comunicar diretamente;
- maior complexidade no gerenciamento de entrada e saída de novos clientes;
- o controle das versões de software é mais difícil de ser mantida, por estar distribuída entre os clientes.

Com a apelação de sair do tradicional cliente/servidor e utilizar mais o potencial das máquinas que ficam nas pontas da rede (pares), existem várias aplicações hoje, utilizando o modelo de comunicação P2P, e algumas tais como o *Napster* e *Gnutella*.

2.4. A Rede Gnutella

Quando a rede *Gnutella* surgiu, em 2000, foi considerada como a próxima geração do *Napster*, rede que iniciou a febre do compartilhamento gratuito de arquivos de música e que levou as gravadoras a um estado de alerta sobre questões de direitos autorais. Logo descobriu-se que a rede gerava muito tráfego e esta foi reorganizada espelhando-se em estruturas mais eficientes, como a *FastTrack* utilizada pela rede *KazaA*, passando a incorporar o conceito de *ultrapeers*, pares especiais com maior potencial de processamento e capacidade de rede. Estes pares aliviam os pares normais da recepção da maioria do tráfego de mensagens, reduzindo, assim, o tráfego total da rede e deixando-a mais rápida. Com o conceito de *ultrapeers* incorporado à rede *Gnutella,* esta passou a ser

considerada uma rede *peer-to-peer* híbrida. Para utilizar a rede *Gnutella,* um usuário precisa de uma aplicação que promove a conexão com outros pares, formando uma rede espontânea, sem controle centralizado, e que age tanto como cliente quanto como servidor de conteúdo. Um dos mais conhecidos é o software livre *Limewire* **[LIME03]**, a ser utilizado neste trabalho. Cada instância do *Limewire* é chamada de *peer* (par).

Como não há servidores centralizados, quando um usuário quer entrar na rede *Gnutella,* ele, inicialmente, conecta-se a um dos vários *ultrapeers*, quase sempre disponíveis. Estas máquinas encaminham, então, informações como endereço IP e porta de rede para outros pares *Gnutella*.

Uma vez conectado à rede, os pares interagem entre si através de mensagens do tipo: *Ping*, *Pong*, *Query*, *Rich Query*, *Query Response* e *Get*/*Push*. Características do protocolo *Gnutella* impedem que mensagens sejam re-encaminhadas para outros pares da rede indefinidamente.

Apesar da rede *Gnutella* ter sido um pouco obscurecida por redes eficientes como o *FastTrack* (protocolo proprietário da aplicação *KazaA*) ela foi escolhida como motivo de estudo neste trabalho pela simplicidade conceitual e por ser um software livre, o que proporciona que experimentações as mais diversas e interessantes possam ser realizadas e avaliadas, como é o caso da implementação de jogos multiusuário. A existência de aplicações de jogos na rede *Gnutella* é desconhecida pelo autor deste trabalho - a maioria do conteúdo compartilhado na rede é de arquivos de música.

2.4.1. O Protocolo Gnutella

Toda a comunicação no *Gnutella* acontece sobre o protocolo TCP/IP. Uma vez que a conexão esteja estabelecida entre dois *servents*, a string de conexão:

"GNUTELLA CONNECT/<versão do protocolo>\n\n"

deve ser enviada por um dos clientes (versão "0.4"). O *servent* que responder enviará a mensagem

"GNUTELLA OK\n\n"

estabelecendo, assim, uma conexão *Gnutella* entre esses dois *servents*. Qualquer outra resposta à string de conexão original será considerada uma rejeição pelo *servent* inicial.

Depois que a conexão estiver estabelecida, dois *servents* comunicam-se entre si através de trocas de descritores de protocolo *Gnutella*. O protocolo *Gnutella* também define as regras de como esses descritores são trocados entre os nós.

Tabela 2 - Descritores Gnutella

Descritor	Descrição
Ping	Usado para descobrir *hosts* ativos na rede. Um *servent* que recebe um descritor *Ping* deve responder com um ou mais descritor *Pong*.
Pong	A resposta a um *Ping*. Inclui o endereço de um *servent Gnutella* conectado e informações sobre os dados que este estiver disponibilizando na rede.
Query	O mecanismo principal para fazer buscas na rede distribuída. Um *servent* que recebe um descritor *Query* irá responder com um *QueryHit* se o que está sendo procurado estiver disponível no conjunto dos seus dados locais compartilhados.
QueryHit	A resposta a um *Query*. Este descritor provê informações suficientes para que os dados encontrados possam ser adquiridos.
Push	Um mecanismo que permite a um *servent* que está atrás de um *firewall* contribuir com a rede.

Como o protocolo *Gnutella* existe sobre o TCP/IP, os endereços IP no formato IPv4 são usados como parte dos descritores *Gnutella* para a identificação de pares (por exemplo, quando estiver estabelecendo *downloads* entre dois *servents).*

Cada descritor é precedido por um cabeçalho, conforme a figura 6. Neste cabeçalho, "*Descriptor* ID" é uma string de 16 bytes que identifica unicamente o descritor na rede. "*Payload Descriptor*" identifica o tipo do descritor (i.e., 0x00 = *Ping*,

0x01 = *Pong*, 0x40 = *Push*, 0x80 = *Query*, 0x81 = *QueryHit*). O contador TTL e o contador de *hops* (quantas vezes o descritor foi enviado de um nó a outro na rede) devem ser conforme a equação TTL(0)=TTL(I)+*Hops*(I). Isto é, em cada *hop* na rede, o contador TTL deve ser decrementado em um, enquanto o contador de *hop* deve ser incrementado em um. Como dito anteriormente, a transferência desse descritor termina quando o contador TTL chega em zero.

	Descriptor ID		*Payload Descriptor*	*TTL*	*Hops*	*Payload Length*	
Offset	*0*	*15*	*16*	*17*	*18*	*19*	*22*

Figura 6 - Cabeçalho *Gnutella* [Prot04]

Regras de Roteamento de Descritores do Protocolo

1. Descritores *Pong* devem apenas ser transmitidos pelo mesmo caminho que trilhou o *Ping*. Isso assegura que somente os *servents* que rotearam o *Ping* irão ver o *Pong* em resposta. Um *servent* que recebe um descritor *Pong* com o *Descriptor* ID = n, mas não recebeu um descritor *Ping* com *Descriptor* ID = n, deve remover o descritor *Pong* da rede.

2. Descritores *QueryHit* devem ser enviados apenas pelo mesmo caminho que trilhou o descritor *Query*. Isso assegura que apenas os *servents* que rotearam o descritor *Query* irão ver o descritor *QueryHit* de resposta. Um *servent* que recebe um descritor *QueryHit* com o *Descriptor* ID = n, mas não recebeu um descritor *Query* com *Descriptor* ID = n deve remover o descritor *QueryHit* da rede.

3. Descritores *Push* devem ser enviados apenas pelo mesmo caminho que trilhou o descritor *QueryHit*. Isso assegura que apenas os *servents* que rotearam o descritor *QueryHit* irão ver o descritor *Push*. Um *servent* que recebe um descritor *Push* com *Descriptor* ID =

n, mas não viu um descritor *QueryHit* com *Descriptor* ID = n deve remover o descritor *Push* da rede. Um *servent* que recebe um descritor *Push* com *Servent Identifier* = n, mas não recebeu um descritor *QueryHit* com *Servent Identifier* = n deve remover o descritor *Push* da rede. Descritores *Push* são roteados pelo *Servent Identifier*, não pelo *Descriptor* ID.

4. Um *servent* irá encaminhar os descritores *Ping* e *Query* que recebeu a todos os seus *servents* diretamente conectados, exceto para aquele que enviou o *Ping* ou *Query*.

5. Um *servent* irá decrementar o campo TTL do cabeçalho do descritor, e incrementar seu campo *Hops* antes de encaminhar o descritor para algum outro *servent* diretamente conectado. Se, depois de decrementado o campo TTL, o valor do campo TTL for zero, então o descritor não é encaminhado.

6. Um *servent* que recebe um descritor com o mesmo *Payload Descriptor* e *Descriptor* ID que um descritor recebido anteriormente evitará encaminhar esse descritor para os *servents* diretamente conectados, pois eles já devem tê-lo recebido e enviando-o novamente, apenas desperdiçaria largura de banda.

O roteamento das mensagens na rede *Gnutella* pode ser visto na figura 7.

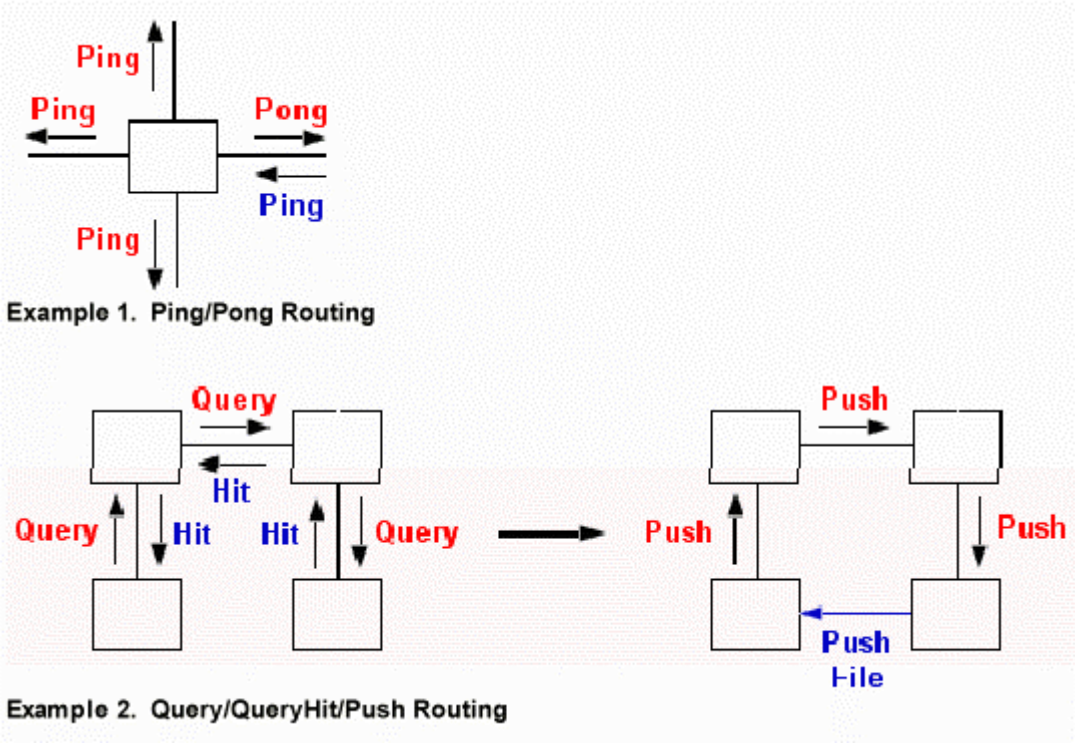


Figura 7 - Exemplos de roteamento de mensagens [Prot04]

2.4.1.1. A Inclusão do RichQuery para Refinamento de Buscas

Com o objetivo de realizar buscas mais refinadas, **[THAD02]** implementou uma modificação no protocolo *Gnutella*, adicionando um campo, o *RichQuery*, no descritor *Query*. Através desse campo, a meta-informação é codificada em XML, ou esquemas XML que definem o formato da informação que um documento pode conter.

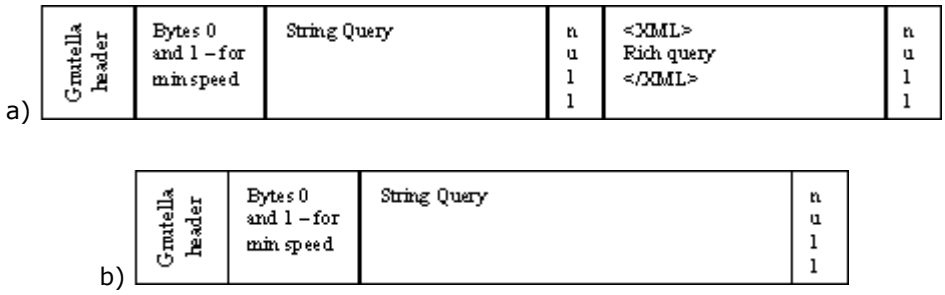


Figura 8 – a) Descritor *Query* com campo *RichQuery* b) Descritor *Query* sem campo *RichQuery*

A figura 8 apresenta o descritor *Query* já alterado com o campo *RichQuery* e antes, sem o campo. Vale lembrar que essa alteração não interfere na busca entre softwares *Gnutella* que têm o campo *RichQuery* e outros softwares que não apresentam esse campo. O que pode ocorrer no caso de um software cliente antigo (sem a *RichQuery*), é que ele apenas não vai reconhecer o campo, mas deve ignorá-lo após o primeiro *Null*. Os clientes novos (já com o campo) entendem que após o primeiro *null* o descritor não termina, mas sim, que há uma *RichQuery*.

Se uma *RichQuery* alcança um novo cliente, a busca continua normalmente e o cliente gera uma *Query Reply* e a envia justamente como uma resposta normal. A única diferença é que o cliente que responde a *RichQuery,* também colocará conteúdo XML no cabeçalho do descritor *Query Reply*.

2.4.2. O Conceito *Ultrapeer* na rede *Gnutella*

O protocolo *Gnutella* original trata todos os pares indiferentemente de suas capacidades de largura de banda, poder computacional e outras propriedades (capacidades de conexão, *uptime*, etc). Os pares na rede comunicam-se usando ou o protocolo *Gnutella* versão 0.4 **[PROT04]**, ou uma versão melhorada que utiliza novos mecanismos de estabelecimento de conexão **[GC01].** Ambos apresentam propriedades interessantes. Novos pares podem entrar na rede a qualquer momento e os pares que estão *online* podem sair a qualquer momento. A habilidade para se ter uma rede segura, sem a dependência de um par em particular (ou um conjunto deles), é uma característica notável que o *Gnutella* traz com sua popularidade. Entretanto, há outras características da rede *Gnutella* **[RIT01]** que a distingue de outros sistemas. Apesar desses aspectos positivos, a rede *Gnutella* apresenta o problema da escalabilidade. Por exemplo, um grande numero de *queries* são enviadas para muitos pares, numa tentativa de assegurar a resposta de poucos deles, causando problema de largura de banda. E para tentar resolver isso, foi proposto pelos desenvolvedores *Gnutella* o esquema de *ultrapeers*. Um *ultrapeer* é um par como os outros, porém apresentando características como alto poder de processamento e alta largura de banda, podendo assumir a "carga de trabalho" de outros pares mais fracos, PC´s ou até mesmo PDA´s e telefones celulares.

Quando um usuário comum quer consultar um item, ele envia uma mensagem para o *ultrapeer* em que está conectado que, por sua vez, envia a mensagem por difusão para os outros *ultrapeers*. Cada *ultrapeer* mantém um índice dos pares que estão a eles conectados. Entretanto, os pares mais fracos apenas recebem as consultas se eles atualmente têm o arquivo solicitado. De outra forma, o *ultrapeer* não encaminha a busca para eles. Nos seus resultados preliminares, os pares comuns não recebem quase nenhum tráfego, enquanto que as conexões dos *ultrapeers* são afetadas pelo aumento no número de conexões. Os *ultrapeers* e pares também utilizam roteamento. Isso permite que a rede funcione mais eficientemente com um menor número de mensagens solicitadas através da rede para busca de arquivos e para conexão. Com essa tecnologia, o *Guntella* pode, significantemente, aumentar os resultados de suas buscas em número e qualidade, e aumentar a escalabilidade da sua rede. A figura 9 ilustra a rede *Gnutella* com a integração de *ultrapeers*.

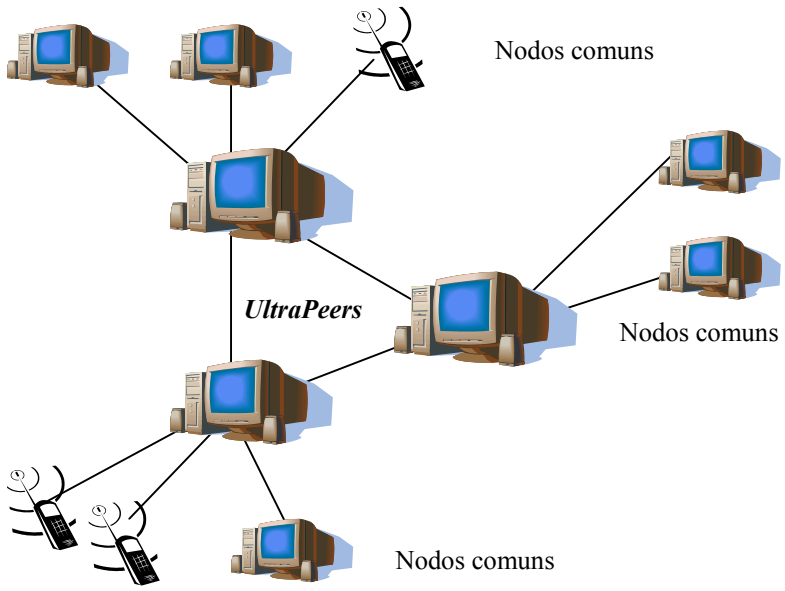


Figura 9 - Modelo da rede *Gnutella* utilizando *Ultrapeers*

Os *ultrapeers* podem, eles próprios, serem pares comuns, assim como um par comum, pode ser um *ultrapeer*. De acordo com **[SIN01]**, um par comum pode ser eleito um *ultrapeer* se ele atender os seguintes requisitos:

- **não estiver atrás de um *Firewall*:** isso pode ser observado olhando se o *host* tem recebido conexões de entrada;

- **sistema operacional em uso:** alguns sistemas operacionais tratam grande número de *sockets* melhor do que outros. *Linux*, *Windows* 2000/NT/XP e o *Mac* OS/X serão melhores super nós do que *Windows* 95/98/ME ou *Mac Classic;*

- **largura de banda suficiente:** é recomendada largura de banda com pelo menos 15 KB/s em tráfego descendente *(downstream)* e 10KB/s em tráfego ascendente *(upstream).* Pode ser analisado pelo processamento máximo de *download* e *upload;*

- ***UPTime* Suficiente:** os *ultrapeers* deveriam ter longo tempo de conexão. Um par comum não poderá tornar-se um *ultrapeer* sem que esteja *online* por um certo tempo, tipicamente 30 ou 40 minutos.

Entretanto, essa arquitetura é mais sensível às falhas dos *ultrapeers* e pode apresentar os mesmos problemas dos servidores centrais como no *Napster*. Este trabalho pretende amenizar esta sugestão, acrescentando um esquema de transferência, em que, sempre que um *ultrapeer* perder a conexão, a mesma seja passada para outro *ultrapeer* ativo.

2.4.3. Limewire

Esse cliente da rede *Gnutella*, o *Limewire*, disponível no *web site www.limewire.org,* apresenta-se como uma ferramenta gráfica utilizando o protocolo *Gnutella*, para compartilhamento de arquivos. Através desse software cliente, muitos usuários têm a possibilidade de acessar a rede *Gnutella* e todo o seu potencial que vai além do simples compartilhamento de arquivos **[LIME03]**.

Alguns dos problemas mais instigantes da ciência da computação estão, hoje, sendo tratados pelas redes *peer-to-peer*. Com isso, o código fonte do *Limewire* pode ser um recurso para pesquisadores e pessoas que o desenvolveram trabalharem na construção de ferramentas que possam auxiliar, ou mesmo senão solucionar problemas**[LIME03]**.

O código fonte do *Limewire* apresenta as seguintes características:

- simplicidade, basta instalar e começar a pesquisar;

- habilidade de buscar por meta informação;
- suporte a *download* de múltiplos hosts;
- tecnologia "*ultrapeer*" que reduz tempo de resposta para a maioria dos usuários;
- suporta chat;
- suporta pesquisa através de *firewalls*;
- conecta-se à rede usando o *GWebCache*, um sistema de conexão distribuída;

Com todas essas características, o *Limewire* foi escolhido, para ser utilizado neste projeto, como o software cliente da rede *Gnutella* na integração com o Padrão MPEG-4 MU, para a implementação de jogos *peer-to-peer*. O conceito de *ultrapeers* é usado para o tratamento dos controladores de sessão, além da rede *Gnutella*, utilizada para a troca de mensagens MPEG-4, os quais serão vistos adiante.

2.5. Considerações Finais

Neste capítulo, foi descrito o modelo *peer-to-peer* e suas características, centrando-se, principalmente no serviço de compartilhamento de arquivos, do qual foi utilizado um modelo para implementação da solução proposta neste trabalho. A tecnologia *Gnutella* foi escolhida, por seu código fonte livre e pela sua simplicidade, apresentando características como os *ultrapeers* os quais puderam ser utilizados para o controle em jogos *peer-to-peer*.

O próximo capítulo descreve os jogos como aplicações *peer-to-peer*, assim como suas características e diferentes variedades.

3. Os Jogos como aplicações Peer-to-peer

De acordo com **[SABA01]**, os jogos de computador têm evoluído por um longo tempo, quase desde o início do desenvolvimento de software. Hoje, os jogos são populares entre uma grande quantidade de usuários e oferecem mais características e possibilidades do que antes, como incríveis gráficos 3D e avançada inteligência artificial.

O termo jogos refere-se a passatempos, empregados com a finalidade de entreter o usuário (jogador). Os jogos necessitam de um conjunto de regras e se competitivos, devem apresentar alguma forma de medir a possibilidade de um jogador ganhar, de acordo com o objetivo principal de cada jogo. Adicionalmente, a maioria dos jogos requer alguma habilidade por parte dos jogadores. Por exemplo, num caso onde a modelagem do mundo real é um aspecto significante de um jogo, esse pode ser chamado de simulação (embora nem todas as simulações possam ser consideradas jogos) **[BAR03]**.

Também, de acordo com **[BAR03]**, os jogos de computador são jogados contra, moderados por, ou jogados usando um computador. Em casos raros, eles podem ser jogados entre computadores.

Existem hoje muitos jogos de computador divididos em diversas categorias, de acordo com seus tipos. *Jones* **[JONES03]** os em jogos de:

- Ação/Tiro

- Aventura/ Ficção Interativa
- Simulação
- RPG
- Estratégia
- Esportes
- Lutas
- Tabuleiros/ *Puzzles*

Passaremos, agora, a descrevê-los mais detalhadamente:

3.1. Os Tipos de Jogos

De acordo com **[LAIRD03]**, não há uma linha classificatória de tipos de jogos pois muitos deles apresentam múltiplos gêneros. Por exemplo, existem jogos de estratégia (um exemplo é o *Dungeon Keeper*) que permitem a humanos realizarem ações como se estivessem jogando um jogo de ação. Também há jogos de ação em que você deve gerenciar recursos e múltiplas unidades, como é o caso do *Battlezone*. Apesar da existência de vários gêneros de jogos, *Laird* e *Lent* **[LAIRD03]** e também **[WOLF00]** os definem como:

Jogos de Ação

Os jogos de ação se constituem um dos gêneros mais populares e envolvem o jogador humano, controlando um personagem num ambiente virtual, usando suas forças para salvar o mundo e combater as forças do mal. Esses jogos variam de acordo com a perspectiva que o jogador tem de seus personagens, a qual pode ser primeira pessoa, em que o jogador vê o que o personagem poderia ver, ou então terceira pessoa, em que o jogador vê não apenas os outros, mas também o seu personagem. Exemplos populares incluem *Doom, Quake*, *Descent, Half-Life*, *Tomb Raider, Unreal Tournament,* e *Halo.*

Role Playing Games

Em RPG´s , um jogador pode controlar diferentes tipos de personagens tais como guerreiros magos ou mesmo um ladrão. O jogador questiona, coleciona e vende itens, luta com monstros e melhora as capacidades do seu personagem (tais como força, magia, rapidez, etc.), tudo em um mundo virtual estendido. Exemplos desse tipo de jogo incluem, *Baldur's Gate, Diablo,* entre outros. Recentemente, os RPG´s multiusuário em massa, (*massively multiplayer role-playing games*) têm sido introduzidos onde milhares de pessoas jogam e interagem num mesmo mundo. Podemos citar como exemplo, *Ultima Online, Everquest,* e *Asheron's Call*.

Jogos de Aventura

Estes jogos e o relacionado gênero de ficção interativa não dão ênfase ao combate armado, mas sim às histórias, enredos e soluções de quebra-cabeças. Os jogadores devem resolver enigmas e interagir com outros personagens para progredir pela aventura. Os primeiros jogos de aventura tais como *Adventure e Zork* foram totalmente baseados em textos, mas os jogos mais recentes apresentam gráficos 3D. Exemplos desses jogos incluem *Infocom series, King's Quest*, e muitos jogos da *Lucas Arts*, tais como *Full Throttle, Monkey Island*, e *Grim Fandango*.

Jogos de Estratégia

Nesta categoria, os jogadores controlam muitas unidades como tanques de guerra ou unidades militares e até mesmo máquinas de guerra alienígenas para fazerem batalhas através da visão de um deus contra um ou mais oponentes. Os jogos de estratégia incluem diferentes tipos de batalha: histórica (*Close Combat, Age of Empires*), realidades alternativas (*Command and Conquer*), futuro da ficção (*Starcraft*), e lendárias (*Warcraft, Myth*). O jogador é colocado em face de problemas como alocação de recursos, escalonamento da produção e organização de defesas e ataques **[LAIRD03]**.

Jogos de Simulação

Os jogos de simulação dão ao jogador controle sobre uma situação ou um mundo simulado. O jogador pode modificar o ambiente e, em até certo ponto, seus habitantes. O entretenimento vem da observação dos efeitos de suas alterações nos indivíduos, sociedade e no mundo. *SimCity* é um exemplo clássico de uma simulação, onde o jogador age como um prefeito e controla as unidades básicas de uma cidade simulada. *The Sims* e o jogo *Black and White* são provavelmente os jogos mais intrigantes. Neles, os jogadores criam personagens individuais com seus próprios objetivos e estratégias para atingi-los mas um deus (o jogador) pode vir e interferir no gerenciamento de ambos, personagem e ambiente.

Esportes

Os jogos de esportes cobrem quase todas as modalidades imagináveis, dos tradicionais esportes com times, como futebol ou baseball, aos esportes individuais como os esportes olímpicos. Muitos destes últimos se encaixam dentro dos jogos de corrida, tais como carros, corrida de barcos, na neve, etc. O gênero corrida será tratado separadamente a seguir. Jogos de times têm o jogador como uma combinação de técnico e jogadores (personagens) ao mesmo tempo em esportes populares, tais como futebol americano, **[WHA99]**, *basket*, futebol entre outros.

Jogos de Corrida *(Racing games)*

Os jogos de corrida operam quase com todos os tipos de veículos: carros, caminhões, motos, barcos, aviões, *skates*, entre outros. Alguns dos primeiros jogos de computador envolveram corridas de carro, evitando obstáculos e tentando vencer outros participantes. Em jogos de corrida, o computador oferece uma simulação de esporte em primeira ou terceira pessoa. O jogador pode controlar um participante num jogo cujo competidor pode ser outro humano ou mesmo computadores.

Jogos de tabuleiros

Os jogos de tabuleiro incluem jogos como xadrez, *Checkers*, ou *Backgammon* e tem esse nome, pois para serem ativados necessitam do tabuleiro, físico ou virtual. Os jogadores podem jogar contra um oponente, ou mesmo com um computador.

Jogos de luta

São jogos envolvendo personagens que lutam usando as mãos, em situações de combate um - a – um, sem o uso de projéteis ou armas. Na maioria desses jogos, os lutadores são representados como humanos ou outros tipos de personagens. Alguns jogos de lutas podem ser classificados no gênero esportes, como é o caso do boxe. *Street Figth* é um exemplo de jogo de luta.

Jogos Educacionais

São jogos criados para ensinar cujo principal objetivo envolve o ensinamento de uma tarefa. Ao invés de ser estruturado diretamente com lições e exercícios, esses programas são estruturados como jogos, com elementos tais como *score*, performance de tempo e pontos dados por cada resposta correta. Exemplos incluem: *Alpha Beam with Ernie; Math*; *Mario's Early Year,* entre outros.

Com base em todos os gêneros de jogos acima, fizemos, ainda, um estudo sobre os principais requisitos dos jogos, características necessárias não apenas na construção, mas também no gerenciamento do jogo. Descrevemos os requisitos a seguir.

3.2. Requisitos de Jogos

Os requisitos de jogos, de uma forma geral, sem especificar os que são apenas para jogos *peer-to-peer,* ou para jogos cliente/servidor são:

Consistência

O requisito de consistência implica que, mesmo que cada participante compute sua própria visão local do jogo, uma técnica de sincronismo distribuído é necessária para garantir que os participantes computem estados do jogo tão similares quanto possível. As entidades devem também se recuperar de informações perdidas ou atrasadas **[DAROSA99]**.

Sincronismo

Jogos distribuídos que utilizam a Internet precisam ainda resolver o problema do sincronismo. Como os atrasos da rede são diferentes para qualquer par de participantes na Internet, mecanismos de sincronização precisam ser introduzidos para permitir que pacotes enviados "ao mesmo tempo" sejam processados "juntos" por qualquer participante. Outro aspecto importante da sincronização é que todos os participantes devem exibir o mesmo estado global do jogo simultaneamente **[DAROSA99]**.

Escalabilidade

Escalabilidade é a arte de desenvolver um sistema que pode crescer, com novas partes, se necessário, sem ter a necessidade de reescrever sessões inteiras toda vez que algum componente é adicionado. Uma arquitetura distribuída permite mais facilmente um particionamento natural do jogo à medida que o número de jogadores aumenta, enquanto que num jogo cliente/servidor, o aumento excessivo no número de usuários pode causar gargalo e até perda de conexão devido ao aumento no trafego de mensagens **[DAROSA99]**.

Persistência

Em jogos como RPG ou o que chamamos de *Persistent Multiplayer*, um humano pode jogar com diferentes tipos de personagens como um guerreiro, um mago entre outros. O jogo tem seu andamento com perguntas, colecionando e vendendo itens, lutando com monstros e expandir as capacidades do personagem (tal como aumentar o poder de um mago), tudo num mundo virtual estendido. Inteligência artificial é usada para controlar inimigos, parceiros dos jogadores e outros personagens de suporte. Os jogos multiusuário em massa (*The massively multiplayer games*) oferecem uma oportunidade adicional para usar

inteligência artificial para expandir e melhorar as interações sociais do tipo jogador-jogador **[KEE99]**.

Em jogos de estratégia, ou também conhecidos como jogos *Non-persistent Multiplayer*, o humano controla muitas unidades (unidades militares, como por exemplo, tanques de guerra) para fazer batalhas da visão de um Deus contra um ou mais de seu oponentes. Inteligência artificial é utilizada em duas regras: (1) como um controle para o comportamento detalhado de unidades individuais que um humano comanda e (2) como um oponente estratégico que deve jogar com um humano. **[KEE99]**.

Interatividade

Interatividade significa a capacidade que o usuário tem de interagir com um ambiente. Os jogos devem apresentar suporte para grandes mundos com um alto grau de interatividade, apresentando interatividade com PC´s e NPC´s. Também pode apresentar tipos de interatividade, como por exemplo:

- Avatar x Ambiente;
- Avatar x Avatar;
- Avatar x Objeto Dirigido por Simulação;

Uma rica interação significa o uso de formas mais naturais de interação.Interação são descrições, ilustrações, representações ou manifestações de ações engatilhadas por jogadores ou pelo ambiente de jogo.

Se há uma palavra com muitos significados nos jogos, essa é a "interatividade". Em jogos de aventura, "interatividade" significa ser capaz de manipular o ambiente. Em jogos de ação, significa ter controle sobre o personagem principal. Alguns jogos combinam ambos tipos de interatividade com excelentes resultados **[KENT].**

Extensibilidade

Uma aplicação extensível é aquela que apresenta várias outras formas de comunicação, tornando-se assim viável a diferentes modelos de comunicação, como por exemplo, *peer-to-peer*, cliente/servidor e o modelo rede de servidores **[SMED02]**.

No *peer-to-peer* há um conjunto de pares conectados formando uma rede. O modelo *peer-to-peer* é usado em jogos multiusuário porque é simples de realizar (não há burocracia a cumprir) e de expandir a partir de um jogo simples. Porém não é escalável devido a sua estrutura hierárquica. O *Peer-to-peer* é mais comum em jogos multiusuário quando o número de usuários é pequeno, ou quando estão jogando numa LAN.

No cliente/servidor, um nó tem a função de servidor. Agora, toda a comunicação é manipulada pelo nodo servidor, enquanto que aos outros nodos resta o papel de clientes. Pode ser escalável porém apresenta problemas como formação de gargalo no servidor. Esse modelo é usado em servidores comerciais *online,* assim como nos clássicos MUDs.

O outro modelo de comunicação é a rede de servidores caracterizada por diversos servidores interconectados. Aqui, a comunicação pode ser vista como uma rede *peer-to-peer* de servidores sobre um conjunto de sub-redes cliente/servidor. A rede de servidores apresenta uma vantagem que é a grande escalabilidade, porém pode incrementar a complexidade da manipulação do tráfego na rede.

A extensibilidade para outras plataformas e outros SO's também deve ser levada em conta, visto que os jogos multiusuário podem ser jogados por qualquer usuário, em qualquer rede e em qualquer dispositivo.

Segurança

A segurança *online* tem se tornado muito importante na industria do entretenimento. Em se falando de jogos multiusuário, métodos de segurança, como impedimento de acesso não autorizado, ou ainda impedindo que um determinado usuário não altere o estado do jogo para ganho próprio, entre outros métodos, devem ser utilizados.

Em **[KIR97]**, dois tipos de objetivos de segurança foram reconhecidos para jogos *online*: Proteção de informações sensíveis, como, número do cartão de crédito, e o

oferecimento de um campo de jogo limpo, por exemplo, manipulação do jogo. A segurança também envolve outros fatores que são:

- proteção a servidores de jogos, devido a invasão de *hackers*, podendo causar problemas de autenticação de outros usuários e até atrasos no andamento do jogo;

- proteção dos clientes, pois os *hackers* podem agir como um usuário válido, alterando e destruindo todas as informações e perfis de um usuário invadido;

- negação de um serviço, ou "*denial-of-service*", é quando os *hackers* impedem usuários legais de acessarem os jogos ou suas informações;

- o servidor passa a controlar as ações dos clientes, por exemplo, exigindo com que o cliente se atualize, fazendo assim um *download* da atualização que sem que ele perceba, traz consigo *trojans,* ou outros arquivos inválidos;

- os *hackers* ainda podem modificar o software cliente para que ele faça coisas inesperadas, quando permitirem a invasão de outros usuários.

Para esses problemas, **[KATO]** oferece soluções como sistema de detecção de intrusos, rígidos métodos de controle de autenticação, assinaturas digitais, melhores e mais seguros métodos de autenticação cliente/servidor com uso de criptografia, armazenamento de informações dos clientes em servidores seguros e ferramentas de monitoramento do sistema.

Latência

A Latência indica o tempo decorrente de uma mensagem, desde o envio dessa mensagem por um usuário, até que todos os outros recebam a mesma. Também a variação da latência sobre um determinado tempo, isto é, *Jitter*, é outra característica que afeta uma aplicação interativa. Para sistemas interativos em tempo real, como os jogos Multiusuário, a latência entre 0.1 e 1.0 segundos é aceitável . Por exemplo, o padrão DIS especifica que a latência da rede deve ser menor que 100 ms **[NEY97]**.

O limiar para que a latência fique inconveniente para o usuário depende do tipo de aplicação. Em um jogo de estratégia em tempo-real, uma latência mais alta (até mesmo até 500 ms) pode ser aceitável contanto que permaneça estática **[BET01]**. Por outro lado, jogos que requerem muita atenção do usuário como os atiradores de primeira pessoa precisam de uma latência de no máximo 100 ms.

Devemos considerar, também, latência no processamento, pois podem haver usuários em dispositivos com uma menor capacidade de processamento, jogando com usuários em super computadores com uma alta capacidade de processamento.

Outros exemplos de latência em três categorias de jogos, podem ser vistos tais como:

- Jogo de Guerra: latência aceitável de 250 à 500ms.
- *First Person Shooters*: latência aceitável de 40 à 150 ms.
- Jogos de corrida: latência aceitável de 100 a 200 ms, e acima disso, o veiculo torna-se incontrolável **[JAMIN]**.

Largura de Banda

Largura de banda refere-se à capacidade de transmissão de uma linha de comunicação, tal como uma rede. Pode ser definida como uma proporção da quantidade de dados transmitidos por unidade de tempo. Em redes WANs, a largura de banda varia de dezenas de kbps até dezenas de Mbps, dependendo da rede. Em LANs, a largura de banda apresenta-se muito maior, variando de 10 Mbps até 1 Gbps. Entretanto, LANs têm um tamanho limitado e suportam um número limitado de usuários, enquanto que WANs permitem conexões globais.

Os requisitos de largura de banda dependem muito do número e distribuição dos usuários e, é claro, dependem também das técnicas de transmissão que podem ser:

- *Broadcast*, quando as mensagens são enviadas a todos os participantes **[MAC97]**. Obviamente, essa técnica pode gerar problemas caso o número de usuários venha a crescer.

- *Unicast*, em que a comunicação é feita diretamente entre dois *hosts*, e a mensagem é enviada diretamente. Porém, uma vez que a maioria das mensagens é enviada a múltiplos receptores, o *Unicast* desperdiça largura de banda.

- *Multicasting*, em que a comunicação é feita entre uma origem e um conjunto de receptores que pertencem a um mesmo grupo *multicast*. Essa técnica permite entrar em grupos que os interessam, ou seja fazer parte de um grupo *multicast* **[MAC95]**.

Como vimos anteriormente, a largura de banda também pode variar muito dependendo da rede utilizada. Com base nesse argumento, os jogos multiusuários necessitam de um suporte para tratamento da largura de banda, pois podemos encontrar usuários numa rede *wireless*, por exemplo, com uma largura de banda de até 2Mbps se o usuário estiver numa rede 3G (terceira geração), e se estiver parado, jogando com outros usuários que estejam em uma LAN, com largura de banda disponível de até 1 Gbps.

QoS

Jogos que utilizam a infra-estrutura existente da Internet enfrentam a falta de suporte a requisitos de *Quality of Service* (QoS). Em redes de comunicação de dados, QoS implica em um compromisso dos dois lados: o usuário especificará e controlará com rigor seu tráfego, e o provedor de serviço de rede fornecerá garantias de QoS mas deverá descartar o excesso de demanda por parte do usuário. A Internet oferece o outro extremo ao aceitar qualquer demanda, dando uma QoS na base do "melhor esforço" e confiando que os usuários serão comportados para garantir justiça na alocação de recursos de rede **[DAROSA99]**.

3.3. O Modelo de Comunicação dos Jogos

Os jogos podem ser implementados em dois modelos de comunicação de redes: o cliente/servidor e o *peer-to-peer* **[SABA01]**.

No primeiro, os jogadores não trocam os dados diretamente uns com os outros, mas enviam e recebem toda a informação solicitada através de um servidor. A vantagem dessa arquitetura é que o jogador manterá apenas uma simples conexão com o servidor, não

importando a quantidade de usuários conectados. Isso reduzirá o trafego de mensagens na rede uma vez que os usuários as enviam apenas uma vez (para o servidor).

Apresenta como vantagens:

- segurança, difícil de ser violada pois os usuários estarão conectados ao jogo através de uma senha;
- toda a atualização, extensão do jogo e algum tipo de troca serão feitos apenas no servidor, sem a necessidade de troca dos softwares clientes.

As desvantagens são:

- todas as mensagens seguirão sempre pelo caminho (cliente – servidor – outros clientes) e com isso encontrarão o problema da latência da rede, que será alto;
- devido ao aumento no numero de conexões de usuários, o servidor formará gargalo, impedindo conexões e aumentando a latência.

Já o *peer-to-peer* tem como principal característica as trocas de mensagens entre os usuários conectados. Com isso, não há necessidade de um servidor central para as mensagens e muito menos para alguma influência no jogo. Toda comunicação e cálculos são feitos pelos próprios usuários e isso remove o gargalo da rede.

Esse modelo apresenta algumas vantagens:

- maior agilidade com relação ao modelo cliente/servidor, pois as mensagens são entregues diretamente de usuário a usuário evitando gargalo;
- se um usuário perder a conexão os outros usuários permanecem inalterados.

Por outro lado apresentam algumas desvantagens:

- não apresentam segurança, tornando-se vulneráveis a *hackers*;

- em uma arquitetura puramente *peer-to-peer*, ou seja, sem a presença de um ponto central, o tráfego na rede é muito maior que no modelo cliente servidor.

Há que se considerar, ainda, que a maioria dos jogos atuais oferece algum tipo de modo multiusuário, dando aos jogadores a possibilidade de competirem uns com os outros, em um ambiente virtual. Um jogo multiusuário, segundo **[BAR03]**, é aquele em que diversos jogadores jogarão simultaneamente. Seu objetivo é dar aos jogadores a impressão de interação num mundo virtual comum e isso pode variar de um simples jogo de xadrez às batalhas em alta velocidade num jogo de ficção científica. Os jogos serão *massive multiplayer games*, que podem ser manipulados por diversas centenas ou mais de jogadores, ou então por pequenos grupos, limitados de 16 a 32 jogadores e, usualmente, requerem que todos os jogadores estejam presentes antes do início do jogo (como exemplo temos jogos de estratégia, jogos de ação 3D entre outros).

As capacidades multiusuário representam uma parte importante de quase todos os jogos de computador hoje desenvolvidos. Não importa qual o tipo de jogo; jogar contra oponentes humanos reais, é simplesmente mais desafiador e motivante. A realização de tais jogos requer, porém, a comunicação entre os jogadores, e os programadores têm que lidar com algumas limitações causadas pela rede. Dependendo do tipo de jogo, diferentes medidas serão tomadas para resolver esse problema.

Neste projeto, os jogos multiusuário *peer-to-peer* são objeto principal. Isto porque apresentam o desafio de manter um ambiente consistente e totalmente distribuído ao mesmo tempo. Neste capítulo, pois, eles foram descritos e classificados de acordo com seu gênero, assim como os requisitos desses jogos e os jogos *peer-to-peer*, que apresentam como principal característica a necessidade de um ponto de controle, problema cuja solução buscamos propor neste trabalho.

3.4. Jogos Peer-to-Peer

Um jogo *Peer-to-Peer* é aquele que apresenta o modelo de comunicação par-a-par, ou seja, as máquinas são interligadas entre si através de uma rede. O Jogo então é manipulado pelas máquinas dos usuários, que contém um software cliente (o jogo), comunicando-se diretamente numa mesma sessão, sem a presença de um servidor. Por

exemplo, quando um usuário se movimenta, sua máquina enviará atualizações para todas as outras máquinas a ele conectadas **[MSDN]**.

Uma outra definição importante é a definição de sessão. De acordo com **[MSDN]**, uma sessão *peer-to-peer* consiste de uma coleção de usuários conectados por uma rede.

Um jogo *peer-to-peer* pode ser organizado através de duas formas, que são:

- os clientes comunicando-se diretamente, como por exemplo, uma LAN;
- os clientes comunicando-se diretamente, porém passando por um servidor (Jogos baseados na Internet).

Uma vez que uma sessão de um jogo é iniciada, todas as mensagens passarão de usuários a usuários. Caso um servidor esteja envolvido, conforme citado acima, ele manipulará tarefas como atualização de sua lista de usuários membros (participantes de uma sessão) quando um jogador saiu do jogo, ou possibilitando a entrada de um novo usuário que solicitou entrada na sessão (jogo) **[MSDN]**.

O jogo *peer-to-peer* multiusuário contém, porém, um fator crítico, uma vez que apesar de apresentar um modelo de rede descentralizado, necessita de um ponto de controle. Esse ponto de controle será uma das máquinas da rede *peer-to-peer*, tendo como função, por exemplo, o controle da entrada e saída de usuários na sessão de um jogo. Esse ponto de controle poderá apresentar problemas como uma perda de conexão necessitando, pois, receber um tratamento especial. O controle de sessões em aplicações *peer-to-peer,* tratado pela literatura, será visto no próximo item.

3.4.1. O Controle de Sessões em Aplicações Peer-to-Peer

Como visto anteriormente, as aplicações *peer-to-peer*, em especial os jogos, necessitam um controle que trate das mensagens durante uma sessão. Esse controle é atribuído a terminais de usuários que tenham capacidades para suportar o tratamento de mensagens. Esse tipo de controle, porém, pode apresentar problemas, caso ocorra que um terminal controlador venha a perder a conexão.

Para esse tipo de problema é necessário haver a atribuição de um novo controlador, para que o jogo tenha sua continuidade. Isso pode ser tratado de várias formas, como analisaremos a seguir.

De acordo com **[MSDN]**, os jogos *peer-to-peer* transmitem suas atualizações de máquina em máquina para todas as outras conectadas. Porém, como já citamos, anteriormente, esses controladores terão capacidades para suportar migração para outro par em caso de perda de conexão.

Uma das maneiras para que essa migração seja feita é através das APIs *DirectPlay*, que são componentes do Microsoft *DirectX*, e que habilitam a escrita de aplicações de rede, como por exemplo, jogos multiusuário **[MSDN]**. Quando um terminal controlador (*host*) sai de uma sessão, podem ocorrer dois resultados diferentes, que são:

- a sessão termina;
- o controlador transfere todo o controle para outra máquina e a sessão continua.

Quando o controlador perde a conexão, uma variável, indica a migração do "*host*" ou controlador, que é feita por um protocolo manipulado pelo *DirectPlay*. O *Host Migration*, como é chamado, automaticamente transfere as responsabilidades do controlador para outra máquina em potencial. Todos os membros da sessão receberão após a migração, uma mensagem contendo o endereço do novo controlador.

Um outro esquema, descrito em **[GOLD01]**, mostra como é feita a seleção e a migração de um controlador numa rede *peer-to-peer,* usando ciência do contexto, porém, neste caso, uma rede *peer-to-peer* móvel. Os pares conectam-se uns aos outros através de seus vizinhos, os quais agem como roteadores. Sempre que um par estabelece várias conexões e apresenta mais do que *N* vizinhos, ele realiza a eleição do líder **[GAR82]**, para eleger um ponto de agregação (controlador) o qual manipulará as solicitações dos pares em um grupo, diminuindo, assim, o fluxo de mensagens de difusão. Devido à volatilidade da rede, o status desse controlador será monitorado e às vezes até trocado (troca de controlador).

Quando um par comum deseja fazer uma desconexão, ele notifica o controlador e sai. Nesse caso, o controlador atualiza seu registro. Porém, caso um par venha a falhar ou sair sem notificação, o controlador tratará essa situação através de mensagens

(*heartbeats*) e quando um par não responde, o controlador atualizará seu registro, anotando a saída do par **[GOLD01]**.

O par controlador pode executar as seguintes operações:

- inicio: Uma vez que a eleição do líder terminou, o par controlador anuncia sua presença e começa a registrar as informações dos pares assinantes (conectados ao controlador);
- serviço de atualização de informações: o par controlador pode trocar informações com outros controladores para atualizar suas informações;
- desconexão: o controlador pode retirar-se de serviço por diferentes razões e sempre nesse caso, uma nova eleição de líder é realizada:
    - divisão de um grupo de pares: quando o grupo de um controlador se torna muito grande, este retira-se para que haja uma nova divisão dos pares. Com isso, a eleição do líder é novamente chamada;
    - união de grupo: quando o controlador apresenta poucos pares conectados a ele, retira-se e tenta conectar-se a outro controlador, juntamente com seus pares.
    - Outras razões: o controlador pode retirar-se por problemas como, pouca bateria no seu dispositivo, recursos escassos, pouco poder computacional, entre outros.

Baseado nessas duas formas de tratamento de controle, nosso trabalho propõe a atribuição de um controlador e suas possíveis migrações para outras máquinas através de hierarquia de pares, tema este do capítulo 5 deste projeto.

3.5. Considerações Finais

Neste capítulo, foram mostrados os jogos de computador e seus diversos gêneros, assim como também que os jogos podem ser executados em redes cliente/servidor e

outros que podem ser executados em redes *peer-to-peer*. Foram descritos os requisitos de jogos e que para os multiusuário, a latência é um fator de fundamental importância. Como solução propõe-se neste trabalho, o uso dos jogos multiusuário que suportam apenas um pequeno grupo de jogadores, uma vez que será utilizada uma rede *peer-to-peer* híbrida, a qual não suporta uma grande quantidade de usuários. O padrão MPEG-4 e como ele pode controlar mensagens de um jogo através de seus componentes serão objetos do capítulo que segue.

4. O Padrão MPEG-4

O MPEG-4 **[N4264]** é um padrão da ISO/IEC (ISO/IEC JTC1/SC29/WG11) desenvolvido pelo MPEG, responsável também pelo desenvolvimento dos padrões MPEG-1, usado para criação de vídeo interativo em CD-ROM, e MPEG-2, usado para DVD e televisão digital. O MPEG-4 representa um esforço internacional que envolve centenas de pesquisadores e engenheiros do mundo inteiro.

O MPEG-4 suporta aplicações em três áreas: televisão digital, aplicações gráficas interativas (conteúdo sintético) e multimídia interativa (*World Wide Web*, distribuição de e acesso para conteúdo). O padrão provê elementos tecnológicos unificados que habilitam a integração da produção, distribuição e paradigmas de acesso a conteúdos destes três campos **[N4668]**. O MPEG-4 trata os aspectos de composição, sintetização, compressão, sincronização e distribuição de objetos áudio-visuais, além de tratar da descrição das cenas.

O padrão MPEG-4 possui um formato de arquivo binário, o BIFS, que encapsula os segmentos elementares (fluxos de dados) do conteúdo da cena para apresentá-los no visualizador MPEG-4 (*Player*). Isto faz com que o tamanho final do arquivo, contendo o ambiente virtual, seja pequeno e, portanto, mais rápido para ser transportado via rede de comunicação **[N4264]**

Uma cena no padrão MPEG-4 é composta por objetos áudio-visuais que são referidos como Fluxos Elementares. Os Fluxos Elementares são dados encadeados recebidos da saída do *buffer* de um codificador, independente de seu conteúdo. A integridade de um Fluxo Elementar é assumida ser preservada de fim-a-fim entre dois sistemas. Um Fluxo Elementar contém uma representação codificada do conteúdo de dado, podendo ser: objetos de áudio ou de vídeo, informações de descrição da cena (BIFS), informações de envio para

identificar fluxos ou para descrever as dependências lógicas entre os fluxos (descritores) e, ainda, informações relacionadas ao conteúdo do objeto (Fluxos OCI). Tais Fluxos Elementares são produzidos e consumidos pelas entidades: camada de compressão, codificador e decodificador, **[DUARTE02]**.

A representação de uma cena MPEG-4 se faz através dos BIFS que fornecem um *framework* completo para as máquinas de apresentação nos terminais MPEG-4. O BIFS suporta uma mistura de várias mídias MPEG-4, como gráficos 2D e 3D, tratamento de interatividade, e negociação com mudanças locais ou remotas da cena. Ele foi projetado como uma extensão da linguagem VRML 2.0 em formato binário. Todos os objetos audiovisuais contidos numa cena MPEG-4 são representados na forma de uma árvore hierárquica.

Atualmente o padrão MPEG-4 **[N4264]** está dividido em sete frentes de trabalho: *Systems, Visual, Audio, Conformance testing, Reference software,* DMIF e *Optimized software* for MPEG-4 *visual tools*.

O Sistema MPEG-4 descreve um sistema para comunicação de cenas audiovisuais interativas, incluindo os seguintes elementos:

- a representação codificada de objetos naturais ou sintéticos, 2D/3D, que podem ser manifestados em forma de áudio e/ou visualmente (objetos audiovisuais);
- a representação codificada do posicionamento espaço/temporal de objetos audiovisuais, bem como seus comportamentos em reposta à interação (descrição da cena);
- a representação codificada das informações relacionadas ao gerenciamento de fluxos de dados (sincronização, identificação, descrição e associação de conteúdo);
- uma interface genérica das funcionalidades da camada de entrega dos fluxos de dados;
- um motor de aplicação para controle programático no *player*: formato, entrega do Java *byte code* baixados (*downloaded*) bem como seu ciclo de execução e comportamento através das APIs e;

- um formato de arquivo para conter a informação de mídia da apresentação MPEG-4 em um formato flexível e extensível para facilitar a troca, gerenciamento, edição e apresentação das mídias.

A completa operação de um sistema de comunicação das cenas audiovisuais pode ser descrita como:

- No transmissor, as informações das cenas audiovisuais são comprimidas, suplementadas com informações de sincronização e passadas à camada de entrega que multiplexa as cenas audiovisuais em um ou mais fluxos binários codificados que são transmitidos ou armazenados.

- No receptor, estes fluxos são desmultiplexados e descomprimidos. Os objetos de mídia são compostos de acordo com a descrição da cena e as informações de sincronização e apresentados ao usuário final, que irá interagir com a apresentação. Informações de interação se processarão, localmente, ou enviadas ao transmissor.

O Sistema MPEG-4 define as seguintes ferramentas:

- um modelo de terminal para o gerenciamento de buffer e tempo;

- uma representação codificada de informações de descrição da cena áudio-visual interativa (BIFS);

- uma representação codificada de meta dados para a identificação, descrição e dependências lógicas dos fluxos elementares (descritores de objeto ou outros descritores);

- uma representação codificada de sincronização de informação (SL);

- uma representação codificada para descrição de informações de conteúdo áudio-visual (OCI);

- uma interface para sistemas de gerenciamento e proteção de propriedades intelectuais (IPMP);

- uma representação de fluxos elementares individuais em um fluxo simples (*FlexMux*) e;

- um motor de aplicação (MPEG-*Java* – MPEG-J).

4.1. A Arquitetura do Padrão MPEG-4

A representação de informações que descreve uma cena áudio-visual interativa, contendo informações áudio-visuais e informações associadas à descrição da cena, foi especificada no *Final Committee Draft of Internation Standard*. A entidade que recebe e apresenta a representação codificada de uma cena áudio-visual interativa é genericamente citada como um "terminal áudio-visual", que pode corresponder a uma aplicação *standalone* ou ser parte de um sistema de aplicação **[N2201]**.

As operações básicas realizadas no terminal são as que seguem:

- **obter acesso à sessão inicial** - O terminal obtém acesso às informações a partir de uma sessão inicial provida da disponibilização do conteúdo das informações;

- **estabelecer contextos de sessão e uma interface de entrega** - Utilidade para estabelecer contextos de sessão e uma interface para a camada de entrega o qual separa fluxos como o meio de armazenamento e transporte;

- **prover a localização dos fluxos elementares** - Provê a localização de um ou mais fluxos elementares os quais são partes da representação do conteúdo previamente codificadas. Muitos desses fluxos podem ser agrupados, usando uma ferramenta de multiplexação como, por exemplo, *FlexMux tool*.

Cada fluxo elementar contém somente um tipo de dado. Como exemplo pode-se citar fluxo de imagens (JPEG) ou áudio (G723). Os fluxos elementares são decodificados usando seus decodificadores de fluxo específicos. Os objetos áudio-visuais são compostos de acordo com as informações da cena e apresentados para o terminal fazer, futuramente, a interação com os usuários. Todos esses processos são sincronizados de acordo com o Modelo

de Decodificação do Sistema, usando a informação de sincronização estabelecida na camada de sincronização (SL) **[N4264]**.

A arquitetura composta por seus componentes, como ilustrado na Figura 10, são detalhados a seguir.

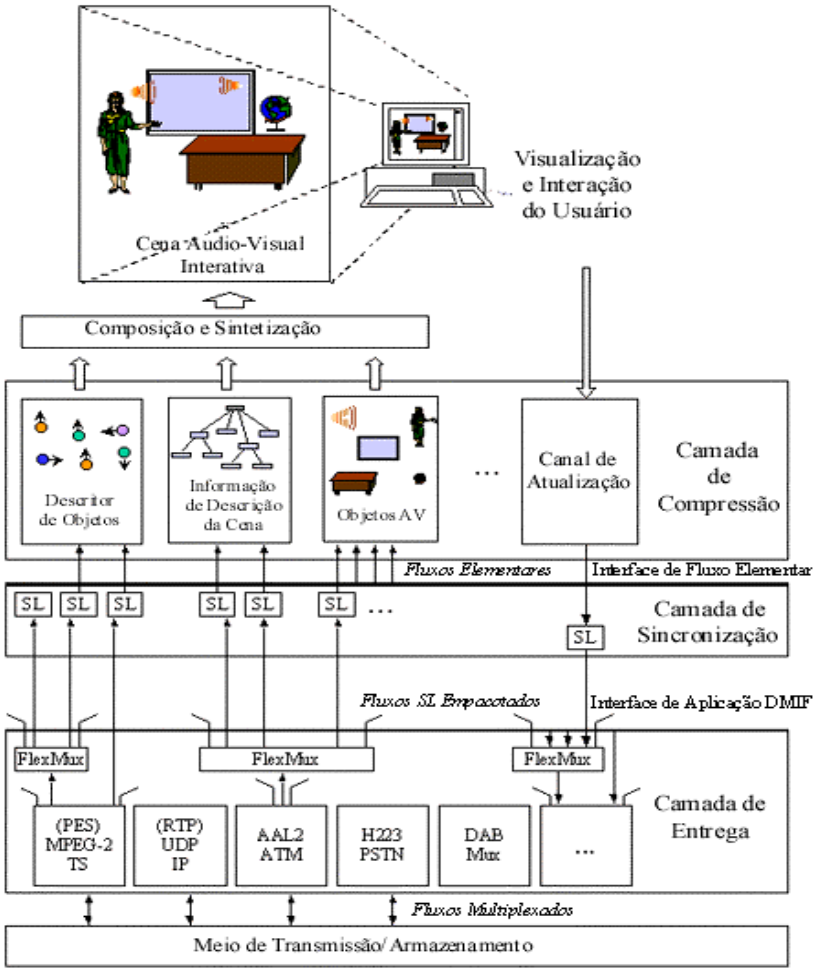


Figura 10 - Arquitetura do Terminal MPEG-4 [N4264].

Modelo de Terminal – Modelo de Decodificador do Sistema (SDM)

O SDM fornece uma visão abstrata do comportamento de um terminal, conforme a especificação do padrão MPEG-4. Sua proposta é permitir a um transmissor predizer como o receptor se comportará em termos de gerenciamento de *buffer* e

sincronização, quando ocorrer a reconstrução da informação áudio-visual que envolve a apresentação **[N4264]**.

O SDM especifica:

- a interface por acessar fluxo de dados desmultiplexados (*Stream Multiplex Interface*);
- decodificação de *buffers* para dados comprimidos para cada fluxo elementar;
- o comportamento de decodificadores de objeto de mídia;
- memória de composição para dados descomprimidos para cada objetos de mídia e o comportamento de produção para o compositor.

Camada de Entrega - Multiplexação de Fluxos

A Multiplexação de Fluxos é usada para transmitir e armazenar conteúdo. Somente a interface dessa camada é especificada (denominada Interface de Aplicação DMIF (DAI)), e define não somente uma interface para a entrega de dados encadeados (*streaming data*), mas também informações de sinalização requeridas para o estabelecimento e encerramento de canais e sessões.

Camada de Sincronização de Fluxos - (SL)

Os fluxos elementares são quaisquer tipos de dados encadeados.

Um fluxo elementar pode conter:

1. ***Descritores de Objetos:*** O propósito da descrição de objetos é identificar e descrever fluxos elementares e associá-los adequadamente a uma descrição da cena auditivo-visual. Os descritores de objeto são carregados como fluxos elementares. Cada descritor de objeto é nomeado com um identificador (O*bject Descriptor ID*) que é único

dentro de uma extensão de nome definida. Este identificador é usado para associar objetos áudio-visuais na descrição da cena, como um descritor particular de objeto, assim os fluxos elementares relacionaram, em particular, àquele objeto. Descritores de fluxos elementares também incluem informação sobre o formato de codificação, informação de configuração para o processo de decodificação e a camada de sincronização, como também qualidade de exigências de serviço para a transmissão do fluxo e identificação de propriedade intelectual.

2. ***Informações de conteúdos de Objetos*:** Informações de conteúdos de objetos (OCI) carregam a informação descritiva sobre os objetos áudio-visuais. Os descritores de conteúdos principais são: descritores de classificação do conteúdo, descritores de palavras chaves, descritores de classificação, descritores de linguagens, descritores textuais, e descritores sobre a criação do conteúdo. Podem, ainda, ser incluídos descritores de OCI diretamente no descritor de objeto relacionado, ou a um descritor de fluxo elementar, ou, se for variante de tempo, pode ser levado por si só em um fluxo elementar. Um fluxo de OCI é organizado em uma sucessão de entidades pequenas, sincronizadas, chamados eventos, os quais contêm um conjunto de descritores OCI. O fluxo OCI pode ser associado a descritores de objetos de múltiplos.

3. ***Fluxos da Descrição da Cena*:** A descrição endereça a organização de objetos audiovisuais numa cena, em ambos atributos espacial e temporal. A descrição consiste em uma hierarquia codificada (árvore) de nós com atributos e outras informações. Os nós da folha nesta árvore correspondem aos dados áudio-visuais elementares, considerando que nós intermediários se agrupam a este material para formar objetos áudio-visuais, e outras operações em objetos áudio-visuais (nós da descrição da cena). A descrição de cena pode evoluir com o passar do tempo usando atualizações onde, mecanismos de interatividade são integrados com a informação da descrição da cena, na forma de recursos de eventos e rotas, bem como sensores (nós especiais que podem ativar eventos baseados em condições específicos). Estes recursos de eventos e rotas

fazem parte de nós da descrição da cena, e assim permite a união de comportamento dinâmico e interativo com a cena específica.

4. ***Fluxos Áudio-visuais*:** Os dados audiovisuais reconstruídos são projetados, disponibilizando o processo de composição para uso potencial durante a retribuição da cena.

Tais fluxos são transportados como fluxos SL empacotados (*SyncLayer-packetized – SL-Packet*) na Interface de Aplicação DMIF, fornecendo informações de tempo e sincronização, bem como informações de fragmentação e de acesso aleatório. A camada de sincronização extrai esta informação de tempo para capacitar a decodificação sincronizada e, subseqüentemente, a composição do dado do fluxo elementar.

Camada de Compressão

A camada de compressão recebe o dado em seu formato codificado e desempenha as operações necessárias para reconstruir a informação original. A informação decodificada é então usada pelo terminal de composição, renderização e subsistemas de apresentação e, finalmente, é entregue ao visualilzador ou Player MPEG-4 para uso e interação do usuário.

4.2. O Padrão MPEG-4 MU

O Padrão MPEG-4 MU trata ambientes virtuais 3D compartilhados, ou ainda um ambiente virtual 3D onde há mais que um usuário interagindo com o ambiente e com outros usuários em tempo real. As ações multiusuário são bem simples e o seu relacionamento estrutural pode ser definido como **[LIVI97]**:

- modificações na cena a ser iniciadas por qualquer cliente;
- modificações com potencialidade para serem sincronizadas em todos os clientes;

- para todas as modificações, cada objeto, ou é a sua origem, com a responsabilidade para espalhar a sua notícia de modificação, ou é um receptor de notícias de modificação, e deve realizar a ação apropriada;
- nem todas as modificações são realizadas de uma só vez; a otimização do desempenho depende de saber o que é mais provável mudar em seguida;
- nem todas as notícias de modificação precisam ser comunicadas, a otimização depende também de ter conhecimento de quem necessita saber o que e o como.

Um dos maiores desafios presente nos ambientes virtuais 3D compartilhados é manter a consistência das informações entre os múltiplos usuários.

Quando se fala em ambientes virtuais 3D compartilhados, alguns conceitos que a ele estão relacionados são muito importantes, pois, nesses ambientes estão compartilhados não só o espaço virtual, mas, também, os objetos contidos neles. Sendo assim, pertencem a um ambiente virtual 3D compartilhado, seções e zonas de compartilhamento e objetos com estado compartilhado ou não **[N3205]**.

De acordo com a especificação do MPEG-4 MU **[M3874]**, uma seção é um recipiente para uma ou mais zonas de compartilhamento; uma zona é um recipiente para um ou mais objetos compartilhados ou não; e um objeto compartilhado pode ser definido como sendo qualquer objeto no ambiente virtual 3D que possui um estado de compartilhamento. Dessa forma, um ambiente só é declarado como sendo compartilhado através da definição de uma seção de compartilhamento no ambiente.

Após a inserção de uma seção de compartilhamento, várias zonas podem ser definidas, contendo vários objetos compartilhados. A inserção de zonas de compartilhamento em um ambiente virtual 3D, possui várias características que favorecem tal delimitação. As principais são: **[ROEH01]**:

- melhora o desempenho de renderização;
- redução dos requisitos de memória, até mesmo para mundos extensos;
- agilização do carregamento dos mundos;
- a habilidade para designar propriedades (iluminação, nevoeiro, etc) diferentes para partes diferentes do mundo;
- maior gerenciamento na detecção de colisão;
- auxílio na limitação de atualizações num sistema multiusuário;

Desta forma, a delimitação de ambientes virtuais 3D, em seções e zonas de compartilhamento, auxilia tanto no projeto e desenvolvimento de ambientes com grandes extensões, como nas questões de gerenciamento de atualizações.

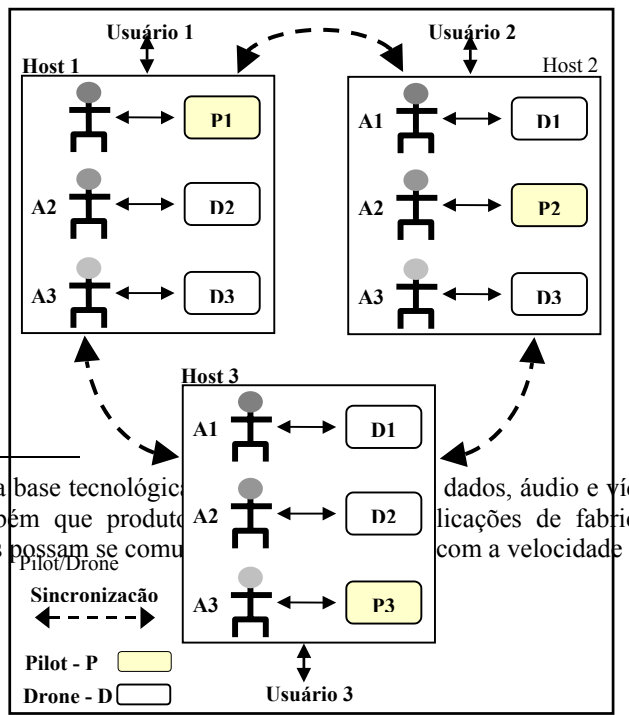
4.2.1. Arquitetura do Padrão MPEG-4-MU

O MPEG-4 MU, tem como objetivo definir uma estrutura para suporte à implementação de aplicações multiusuário e suas soluções. A interação entre os múltiplos usuários se define de acordo com os seguintes itens **[W4415]:**

- objetos compartilhados: objetos arbitrários que podem ser declarados como compartilhados e serem carregados e incorporados em ambientes MU;
- mundo: ambiente MU criados por diferentes provedores de conteúdo com possibilidade de ser visto por diferentes navegadores MU;
- navegadores (*Browsers*)*:* qualquer navegador a ser usado para visualizar e interagir com qualquer conteúdo MU;

Esses itens referentes à interoperabilidade implicam em que o formato de um conteúdo MU e o protocolo de mensagens relacionados aos dados multiusuários são padronizados.

A arquitetura para o tratamento de múltiplos usuários, em um ambiente MPEG-4, é baseada na especificação ITU H.323[1] e no mecanismo *Pilot/Drone*, proposto na



[1] O padrão H.323 provê uma base tecnológic dados, áudio e vídeo em redes baseadas no protocolo IP. Permite também que produto licações de fabricantes diferentes possam interoperar e que os usuários possam se comu com a velocidade da rede **[PINH00]**.

especificação do *Living Worlds* **[LIVI97]**, que define um *Pilot* como sendo cópias mestres dos nós nos clientes ou servidores. Define, ainda, um *Drone* como sendo instancias locais de nós que contatam seus correspondentes *Pilots* quando mudanças são distribuídas. Sendo assim, os *Drones* só existem no grafo da cena de um terminal cliente. Como vemos na figura 11 o retângulo amarelo, representa o *Pilot* P correspondente de cada avatar; o retângulo branco, as réplicas de cada *Pilot*, denominadas *Drones* e as linhas pontilhadas a sincronização entre os correspondentes *Pilots* (P*n*) e seus *Drones* (D*n*).

Figura 11 – Mecanismo de Sincronização entre Pilot e Drone [adaptada de W4415].

O conceito do mecanismo *Pilot/Drone*, ilustrado na figura 11, pode ser entendido da seguinte forma: todos os objetos compartilhados possuem *pilots*, ou seja, são capazes de originar comportamentos e mudanças de estados; todos os objetos compartilhados são inicialmente atribuídos como sendo *drones* que replicam localmente somente comportamentos e mudanças de estados originados em outro local. A exceção é o avatar, que possui sua chave atribuída por definição como ligada (*on*), ou seja, ele é designado como *pilot*.

O *avatar* tem o seu estado atribuído para *drone* numa situação em que o usuário deixa de pilotar sua representação no ambiente virtual 3D, como, por exemplo, receber uma ação de um outro *avatar*.

O mecanismo *Pilot/Drone* utiliza mensagens de solicitação/atualização, para garantir sincronização e consistência do ambiente virtual 3D entre todos os terminais MPEG-4. Há dois cenários possíveis para garantir a sincronização: *peer-to-peer* e centralizado (Cliente-servidor) usando um *BookKeeper* **[W4415]**.

Para a sincronização do mecanismo *Pilot/Drone,* há componentes que controlam e gerenciam mensagens de requisição e atualização dos estados/comportamentos dos objetos compartilhados e para o gerenciamento desse controle, a arquitetura atual do padrão MPEG-4 foi estendida, como comprovaremos a seguir.

4.2.2. Componentes da Arquitetura MPEG-4-MU

A Figura 12 **[W4415]** representa os componentes da arquitetura MPEG-4 MU, especificados pelo padrão e aqueles que são deixados para os fabricantes de navegadores e de sistemas para a *web*, tais como o Controlador de Sessão *MUTech* (*MUTech Session Controller* - MSC), o *MUTech Bookkeeper* (MBK) e o Manipulador de Mensagem *MUTech* (*MUTech Message Handler* - MMH).

MUTech Session Controller (MSC)

Iniciando a descrição da arquitetura, apontamos o primeiro módulo que é parte do plano de controle. O MSC configura canais de controle, associando identificadores de clientes (*clientIDs*), mantendo uma visão geral das capacidades e endereçamento IP, informações importantes para roteamento de mensagens. Cada cliente terá um canal de controle permanente no MSC, que oferecerá aos clientes a lista das zonas existentes, assim como notificações de serviços de clientes que entram/saem, adicionar/remover zonas. Uma lista de quais zonas os clientes estão participando será mantida. Essas informações podem ser requisitadas pelos clientes, e usadas, também, pelo *MUTech Bookkeeper*. Informações sobre as capacidades dos clientes serão encontradas no MSC. O MSC controla acesso às zonas numa sessão, e manipula pedidos de clientes que querem entrar ou sair de uma zona. O MSC inicia o *MUTech Bookkeeper*.

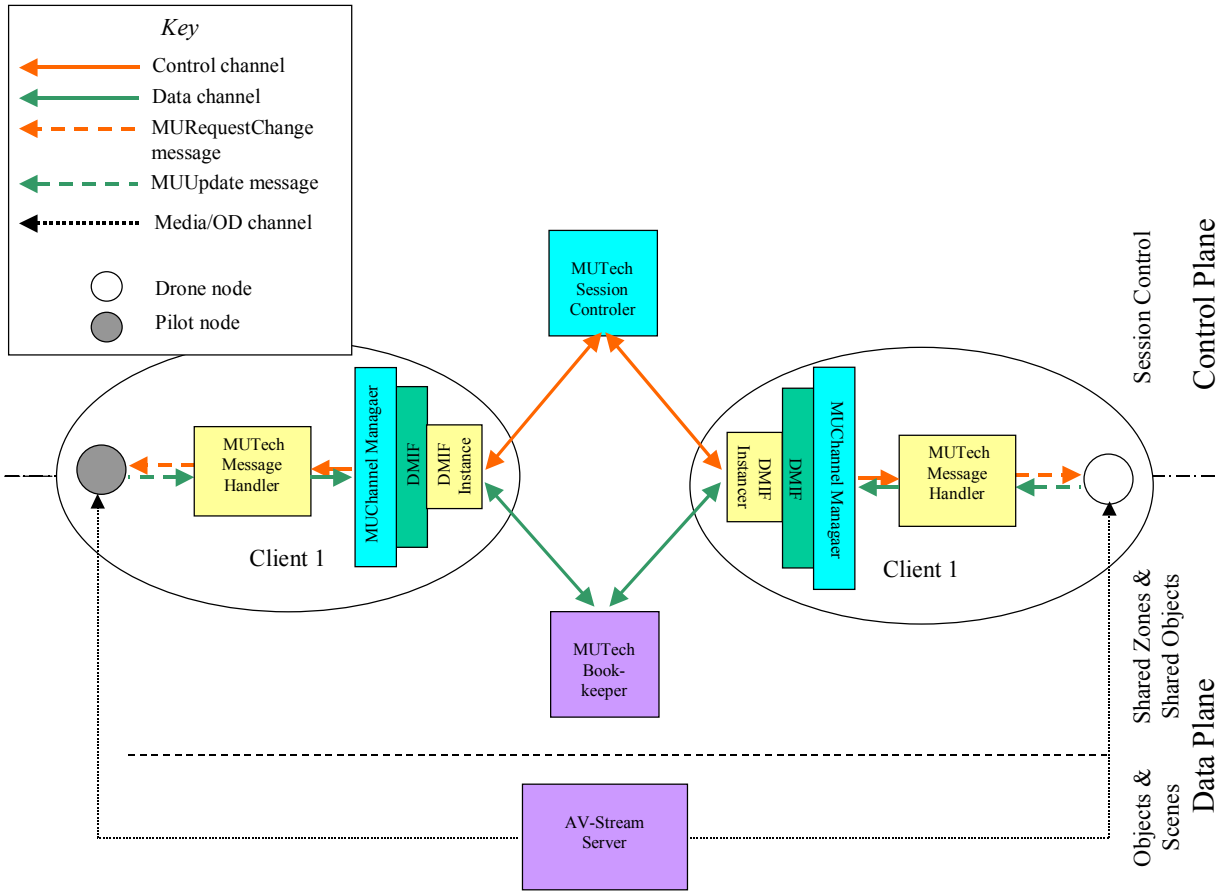


Figura 12 - Arquitetura Pedido/Atualização [W4415]

MUTech Bookkeeper (MBK)

O *MUTech BookKeeper* é um módulo que faz parte do plano de dados. Ele é responsável pela sincronização do estado dos objetos compartilhados, além de rotear mensagens *Pilot-drones*. Ele pode assumir a responsabilidade de pilotar os objetos compartilhados com uma zona, mas também pode transferir a pilotagem de uma zona contendo o pedido de um cliente. O MBK saberá onde o nó de um *Pilot* está a todo o momento. O MBK mantém uma cena gráfica completa contendo todas as zonas e nós juntamente com a sessão.

Se o MBK está pilotando um objeto:

- todas as mensagens *RequestChange* (solicitação de mudança) para esse objeto são processadas pelo MBK;

- o MBK gera as *UpdateMessages* (mensagens de atualização) necessárias e as distribui para todos os clientes que estão compartilhando um objeto;

Se o MBK não está pilotando um objeto:

- se o MBK recebe uma mensagem *RequestChange* de um nó que não está sendo pilotado, o MBK dirige essa mensagem a um cliente pilotando o nó;
- recebida uma mensagem *UpdateMessages* de um cliente pilotando o nó, o MBK distribui a mensagem para todos os outros clientes que estão compartilhando o nó que foi alterado.

MUTech Message Handler (MMH)

Convém descrever, ainda, o manipulador de mensagens que tem como funções principais:

- oferecer uma interface ao terminal do cliente para envio e recebimento de mensagens MU;
- empacotar diversas mensagens MU que irão juntar-se para formar o pacote SL;
- endereçar as mensagens com o correto identificador do cliente antes de enviá-la ao longo do canal *Request/Update* aberto no DMIF.

O MMH apresenta uma interface denominada *MAI, (Multi-user Application Interface*), interface de aplicação MU, a qual seria usada por outros componentes na aplicação, como por exemplo, o MPEG-J. Isso oferece uma conveniente forma de tais módulos afetarem a cena e assegurar que as modificações sejam distribuídas. O MPEG-J poderia, tipicamente, enviar uma solicitação relacionando os nós a serem modificados e o pedido roteado via MMH até o *Pilot* para validação **[W4415]**.

MUChannelManager (MCM)

O gerenciador de canal é um componente responsável por configurar fluxos de controle e dados entre um cliente e o MSC, através da interface DAI. O MCM oferece uma interface conveniente para o MMH acessar funcionalidades no DAI. Atributos do canal serão descritos usando um descritor de objetos OD e um descritor de fluxos elementares ESD, que permitirá a inclusão de parâmetros como direção, informações de qualidade de serviço e informações de conteúdo de objetos a serem especificadas na configuração do canal **[W4415]**.

As mensagens MU serão enviadas via MCM e o DMIF através de um canal aberto no servidor DMIF e nos módulos dos clientes. Os fluxos MU serão associados através de descritores de objetos a um tipo de *stream* MU. Cada nó *MUSession* tem um descritor de objetos anexado a uma URL para o *MUSessionController* associado.

O OD apresenta dois ESDs os quais:

- Um ESD descreve o fluxo contendo *MU-commands* de modo *downstream* (servidor para cliente)

- Um segundo ESD descreve o fluxo contendo *MU-commands* de modo *upstream* (cliente para servidor). Possui uma *flag streamDependenceFlag* de dependência de fluxo apontada para *true* e o campo *dependsOn_ES_ID* apontado para o *ES_ID* do canal de *downstream*.

O servidor de fluxos DMIF oferece suporte em tempo real para clientes MU através das APIs *LIVE* e *SINK* e também DLLs. A comunicação é feita de cliente para servidor e servidor para cliente, respectivamente. Cada cliente conectado ao servidor de fluxos tem suas próprias ou compartilhadas MU_LIVE e MU_SINK DLLs. Para simplicidade da arquitetura, é previsto a implementação MU usando 2 (duas) DLLs por cliente conectado, como visto em **[W4415]**.

Instância MUTech (*MUTech Instance*)

É uma instância DMIF que permite a uma aplicação abrir canais de mensagens e transferir mensagens de/para MSC. Essa instância estabelece um acordo com o DPI, e faz a comunicação de forma transparente com o MSC para a aplicação. Ela também pode suportar

comunicação ponto-a-ponto entre os clientes para que o MSC e o MBK não necessitem ser usados para roteamento das mensagens de solicitação e atualização (*Request* e *Update*).

O *MUTech Instance* será implementado por um DMIF existente, suportando *up-* e *downstreaming* e outra opção é a entrega de fluxos elementares solicitados pelo *MUTechMessage Handler* multiplexado juntamente com a sessão principal de uma *AV-Scene*.

Servidor de fluxo (*AV Stream Server*)

Esse módulo é um local seguro na arquitetura para um servidor que supre os nós da cena com fluxos MPEG-4 apontados pelo descritor de objetos. Isso permite o uso de *unicast* ou *multicast*, arquivos locais, entre outros fatores para distribuição e compartilhamento de fluxos entre os terminais clientes, sem ter que mudar a descrição da cena.

MUSession

Cada cena contém um nó *MUSession,* que especifica a URL/ID do descritor de objeto onde o *MuTech Session Controller* pode ser contatado.

4.2.3. A Transferência de Pilotagem

A Transferência de Pilotagem é uma forma eficiente de remover, temporariamente, o atraso de rede que os clientes experimentam quando estão participando em uma sessão multiusuário. Ela pode ser útil e, talvez, também crucial para alguns tipos de aplicações, como, por exemplo, desenvolvimento de projetos. Combinando o uso de *locking* e a transferência do *Pilot* teremos também alguns benefícios, como o controle de concorrência.

O MBK se responsabilizará pela pilotagem dos objetos compartilhados. Quando isto não acontece, o *Pilot* é um dos terminais-cliente participantes. Apresentamos abaixo, os dois casos de pilotagem do objeto:

- **Quando o MBK pilota um objeto**
    - todas as mensagens de requisições para um objeto são processadas pelo MBK;

- o MBK gera mensagens de atualização (*UpdateMessages*) e então as distribui para todos os clientes que compartilham o objeto.

- **Quando o MBK não pilota um objeto**
  - o MBK recebe uma mensagem de requisição (*RequestChange*) de um terminal-cliente que não é o *Pilot* e repassa essa requisição para o terminal-cliente que é o *Pilot* do nó objeto;
  - o MBK recebe mensagens de atualizações (*UpdateMessages*) do terminal-cliente que pilota o nó e então distribui as mensagens para todos os terminais-cliente que compartilham o objeto.

4.2.4. Propagação das Mensagens

A propagação das mensagens entre *Pilots* e *Drones* ocorre de forma diferente, dependendo do cenário em que se encontram em um determinado instante de tempo, conforme descrição que segue:

- o círculo amarelo representa um objeto compartilhado em uma *MUZone* com o campo *pilot = true*;
- os círculos azuis são objetos compartilhados em uma *MUZone* com o campo *pilot = false*, portanto estes objetos são os *Drones* de seus respectivos *Pilots*.

Cenário 1: *Pilot* no Servidor

1. O cliente 1 interage com o nó (objeto compartilhado), efetuando alguma mudança de estado. Este nó é um *Drone*.
2. O *Drone* então, envia uma mensagem de requisição de alteração para o MBK, o qual passa esta mensagem para o correspondente nó *Pilot*.
3. Uma mensagem de atualização é, a seguir, enviada do *Pilot* para seus *Drones* através do MBK, e o *Pilot* então se atualiza.

O atraso na rede para esta operação é, aproximadamente, duas vezes o atraso na rede para ambos os clientes, como ilustra a 13 .

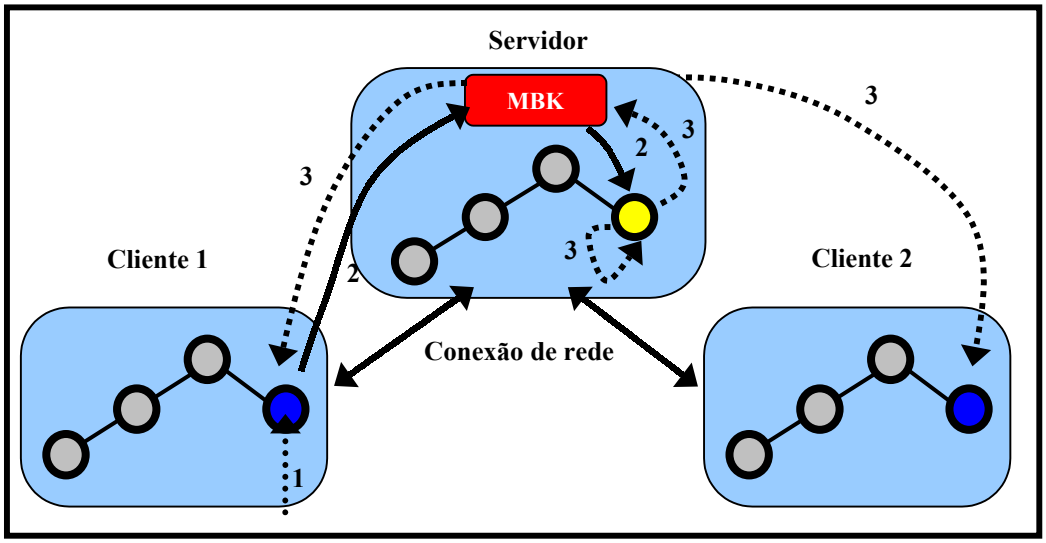


Figura 13 – *Pilot* no Servidor

Cenário 2: *Pilot* no Terminal-Cliente 1

1. O cliente 1 interage com o nó (objeto compartilhado), mudando seu estado. Este nó é um *Pilot*.

2. O *Pilot*, envia, então, uma mensagem de atualização para o MBK, o qual passa esta mensagem para os correspondentes nós *Drone*. Nesse mesmo tempo, o *Pilot* se atualiza também.

O atraso para esta operação é desprezível para o terminal cliente 1, conforme a figura 14.

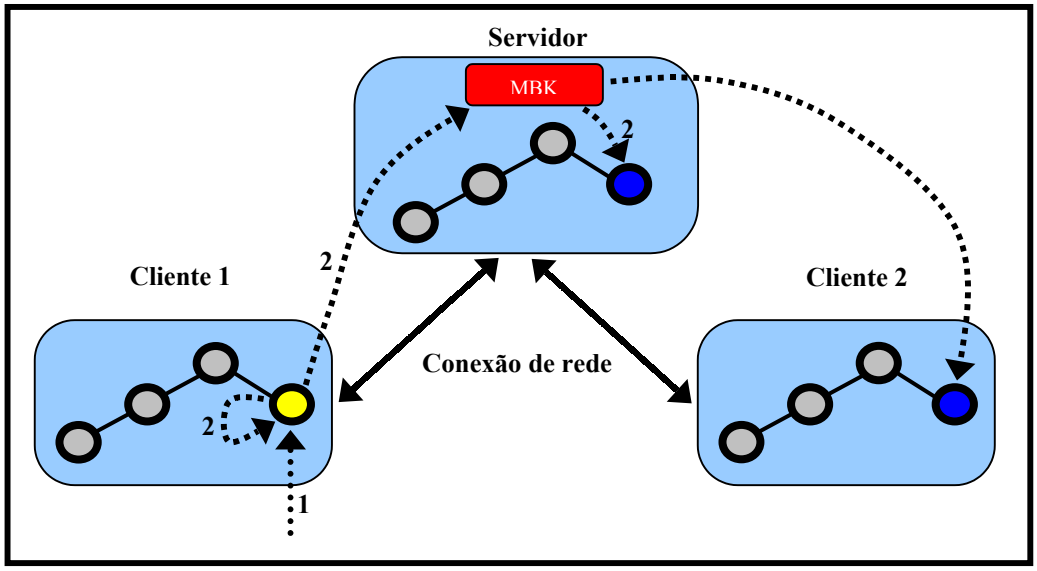


Figura 14- *Pilot* no Terminal-Cliente 1.

Cenário 3: *Pilot* no Terminal Cliente 2

1. O cliente 1 interage com o nó (objeto compartilhado), efetuando uma mudança no estado. Este nó é um *Drone*.

2. O *Drone* por sua vez, envia uma mensagem de requisição de alteração para o MBK, o qual passa esta mensagem para o correspondente nó *Pilot* (agora no terminal cliente 2).

3. Uma mensagem de atualização é enviada, então, do *Pilot* para seus *Drones* através do MBK, e o *Pilot* então se atualiza.

No terminal-cliente 1 o atraso para esta operação é aproximadamente quatro vezes o atraso da rede. O terminal-cliente 2 receberá uma atualização com um atraso de cerca de duas vezes o atraso da rede depois de iniciada uma mudança no terminal-cliente 1, ilustrado na 15.

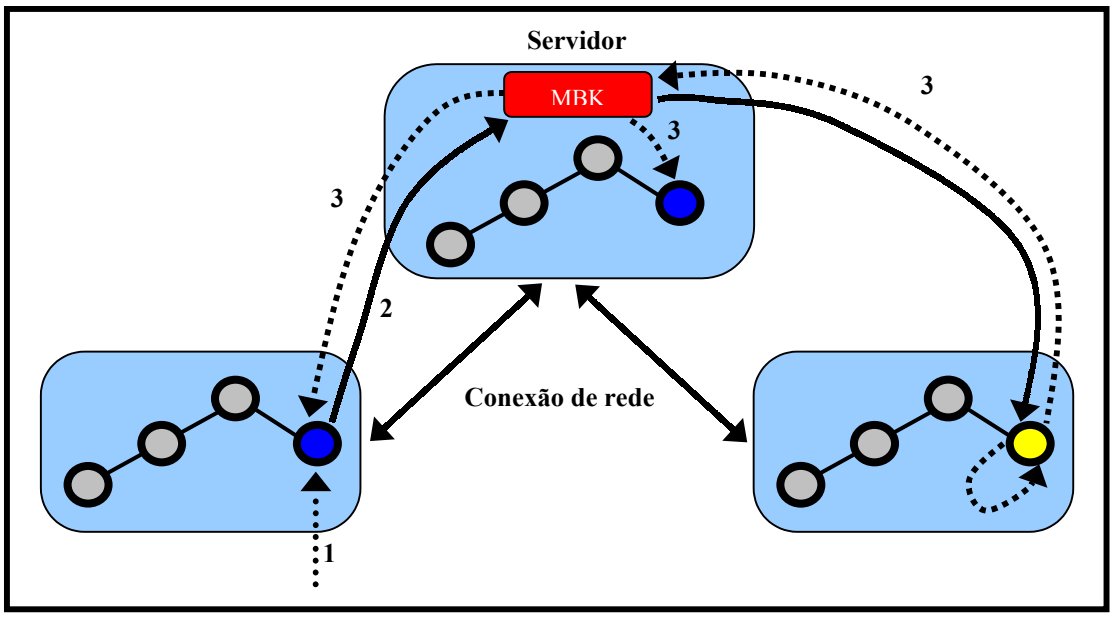


Figura 15- *Pilot* no Terminal-Cliente 2.

As mensagens que são propagadas de acordo com cada cenário discutido a pouco são fundamentais para a sincronização entre *Pilot* e *Drone* no Ambiente Virtual Multiusuário. No Padrão MPEG-4 MU todas as mensagens são encapsuladas em forma de fluxos que trafegam entre clientes e servidor. Estas mensagens são definidas conforme o fluxo *MUCommandStream* **[W4415]** o que é um novo tipo de fluxo definido para transportar mensagens multiusuário.

4.2.5. Modelo de Comunicação *Peer-to-Peer*

No cenário *peer-to-peer,* a sincronização ocorre no terminal que pilota o objeto compartilhado, onde um *pilot* recebe mensagens de requisição de mudanças dos *drones*. Ele aceita ou nega as atualizações e distribui os comandos de atualização para si e para todos os seus *drones*. Pode também receber requisições locais para mudanças (via API MPEG-J) e tratá-las da mesma forma. O *drone* replica as modificações ocorridas no objeto, pelo envio de requisições de mudanças dos campos ao *pilot*.

4.2.6. Modelo de Comunicação Cliente-Servidor

De acordo com **[W4415],** o *Bookkeeper* é responsável por manter o estado dos objetos compartilhados. Um *bookkeeper* recebe mensagens de requisição de mudança e distribui comandos de atualização BIFS como resposta.

Um cliente envia mensagens de requisição de mudança ao *bookkeeper*, o qual recebe tais mensagens e atua de acordo com os privilégios de acesso. O *bookkeeper* distribuirá as mensagens de mudança de modo a sincronizar o estado do objeto compartilhado entre os clientes subscritos. Neste cenário, o *bookkeeper* gerencia os deveres do *pilot*. Um cliente solicitará a propriedade de um objeto pelo ganho dos direitos de acesso e transferência de propriedade. O proprietário concederá ou negará acesso ao objeto de posse. O *bookkeeper* fornecerá serviços de objetos compartilhados, como persitencia, transações para transferência de propriedade e notificação de adição/remoção de eventos.

4.3. Aplicações

Existem cinco tipos de aplicações classificadas pelo MPEG-4. Estas aplicações são selecionadas por três critérios**[VIDAL97]**:

- **Limites de tempo** - aplicações podem ser em tempo real ou não. Uma aplicação em tempo real é simultaneamente adquirida, processada, transmitida e potencialmente usada pelo receptor.

- **Simetria das facilidades de transmissão** - aplicações são classificadas como simétricas ou não. Aplicações simétricas são aquelas em que equivalentes facilidades de transmissão estão disponíveis em ambos os lados do enlace de comunicação.

- **Interatividade** - aplicações são interativas ou não. Aplicações interativas são aquelas em que o usuário tem controle individual da apresentação, ou somente no nível de controle da mídia de armazenamento ou também no escalonamento da seqüência do fluxo da informação dependendo das escolhas do usuário.

As aplicações, por sua vez, são apresentadas em cinco tipos que são **[VIDAL97]**:

Aplicação Classe 1 (Tempo real / Simétrica / Interativa)

O usuário tem controle individual sobre a apresentação. A quantidade de dados transmitidos é a mesma em ambas as direções. Exemplos: videoconferência, vídeotelefonia, consulta remota a especialista com simetria.

Aplicação Classe 2 (Tempo real/ Assimétrica / Interativa)

As aplicações são interativas, mas o receptor envia uma pequena quantidade de dados independente dos dados enviados pelo transmissor. Exemplos: controle ou monitoramento remoto e consulta remota assimétrica à especialista.

Aplicação Classe 3 (Não tempo real / Simétrica / Interativa)

O usuário pode controlar o fluxo de dados através do canal de dados de controle. Aplicação típica é o correio eletrônico.

Aplicação Classe 4 (Não tempo real / Não simétrica / Interativa)

O usuário tem controle individual da apresentação sobre a informação armazenada em banco de dados. Exemplos: jogos, vídeo sob demanda, *teleshopping*, noticiário eletrônico, etc.

Aplicação Classe 5 (Não tempo real / Não simétrica / Não interativa)

O usuário não tem controle sobre apresentação. Aplicações típicas são apresentações multimídias, onde a interatividade não existe.

O MPEG-4 fornece, ainda, o suporte à representação baseada em objeto, como visto em **[VIDAL97]**. Os padrões correntes de compressão de vídeo transmitem um quadro inteiro de vídeo em um único fluxo de bits *(bitstream)*. O MPEG-4 codificará objetos audiovisuais em quadros separadamente. Os objetos serão compostos em um quadro no decodificador. Objetos codificados separadamente fornecem três benefícios:

- **reusabilidade** - a abordagem orientada a objeto permite aos autores reusarem material audiovisual mais rapidamente;
- **escalabilidade** - objetos podem ser codificados usando diferentes resoluções espaciais e temporais. A largura de banda adicional pode ser alocada para objetos mais importantes. A resolução do objeto pode ser ajustada para casar com a capacidade do meio de transporte;
- **interatividade** - por causa dos objetos audiovisuais serem compostos em quadros no decodificador, o usuário pode controlar a saída.

A figura 16 ilustra uma aplicação de difusão de notícias *(news broadcast)*, na qual se usam estas três funcionalidades. Os quatro objetos de vídeo incluídos na figura são: o vídeo da notícia, o vídeo do apresentador, o texto e um relógio. Os dois objetos de áudio são a voz do apresentador e o áudio da notícia. O usuário pode selecionar quais dos objetos serão usados para compor o quadro.

O expectador deseja remover o vídeo do apresentador e usar somente o vídeo da notícia. O apresentador poderia narrar a cena fora da câmera. O expectador desligará o áudio do apresentador e escutará o áudio da notícia enquanto lê o texto. Por isso, os objetos são escaláveis, quantidade de largura de banda variável pode ser alocada para diferentes objetos. Ao vídeo da notícia pode ser dada uma maior largura de banda do que o vídeo do apresentador, desde que o expectador esteja usualmente olhando o vídeo da notícia. O texto requer menor largura de banda. O fluxo de bits codificado da notícia pode ser armazenado em uma biblioteca por outras organizações e reutilizado no futuro.

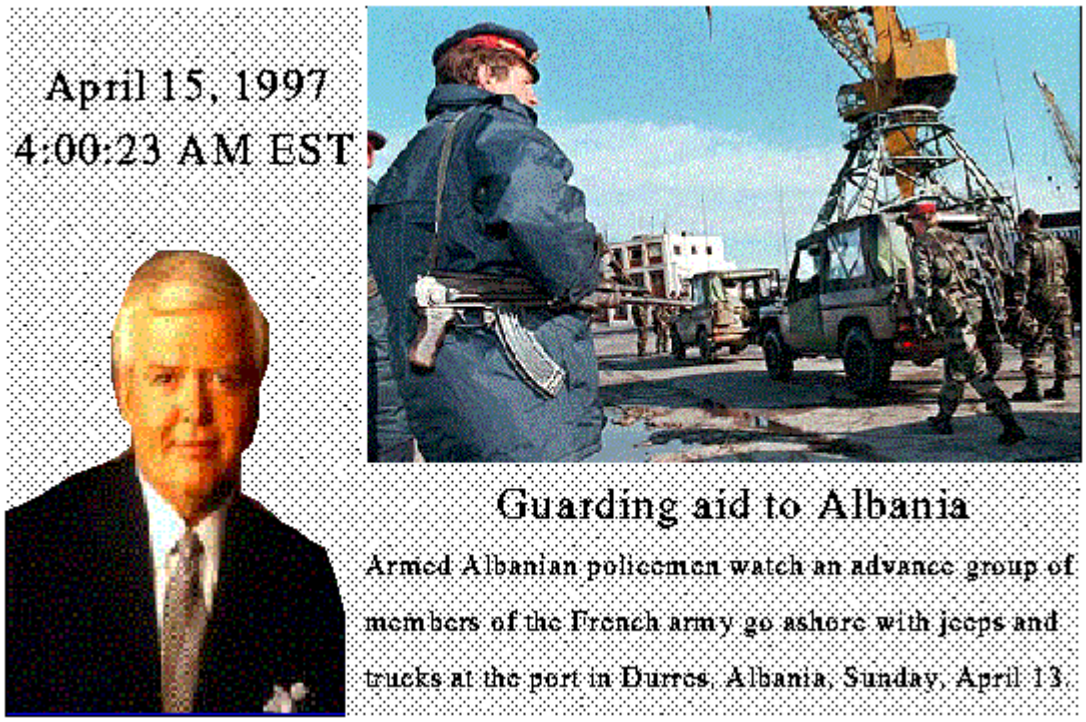


Figura 16 - Exemplo de uma aplicação MPEG-4 [VIDAL97]

4.4. Considerações Finais

O MPEG-4 é um padrão que tem tudo para repetir o sucesso do MPEG-1 e MPEG-2, dois padrões que mudaram o aspecto audiovisual das aplicações atuais. O MPEG-4 foi desenvolvido pelos melhores especialistas de Instituições de Ensino e de laboratórios de empresas multimídia.

O padrão orienta empresas e servirá como motivação para a geração de novos serviços e aplicações a serem desenvolvidos por estas empresas e por empresas desenvolvedoras de visualizadores para a *Web*.

É um padrão que sempre aceita novas características, preservando a compatibilidade entre aplicações existentes para responder a demanda do mundo das aplicações e suportando as expansões tecnológicas.

A sua extensão, o MPEG-4 MU, não está totalmente implementado, apresentando apenas soluções ainda por ser avaliadas e, portanto, sendo alvo de contribuições no mundo todo. O tratamento de múltiplos usuários em diversos dispositivos, a manipulação do *player* MPEG-4 para implementação em dispositivos com capacidade limitada, o uso em

ambientes colaborativos multiusuário, enfim, são todos elementos de desafios que estimulam a exploração desta área.

Ao concluir este capítulo, vimos que o MPEG-4 MU deixa a cargo dos desenvolvedores a implementação de alguns componentes de sua arquitetura, por exemplo o MSC e o MBK, respectivamente os controladores de sessão e atualização da cena multusuário. Essas implementações seguirão o padrão MPEG, porém, este não especifica a forma de como desenvolver esses componentes. Por essa razão, neste trabalho foi feito um estudo de como implementar o MSC e MBK para que pudessem ser desenvolvidos numa rede *peer-to-peer*, baseado em todos as características e desafios já analisados.

No capítulo seguinte, descreveremos uma solução, incluindo a implementação dos componentes MSC e MBK e apresentaremos uma analise do comportamento desses componentes, atuando em uma rede *peer-to-peer*.

5. Uma Solução *Peer-to-Peer* Híbrida para Jogos Virtuais Multiusuário baseada no padrão emergente MPEG-4 -MU

5.1. Introdução

Descreveremos aqui uma solução *peer-to-peer* híbrida para a implementação de jogos multiusuário, baseada no padrão MPEG-4 MU. A rede *peer-to-peer* híbrida em questão é a rede *Gnutella*, a qual apresenta nodos mais poderosos, os *ultrapeers,* capazes de diminuir o fluxo de mensagens na rede.

A solução proposta é para que usuários, através de dispositivos que possam variar de celulares, PDAs, *set-top-boxes*, PCs, a servidores de alto desempenho, componham a rede *Gnutella*. Esses usuários, munidos do software cliente *Limewire* para compartilhamento de arquivos na rede *Gnutella*, do *player* MPEG-4 para visualização de jogos 3D e dos componentes responsáveis pelo controle de sessão e sincronização dos jogos, venham a interagir entre si para a troca de arquivos de jogos, bem como para o controle do jogo.

O software cliente *Limewire* **[LIME03]** foi estendido para suportar uma busca refinada de arquivos de jogos - até então, o *Limewire* suportava a busca de softwares de maneira genérica. Com esta extensão, um usuário da rede *Gnutella* receberá, agora, informações sobre jogos para que, a partir de dados do tipo: tempo de resposta de conexão, número de usuários, etc., o usuário possa selecionar o jogo de seu interesse. Os componentes MSC e MBK, definidos pelo padrão emergente MPEG-4 MU, para controle de sessão e sincronização do jogo, respectivamente, foram implementados neste trabalho, com o auxílio de dois alunos de iniciação científica. A especificação original desses componentes foi também estendida e implementada para suportar a transferência de controle de sessão quando um nodo controlador deixa a rede *Gnutella*, de forma normal ou anormal. O *Limewire* foi, finalmente integrado aos componentes do padrão emergente MPEG-4 MU. O estabelecimento

de sessões e zonas foram testados, assim como a transferência de controle de um nodo para outro, no caso de perda de conexão.

Apresentamos, a seguir, a descrição mais detalhada da solução.

5.2. Fase do Pré-Jogo: Busca e *Download*

Nesta fase, serão descritas as operações de Busca e *Download* de jogos, que foram integradas no *Limewire*.

5.2.1. A Busca de Jogos

Na fase de busca de jogos, o usuário inicia o *Limewire* para conectar-se à rede *Gnutella*. Ao ser conectado, o usuário seleciona a opção de busca de jogo que ativa uma *RichQuery*, que é uma operação de busca mais detalhada, suportada pela rede *Gnutella*, contendo meta-informação sobre o jogo a ser buscado. Por exemplo, um usuário pode querer buscar um jogo específico, de nome "*Moving Target*", com a seguinte meta-informação: criado pelo Laboratório de Realidade Virtual em Rede – LRVnet, da Universidade Federal de São Carlos em janeiro de 2003.

Como resultado da *RichQuery,* uma lista de usuários da rede *Gnutella*, que contenham este arquivo de jogo, é mostrada ao usuário que solicitou a busca. Sem a extensão implementada aqui, um usuário procurando por "*Moving Target*" no sistema de busca atual na rede *Gnutella*, poderia até conseguir a resposta (lista de arquivos encontrados), porém junto com outros arquivos, os quais poderiam não interessar ao usuário.

Com a operação *RichQuery*, os usuários que disponibilizam os arquivos de jogos fornecerão informações mais precisas para que, outros usuários, que buscam esses arquivos de jogos, sejam capazes de encontrar de forma mais eficiente a informação que deseja.

Cada arquivo, numa biblioteca de arquivos compartilhados, será associado a um conjunto de múltiplos rótulos *(tags)* de meta-informação, que utiliza XML para a representação dessas informações. Um esquema XML (que define o formato da informação que um documento deve ter) para jogos, conterá as seguintes informações: nome, autor, gênero, palavra-chave, mono ou multiusuário, jogo 2D ou 3D, dentre outras. O anexo A

descreve o código XML criado como o esquema jogo. Uma vez que essa informação esteja associada ao arquivo, os usuários podem buscarão um jogo de forma mais específica.

O *Limewire* já apresenta a alteração contendo a *RichQuery*, porém, apresenta limitações para a busca de jogos. Um usuário poderia até procurar e encontrar o arquivo *MovingTarget*.mp4, porém, esse não traria informações sobre sessões, usuários e também não ativaria o *Player* MPEG-4. Para isso, foram necessárias as seguintes alterações no código fonte do *Limewire*: o esquema "jogos" foi criado e adicionado ao *Limewire* o que permitiu uma *RichQuery* para jogos. A interface gráfica do *Limewire* também foi adaptada para disponibilizar a busca de jogos, como mostra a figura 17. Com essas alterações, o cliente *Gnutella* passa a ser capaz de efetuar buscas mais especificas de jogos MPEG-4.

Ainda na fase de busca, o *Limewire* foi estendido para que buscasse não apenas os nodos contendo um jogo específico, mas também, o controlador de sessão ativo daquele jogo. O *Limewire* verifica, junto a um nodo, se este é um nodo *ultrapeer* e se sim, se é um controlador de sessão ou não, checando uma variável binária, *isMSC,* que é parte do controlador MSC e que é colocada em 1 como resultado da vontade do usuário em ceder a sua máquina para que esta seja usada como controladora. Os usuários fazem o mesmo atualmente, quando se voluntariam como *ultrapeers*, disponibilizando a sua máquina para o melhor compartilhamento de arquivos na rede *Gnutella*. Caso o nodo não seja um *ultrapeer*, ele também não pode escolher ser um controlador de sessão. Ele pode entretanto, resolver ser um nodo *ultrapeer*, mas não um controlador MSC.

Encontrado um controlador de sessão, o *Limewire* conecta-se a este nodo para obter informações, tais como: sessões existentes, número de usuários participando de cada sessão, o tempo de resposta do controlador (através do *ping*) etc. Caso o *Limewire* não encontre nenhum controlador, essas informações surgirão em branco na tela de busca do usuário, mostrada na figura 18.

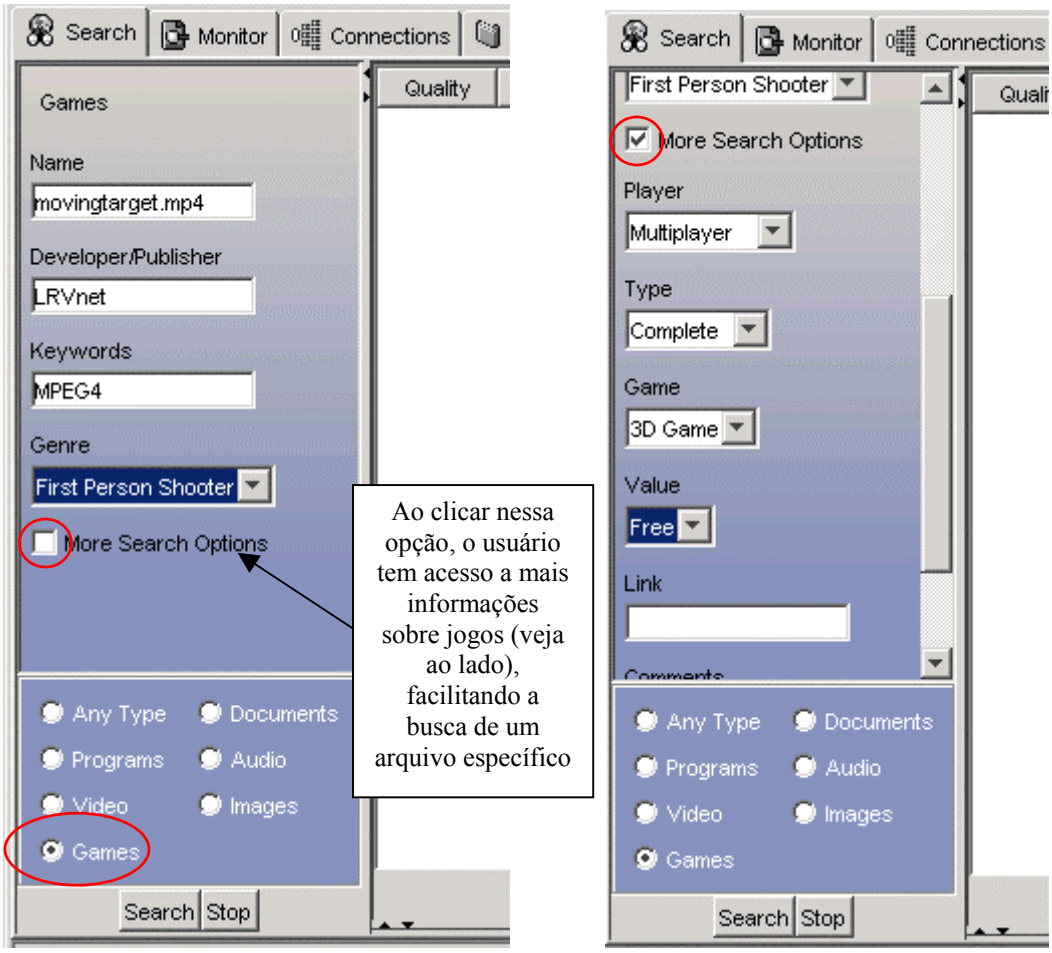


Figura 17 - Busca por jogos – Característica adicionada ao código do *Limewire*

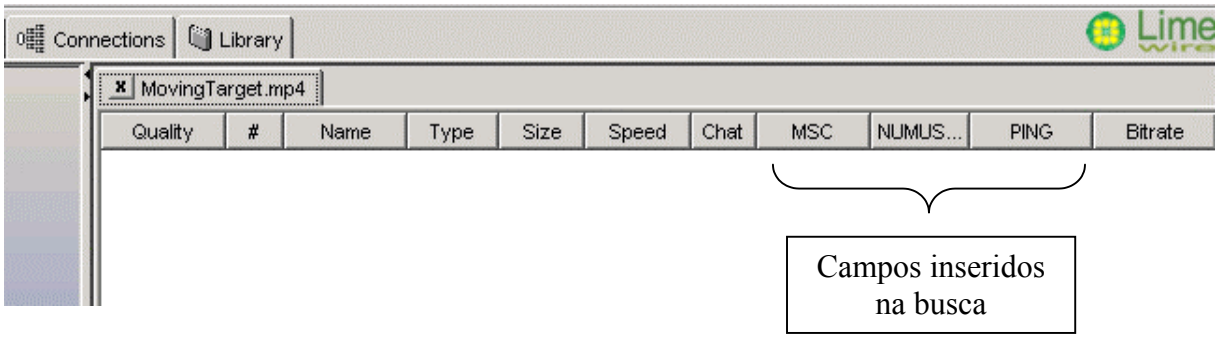


Figura 18. Janela de busca com campos adicionados

As informações mostradas na janela de busca são o resultado da chamada de uma função externa, vinda do controlador, executada pelo *Limewire* quando o usuário clica no botão busca (*Search*). Nesse caso, é estabelecida uma conexão, através de *socket,* com o endereço IP do controlador de sessão (o qual é identificado na busca). A seguinte função é usada para isso:

```
public void conecta() {
     try {
          long inicio, fim;
          inicio = System.currentTimeMillis();
          socket = new Socket(endereco, porta);
          fim = System.currentTimeMillis();
          pingms = fim - inicio;
     }
     catch (UnknownHostException e) { }
```

Estabelecida a conexão, o *Limewire*, em posse das informações do controlador, disponibiliza-as na tela de busca. Ao escolher o melhor arquivo, o usuário fará o *download* do jogo. As alterações feitas no *Limewire* para o suporte ao *download* de jogos serão apresentadas a seguir.

5.2.2. O *Download* do Jogo

Após encontrar o arquivo desejado em alguns outros nodos *Gnutella*, o usuário clica duplamente sobre este arquivo, salvando-o em sua máquina ou dispositivo. Os arquivos trarão, juntamente com o código mp4 do jogo, o código de sincronização do jogo (MBK) que, numa versão futura será parte de um *player* MPEG-4 MU, as sessões abertas, as zonas correspondentes a cada sessão, o número de usuários jogando, o tipo de conexão, tempo da conexão e o campo *isMSC* (que indica se o nodo controla ou não uma sessão). Para que o usuário possa jogar o jogo, ele já deverá ter instalado em sua máquina o *Player* MPEG-4, incluindo os componentes controlador de sessão (MSC) e o controlador de atualização da cena multiusuário (MBK). A tela que contém essas informações, chamada de *Game Setup,* está representada na figura 19.

Conforme já afirmamos, anteriormente, assim que o *download* terminar, e o usuário clica sobre o arquivo, a tela *Game Setup* é mostrada, momento no qual as seguintes situações podem ocorrer:

- Caso o endereço do controlador não exista mais (perda de conexão), a tela *Game Setup* aparecerá em branco; assim uma nova busca por controladores daquele jogo será feita, atualizando as informações de sessão e zonas, clicando no botão **atualizar**. Após isso, o usuário escolherá uma sessão e entrar normalmente no jogo.

- Caso não haja ninguém controlando a sessão, a tela *Game Setup* aparecerá novamente em branco, e o usuário poderá escolher criar uma sessão, clicando no botão **New session**. Assim que ele a criar, passará a controlá-la. Porém, a máquina do usuário terá atributos que a tornem-na habilitada a ser um controlador de sessão, (basicamente, que seja um nodo *ultrapeer*). Se a máquina do usuário não puder ser um controlador, uma nova busca será iniciada até que seja encontrado um nodo capaz de controlar aquele jogo.

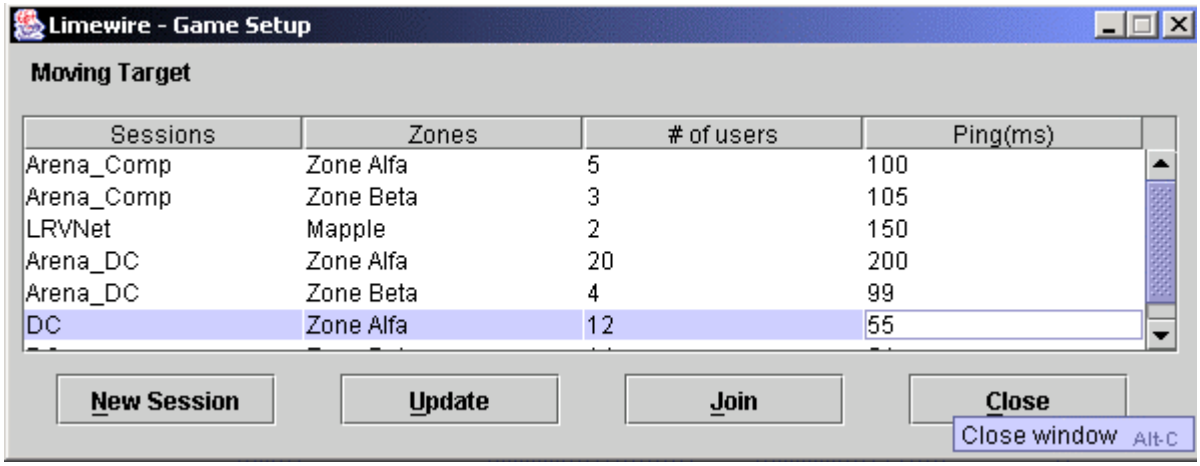


Figura 19. Tela *GameSetup*

Ao selecionar uma sessão, o usuário clicará no botão ***Join*** que ativará o *Player* MPEG-4, carregará, de forma automática, o jogo no ambiente MPEG-4, além de emitir solicitação de entrada na sessão, deixando o usuário pronto para jogar.

5.3. Fase do Jogo

Nesta fase, são descritas operações como a entrada em determinada sessão, entrada em zonas, o gerenciamento do jogo, o qual inclui a transferência de controle, e também como é feita a sincronização do jogo entre todos os usuários participantes.

5.3.1. Iniciação e associação do usuário em uma sessão e zona

Ao entrar no jogo, o usuário optará por criar uma nova sessão (caso atenda os requisitos para isso) ou entrará numa sessão existente.

Considerando que o usuário opte pela entrada em uma sessão, isso será feito através de uma série de trocas de mensagens entre a aplicação (jogo), que solicita a entrada, e

o componente MSC implementado no nodo controlador ativo. A cena inicial MPEG-4 é passada para o usuário que estiver entrando no jogo, e contém um descritor de objeto (OD) para a cena e a representação binária da cena (BIFS). A cena contém um nó reservado para a parte compartilhada da cena, nó *MUSession,* e um descritor de objeto associado. O OD oferece as informações necessárias para a aplicação contatar o MSC (por exemplo, o endereço IP do controlador de sessão). O usuário então, envia ao MSC uma mensagem de solicitação, *MUJoinSessionRequest*, requisitando permissão para entrada na sessão. Recebida a mensagem de solicitação, o MSC do controlador ativo solicita ao usuário suas capacidades (capacidades de áudio, suporte AV entre outros) através da mensagem *MURequestCapabilities*. O usuário retorna então uma mensagem *MUParticipantCapabilities*. 0Essa mensagem testa se o usuário tem capacidades de rede, como por exemplo, capacidades *Multicasting*. Se conceder a entrada, associa um ID ao terminal do usuário (nodo contendo o *Player* MPEG-4) e envia uma mensagem *MUJoinSessionAcknowledge*. Assim que a aplicação do usuário receber esta mensagem, o campo *enable* do nó *MUSession* é alterado para *TRUE* ou *FALSE*, dependendo se o MSC concede ou nega a entrada do usuário na sessão **[N4272]**.

A figura 20 ilustra a troca de mensagens entre a aplicação e o MSC no controlador ativo, *JoinSessionRequest e JoinSessionAcknowledge*.

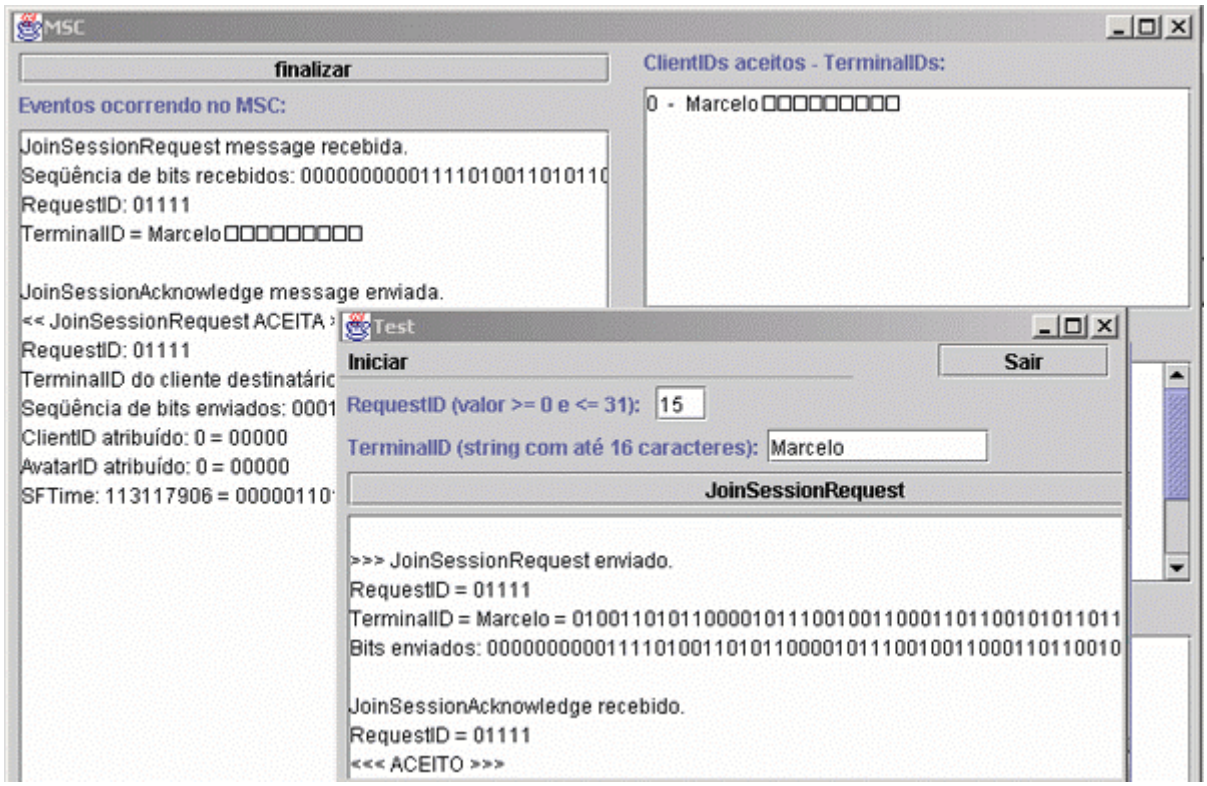


Figura 20 - Interação entre a aplicação e MSC no controlador ativo– *JoinSessionRequest*

Quando o usuário entra na sessão, o componente responsável pela sincronização do jogo - MBK gera uma mensagem de atualização da cena (BIFS *command*), contendo as zonas de alto nível e as envia a todos os nodos contendo os usuários participantes daquela sessão. Recebida essas zonas, a aplicação, em cada nodo que suporta um usuário participante, reflete essas zonas no grafo da cena localmente, isto é, atualiza o nó *MUSession* de acordo com as informações recebidas pela *MUUpdateMessage*.

Construídas as zonas na cena do usuário, cada aplicação local envia uma solicitação ao MSC no controlador do tempo da sessão, através da mensagem *MURequestSessionTime*, que retorna este tempo via mensagem *MUSessionTime*. Esse tempo é contado a partir do momento que o usuário entrou na sessão, para verificar, por exemplo, quem foi o usuário que entrou primeiro na sessão e aquele que tem mais tempo conectado.

O usuário então, escolhe uma ou mais zonas que deseja entrar, o que engatilha o envio de uma mensagem de solicitação, *MUJoinZoneRequest* (uma mensagem para cada zona), para o MSC no controlador, que retorna ao usuário a mensagem *MUJoinZoneAcknowledge* (uma para cada zona solicitada), caso a solicitação possa ser atendida. O tratamento dessa mensagem é feito pelo usuário que colocará ou não políticas para negar ou aceitar as solicitações.

A partir de então, o usuário estará efetivamente participando de uma sessão multiusuário e todos os outros usuários participantes passam a receber mensagens de atualização, *UpdateMessage*, indicando a entrada de um novo usuário, além de qualquer modificação que ocorra no jogo.

A figura 21 ilustra, num diagrama de seqüência UML, a troca de mensagens de solicitação de entrada numa sessão (via MSC) e de atualização de cenas (via MBK).

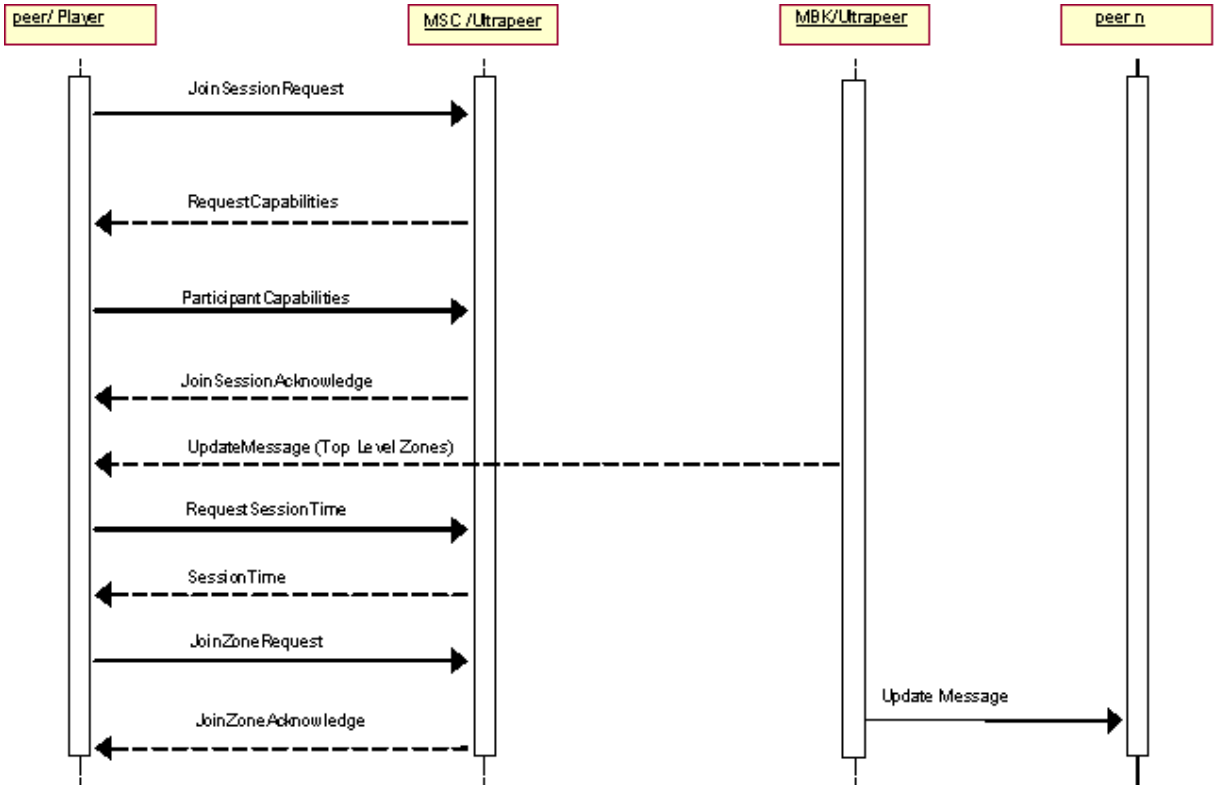


Figura 21 – Diagrama de seqüência UML de entrada numa sessão e zona

A figura 22 mostra um cenário de troca de mensagens entre a aplicação na máquina do usuário (nodo) e o controlador de sessão (MSC no controlador ativo). Na figura, os controladores de sessão, incluindo o controlador ativo, são representados como nodos *ultrapeer*. Os nodos folhas (nodos das pontas), são representados como pequenos computadores (possivelmente com menor potencial de processamento e de rede), caracterizando assim uma rede *Gnutella.* O esquema de troca de mensagens para entrada numa sessão é ilustrado, através de setas e linhas pontilhadas descritos abaixo.

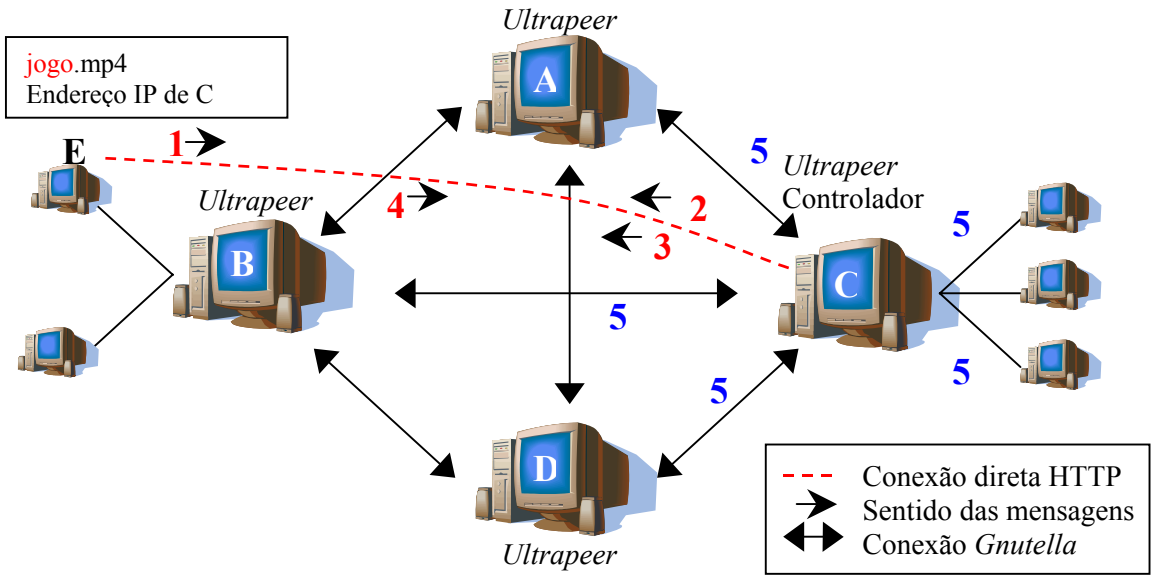


Figura 22 – Cenário de troca de mensagens entre a aplicação no nodo do usuário e o MSC no controlador, para entrada em sessão e zona

Conforme mostrado na figura 22, o nodo C é um *ultrapeer* controlador ativo de sessão, isto é, um controlador que assegura o controle da sessão de um jogo, até que ocorra uma desconexão. Supondo que o nodo "E" acabou de efetuar o *download* do jogo e está querendo entrar numa sessão, este inicia o estabelecimento de uma conexão direta com "C". O nodo "E" sabe o endereço do nodo "C", através da URL[2] de "C" que é recebida juntamente com a aplicação (jogo) no momento do *download* do jogo, e envia uma mensagem de solicitação de entrada (mostrada em 1 na figura 22). A resposta à solicitação de entrada é enviada do nodo "E" para o nodo "C", como mostrado em (2), e será positiva ou negativa dependendo da política adotada pelo usuário. Após a confirmação de entrada na sessão (positiva), o nodo "E" recebe uma lista atualizada das sessões e zonas (3). Após selecionar a zona, o nodo "E" solicita ao nodo "C" a entrada em uma determinada zona (4). Mediante a confirmação de entrada, o usuário no nodo "E" passa a participar da sessão multiusuário

[2] Nesse caso, a URL que é passada é o endereço IP da máquina de onde foi feito o *download* do jogo.

controlada por “C”, que controla a sessão. O MBK notifica então a entrada do novo jogador para os outros nodos participantes(5).

5.3.2. Gerenciamento do Jogo

Após a associação do usuário numa sessão e zona, a rede *peer-to-peer Gnutella* não é mais utilizada em todo o seu potencial, uma vez que apenas grupos de usuários associados trocam mensagens entre si para as atualizações das cenas compartilhadas. Essas atualizações são feitas através do componente MBK existente em todos os nodos participantes de uma sessão de jogo. Já, o controle do jogo, feito pelo componente MSC num nodo *ultrapeer*, deve ser garantido, para o sucesso da aplicação de jogos em redes *peer-to-peer* híbridas. Para isso, mais que um *ultrapeer* é preparado para ser controlador de sessão, conforme descrito a seguir.

Como visto no capitulo 3, forma descritos na literatura algumas formas de tratamento de transferência de controle para que uma sessão permaneça sempre ativa **[MSDN]**, porém, nenhuma delas trata da transferência de controle em redes *peer-to-peer* híbridas. Assim, propomos aqui um esquema de transferência de controle de sessão de um nodo *ultrapeer* para outro nodo *ultrapeer*, de tal forma que o controle de sessão permaneça ativo sempre que houver usuários participantes nela.

Seria difícil contemplar o uso de *ultrapeers* como controladores de sessão se esta função exigisse muito tempo de processamento. Medições realizadas em duas máquinas do LRVNet, uma com processador Pentium III 550 MHz, com 256 MB de RAM e outra com processador Pentium II dual 450 MHz, também com 256 MB de RAM, mostraram que o tempo de processamento que o *ultrapeer* gasta para monitorar a sessão é ínfimo. Para que o jogo não seja interrompido com a saída do *ultrapeer* que controla uma sessão, se um *ultrapeer* deixar a rede por qualquer motivo (desconexão normal ou anormal), outro *ultrapeer* substituto assume a sessão para que esta não termine enquanto houver participantes. Assim, uma lista é gerada contendo possíveis controladores de sessão prontos para assumir o controle, caso o *ultrapeer* controlador atual saia de cena (a lista segue a ordem de conexão – o primeiro que estabeleceu conexão e cumpre os requisitos para ser um *ultrapeer*, é o próximo a assumir o controle). Caso a transferência de controle não possa ser feita (por falta absoluta de máquinas que possam ser controladoras), a sessão então será terminada.

O esquema de seleção de *ultrapeers* controladores de sessão substitutos funciona da seguinte maneira: um *ultrapeer* que é o controlador atual de sessão, ao receber solicitação de conexão de um nodo cliente, primeiro verifica se este nodo se voluntariou para ser controlador de sessão (variável *isMSC* em 1), juntamente com a opção de ser ou não um *ultrapeer*. Se ambas as condições forem satisfeitas, este nodo será o controlador de sessão número dois na hierarquia dos controladores de sessão. Com isso, este nodo passa a receber informações de atualização de sessão, bem como passa a monitorar o estado de conexão do controlador atual de sessão. Se o nodo que solicita conexão a um *ultrapeer* controlador atual de sessão não satisfizer as duas condições mencionadas acima, este simplesmente entrará na sessão como um nodo comum. No momento, até quatro controladores substitutos serão mantidos em estado de latência (prontos para assumir o controle da sessão). Se algum dos substitutos desconectar, outro substituto será buscado e preparado de forma automática.

Os controladores substitutos executam duas funções no estado de latência: recebem informações de atualização de sessão, enviadas pelo controlador ativo, e monitoram o estado de conexão do controlador ativo. Assim que o próximo substituto da lista "percebe" que o controlador ativo desconectou, ele inicia o estabelecimento de conexão com todos os jogadores da sessão, passando a sua URL, isto é, o endereço do novo controlador de sessão. Essa passagem da nova URL é feita através da conexão via *socket* do novo *ultrapeer* controlador com todos os outros nodos que estavam conectados ao antigo controlador. A forma como um nodo controlador substituto "percebe" que o nodo controlador ativo desconectou será descrita a seguir. Um *ping* é enviado dos controladores substitutos ao controlador ativo. Para cada controlador substituto, um tempo base é identificado (que será alterado dinamicamente) baseado na média de tempo de resposta dos *pings* enviados. Se o tempo de resposta atual de um *ping* que acabou de ser enviado for duas vezes maior que o tempo base, isto será considerado como desconexão do nodo controlador ativo e o próximo nodo controlador substituto na hierarquia assumirá o controle.

O tempo de transferência de controle de um *ultrapeer* para outro, isto é, o tempo decorrido entre o momento em que se percebeu a desconexão até o momento em que o controle passa a ser realizado pelo novo *ultrapeer* controlador de sessão, quando este está pronto para receber novas solicitações de conexão, foi medido e é mostrado no gráfico da figura 23. Foi observado que o que mais pesa no tempo total da transferência de controle é o estabelecimento das conexões entre o novo *ultrapeer* controlador e os nodos de todos os jogadores da sessão. Ainda assim, o tempo é imperceptível, do ponto de vista dos jogadores.

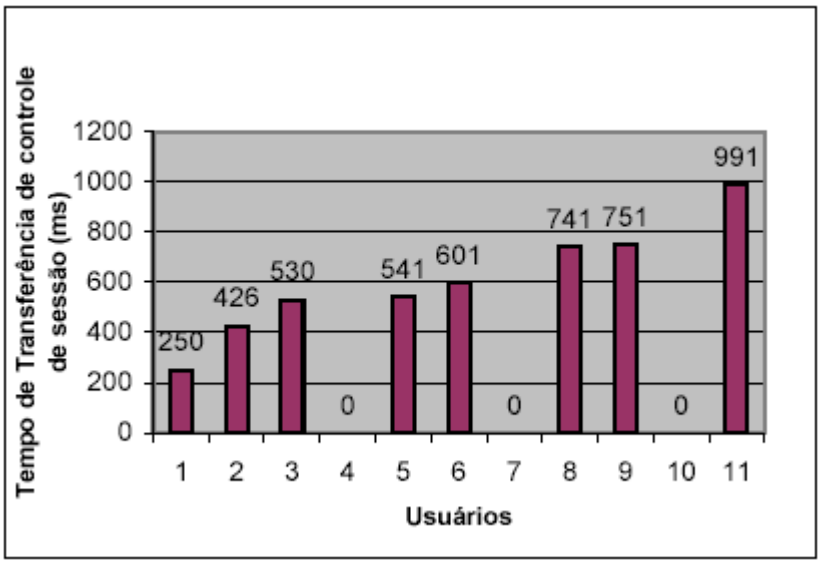


Figura 23 - Tempo de transferência de controle pelo número de usuários

A conexão entre o *ultrapeer* controlador ativo e um nodo controlador substituto está representada na figura 24:

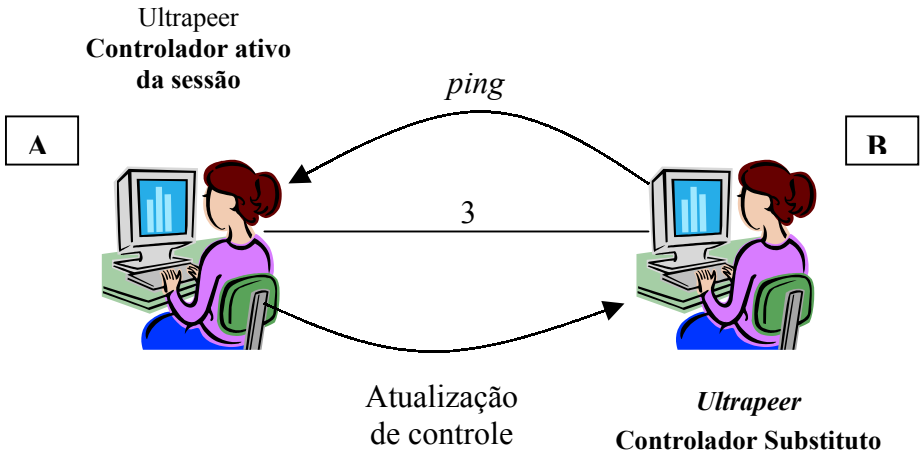


Figura 24 – Conexão entre controlados MSC ativo e possível controlador

Conforme mostra a figura 24, o nodo "A" é um *ultrapeer* controlador ativo de sessão e aguarda conexões de outros nodos que desejam conectar-se em sessões sob seu controle. "B" então, um nodo controlador substituto, encontra-se em estado latente, apenas recebendo mensagens de atualização do controlador "A". Assim que "B" percebe a perda de conexão, o controlador substituto assume o controle. Esse novo *ultrapeer* controlador contém os endereços IP's de todas as máquinas conectadas ao antigo controlador ativo. Com isso, o

novo *ultrapeer* controlador de sessão estabelece uma conexão com todos os outros usuários, mantendo assim a sessão ativa.

Sempre que um novo usuário deseja entrar numa sessão, este repetirá a seqüência descrita acima e, assim, sucessivamente até que pelo menos quatro controladores substitutos sejam identificados possam ser preparados para assumir o controle.

Caso o nodo que deseje entrar na sessão não seja elegível como possível controlador, uma conexão normal, de nodo comum para controlador de sessão ativo, será estabelecida com esse nodo.

5.3.3. Controle de Atualizações

Também na fase do jogo, é tratada a atualização dos objetos e dos usuários participantes do jogo, através do componente MBK, parte integrante do padrão MPEG-4 MU, e amplamente descrito no capítulo sobre o padrão emergente MPEG-4MU. De acordo com o padrão, todo piloto de um objeto é o MBK da zona correspondente - se um usuário é o piloto de uma zona, ele é então o MBK daquela zona. Assim, uma atualização na zona daquele usuário, só é possível de ser feita pelo piloto daquela zona. Se um usuário deseja fazer uma alteração em um objeto que está em uma zona na qual ele não é o piloto, uma solicitação de transferência de pilotagem deve ocorrer. Se a transferência de pilotagem for aceita, a atualização é feita e difundida para todos os MBKs em nodos de usuários participantes. Um nodo MBK também armazenará onde o piloto de um objeto compartilhado está, o tempo todo, além de manter o grafo da cena MPEG-4 completo.

Em suma, todos os nodos de usuários participantes de uma sessão, possuem o MBK em sua aplicação. Através dele, é possível verificar quem é o proprietário de um objeto, solicitar a pilotagem de um objeto ou então solicitar alterações. O MBK é sempre atualizado juntamente com todos os outros controladores de atualização da rede. A atualização de um objeto compartilhado se encontra ilustrada na figura 25.

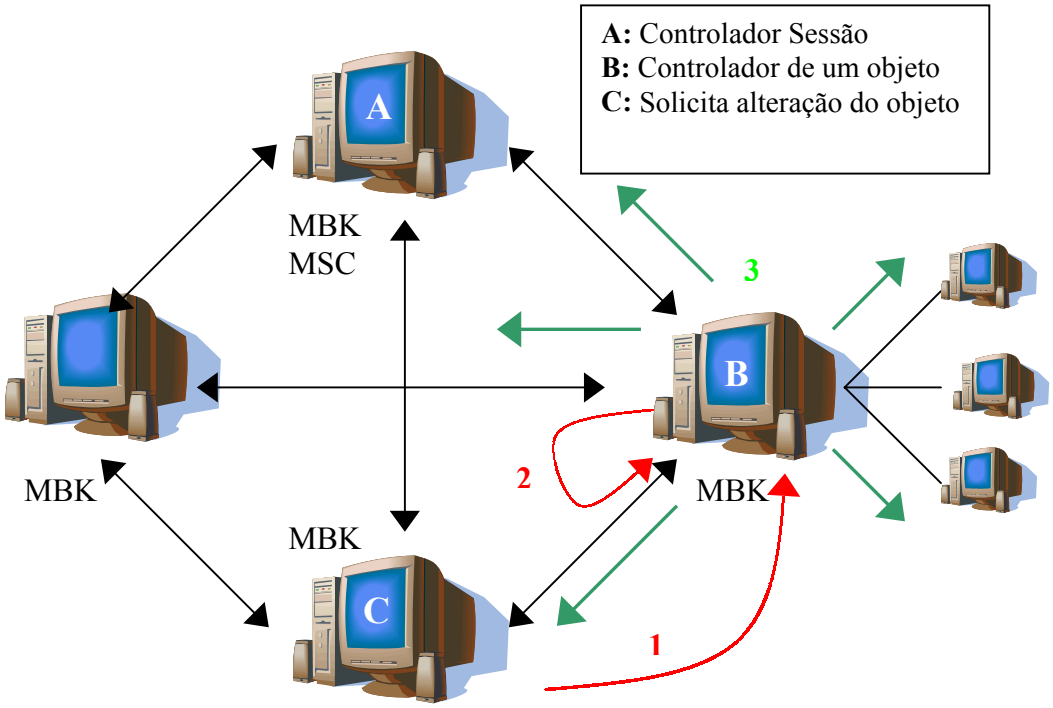


Figura 25 – Atualização de um objeto compartilhado

A figura 25 mostra como é feita a atualização de um objeto compartilhado. O nodo do usuário "A" é um *ultrapeer* e, atualmente, tem o controle da sessão. Os nodos "B" e "C" são nodos cujos usuários participam da sessão multiusuário controlada por "A". Porém, "B" é atualmente o proprietário de um objeto compartilhado em uma zona X. Observa-se que todos os nodos de usuários possuem o MBK. Supondo que "C" queira fazer uma alteração no objeto compartilhado, ele enviará uma mensagem para "B", solicitando uma alteração (1). Assim que a solicitação para alteração for concedida (2), "B" (2), atualiza o seu próprio objeto e encaminha mensagens de atualização (3) para todos os nós conectados àquela sessão.

Ao longo do jogo, novos nodos usuários entrarão e outros deixarão a sessão que se manterá contínua por tempo ilimitado (desde que haja um controlador de sessão ativo). Quando não houver mais qualquer usuário participante numa sessão, esta será encerrada pelo nodo controlador de sessão ativa. No próximo item, esta questão será tratada.

5.4. Fase de Desconexão

Durante o jogo, os usuários deixarão uma sessão de duas formas diferentes: normal e anormal.

A forma anormal ocorre quando um usuário perde a conexão e deixa a sessão multiusuário por problemas no seu dispositivo, tais como: travamento do sistema operacional, redução de carga da bateria, no caso de dispositivos portáteis, entre outros. Para o tratamento desse tipo de desconexão, o controlador de sessão ativa monitorará todos os usuários, a fim de verificar sua integridade durante a sessão. Isso é feito sempre que o controlador da sessão perceber alguma desconexão com nodos de usuários participantes. Percebendo a desconexão, uma mensagem de atualização é enviada para todos os outros, notificando a saída daquele nodo do jogo. Ao receber esta mensagem, o MBK, em cada nodo de usuário participante, removerá todas as entidades compartilhadas relacionadas ao usuário no nodo que perdeu a conexão e retirará aquele usuário de todas as zonas.

Já a forma normal ocorre, de acordo com **[N4272]**, quando um usuário, por vontade própria, desejar sair de uma sessão e, logicamente de uma ou mais zonas. Esse usuário envia ao controlador de sessão ativa uma mensagem de solicitação de saída da(s) zona(s), *MULeaveAllZonesRequest*. Recebida a solicitação, o controlador de sessão ativa responde ao usuário com uma mensagem de confirmação de saída da(s) zona(s). Em seguida, o usuário envia uma nova mensagem ao controlador se sessão ativa solicitando a saída da sessão multiusuário, *MULeaveSessionRequest*, conforme mostra a figura 26. Recebida a solicitação, o controlador de sessão confirmará a saída através do envio de mensagem de confirmação, e o usuário, finalmente, se desconectará do controlador de sessão ativa. A dinâmica dessa troca de mensagens encontra-se no diagrama UML da figura 27.

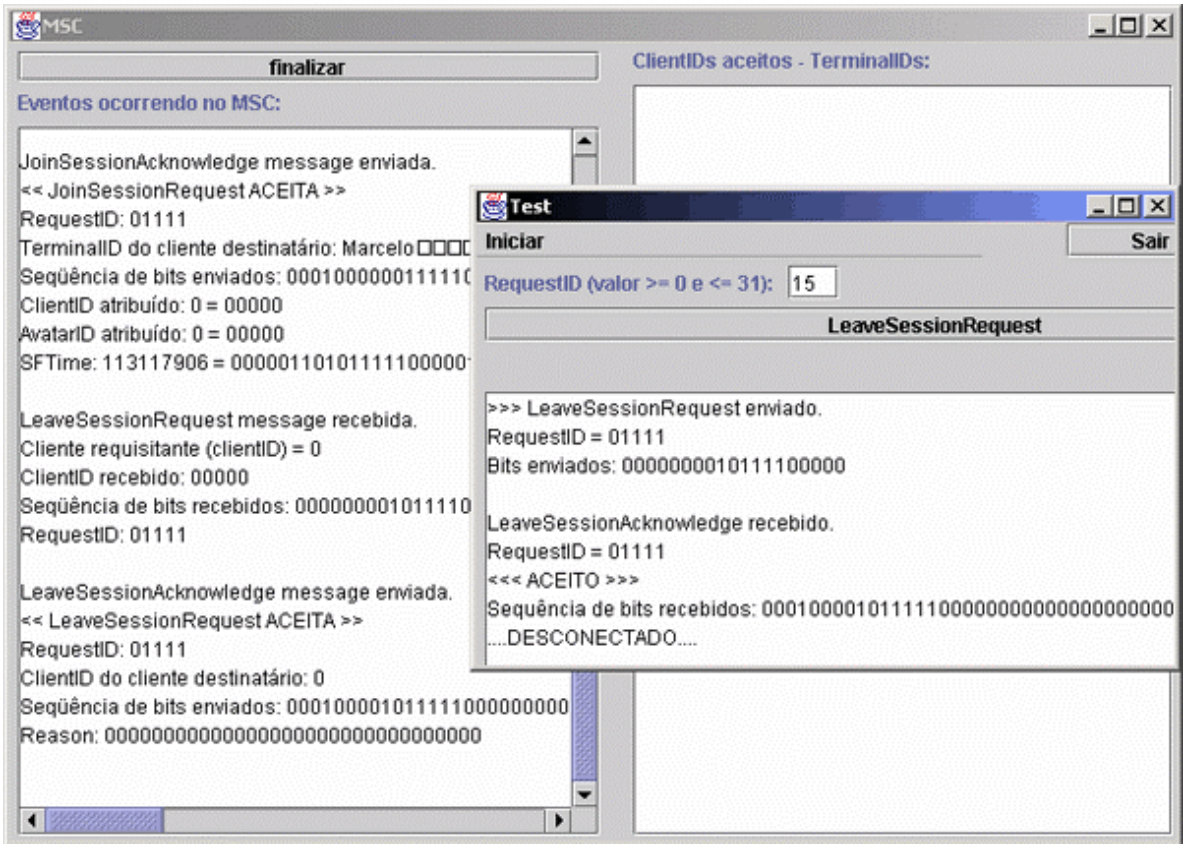


Figura 26 -. Interação entre aplicação e MSC – *LeaveSessionRequest*

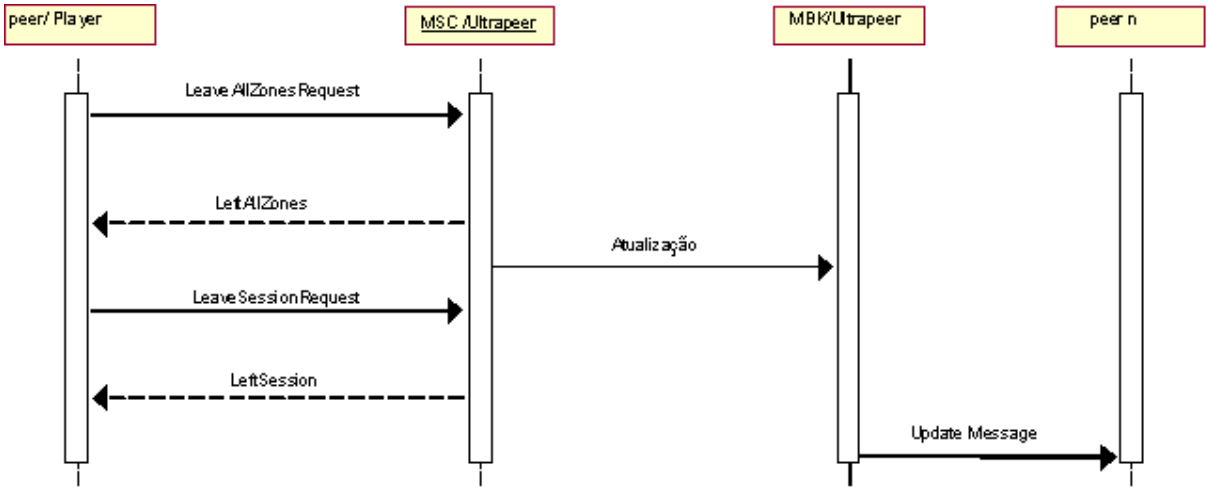


Figura 27 – Diagrama de seqüência UML de desconexão do Jogo – Forma Normal

5.5. Um Cenário da Integração

Um cenário multiusuário ilustrativo está registrado na figura 28 onde todas as entidades discutidas neste capítulo se integram. Nesse cenário, desde telefones celulares de terceira geração, PDAs e PCs comuns, podem ser interligados numa rede *Gnutella*.

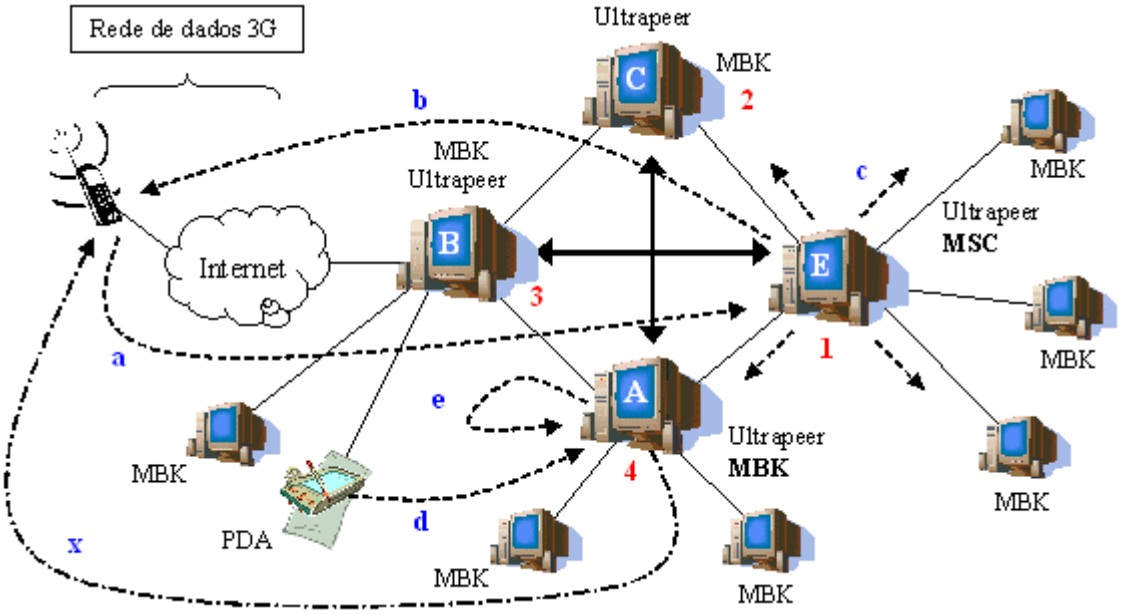


Figura 28 - Cenário de integração da rede *Gnutella*

A figura 28, representa, de forma integrada, algumas operações, das inúmeras que podem ocorrer ao mesmo tempo, com a integração das tecnologias das redes *peer-to-peer* híbridas, com o padrão emergente MPEG-4 multiusuário. Na figura, o *ultrapeer* "E" é um controlador de sessão ativa. O *ultrapeer* "A", está pilotando um objeto compartilhado, através do componente MBK. Um aparelho celular de terceira geração solicita a entrada numa sessão ao controlador de sessão ativa "E", o qual analisa as capacidades do solicitante. Verificando a impossibilidade de controle por parte do celular, o controlador de sessão autoriza a sua entrada na sessão como nodo comum (b).

A ilustração mostra também, o caso de um PDA que deseja fazer uma alteração em um objeto compartilhado, cujo proprietário é o *ultrapeer* "A". O PDA, através de seu MBK, solicita a alteração ao MBK de "A" (d). Este executa a alteração no seu objeto e a distribui (atualiza) para todos os outros nodos participantes, inclusive o PDA que solicitou a alteração.

A figura representa ainda, os nodos *ultrapeers* e seus respectivos números na hierarquia de nodos controladores substitutos. O controlador de sessão ativa (1) é o nodo "E". É ele o responsável por verificar se as máquinas que estão solicitando a entrada em uma sessão serão ou não possíveis controladoras, classificando-as numa hierarquia de acordo com a ordem de chegada. Esses nodos, assim classificados, monitoram-se continuamente, através do envio de mensagens *ping*. O próximo nodo na hierarquia assume quando o tempo de

conexão com o controlador de sessão ativa for maior que duas vezes o tempo base, obtido como a média do atraso da rede (o tempo base pode ser configurado de forma dinâmica).

5.6. Considerações Finais

Neste capítulo, foi descrita uma solução para jogos multiusuário como aplicações *peer-to-peer* híbridas, baseada no padrão MPEG-4 MU. Para isso, o software *Limewire*, um cliente da rede *Gnutella* para compartilhamento de arquivos, foi estendido para suportar a busca e o *download* de jogos da rede *Gnutella*. Dois componentes do padrão emergente MPEG-4 multiusuário, o MSC e MBK, responsáveis pelo controle de sessões e pela sincronização do jogo, respectivamente foram implementados e testados. Estes componentes foram integrados ao cliente *Limewire* e à arquitetura da rede *Gnutella* (*ultrapeers*) para o suporte a jogos com múltiplos participantes.

Adaptado para busca de jogos através de um conteúdo mais específico (busca por meta-dado), o *Limewire* traz ao usuário não apenas o arquivo do jogo, mas também informações como as sessões ativas daquele jogo, quantidade de usuários em jogo, a latência da conexão de rede (através do *ping*) para que o usuário possa selecionar melhor o jogo que deseja. Ao terminar o *download* do arquivo, o usuário pode clicar sobre uma sessão desejada, mostrada numa tela intermediária do *Limewire*. Ao escolher a sessão, o *Limewire* ativa automaticamente o *player* MPEG-4, passando também o endereço do controlador da sessão ativa (que foi baixado juntamente com o arquivo do jogo na fase busca), ou então o endereço do próprio nodo do usuário, caso este deseje criar uma sessão própria.

Após sua entrada no jogo, os nodos dos usuários participantes trocam mensagens de atualização (sincronização do jogo) de forma direta, através do componente MBK. Se um usuário comum pretende sair normalmente de uma sessão, ele enviará uma solicitação de saída ao controlador que desconectará o usuário da sessão e respectivas zonas e atualizará os outros usuários. Caso a saída seja anormal, o nodo controlador de sessão ativa apenas desconectará esse usuário e atualizará os outros nodos conectados. Porém, se a desconexão ocorrer com um controlador de sessão ativa, um mecanismo descrito transferirá o controle de sessão para nodos controladores substitutos. Este mecanismo foi testado e medido de forma preliminar, mostrando que o tempo da transferência de controle é relativamente

baixo e praticamente imperceptível nos nodos da rede. Na próxima sessão de conclusões detalharemos melhor as contribuições geradas por este trabalho.

6. Conclusão

O objetivo deste trabalho foi a implementação de uma estrutura de suporte a jogos virtuais 3D em rede, baseada no padrão emergente MPEG-4 MU (*MultiUser*) e implementada na rede *peer-to-peer* híbrida Gnutella. Para isso, o código *Limewire,* um software cliente da rede Gnutella foi modificado de modo a incluir um serviço de busca de jogos e de sessões ativas. Mais ainda, os componentes MSC e MBK especificados pelo padrão emergente MPEG-4 MU foram implementados e integrados à rede *Gnutella* para controle de sessão de jogos e atualização das cenas, respectivamente. Um esquema de transferência de controle de sessões foi implementado e testes preliminares indicam que o tempo de transferência do controle de uma sessão de um *ultrapeer* para outro é praticamente imperceptível para os jogadores. Uma das vantagens resultantes deste trabalho é que qualquer usuário que tenha um jogo e deseje iniciar e controlar uma sessão poderá fazê-lo sem que para isso tenha que se registrar em um servidor de conteúdo.

Como a solução implementada utiliza o *Player* MPEG-4 e o padrão emergente MPEG-4MU, isto aumenta a possibilidade de uso de dispositivos com diferentes capacidades, uma vez que o *Player* MPEG-4 está sendo disponibilizado para celulares, PDAs, *set-top-boxes* e outros. Mais ainda, o uso de *ultrapeers* como controladores de sessão proporciona menor latência de rede para os nodos que se conectam neles pois, normalmente, os *ultrapeers* encontram-se mais próximos desses nodos (no modelo cliente-servidor, o servidor pode estar localizado longe de um nodo, aumentando a latência de rede deste e submetendo o jogador a ele conectado, a injustiças quando numa situação de jogo, um usuário tem que ser mais rápido que outro). O uso de *ultrapeers* num modelo *peer-to-peer* híbrido apresenta vantagens quanto

ao tráfego, pois diminui o fluxo de mensagens que são transferidas de uma máquina para outra, quando comparado a uma rede *peer-to-peer* totalmente descentralizada.

Nosso estudo proporcionou uma melhor compreensão das questões envolvidas na implementação jogos multiusuário em redes *peer-to-peer* híbridas, além de mostrar que o suporte multiusuário definido pelo grupo MPEG é um esquema promissor no suporte a jogos multiusuário de maneira geral. Também vimos que, dispositivos de menor capacidade de processamento e armazenamento poderão utilizar nodos de maior capacidade de processamento, como é o caso de um celular participando de uma sessão e conectado a um *ultrapeer,* este responsável por fazer o controle de sessão para o dispositivo celular. Os dispositivos de menor capacidade apenas tratarão mensagens e receberão atualizações que serão mostradas em sua tela.

Com este trabalho, avaliou-se a possibilidade de implementação de jogos multiusuário em redes dinâmicas como é o caso das redes *peer-to-peer*.

6.1. Contribuições

A pesquisa desenvolvida contribuirá com outras pesquisas realizadas pelo LRVNet na avaliação do padrão MPEG-4 como um padrão viável no suporte a aplicações multiusuário. As referidas contribuições são as que seguem relacionadas.

- Implementação dos componentes MSC e MBK em conjunto com outros três alunos: um de mestrado, que também usa o MSC para controle de sessão em ambientes colaborativos, e dois de iniciação científica. Os componentes MSC e MBK são utilizados para suporte aos controles de jogos e outras aplicações para operações tais como gerenciamento de sessão, controle de concorrência e manutenção da consistência;

- Alterações desenvolvidas no software cliente *Limewire*, da rede Gnutella, juntamente com um aluno de iniciação científica. As alterações serão repassadas para a comunidade de desenvolvimento *Gnutella*.

6.2. Trabalhos Futuros

Os seguintes trabalhos futuros deverão ser realizados:

- Montagem de teste em escala de pelo menos 10 sessões de 30 usuários participantes cada (em cooperação com outras Instituições) para avaliação qualitativa do tempo de resposta no gerenciamento das sessões e atualização de cenas;
- Avaliação quantitativa de atraso na sincronização das cenas;
- Análise da latência da rede durante a troca de mensagens MPEG-4;
- Implementação de outros tipos de jogos, para testes na estrutura implementada;
- Variação dos requisitos para atribuição de controle de sessão a nodos que atendam esses requisitos;
- Avaliar o tempo de permanência ativa de *ultrapeers* quando a aplicação for jogos;
- Pesquisa junto a comunidade *Gnutella* para saber se os usuários dessa rede aprovam a idéia de disponibilizar suas máquinas como controladores de sessão (assim como muitos já disponibilizam-nas como *ultrapeers* para o compartilhamento de músicas);
- Disponibilizar o código para a comunidade de desenvolvimento do *Gnutella*;

Realizados os trabalhos futuros, novas medições poderão ser avaliadas e estudadas e conseqüentemente novas contribuições poderão ser implementadas. Vimos também que, o projeto envolvendo o padrão emergente MPEG-4 MU juntamente com a rede *Peer-to-Peer Gnutella* poderá ser bastante promissor, possibilitando o surgimento de novas linhas de pesquisa nas mais diversas áreas.

Referencias

[AIT]	Aitken D. et all. **Peer-to-Peer Technologies and Protocols**. Disponível em URL: http://ntrg.cs.tcd.ie/undergrad/4ba2. 02/p2p/. Consultado em 02/06/2003.
[APL03]	Oliveira, D. **Aplicações Peer-to-Peer**. Trabalho de Graduação para a disciplina Redes de Computadores II – Universidade Federal do Rio de Janeiro – 2002. Disponível em URL: http://www.gta.ufrj.br/~rezende/cursos/eel879/trabalhos/p2p/ Consultado em 09/01/2003.
[AND02]	Anderson, D., Cobb, J., Korpela, E., Lebofsky, M.,Werthimer, D. 2002. **SETI@home: An Experiment in Public-Resource Computing**. To appear in CACM, November 2002.
[BAR03]	Bartle R.**Interactive Multiuser Computer Games**. Disponível em URL: http://www.iol.ie/~ecarroll/mud/mudreport.html. Consultado em 14/05/2003
[W4415]	ISO/IEC JTC1/SC29/WG11 – N4415 – **MPEG-4 Systems.** MPEG, ISO. Pattaya. Dezembro 2001.
[BAL03]	Balakrishnan H., et all. **Looking Up Data in P2P Systems**. Communications of the ACM. Fevereiro 2003
[BET01]	Bettner, P. e Terrano, M. "**1500 archers on a 28.8: Network programming in Age of Empires and beyond**", in The 2001 Game Developer Conference Proceedings, San Jose, CA, Mar. 2001.
[CRO02]	Cronin, E., et all. **An Efficient Synchronization Mechanism for Mirrored Game Architectures.** NetGames2002 Abril 16-17, 2002, Braunschweig, Germany

[CLAR00]	Clarke, I., Sandberg, O., Wiley, B., and Hong, T. Freenet: **A distributed anonymous information storage and retrieval system**. In *Proceedings of ICSI Workshop on Design Issues in Anonymity and Unobservability*. Berkeley, California (June 2000); freenet.sourceforge.net.
[DIOT99]	C. Diot and L. Gautier.(1999) "**A distributed architectures for multiplayer interactive applications on the internet**". IEEE Networks magazine, 13(4)
[DIV96]	O. Hagsand. **"Interactive MultiUser VEs in the DIVE System"**. IEEE, 1996. 3(1), http://www.sics.se/dce/dive/dive.html
[DUARTE02]	Duarte, F. V. **Implementação e avaliação de um ambiente de modelagem colaborativa baseada no padrão MPEG-4.** Exame de Qualificação. PPG-CC, UFSCar, São Carlos, Maio de 2002
[GAR82]	Garcia-Molina, H. . **Elections in a Distributed Computing System.** IEEE Trans. On Computers, 31:48-59 Janeiro de 1982.
[GC01]	Bildson, G. "**PROPOSAL: Handshaking Protocol**", The Gnutella Developer Forum (GDF), Agosto 2001. Disponível em URL: http://groups.yahoo.com/group/the_gdf/message/2010 Consultado em 12/02/2003
[GEN01]	**Genome@Home**. 2001. Disponível em URL genomeathome.stanford.edu. Consultado em 16/03/2003
[CIG02]	Ciglaric, M. and Vidmar, T. **Position of Modern Peer-to-Peer Systems Architecture.** Conference Theme Communications and Information Technology for Development IEEE. MELECON 7-9 May 2002
[GNU03]	**Gnutella.com** Disponível em URL: www.gnutella.com. Consultado em 16/03/2003

[GOLD01]	Gold, R., Mascolo, C. . **Use of Context-Awareness in Mobile Peer-to-Peer Networks**. Workshop on Future Trends of Distributed Computing Systems, IEEE. 2001
[GRO]	**Groove Networks**. Disponível em URL: http://www.groove.net Consultado em 06/05/2003
[GROO01].	**Groove Networks Product Backgrounder**. Groove Networks White Paper. Disponível em URL: www.groove.net/pdf/groove_product_ backgrounder.pdf. Consultado em 16/04/2003
[HER03]	Herlihy M. and Mohan A. **Peer-to-Peer Multiplayer Gaming Using Arrow Multicast: Peer-to-Peer Quake.** The 23rd International Conference on Distributed Computing Systems IEEE May 9-22, 2003, Rhode Island,USA
[JAMIN]	Jamin, S. **About Online Multiplayer Games.** Disponível em URL: http:// www.cs.umn.edu/Research/networking/itrworkshop/slides/sugih_jamin .pdf. Consultado em 23/04/2003
[JONES03]	Jones, R. M. **Design and Implementation of Computer Games.** Disponível em URL: http://www.cs.colby.edu/~rjones/courses/ cs397/fall2000/genres.html. Consultado em 20/03/2003.
[JXTA]	**Project JXTA,** disponível em: http://www.jxta.org, consultado em 05/05/2003.
[KATO]	Kato J. e Zojonc R.. **Security Analysis of Massively Multiplayer Online Games.** Disponível em: islab.oregonstate.edu/koc/ece478/ proj/2002RP/KZ.pdf. Consultado em 12/02/2003.
[KEE99]	Keegan P. e Laird, J. E. **Games Survey Analysis**. Disponível em URL: http://www.rpi.edu/~millmd/project/surveyanalysis.html. Consultado em 20/05/2003

[KENT]	Kent, S. L. **Engines And Engineering - What to expect in the future of PC games.** Disponível em URL: http://www.gamespy.com/ futureofgaming/engines/index4.shtml. 31/10/2002
[KIR97]	Kirmse, A. e Kirmse C., "**Security in online games**," Game Developer, vol. 4, no. 4, pp. 20–8, July 1997.
[KUB03]	Kubiatowics J., **Extracting Guarantees from Chaos.** Communications of the ACM. Vol. 46. N 2. Fevereiro/2003
[LAIRD03]	Laird, J. E., Lent, M. van. **The Role of AI in Computer Game Genres**. Disponível em URL: http://ai.eecs.umich.edu/people/ laird/papers/book-chapter.htm. Consultado em 12/04/2003.
[LIME03]	**Limewire.Org.** Disponível em URL: http://www.limewire.org. Consultado em 11/02/2003
[LIVI97]	**Living Worlds Proposal Draft 2**. URL: http://www.vrml.org/WorkingGroups/living-worlds/draft_2/ index.htm. Consultado em 12/04/02
[MASS95]	Chris Greenhalgh and Steve Benford (1995). "**MASSIVE: a Distributed Virtual Reality System incorporating Spatial Trading"**. In Proc, IEEE 15th Int. Conference on Distributed Computing Systems
[MAU02]	Mauve, M., Fischer, S. and Widmer, J., (2002**) A Generic Proxy System for Networked Computer Games**, in ACM NetGames 2002, pp 25-28, Germany.
[M3874]	Grahn, H. & Grini, I. **Working Draft for MPEG-4 Multi-Users worlds architecture and tools.** ISO/IEC JTC1/SC29/WG11, January 2001
[MAC95]	Macedonia, M. R. , **A Network Software Architecture for Large Scale Virtual Environments**, Ph.D. thesis, Naval Postgraduate School, Monterey, CA, June 1995.

[MAC97]	Macedonia, M. R. e Zyda, M. J.. "**A taxonomy for networked virtual environments**," IEEE Multimedia, vol. 4, no. 1, pp. 48–56, 1997.
[MIL02]	Milojicic, D. S. et all. **Peer-to-peer Computing**. Disponível em URL:http://citeseer.nj.nec.com/milojicic02peertopeer.html Consultado em 17/04/2003
[MSDN]	**Topologia P2P para jogos - MSDN – Microsoft Software Developers Network.** Disponível em URL: - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/play/intro/peertopeertopology.asp Consultado em 12/02/2003
[N2201]	ISO/IEC N2201, MPEG-4 Systems. May, 1998.
[N3205]	ISO/MPEG N3205, **Multiusers Technology (Requeriments and Applications) MPEG**, ISO, Maui, Dezembro de 1999.
[N4264]	**Overview of MPEG-4 Standard**, (V.18 – Singapore Version), March 2001
[N4272]	**MPEG-4 - Working Draft 3.0 of ISOIEC 14496-1** july 2001
[N4668]	ISO/IEC N4668, MPEG-4 Overview - (V.21 - Jeju Version), March, 2002.
[NEY97]	David L. Neyland, **Virtual Combat: A Guide to Distributed Interactive Simulation**, Stackpole Books, Mechanicsburg, PA, 1997
[OPEN]	**OpenNap: Open Source Napster Server,** disponível em: http://opennap.sourceforge.net/**,** consultado em 26/05/2003.
[PINH00]	Pinheiro, C. B. Tutorial - **Especificação ITU H.323.** URL: http://penta2.ufrgs.br/h323/indice.htm, Consultado em 10/12/01.

[PROT04]	**The Gnutella Protocol Specification v 0.4. Document Revision 1.2.** Disponível em URL: http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf. Consultado em 23/04/2003
[RIT01]	Ritter, J. . **Why Gnutella Can't Scale. No really.** Disponível em URL: http://www.darkridge.com/~jpr5/doc/gnutella.html. Fevereiro de 2001. Consultado em 02/06/2003
[ROEHL01]	Roehl, B. **Adding Regions to VRML.** URL: http://ece.uwaterloo.ca/~broehl/vrml/regions.html, Consultado em 04/04/02.
[DAROSA99]	da Rosa, A. **Tendências em jogos para Múltiplos Jogadores.** Trabalho de Sistemas de Informação Distribuídos (MSc-UFRGS), 1999. Disponível em URL: www.urcamp.tche.br/~alexsand/personal/JogosEmRede.pdf. Consultado em 12/03/2003
[SABA01]	Sabadello M. **Small Group Multiplayer Games.** Viena University of Technology, Austria. Abril/Maio 2001
[SCHOL02]	Schollmeier, R. **A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architecture and Applications**. First International Conference on Peer-to-Peer Computing. Agosto 27 - 29, 2001 Suécia
[SETI03]	**SETI@home: The Search for Estraterrestrial Intelligence**. Disponível em URL: http://setiathome.ssl.berkeley.edu/ Consultado em 07/05/2003
[SIN01]	Singla A., Rohrs C.. **Ultrapeers: Another Step Towards Gnutella Scalability. Working Draft.** Limewire LLC.Dezembro de 2001. Disponível em URL: http://rfd-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm. Consultado em 13/06/2003

[SCHO03]	Schoder, D. FischBach, K.. **Peer-to-Peer Prospects**. Communications of the ACM. Fevereiro 2003
[SMED02]	Smed, Kaukoranta, and Hakonen, **Aspects of Networking in Multiplayer Computer Games**, The Electronic Library, 20(2):87-97, 2002.
[SZO02]	Szoke M. **Peer-to-Peer Networks. Business Model and Enterprise Application.** Tensa Technology. Janeiro 2002
[THAD02]	Thadani S. . **Meta Information Searches on Gnutella Network.** Limewire LLC. Disponível em URL: http://www.limewire.com/index.jsp/metainfo_searches. Consultado em 13/06/2003
[UD03]	**United Devices**. Disponível em URL: http://www.ud.com/home.htm. Consultado em 16/04/2003
[VIDAL97]	Vidal, P.C.S. **Evolução do padrão MPEG.** Evolução do Padrão MPEG (23 Dec. 1997). URL: http://www.gta.ufrj.br/~vidal/mpeg/mpeg.html. Consultado em 27/05/2002.
[WHA99]	Whatley, D. **Designing Around Pitfalls of Game AI**. In Proceedings of the Game Developers Conference, San Jose, CA, 1999.
[WOLF00]	Wolf, M. J. P. **Genre and the Video Game.** Chapter 6 of The Medium of the Video Game. Disponível em URL: http://www.robinlionheart.com/gamedev/genres.xhtml. Consultado em 13/ 04/2003
[ZAI03]	Zaiantchick, J. B**. Análise sobre o impacto da tecnologia peer-to-peer.** Disponível em URL: http://www.ime.usp.br/~is/ddt/mac339/projetos/2001/demais/bello. Consultado em 07/05/2003

Anexo A

O Esquema Games

O código a seguir representa o esquema XML utilizado no *Limewire*, para refinar busca por jogos.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<schema xml:lang="en"
 xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:i18n="http://www.limewire.com/schemas/i18n"
i18n:locales="http://www.limewire.com/schemas/i18n/games.local
es.xml"
 targetNamespace="http://www.limewire.com/schemas/games.xsd">
  <element name="games" minOccurs="1" maxOccurs="1">
    <complexType>
      <element name="games" minOccurs="1"
maxOccurs="unbounded" type="gameType"/>
    </complexType>
  </element>
  <complexType name="gameType">
    <all>
      <attribute name="Name" type="string"/>
      <attribute name="Developer/Publisher" type="string"/>
      <attribute name="Keywords" type="string"/>
      <attribute name="Genre">
        <simpleType base="string">
          <enumeration value="Adventure"/>
          <enumeration value="Arcade"/>
          <enumeration value="Board"/>
          <enumeration value="Cards"/>
          <enumeration value="Fight"/>
          <enumeration value="First Person Shooter"/>
          <enumeration value="Kids Education"/>
          <enumeration value="Plataform"/>
          <enumeration value="Puzzles"/>
          <enumeration value="Racing"/>
          <enumeration value="RPG"/>
          <enumeration value="Simulators"/>
          <enumeration value="Spaceship Shooters"/>
          <enumeration value="Sports"/>
          <enumeration value="Strategy"/>
          <enumeration value="Versus Fight"/>
        </simpleType>
      </attribute>
      <attribute name="Player">
        <simpleType base="string">
          <enumeration value="Single Player"/>
```

```
            <enumeration value="Multiplayer"/>
          </simpleType>
        </attribute>
        <attribute name="type">
          <simpleType base="string">
            <enumeration value="Demo"/>
            <enumeration value="Complete"/>
            <enumeration value="Patches"/>
            <enumeration value="Expansion"/>
          </simpleType>
        </attribute>
        <attribute name="Game">
          <simpleType base="string">
            <enumeration value="2D Game"/>
            <enumeration value="3D Game"/>
          </simpleType>
        </attribute>
        <attribute name="Value">
          <simpleType base="string">
            <enumeration value="Free"/>
            <enumeration value="Pay"/>
          </simpleType>
        </attribute>
        <attribute name="link" type="uriReference"/>
        <attribute name="comments">
          <simpleType base="string">
            <maxInclusive value="100"/>
          </simpleType>
        </attribute>
        <attribute name="action" type="string"/>
      </all>
    </complexType>
</schema>
```

Anexo B

Os principais componentes que controlam a arquitetura do padrão MPEG-4 MU, o MSC e o MBK, foram escritos usando a linguagem JAVA.

Os componentes estão divididos em três partes: primeiro o MSC e suas classes, em seguida o MBK e suas classes e, por fim, as mensagens utilizadas para controle do MSC e MBK e suas respectivas classes.

As definições seguem a ordem:

- nome da classe;
- o que faz;
- cabeçalho da classe.

MSC – MUTech SessionControl

- **Classe MSC Manager**

É a responsável pelo gerenciamento de todas as operações realizadas pelo MSC

```
public class MSCManager
```

- **Classe Client Manager**

É a responsável pelo gerenciamento dos clientes, para conexões , recebimento e envio de mensagens, entre outras tarefas.

```
public class ClientManager extends Thread
```

- **Classe InfoMngr**

É responsável por comandar os vários InfoSupplier. Cada objeto instanciado InfoSupplier administra uma conexão com socket TCP/IP provendo informações sobre o ambiente comandado pelo MSC/MBK.

```
public class InfoMngr extends Thread
```

- **Classe InfoSupplier**

É a encarregada de estabelecer conexões com clientes ou eventuais futuros clientes do MSC para prover determinadas informações sobre o que ocorre na sessão e zonas que o MSC/MBK está tomando conta atualmente. Cada objeto InfoSupplier instanciado estabelece um socket com algum programa, sendo esse socket TCP/IP e que troca Strings.

```
public class InfoSupplier extends Thread
```

- **Classe InterComMngr**

É a responsável, por controlar as conexões entre MSCs e verificar se há possíveis MSCs controladores de sessão tentando estabelecer conexão.

```
public class InterComMngr extends Thread
```

- **Classe InterComUnit**

É a responsável pela intercomunicação conexão entre os MSCs e pelo controle da hierarquia, ou seja, qual MSC entrou primeiro, quem é o segundo da hierarquia, e assim por diante

```
public class InterComUnit extends Thread
```

- **Classe MSCInterface**

É a responsável pela construção da interface do MSC

```
public class MSCInterface extends JFrame
```

- **Classe MessagesTreatment**

É a responsável pelo tratamento das mensagens MPEG-4 e suas possíveis respostas

```
public class MessagesTreatment
```

MBK – MUTech Bookkeeper

- **Classe MBKManager**

É a responsável pelo gerenciamento das funções do MBK

```
public class MBKManager
```

- **Classe MBKInterface**

É a responsável pela construção da interface do MBK

```
public class MBKInterface extends JFrame
```

- **Classe MsgHandler**

É a responsável pela manipulação das mensagens de atualização do MBK

```
public class MsgHandler
```

Mensagens MPEG-4

- **Classe JoinSessionRequest**

É a responsável pela manipulação da mensagem de entrada numa sessão multiusuário

```
public class JoinSessionRequest
```

- **Classe JoinSessionAcknowledge**

É a responsável pela manipulação das mensagens de permissão/negação a entrada de uma sessão multiusuário

```
public class JoinSessionAcknowledge
```

- **Classe JoinZoneRequest**

É a responsável pela manipulação da entrada numa zona multiusuário

```
public class JoinZoneRequest
```

- **Classe JoinZoneAcknowledge**

É a responsável pela manipulação das mensagens de permissão/negação a entrada de uma zona multiusuário

```
public class JoinZoneAcknowledge
```

- **Classe LeaveSessionRequest**

É a responsável pela manipulação das mensagens de saída de uma sessão

```
public class LeaveSessionRequest
```

- **Classe LeaveSessionAcknowledge**

É a responsável pela manipulação das mensagens de permissão/negação a saída de uma sessão multiusuário

```
public class LeaveSessionAcknowledge
```

- **Classe LeaveAllZonesRequest**

É a responsável pela manipulação das mensagens de saída de todas as zonas

```
public class LeaveAllZonesRequest
```

- **Classe LeaveAllZonesAcknowledge**

É a responsável pela manipulação das mensagens de permissão/negação a entrada de todas as zonas

```
public class LeaveAllZonesAcknowledge
```

- **Classe LeaveZonesRequest**

É a responsável pela manipulação das mensagens de saída de uma zona

```
public class LeaveZonesRequest
```

- **Classe LeaveZonesAcknowledge**

É a responsável pela manipulação das mensagens de permissão/negação a saída de uma zona

```
public class LeaveZonesAcknowledge
```

- **Classe MURequest**

É a responsável pelas mensagens de solicitação (*Join Session, leave session*, entre outras).

```
public class MURequest
```

- **Classe MUUpdate**

É a responsável pelas mensagens de propagação das modificações da cena.

```
public class MUUpdate
```

- **Classe ZonePilotShipRequest**

É a responsável pela mensagem de solicitação de Pilotagem

```
public class ZonePilotShipRequest
```

- **Classe ZonePilotShipAcknowledge**

É a responsável pela manipulação das mensagens de permissão/negação em resposta ao pedido de pilotagem de um objeto por um usuário.

```
public class ZonePilotShipAcknowledge
```

- **Classe ZoneReleasePilotShipRequest**

É a responsável pela manipulação das mensagens de fim de pilotagem.

```
public class ZoneReleasePilotShipRequest
```

- **Classe ZoneReleasePilotShipAcknowledge**

É a responsável pela manipulação das mensagens de permissão/negação ao fim da pilotagem

```
public class ZoneReleasePilotShipAcknowledge
```

Anexo C

Artigo enviado para o WEBMÍDIA 2003, que ocorreu em Novembro, 2003 na cidade de Salvador, Brasil.

Uma solução peer-to-peer híbrida para a implementação de jogos em rede baseada no padrão emergente MPEG-4 multiusuário

Marcelo Martins Laffranchi[1], Regina Borges de Araujo[1]
[1]Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
Caixa Postal 676 – São Carlos – SP – Brasil
{laffranchi,regina}@dc.ufscar.br

Abstract. This paper describes the implementation of an architecture to support network 3D virtual games based on the emerging standard multiuser MPEG-4 in a hybrid peer-to-peer Gnutella network. For that, the source code that supports the Gnutella network was modified in order to include a games search service. Two components defined and specified by the emerging standard MPEG-4MU were implemented and integrated into the Gnutella network for games session control and scenes updating. A preliminary evaluation shows that hybrid peer-to-peer networks can support multiuser games, by overcoming the limitations of client-server and pure peer-to-peer communication models.

Resumo. Este artigo descreve a implementação de uma estrutura de suporte a jogos virtuais 3D em rede, baseada no padrão emergente MPEG-4 multiusuário em uma rede *Gnutella peer-to-peer* híbrida. Para isso, o código que implementa a rede *Gnutella* foi modificado de modo a incluir um serviço de busca de jogos e de sessões ativas. Dois componentes definidos e especificados pelo padrão emergente MPEG-4MU foram implementados e integrados à rede *Gnutella* para controle de sessão de jogos e atualização das cenas. Uma avaliação preliminar desses componentes mostra que as redes híbridas *peer-to-peer* podem suportar jogos multiusuário, superando as limitações dos modelos de comunicação cliente-servidor e *peer-to-peer* puro.