UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO DE UMA FERRAMENTA DE COMUNICAÇÃO EM REDE *P2P* PARA JOGOS *MULTIPLAYER* EM FLASH

TRABALHO DE GRADUAÇÃO

Celito Muck Felipetto

Santa Maria, RS, Brasil

2010

DESENVOLVIMENTO DE UMA FERRAMENTA DE COMUNICAÇÃO EM REDE *P2P* PARA JOGOS *MULTIPLAYER* EM FLASH

Por

Celito Muck Felipetto

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (USFM, RS), como requisito parcial para a obtenção de grau de
Bacharel em Ciências da Computação

Orientador: Prof. Dr. Cesar Tadeu Pozzer

Trabalho de Graduação 302
Santa Maria, RS, Brasil
2010

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

DESENVOLVIMENTO DE UMA FERRAMENTA DE COMUNICAÇÃO EM REDE *P2P* PARA JOGOS *MULTIPLAYER* EM FLASH

elaborado por
Celito Muck Felipetto

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Prof. Dr. Cesar Tadeu Pozzer
(Presidente/Orientador)

Prof. Dr. Raul Ceretta Nunes (UFSM)

Prof. Msc. Fernando Bevilacqua (UFFS)

Santa Maria, 3 de dezembro de 2010

RESUMO

Trabalho de Graduação

Curso de Ciência da Computação

Universidade Federal de Santa Maria

DESENVOLVIMENTO DE UMA FERRAMENTA DE COMUNICAÇÃO EM REDE *P2P* PARA JOGOS *MULTIPLAYER* EM FLASH

Autor: Celito Muck Felipetto
Orientador: Cesar Tadeu Pozzer
Local e data da defesa: Santa Maria, 10 de dezembro de 2010.

Torna-se cada vez mais comum a funcionalidade de multiplayer em jogos. Isso aumenta a complexidade do projeto de criação de jogos. A produtora de jogos Decadium Studios, que produz jogos com essa funcionalidade, investe para resolver problemas de comunicação entre jogos. Com a parceria da Decadium, esse projeto desenvolveu uma ferramenta que organiza salas para partidas multiplayer e efetua a comunicação durante partida sem a necessidade de um servidor dedicado. Essa ferramenta de comunicação via rede foi criada para os jogos já desenvolvidos e futuros projetos da Decadium Studios. O trabalho foi desenvolvido na linguagem Action Script 3.0 e deve ser integrado a jogos desenvolvidos na ferramenta proprietária Adobe Flash. Visando a reutilização dos jogos já desenvolvidos da empresa, ela desenvolveu uma biblioteca nomeada Simskeleton com códigos comuns aos seus projetos desenvolvidos em Flash. Os atuais jogos da empresa que possuem modo de jogo *multiplayer* são todos dependentes de um servidor SmartFox. Visto que tal servidor não é de controle da empresa, mas sim de um dos diversos clientes que a empresa mantém, o projeto visa dar uma solução a isso. A ferramenta desenvolvida foi criada para ser integrada à Simskeleton e em outros projetos que não usam a biblioteca, para que, assim, esses possam se comunicar de forma Ponto-a-Ponto (P2P) sem a necessidade de um servidor para troca de informações nas diferentes instâncias dos mesmos jogos.

ABSTRACT

Undergraduate Final Work
Graduation in Computer Science
Federal University of Santa Maria

DEVELOPMENT OF A P2P NETWORK COMMUNICATION TOOL FOR MULTIPLAYER GAMES IN FLASH

Author: Celito Muck Felipetto
Adviser: Prof. Dr. Cesar Tadeu Pozzer
Data and Local: December 2010, Santa Maria

Multiplayer functionality in games is becoming very usual. It increases the complexity of game creation projects as in. The game producer Decadium Studios, that creates games with that functionality, makes investments as to solve problems of communication in games. Alongside with Decadium, this project developed a tool that organizes rooms for multiplayer matches and resolves communications during the game without the need of a dedicated server. This network communication tool was created for games already made and also for future projects of Decadium Studios. This paper's tool was developed using Action Script 3.0 and as such must be integrated in games developed in the proprietary tool Adobe Flash. Aiming to reuse its games, the company developed a library named Simskeleton with codes that are common to all their Flash based projects. Present games of the company that have a multiplayer mode are dependants of a SmartFox server. As the server is not owned by the company, but by one of the many costumers it has, this project tries to bring a solution to this matter. The tool was created to be integrated to Simskeleton and to other projects that don't use the library, so that they can communicate Point-to-Point (P2P) without the need of a server for information exchange between different instances of the same game.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

P2P	*Peer-to-peer*
SDK	Software Develop Kit
IDE	Integrated Development Environment
LAN	Local Area Network
WAN	Wide Area Network
SO	Sistema Operacional
IDL	Interface Description Language
FPS	Frames per Second
API	Application Programming Interface
CG	Computação Gráfica
BBS	Bulletin Board System
MIT	Massachusetts Institute of Technology
MUD	Multi-User Dungeon

SUMÁRIO

1 INTRODUÇÃO

Os jogos eletrônicos, que tomaram grande espaço no mercado de trabalho, tornam-se cada vez mais complexos em suas mais diversas características. Vemos uma evolução de jogos cada vez mais interativos e com diferentes interfaces e cada vez mais compromissados com o realismo e detalhamento de suas cenas. Junto a tal avanço de interatividade, surgiu a tendência dos jogos em terem diferentes instâncias em diferentes máquinas se comunicando, assim proporcionando aos seus jogadores partidas *multiplayer* (do inglês, multi-jogador).

Dentro desse contexto, surgem várias peculiaridades no modo de implementação da comunicação entre jogos. Existem certas características dos jogos que devem permanecer consistentes no decorrer dessa comunicação, tais como: a interatividade dos jogadores, a continuidade e consistência de informação (posição de personagens, mudanças no cenário, etc.) entre outras características específicas de cada jogo.

Surgem então, com essas necessidades, problemas na hora do desenvolvimento de jogos com a capacidade de serem jogados em modo *multiplayer*. Para se desenvolver o modo *multiplayer* se utiliza ou se desenvolve algum *middleware* de comunicação. Em sistemas distribuídos, um *middleware* se classifica como uma camada abstrata a aplicação que efetiva a comunicação com outras máquinas de acordo com protocolos definidos. Logo, conhecendo as peculiaridades do jogo, uma das grandes dificuldades dos programadores é escolher a melhor arquitetura na construção de um *middleware* e no melhor conjunto de protocolos que esse deve conter.

Para se escolher a implementação de *middleware* que melhor se encaixa às necessidades do jogo a ser implementado, deve-se estudar a estrutura e limitações do projeto, tais como servidores disponíveis, capacidade de processamento e banda deles, limitações das máquinas dos possíveis usuários, dificuldade da implementação e preço de ferramentas proprietárias

dentre outras limitações. Visto essa área de estudo esse trabalho relata a implementação do modelo de *middleware* em P2P com protocolo UDP para o módulo *multiplayer* em uma biblioteca para o desenvolvimento de jogos.

1.1 Contextualização

A biblioteca a qual será adicionado essa nova ferramenta de *middleware* será a Simskeleton, uma biblioteca cujos direitos autorais são restritos a sua criadora, a empresa Decadium Studios. Essa empresa surgiu de uma iniciativa de cinco alunos do curso de Ciência da Computação no ano de 2003. A empresa já trabalhou com diversos títulos de jogos entre outros sistemas. Agora mais especializada em jogos casuais, a empresa mantém vários clientes em diversos países.

A empresa criou vários jogos com módulo *multiplayer* para a sua cliente, a empresa Sim, localizada na Alemanha. Visando a qualidade e a produtividade nas atividades de programação para criação dos jogos, os programadores da empresa concretizaram uma biblioteca usada atualmente para a produção de diversos jogos na plataforma Flash. Essa biblioteca foi batizada de Simskeleton.

Dando continuidade ao trabalho até então desenvolvido, esse trabalho complementa a biblioteca Simskeleton com mais uma arquitetura de sua camada de comunicação entre suas diferentes instâncias. Essa arquitetura para a biblioteca é baseada no modelo *peer-to-peer* (P2P) de comunicação, muito difundido em sistemas distribuídos, e utiliza o protocolo de comunicação *Real Time Midia Flow Protocol* (RTMFP), um protocolo da Adobe que utiliza como base de transporte o *User Datagram Protocol* (UDP).

1.2 Objetivo do trabalho

O trabalho aqui desenvolvido tem como objetivo ampliar as funcionalidades da Simskeleton no aspecto de comunicação *multiplayer*. Os jogos que foram desenvolvidos com base na Simskeleton terão a capacidade de efetivar sua comunicação pelo sistema P2P criado. Essa funcionalidade estará disponível também para os jogos futuros criados com base na biblioteca. A nova ferramenta de comunicação também permitirá que os jogos disponibilizados pela empresa sejam testados em modalidade *multiplayer* pelos seus

jogadores, isso visando a vende e publicidade dos jogos desenvolvidos pela Decadium Studios.

1.3 Estrutura da monografia

Este texto está organizado da seguinte maneira: O capítulo dois descreve toda fundamentação teórica necessária para o desenvolvimento do trabalho, abordando temas como programação de jogos e sistemas distribuídos. O terceiro capítulo trata sobre a situação atual da biblioteca de produção da Decadium Studios e também descreve quais são as modificações a serem feitas e as motivações para isso. No capítulo quatro são descritas quais as bases do projeto teórico e como ele foi planejado. No capítulo cinco são descritas brevemente como funcionam as ferramentas utilizadas como auxilio no decorrer da programação da ferramenta. O capítulo seis trata da implementação desenvolvida e sobre possíveis trabalhos futuros para complementarem a ferramenta. O capítulo final conclui o trabalho citando os resultados atingidos.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo trata dos conceitos de sistemas distribuídos, sistemas P2P e de programação de jogos utilizados para o desenvolvimento desse trabalho. Através da apresentação dessas definições, será possível ter a devida compreensão dos fundamentos utilizados na criação da extensão de biblioteca aqui proposta.

2.1 Sistemas distribuídos

Um sistema distribuído é aquele que está compartilhado entre diversas máquinas, mas, tanto para a camada da aplicação que nele está envolvida quanto para o usuário, aparenta ser um sistema de um único computador. Pode-se encaixar o sistema *multiplayer* desenvolvido nesse projeto como um sistema distribuído. Para uma melhor compreensão sobre o assunto, seus principais aspectos são abordados neste capítulo.

2.1.1 Definições e características de sistemas distribuídos

Em contraste aos sistemas centralizados que dominavam o mercado de sistemas surgiu junto às rede um novo meio de aplicar os mesmos: os sistemas distribuídos. Por definição:

> *"Um sistema distribuído é uma coleção de computadores independentes que parecem aos seus usuários como um sistema único e coerente."* (TANENBAUM e STEEN, 2002)

Uma importante característica de um sistema distribuído é a diferença entre seus vários computadores que o compõe e a maneira como a comunicação entre eles é abstrata para o usuário. Outra importante característica é de que usuários e aplicações podem interagir com um sistema distribuído de uma maneira consistente e uniforme, independente de onde ou quando a interação acontece.

Um sistema distribuído deve ser de fácil expansão e escalabilidade. Essa característica é consequência do sistema ser composto por independentes computadores, ou seja, o sistema deve ser planejado para que o número de computadores seja algo variável visto que podem acontecer adições e remoções de membros do grupo de máquinas e também falhas nas máquinas componentes do sistema. A reposição e concerto de parte do sistema não devem interferir no seu montante.

Para que sistemas distribuídos tenham suporte a um grupo de computadores e sistemas operacionais heterogêneos, esses se organizam em camadas de aplicação que possuem tarefas diferentes. A camada mais baixa é aquela referente ao sistema operacional local. A camada que efetiva a comunicação de maneira abstrata para a aplicação é a camada conhecida como *middleware*. A camada mais próxima ao usuário é aquela em que a aplicação distribuída se encontra. A figura a seguir demonstra a organização dessas camadas de maneira gráfica.

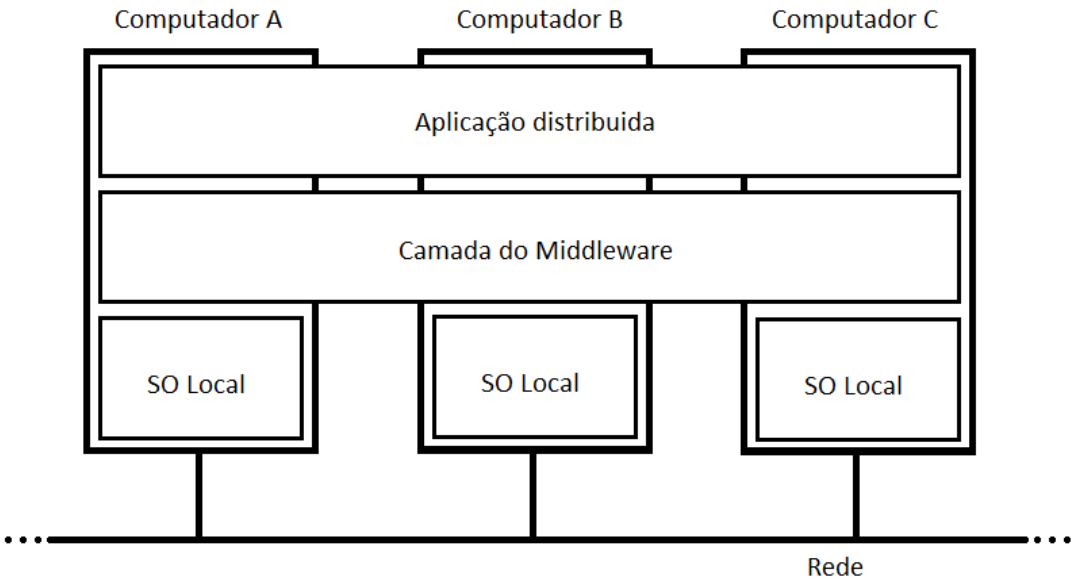

Figura 2.1: Organização de um sistema distribuído em camadas. (COULOURIS, DOLLIMORE e KINDBERG, 2007)

2.1.2 Objetivos de um sistema distribuído

A capacidade de se construir um sistema distribuído não define sua utilidade. Logo, tem-se que definir quais os objetivos em se construir um sistema distribuído para definir se existe a necessidade do mesmo no ambiente em que ele será inserido.

O objetivo principal de um sistema distribuído é facilitar a conexão entre usuários e recursos. Esses recursos podem ser virtualmente qualquer coisa. Os exemplos clássicos são impressoras, computadores, espaços para armazenamento, dados, arquivos, páginas Web e redes. O motivo mais claro do porque a necessidade de conexão entre usuários e recursos é pelo motivo econômico, pois é mais barato manter menos cópias dos recursos sendo que essas são acessadas por vários usuários, como, por exemplo, uma impressora em um andar de um prédio comercial.

Conectar usuários e recursos também permite que mais facilmente usuários trabalhem colaborativamente e trocando informação. Por exemplo, sistemas que trabalham com versionamento de dados (SVN, GIT, etc.) são uteis no trabalho colaborativo na criação de software.

Outro objetivo muito comum na construção de um sistema distribuído é esconder o fato de que os recursos estão distribuídos fisicamente em múltiplos computadores. Chama-se transparência a característica de um sistema distribuído de se apresentar ao usuário e a aplicativos como um sistema de um único computador. Existem diferentes tipos de transparência para diferentes objetivos. Pode-se qualificá-las como a tabela abaixo mostra:

Tabela 2.1: Diferentes formas de transparência em um sistema distribuído (COULOURIS, DOLLIMORE e KINDBERG, 2007)

Transparência	Descrição
Acesso	Esconde as diferenças de como os dados estão representados e como eles são acessados.
Localização	Esconde onde os recursos estão localizados.
Migração	Esconde o fato de que um recurso talvez tenha sido movido para outra localização
Realocação	Esconde o fato de que um recurso talvez precise ser movido para outra localização enquanto é usado.
Replicação	Esconde o fato de um recurso ser replicado.
Concorrência	Esconde o fato de que talvez um recurso seja compartilhado por alguns usuários competitivos entre si.
Falhas	Esconde a falha e a recuperação de um recurso
Persistência	Esconde se o um (software) recurso está em memória ou em disco.

Existe ainda o objetivo de abertura para um sistema distribuído. Um sistema distribuído aberto é aquele que fornece serviços de acordo com regras padronizadas que descrevem a sintaxe e a semântica desses serviços. Essas formalizações de como se dá o serviço também podem ser chamadas de protocolos. Em geral nesse tipo de sistema os serviços que ele provém são especificados em interfaces, as quais são descritas em Linguagem de Definição de Interface (IDL). As definições descritas em IDLs quase sempre apenas apresentam a sintaxe dos serviços. Em outras palavras, elas especificam os nomes de funções que podem ser acessadas por usuários e aplicativos para que o serviço do sistema seja cumprido. Junto aos nomes das funções segue os parâmetros dessas, os valores de retorno, suas possíveis exceções entre outras características. A parte semântica dos serviços normalmente não é descrita ou é simplificada e informal.

2.1.3 Modelos de Arquiteturas de Sistemas Distribuídos

A arquitetura de um sistema é a sua estrutura em termos de componentes especificados separadamente. O objetivo de uma arquitetura é de que ela atenda as demandas que um sistema tem e pode vir a ter. As demandas de um sistema normalmente são referentes à sua confiabilidade, gerenciabilidade, adaptabilidade e rentabilidade.

Existem em sistemas distribuídos várias arquiteturas para as diversas situações e necessidades do projeto para qual esse sistema está sendo construído. As seções a seguir apresentam as arquiteturas mais comuns de sistemas distribuídos.

2.1.4 Cliente-servidor

Essa é a arquitetura mais comum em sistemas distribuídos. Ele é baseado em clientes que hospedam interfaces do sistema que por sua vez tem função de acessar o servidor que contem serviços que devem resolver invocações feitas pelos usuários. Toda necessidade de um cliente em acessar um serviço do servidor gera uma invocação do servidor que por sua vez gera outra mensagem em resposta para aquele devido cliente.

O servidor normalmente é o detentor da maioria dos recursos em um sistema ou de pelo menos as indicações de onde os recursos estão localizados. Isso ocorre para que todos os clientes tenham um consenso de onde procurar os diversos recursos disponibilizados no sistema.

O servidor numa arquitetura cliente-servidor torna-se um problemático ponto fraco na tolerância de falhas por ser o único sanador de invocações e detentor da maioria dos recursos. Um servidor falho significa um sistema todo falho quando assim arquitetado. Então, apesar de ser um sistema fácil de ser implementado, não é considerado um sistema tolerante a falhas.

Esse tipo de sistema também pode ser hierarquizado. Pode haver requisições de um servidor para outros servidores, torando assim o servidor de um sistema o cliente em outro. A figura a seguir demonstra a idéia de hierarquia entre servidor e invocações de clientes.

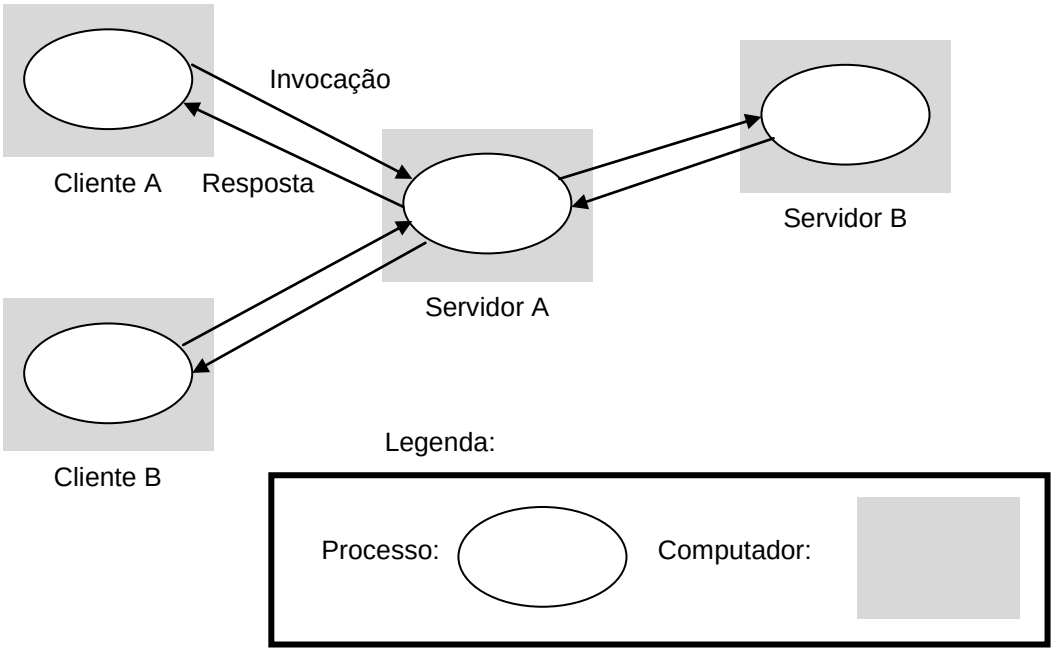

Figura 2.2: Organização de um sistema cliente-servidor. (COULOURIS, DOLLIMORE e KINDBERG, 2007)

2.1.5 Sistemas Peer-to-Peer (P2P)

Diferente do sistema centralizado Cliente-Servidor, nos sistemas P2P todos os membros existentes são hosts capazes de fornecer serviços e recursos ao grupo de usuários que estão conectados. Ou seja, é como se, em um sistema cliente-servidor, fosse extinto a necessidade do servidor e dos clientes fazerem invocações uns aos outros ao invés de todos ao mesmo endereço. Shirky (SHIRKY, 2000) definiu os aplicativos P2P como “aplicativos que exploram os recursos disponíveis nos limites da internet – armazenamento, ciclos de processamento, conteúdo, presença humana”. A figura 2.3 é uma exemplificação de um sistema P2P:

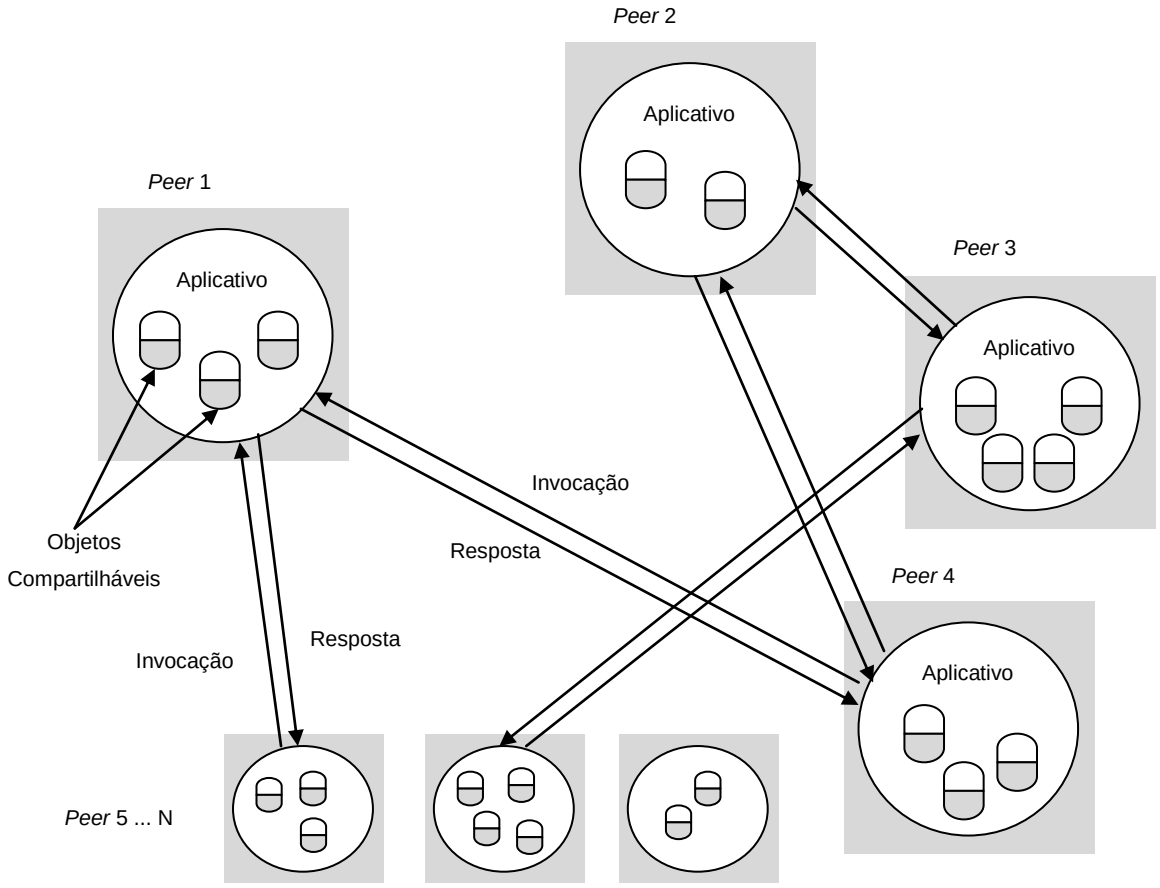

Figura 2.3: Organização de um sistema peer-to-peer. (COULOURIS, DOLLIMORE e KINDBERG, 2007)

P2P não é uma idéia nova: a ARPANET, princípio da internet contemporânea, foi projetada para ser um sistema P2P que conectava certas universidades e instalações do governo dos EUA. Com expansão da ARPANET e o surgimento de vários serviços baseados em protocolos cliente-servidor (Telnet, e-mail entre outros), sistemas P2P não eram tão atrativos, visto o custo e a inviabilidade tecnológica de fazer com que os todos os computadores clientes tivessem hardware e banda suficiente para dividirem o pesar do processamento dos serviços prestados pelos servidores. O ressurgimento dos sistemas P2P foi uma consequência de acréscimo no número e na largura das conexões e na capacidade do hardware dos seus diversos usuários. A internet chega hoje a milhões de usuários. (MARSHALL, 2010)

O sistema P2P mais comentado e que renovou a idéia de compartilhamento de arquivos de maneira massiva foi o Napster (OpenNap, 2001). Este foi um sistema criado em 1999 para o compartilhamento de musicas entre usuários. No seu auge, o Napster, chegou a ter vários milhões de usuários cadastrados. Ele não era um programa puramente P2P por ter

vinculado ao sistema um servidor de localização de arquivos/clientes (*tracker*). Outro sistema, não tão famoso, a usar P2P para sua comunicação foi o sistema *multiplayer* do jogo Doom, o qual independia de quaisquer servidores interagindo com o jogo para que os jogadores pudessem efetivar partidas na rede (SHIRKY, 2000). Bastava que os jogadores identificassem um *Host* (um dos jogadores responsável por orientar o grupo no inicio da partida) que a partir de então todos os jogadores tornavam-se membros igualitários para o grupo como clientes.

2.1.6 TCP/IP e UDP

Para efetivar a troca de dados entre um sistema distribuído certo protocolo de comunicação deve ser definido. Os protocolos usados hoje são formados por diversos outros protocolos com fins mais claros e específicos na tarefa de comunicação entre computadores. Dentro os diversos protocolos existentes para formar protocolo completos de comunicação, pode-se classificar esses entre as cinco camadas da pilha de protocolos TCP/IP. O nome dessa pilha de protocolos vem de Transmission Control Protocol (TCP) e de Internet Protocol (IP) os primeiros dois protocolos de comunicação a serem definidos. (TCP/IP Artigo da Wikipédia, 2010)

O conjunto de protocolos TCP/IP abrange as cinco diferentes camadas que devem trabalhar independentes uma das outras:

- Camada de aplicação: a camada mais próxima ao usuário. Ela deve implementar a interface com o programa que a implementa. Os processos que rodam nessa camada são específicos da aplicação. Praticamente se recebem da aplicação os dados no formato interno ao código e se codificam eles (ou decodificam no retorno) em dados do formato definido no protocolo.
- Camada de transporte: essa é a camada responsável pelo dado conseguir alcançar sem erros seu destino, característica também conhecida como confiabilidade, e pela propriedade dos dados chegarem na ordem correta ao seu destino, propriedade também chamada de integridade. Nem todos os protocolos dessa camada se comprometem com total confiabilidade ou integridade. Visando uma menor quantidade de dados a ser repassado o protocolo que define as regras de transporte pode se comprometer apenas com uma fraca confiabilidade e com nenhuma garantia de integridade como no caso do protocolo UDP.

- Camada de rede: a camada que se deve direcionar através de um protocolo de endereçamento os pacotes de dados através da rede. O exemplo mais conhecido dessa camada é o protocolo de endereçamento IP, hoje difundido por toda internet.
- Camada de enlace: essa camada define protocolos para adição de *headers* aos pacotes que serão despachados pela rede física. Esses protocolos pode ser implementados tanto em nível de software (*device driver*) quanto em de hardware nas placas de rede e outros dispositivos que compõem uma rede. A responsabilidade dessa camada é de endereçamento, roteamento e controle de envio e recepção dos pacotes que vão circular pela rede.
- Camada física: essa camada trata dos protocolos de deslocamento físico de ondas eletromagnéticas ou físicas para a troca de informações.

O projeto de um sistema normalmente não é dependente dos protocolos de comunicação na qual essa deve funcionar, como é o caso do projeto desenvolvido nesse trabalho. Isso ocorre pelo fato dos protocolos estarem diferenciados por camadas que trabalham independentes da implementação das outras. Mas para a implementação do mesmo, foram definidos os protocolos da camada de aplicativo e camada de rede. As outras camadas são dependentes da plataforma e da rede em que o sistema estiver sendo rodado.

O protocolo da camada de aplicação para a implementação desse trabalho é proprietário da Adobe RTMFP (Real Time Media Flow Protocol - RTMFP , 2009). Esse é um protocolo desenvolvido para aplicações criadas com a ferramentas da Adobe e tem sua otimização voltada para troca de grande quantidade de informações que não precisam de confiabilidade em seus pacotes. Essa não necessidade de sistemas para confiabilidade de pacotes é comum em *streamings* de vídeo e áudio ou em jogos. Na sua versão atual, a utilizada pelo Flash Player 10.1, o protocolo RTMFP tem implementado como seu protocolo da camada de transporte padrão o UDP. A adobe prevê possíveis atualizações para que o protocolo aceite outros protocolos de transporte.

O Protocolo UDP implementado pelo RTMFP é um protocolo não confiável descrito pela RFC 768 (POSTEL, 1980). Esse protocolo permite que o aplicativo encapsule seus pacotes IPv4 ou IPv6 para que sejam enviados ao seu destino. Visto sua simplicidade e sua falta de confiabilidade quem o implementa tem a opção de aplicar diversos algoritmos de estruturas de controle como timeouts, retransmissões, *acknowlegments*, controle de fluxo, etc.

O protocolo RTMFP implementa seus próprios algoritmos para melhorar a confiabilidade do UDP. (User Datagram Protocol Artigo da Wikipedia, 2010)

2.2 Conceitos básicos de programação de jogos

Jogos eletrônicos podem ser tanto jogos de computador quanto jogos para console. Um jogo de computador é um aplicativo como qualquer outro programa. Jogos para plataformas que não o computador também são aplicativos, com a diferença de conter características específicas para o console que roda a aplicação. O que diferencia um jogo de uma aplicação são algumas características comuns a todos os jogos.

Um jogo eletrônico deve ser executado em tempo real. Ou seja, sua execução deve passar noção de passagem de tempo e movimento. Essa característica não se compromete com a interatividade do usuário, possibilitando inclusive que a execução de um jogo seja assíncrona a entrada do usuário. A noção de movimento de um jogo está estritamente ligada à taxa de atualizações da tela de jogo. Essa taxa se refere a quantas vezes por segundo é redesenhada a tela para realizar a noção de movimento dos objetos em jogos. Ela também é conhecida como FPS (*frames per second*).

Segundo os experimentos do físico belga Plateau a taxa de quadros por segundo que o olho humano é capaz de captar é de 16 quadros por segundo, isso é a taxa que passa ao espectador uma impressão de continuidade de movimento (Joseph Plateau Wikipedia Article, 2010). As taxas de FPS mais comuns são as do formato NTSC (National Television Standards Committee) cerca de 30 FPS, do formato PAL (Phase Alternation by Line) com aproximadamente 24 FPS e a de muitos monitores de computadores modernos de 60 FPS a 120 FPS (HARRIS, 2000).

Um jogo também deve conter técnicas de computação gráfica (CG) em tempo real. Para aplicação dessas técnicas são necessárias APIs gráficas como OpenGL e DirectX. Essas APIs proporcionam ferramentas para a criação de ambientes ideais para a renderização rápida de cenas dos jogos. Para se conseguir utilizar tais ferramentas é necessária uma placa de vídeo aceleradora.

Todo e qualquer jogo eletrônico também deve conter algum tipo de interface com usuário. Ao menos uma interface de saída visual, usualmente um monitor ou tela semelhante e ao menos uma interface de entrada como teclado e mouse, para maioria de jogos de

computador e também controles, microfones, manches de simulação, luvas e capacetes de realidade virtual entre outros diversos dispositivos.

Integrando esses conceitos básicos de como se define um jogo, o cerne de sua produção se centraliza na programação computacional. A programação de jogos se dá em múltiplas linguagens de programação e diversas plataformas. Existem hoje diversas ferramentas, bibliotecas e *engines* (Game Engine Wikipedia Article, 2010) para o auxilio dos programadores de jogos visto a complexidade na sua produção.

2.3 Programação de jogos *multiplayer*

O primeiro jogo que se tem noticia a ser jogado online foi criado em 1969 por Rick Blomme, estudante do MIT em Massachusetts, EUA. O jogo era do estilo *Bulletin Board System* (BBS) onde um administrador recebia diariamente as jogadas de jogos de tabuleiros pelo sistema da universidade de diversos estudantes e respondia com o resultado das jogadas no tabuleiro como se tivessem sido feitas simultaneamente. Apesar de terem sido criados antes, não foram os BBS que deram origem a série de jogos que hoje se espalham como os mais diversos jogos online. Os jogos *Multi-User Dungeon* (MUD), populares nas universidades da Inglaterra na década de 80, tinham um sistema *multiplayer* muito mais parecido com o que vemos nos jogos hoje em dia. Esses sistemas eram baseados em múltiplas entradas de texto em um servidor. Os textos descreviam pequenas ações que os jogadores pretendiam efetivar como os seus personagens do jogo. Dependendo dos textos inseridos pelos usuários os servidores respondiam propriamente com outro texto que descrevia os eventos desencadeados pela ação executada do jogador (History Of Multiplayer Games, 2007).

Atualmente um grande número de jogos se comunica com outros pela internet ou pela rede. Os jogos vão de partidas de dois jogadores a jogos feitos para suportar milhares de jogadores online simultaneamente como no caso de jogos Massive Multiplayer Online (MMO).

A idéia mais intuitiva para definir uma comunicação entre instâncias de jogos é a de fazer com que os diferentes aplicativos troquem todas as informações referentes à suas execuções do atual frame na mesma taxa que o FPS de um jogo. Em um caso ideal de redes e de projetos sem limitações, essa sempre seria a melhor estratégia para implementação de um sistema

multiplayer. Mas essas limitações existem e são definidas pela largura da banda dos jogadores, pela capacidade dos hardwares dos usuários e dos servidores e pelas restrições do projeto. Visto isso, torna-se praticamente impossível a criação de um jogo que mantenha uma taxa aceitável de quadros por segundos em uma partida *multiplayer* de muitos jogadores. O sistema *multiplayer* de um jogo deve ser um sistema de troca de mensagens em frequência assíncrona a de FPS e com mensagens de conteúdo de extrema importância, ou seja, apenas o necessário para manter a percepção para os jogadores de que o jogo se mantém síncrono.

A maneira com que jogos se comunicam online não está restrita a um tipo de protocolo. Visto isso, existem os mais diferentes sistemas distribuídos para a comunicação entre jogos. Os jogos podem se comunicar com auxílio de um servidor, ou independes de um com troca de mensagens entre os jogadores (P2P). Jogos *multiplayer* podem implementar sistemas tolerantes a falhas dos membros ou bloqueantes. Podem se comunicar com protocolos mais seguros de troca de dados como TCP ou protocolos mais rápidos como UDP. Todo tipo de configuração do sistema distribuído de um jogo é definido segundo suas necessidades e peculiaridades.

Num sistema de comunicação implementado em um jogo *multiplayer* não se é usado apenas a sua confiabilidade em manter a informação coerente como critério eficiência. O desempenho em relação a velocidade de troca de mensagens também é outro critério muito comum a ser avaliado. Para isso, se utiliza o termo *lag* (latência) para se rotular o tempo de atraso no envio e recebimento de uma resposta para certo comando que deve ser enviado. A unidade comum a ser usada para tal medição de tempo é o milissegundo. Uma forma comumente utilizada para medir o *lag* em sistemas multiplayer é o programa *ping* (Ping Artigo da Wikipédia, 2010).

A tolerância de *lag* em um jogo depende do tipo de jogo que está em execução. Em caso de jogos baseados em turnos ou de baixa ação, um *lag* alto não é considerado um problema, pois na execução do jogo a demora de resposta dos outros jogadores ou das próprias ações tomadas não atrapalha as condições de jogo. Já em jogo de resposta rápida, como no caso dos jogos do tipo *first person shooter* (FPS), o *lag* tem reflexo direto nas condições de jogo dos diversos jogadores. Nesses casos o lag deve ser mantido o menor possível. Isso não depende só da estrutura do jogo, mas também de toda estrutura de redes e dos computadores que a integram, pois por mais bem planejado que seja um sistema ele ainda está sujeito a problemas como falhas nas máquinas ou falta de banda na rede para a execução do programa. Logo, para

cada tipo de jogo existem necessidades únicas para a comunicação em rede. São essas questões que devem ser trabalhadas em cada protocolo para que a troca de mensagens seja eficiente e supra as necessidades do projeto em comunicação.

3 BIBLIOTECA SIMSKELETON E MOTIVAÇÕES PARA AS CRIAÇÃO DA FERRAMENTA P2P

Neste capítulo explica-se o funcionamento da biblioteca Simskeleton, mais especificamente a camada de comunicação baseada na arquitetura cliente-servidor. Assim deve-se entender melhor os motivos da atual implementação e do porque e como devem ocorrer as modificações nessa biblioteca.

3.1 Funcionamento Geral da Simskeleton

No decorrer dos anos da empresa Decadium, seus contatos na Alemanha geraram certa demanda de jogos casuais *web-browser*. A empresa Sim (SevenOne Intermedia) (SevenOne , 2010) criou um portal de jogos desenvolvidos em Flash e em linguagem AS3 (Seven Games, 2010), sendo que todos esses partilhavam diversas características, como: pontuações diferenciadas para cada jogador em um *rank*, poderes especiais para jogadores pagantes (*Power-ups*) e a capacidade de multi-jogadores. Alguns desses jogos foram terceirizados para serem desenvolvidos pela Decadium. Como os jogos se assemelhavam em características de desenvolvimento deu-se início a criação de uma biblioteca que contivesse todo código comum e útil aos jogos desenvolvidos para a Sim. Esse concentrado de código foi batizado de Simskeleton. A biblioteca expandiu-se de tal maneira que se tornou útil em várias ocasiões e inclusive para outros jogos que não os voltados para esse cliente.

3.1.1 *Multiplayer* para Jogos da Sim

O trabalho dos programadores para o desenvolvimento do *midleware* engloba duas fases distintas. A primeira, para cada novo jogo, deve-se fazer uma análise para se definir quais devem ser as mensagens trocadas e tentar diminuir ao máximo o número dessas mensagens; depois, na parte de programação, deve-se manter os jogos sincronizados e de evitar erros de

coerência de informações entre as instâncias dos jogos com as mensagens estipuladas. A Simskeleton abstrai a parte mais simples e comum a todas as implementações do modulo *multiplayer*: os protocolos e a efetivação da troca de mensagens entre os jogadores. Ela simplesmente disponibilizava aos vários jogos uma função de envio de mensagem a todos os jogadores dessa seção. Esse processo também é conhecido como *broadcast*.

Para efetivar essa tarefa, a Sim disponibilizava para os seus jogos um servidor Smart Fox (SmartFox, 2010). Esse é um servidor proprietário da companhia gotoAndPlay (gotoAndPlay, 2008) para comunicação massiva entre aplicativos em plataformas como Flash/Flex/Air, Java, Android, .Net, Unity3D, SilverLight entre outros. No sistema de comunicação da Simskeleton seus jogos enviam as mensagens diretamente ao servidor da Sim e esse, por sua vez, fazia o processo de reenviar a todos os outros jogadores envolvidos na partida, efetivando assim um *broadcast* pelo servidor.

Para que fossem iniciadas as partidas em modo *multiplayer* os jogos eram invocados com parâmetros específicos. Esses parametros provinham do própio site que invocava o programa e descreviam o numero de jogadores e como deveria proceder a partida *multiplayer*. Os jogadores, para isso, escolhiam se a execução do jogo deveria ser em modo solo ou modo *multiplayer*. Ou seja, não existia nenhum tipo de interface em jogo para que o jogador pudesse escolher se gostaria de jogar sozinho ou com outros jogadores pela rede. Para tal feito, todos os jogos da Sim eram criados em dois módulos diferentes: um deles simplesmente executava sem comunicação com o servidor durante a partida, e outro em que eles tinham a capacidade de se comunicar com o servidor Smart Fox que saberia a existência e localização dos outros jogadores.

3.2 Criação da ferramenta P2P para a Simskeleton

A fim de demonstrações e propaganda, a Decadium Studios tem interesse que seus jogos tenham a capacidade de se comunicar sem a necessidade de um servidor Smart Fox. Houve então a análise de meios alternativos para comunicação em partidas multi-jogadores para demonstração de seus sistemas. Na análise sugeriu-se a utilização de uma nova tecnologia experimental da companhia Adobe em seus aplicativos Flash: a comunicação P2P. Essa nova tecnologia está apenas disponível para rodar em Flash Players 10.1 e só pode ser compilada com Flex SDK 4 (explicado na seção 5.1).

Um dos propósitos deste trabalho é o planejamento e definição de um conjunto de modificações na Simskeleton para que essa biblioteca disponibilize aos jogos uma ferramenta para comunicação entre os jogadores independente de outros servidores no decorrer da partida, necessitando apenas de um *tracker* de endereço estático, como descrito na seção 2.1.5, para que esse auxilie o cliente a se juntar ao grupo de jogadores. Ademais, decidiu-se que os jogos tivessem também um cenário com interface ao jogador para que esse pudesse escolher qual jogo deseja jogar e pudesse criar salas para jogar entre outros usuários.

4 PROJETO CONCEITUAL

Esse capítulo deve esclarecer como é projeto conceitual da ferramenta implementada nesse projeto. O projeto conceitual é independente de qualquer ferramenta ou protocolo a ser usado.

4.1 Funcionamento dos sistemas *multiplayer* de jogos

Os jogos *multiplayer* apresentam diversos tipos de interações com o usuário. Cada jogo, com suas peculiaridades, apresenta interações específicas; Mas algumas características em muitos deles se assemelham. Essas características comuns dizem respeito principalmente à ordem dos eventos para que o jogador tenha seu jogo conectado a outros. Para meios de entendimentos, divide-se os jogos entre os que possuem partidas e os que não possuem. Uma partida é um evento de jogo definido, com início e fim, que pode ter instâncias diversas e pode ser repetida diversas vezes por um mesmo jogador. Um exemplo de estilo de jogo que não possui partidas são os *Masssive Multiplayer Online Role Plaing Game* (MMORPG) (MMORPG Artigo da Wikipédia, 2010), visto que são jogos onde os jogadores acessam seus personagens numa instância de partida consistente, ou seja, que se mantém mesmo com os jogadores não mais influenciando nela, e que tem seu desenrolar influenciado por todos os jogadores que se encontram no mesmo servidor. No caso de jogos com partidas definidas, que são os que interessam ao sistema aqui desenvolvido, pode-se classificar a interface de interação com o jogador em três fases: pré-partida, partida e pós-partidas.

As funções mais comuns dessas fases são bem definidas. A fase da pré-partida permite ao jogador normalmente visualizar todas as partidas que lhe são permitidas participar e ingressar numa sala com os jogadores que da partida irão participar. O modo de localizar a partida que interessa ao jogador varia, visto que muitas vezes o jogador visa diferentes critérios para sua próxima partida. Os critérios comuns aos jogadores na maioria dos casos são o valor da

medição do ping entre o servidor e o jogo ou entre os demais jogadores, localidade do servidor ou dos outros clientes, afinidade do jogador com os outros, classificação dos jogadores participantes da partida em certa pontuação, entre outros critérios. Desse modo, na pré-partida deve ser apresentado o maior número de dados relevantes para o jogador decidir que partida esse deve jogar. Em casos de complexidade desnecessária ou abstrata ao jogador, o servidor em certos casos define as partidas a serem formadas. Um exemplo disso é o caso do Starcraft 2 (Starcraft 2, 2010) onde segundo a Blizzard (Blizzard Entertainment, 2010), sua produtora, para ser mais justa na forma de escolha de adversários para partidas no jogo essa deveria ser de controle exclusivamente do servidor e não dos jogadores (Match Making Article, 2010). Na sala em que os jogadores se reúnem quando escolhida a partida é muito comum se apresentar as informações relevantes aos outros jogadores e um modo de comunicação com eles. Na figura 4.1 tem-se um exemplo de sala de pré-partida do jogo Starcraft 2.

Figura 4.1: Sala de pré-partida do Starcraft 2 (Starcraft 2, 2010)

Em grande parte, os jogos têm seu modo de partida solo como padrão, normalmente relacionado a uma campanha de fases a serem vencidas para que o jogo se dê por concluído. Também é comum jogos apresentarem um modulo de jogo *skirmish* ou *custom* onde o jogador tem a opção de definir as primitivas da partida a qual vai jogar. Visto que o jogador deve

optar por algum dos módulos que o jogo apresenta é comum se apresentar inicialmente um menu de opções. A figura 4.2 é um exemplo de menu de opções inicial.

Figura 4.2 : Menu de exemplo do jogo Empire Total War (Total War, 2010)

Do mesmo modo que o menu de exemplo, a maioria deles tem sua própria seção na parte de multiplayer, visto que muitos deles apresentam diferentes sistemas de pré-partida. Os sistemas normalmente se diferenciam pela área que englobam, como, por exemplo, um sistema de pré-partida que agrupa jogadores que estão numa mesma LAN, ou jogadores por toda WAN ou até casos de multiplayer sem comunicação externa onde os jogadores dividem a interface do jogo em uma mesma plataforma.

No caso de sistemas para pré-partidas na internet (WAN), essas normalmente, por questões de limitações de banda de internet dos jogadores são disponibilizados servidores dedicados, os quais desempenham papel de *broadcaster* para os jogadores e sincronizadores do jogo, validando ações e eventos para os clientes que nele estiverem conectados. Nem sempre essas ferramentas de pré-jogo são internas aos jogos; muitas vezes elas são implementadas por outras companhias ou até por outros programas, como no caso do Game Spy (Game Spy, 2010), Garena (Garena, 2010) e do Steam (Steam, 2010). Na figura 4.2 temos uma tela de exemplo do Game Spy, onde a ferramenta disponibiliza as partidas conhecidas pelo servidor do Game Spy que estão na fase de pré-partida.

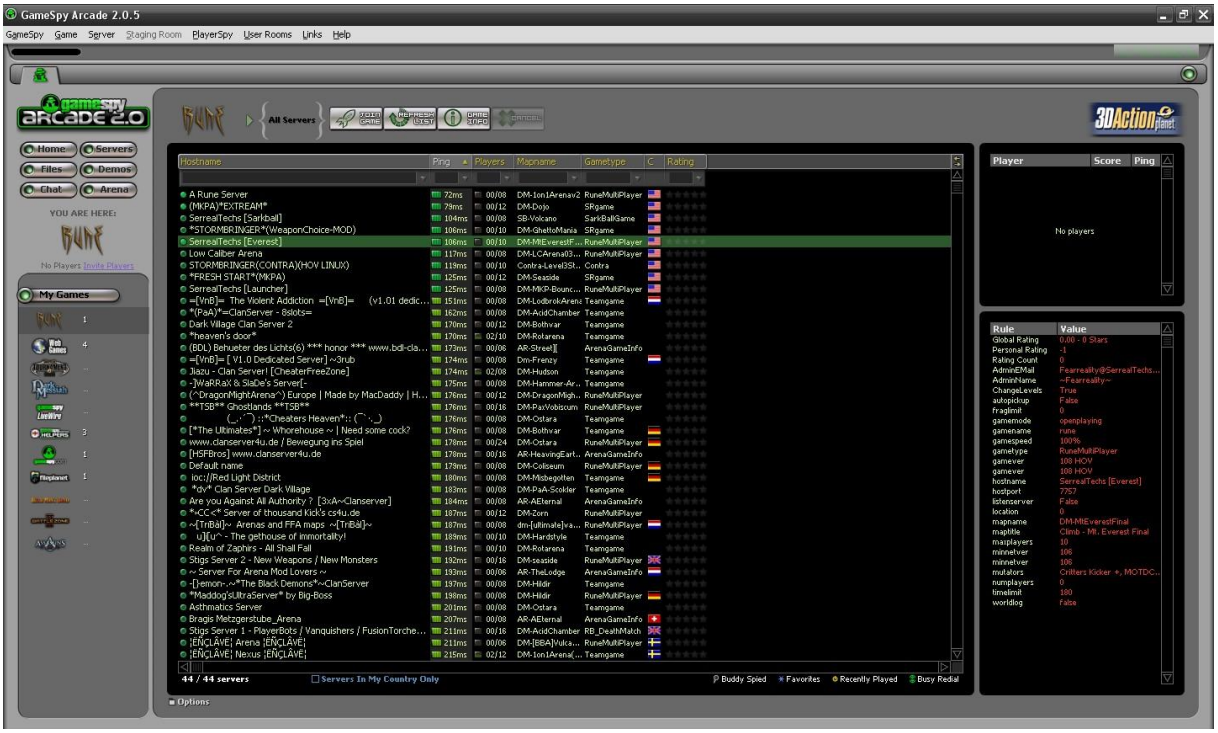

Figura 4.3:Tela de seleção de partida do Gamespy (Game Spy, 2010)

Já em pré-partidas a âmbito de LAN, as ferramentas para se encontrar partidas e se conectar aos jogadores da partida são implementadas internamente aos jogos e geram partidas em sistemas P2P ou com uma das próprias instâncias do jogo como servidor das outras. Isso é feito desse modo, porque LANs não possuem problemas encontrados em conexões em partidas pela internet e visto que LAN possuem em geral uma banda maior entre os computadores membros dessa.

A implementação do processo do modo *multiplayer* no decorrer da partida normalmente é resumida a fazer as trocas de dados do jogo, trocas de mensagens de chat entre os jogadores e/ou trocas de *stream* de áudio/vídeo. A implementação da parte de pós-partida multiplayer é muito dependente das informações relevantes a serem exibidas no final de uma partida, tais como pontuação dos jogadores no decorrer do jogo, classificação dos jogadores, gráficos com a desenvoltura dos jogadores através do tempo, etc.

4.2 Proposta de arquitetura de sistema *multiplayer* baseada em grupos de usuários

Visto as características que os jogos multiplayer atualmente contem, planejou-se uma ferramenta que adicionasse tais funcionalidades e fosse baseada num funcionamento não dependente de um servidor que tivesse de atender todas as requisições dos jogadores, sendo

que essas deveriam ser resolvidas entre o grupo dos mesmos. O projeto organiza o sistema de modo que ele efetue ao jogo toda a fase de pré-partida, comunicação durante a partida e pós-partida. Para tal organização foram definidos os passos dos jogadores a fim de iniciar, efetivar e concluir uma partida de modo abstrato ao jogo. A única preocupação dos programadores será a de preparar a interface do programa com a ferramenta e definir como deve ser a interface do jogador com o programa, para que esse interaja com a ferramenta.

O sistema está organizado em diversos grupos. O grupo de todos os jogadores online que ainda não estão em nenhuma partida será tratado como *Maingroup*. O grupo de jogadores que já está em uma sala exclusiva para os jogadores da mesma partida ainda na fase de pré-partida será tratado como *Roomgroup*.

O primeiro passo de um jogador que deseja se conectar com outros para entrar em alguma partida é o de se juntar ao *Maingroup*. O ato de se juntar ao *Maingroup* nem sempre deve ser uma ação livre, visto que muitas vezes para se jogar uma partida, o jogador deve se identificar com um cadastro num banco de dados para que sejam armazenados seus dados referentes a pontos ou para se validar seu produto, por exemplo. Logo antes do ato de se juntar ao *Maingroup*, o usuário deve passar primeiro por uma fase de *login* para que esse seja identificado. O sistema de *login* deve ser implementado independente do sistema *multiplayer* projetado, visto que esse precisa de um servidor com banco de dados para validação das identificações. O sistema então apenas contém uma interface com o sistema de *login* que não está no âmbito desse projeto. O esquema da figura 4.4 representa a fase de identificação do usuário. O *Login Server* não necessariamente deve ser um servidor diferente do tracker como mostrado na figura.

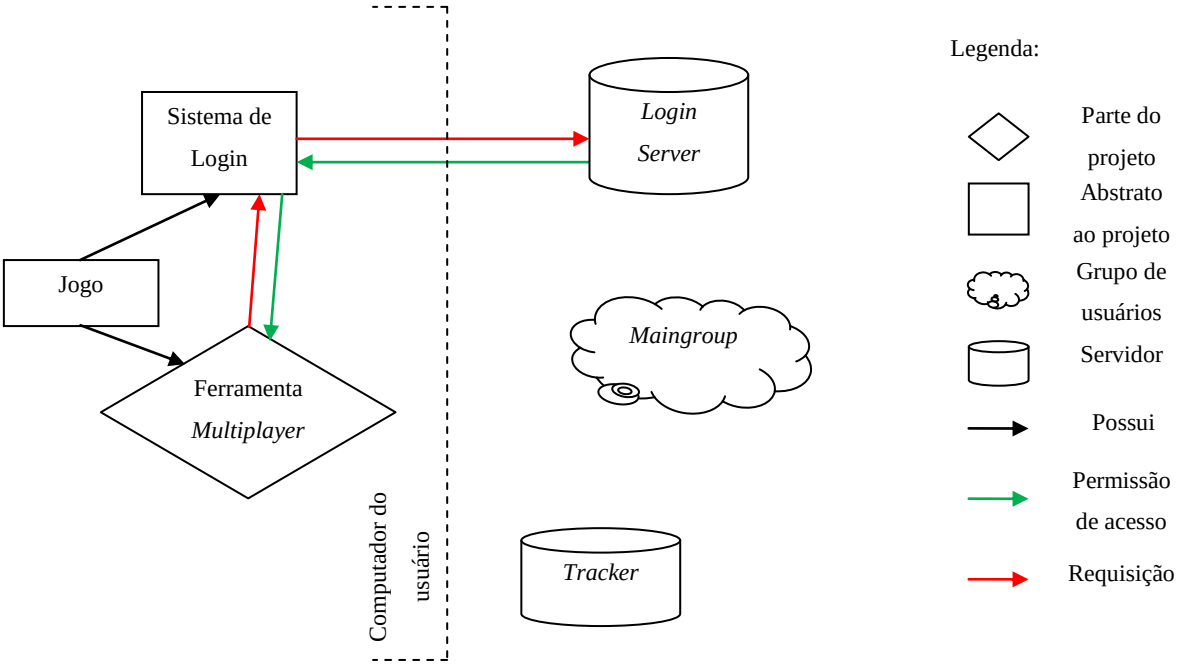

Figura 4.4: Requisição de *login*

Após aprovado pelo sistema de *login* o usuário ainda não pode se juntar ao *Maingroup* visto que ele não conhece o endereço de nenhum dos membros do grupo. Para que então possa se conectar esse deve fazer uma requisição a um servidor *tracker* que identificará o endereço a qual o *middleware* deve se direcionar para se comunicar com os membros do *Maingroup*.

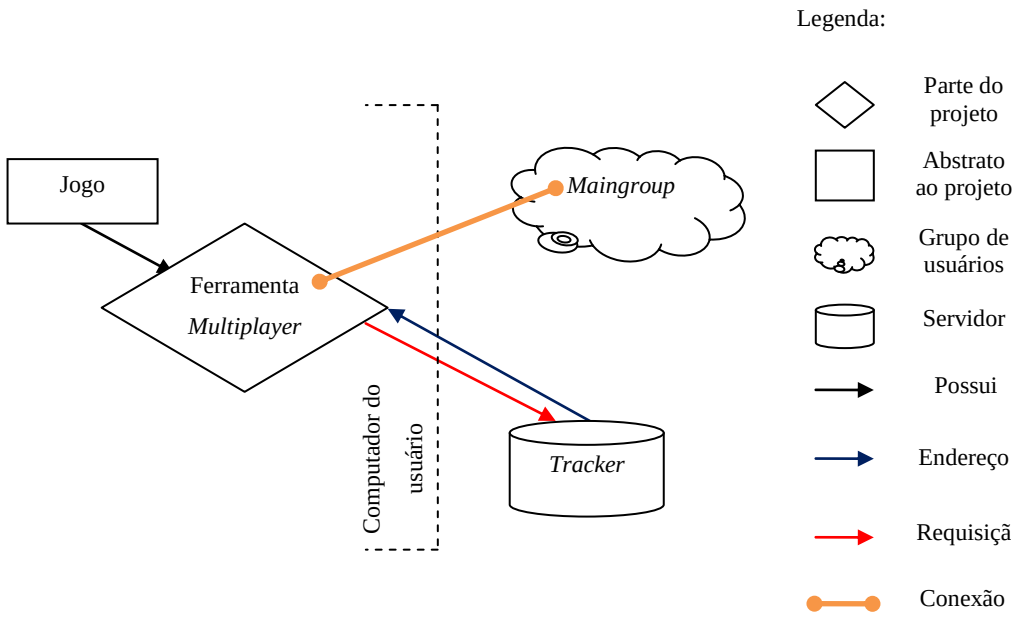

Figura 4.5: Conexão com o *Maingroup*

Junto aos membros do *Maingroup* o novo cliente deve fazer *post* de pedidos de *refresh*. O *refresh* é uma requisição para que os membros respondam com os devidos endereços das partidas que estes estão preparando, ou seja, o endereço dos *Roomgroups* das salas de pré-partidas. Feita então uma lista das salas já criadas o sistema disponibiliza-as ao programa que escolhe qual deve ser o *Roomgroup* que este irá se juntar.

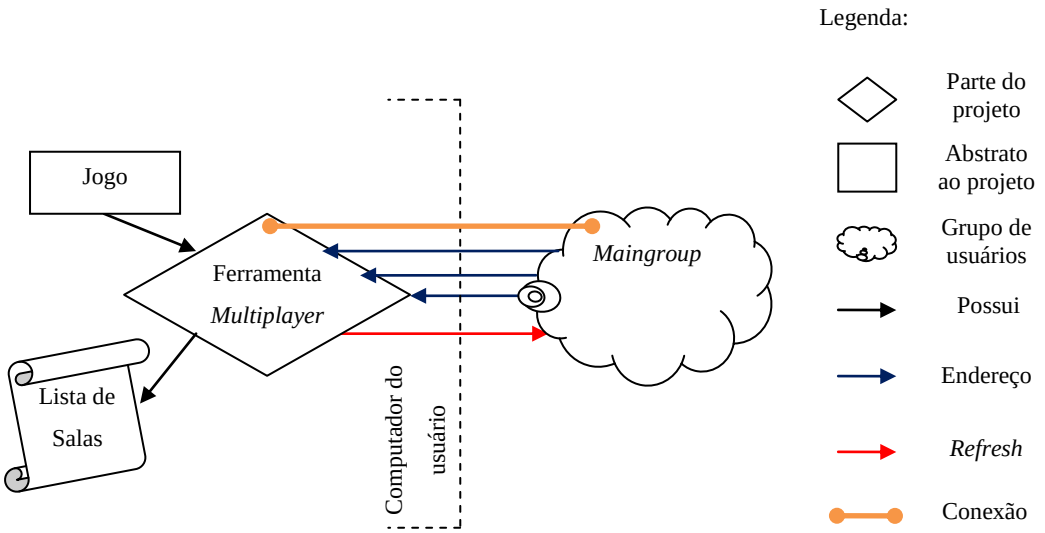

Figura 4.6: Comando *refresh*

Quando o sistema estiver abastecido com a lista de salas de partidas a ferramenta multiplayer irá aguardar a escolha do programa ou do usuário da partida a qual ele deseja participar de acordo com as definições do jogo. Entra-se em contato pelo *Maingroup* com o usuário que enviou o endereço escolhido e avisa-se que se deseja entrar no grupo. Conecta-se então ao *Roomgroup* especificado no endereço recebido anteriormente.

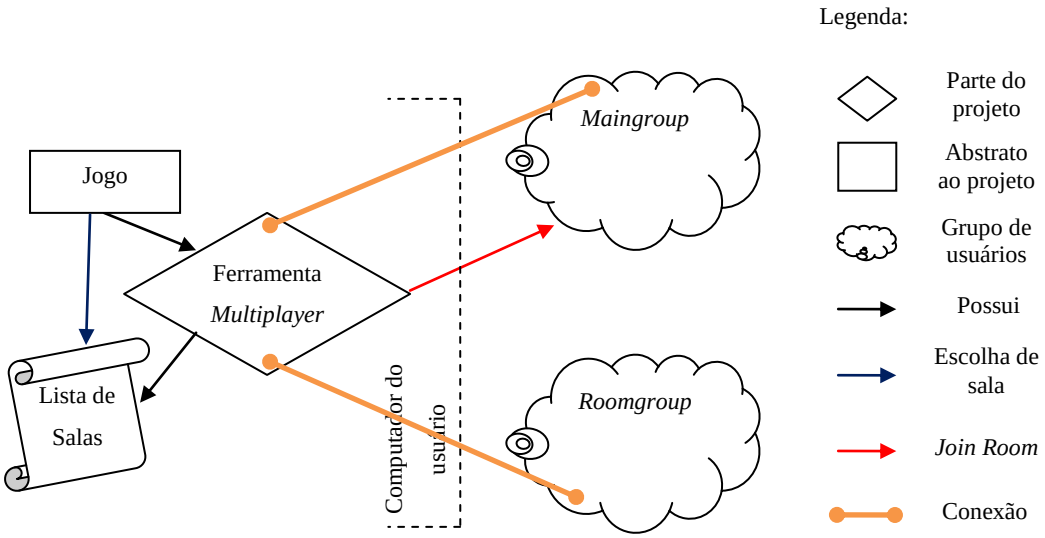

Figura 4.7: Entrada em uma *Roomgroup*

O sistema, além de disponibilizar através do comando *refresh* a lista das salas existentes, também disponibiliza o comando para que seja possível a criação de uma sala nova de pré-partida. A execução desse comando gera um endereço de grupo no qual vai se localizar aquela sala. Depois da criação do endereço esse usuário do *Maingroup* deve começar a responder todos os futuros *refresh* com o novo endereço da *Roomgroup*.

Numa sala de pré-partida, três estados são definidos por padrão para seus membros. O membro pode ser o hospedeiro, pode estar pronto e pode estar não pronto. Uma sala só vai existir enquanto ao menos um jogador estiver nela, e sempre que existir ao menos um jogador na sala o que estiver a mais tempo na sala será definido como hospedeiro. Assim sempre existirá um hospedeiro. Os membros que entrarem na sala serão considerados, por padrão, no estado não pronto. Para que modifiquem seus estados os não prontos devem enviar ao grupo um *post* avisando que estão prontos e assim modificarem seus estados. A única pessoa na sala capaz de iniciar uma partida é o hospedeiro, e esse só vai poder efetivar esse comando se todos os outros jogadores estiverem no estado pronto. Quando dado o comando de início da partida, todos os jogadores se desconectam do *Maingroup* e o *Roomgroup* é usado como meio de transmissão dos dados das partidas.

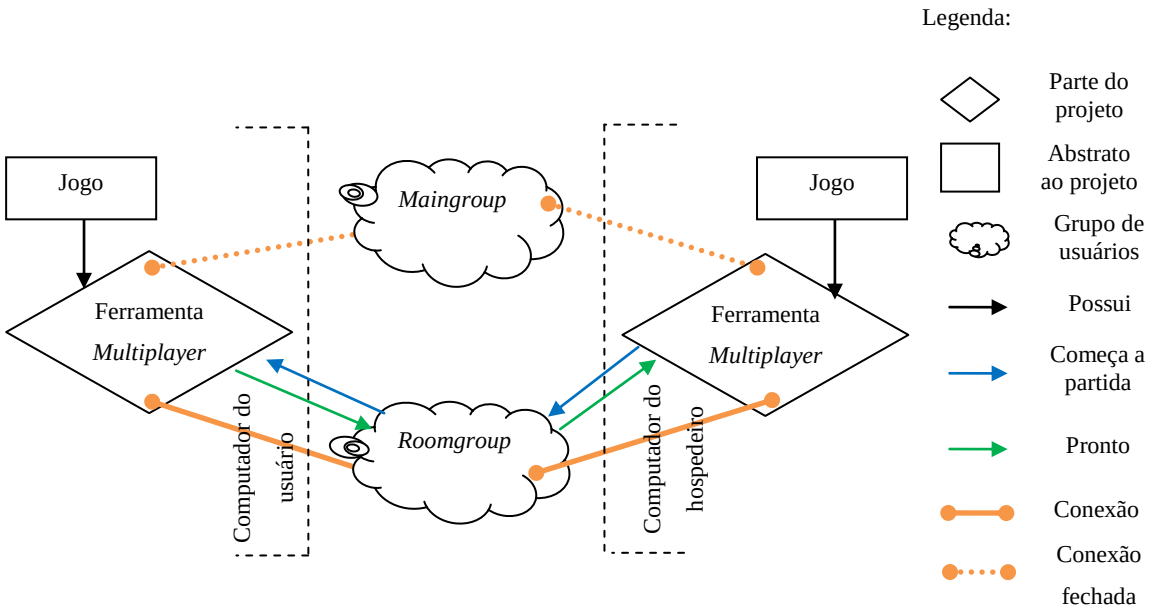

Figura 4.8: Início de uma partida

A ferramenta durante o decorrer da partida apresenta funcionalidades de chats para o caso de existir algum projeto que necessite dessa funcionalidade, e pode oferecer serviços como

medição de *lag* por *ping*. A medição de *ping* é necessária a criação de certos sistemas de compensação de *lag*.

5 FERRAMENTAS USADAS NO DESENVOLVIMENTO DO TRABALHO

Este capítulo descreve o funcionamento das ferramentas usadas no auxílio do desenvolvimento do trabalho aqui proposto. Dentre as ferramentas de auxílio encontram-se a linguagem escolhida, o ambiente de desenvolvimento, IDE para programação e bibliotecas usadas.

5.1 Adobe Flash e Flex SDK 4

Adobe Flash é um software de gráfico vetorial, apesar de suportar bitmap e vídeos, utilizado na criação de animações, jogos e em outros aplicativos que podem funcionar embutidos em um navegador web ou na plataforma Adobe AIR (Adobe AIR, 2010). Antes de ser vendido para Adobe, o Flash era desenvolvido pela Macromedia, empresa especializada na criação de programas para desenvolvimento web.

Os arquivos gerados pelo Adobe Flash estão no formato SWF (Shockwave Flash Files). Esses são visualizáveis através de qualquer navegador que suporte o Flash Player, que é um aplicativo distribuído gratuitamente pela Adobe. Certos sistemas operacionais contêm o Flash Player instalado em sua configuração inicial.

A partir da 5ª versão, a Macromedia expandiu a utilização do Flash com a introdução da capacidade de interpretar códigos Action Script, os quais ampliavam a capacidade do Flash Player em reproduzir aplicativos muito mais complexos que seus antecessores, que, na sua maioria, eram simples animações. A Adobe atualmente disponibiliza o Flash Player de versão 10.1, o qual é capaz de interpretar aplicativos criados no Adobe Flash com linguagem Action Script 3.0 e compiladas em seu mais novo compilador, o Flex SDK 4, que ainda está em fase experimental.

Como já citado na sessão 3.2, certas funções do trabalho desenvolvido estão apenas presentes no compilador Flex SDK 4. Aplicativos desenvolvidos com base nesta SDK devem apenas funcionar no Flash Player 10.1 ou no AIR 2.5, por isto existe a restrição de que o usuário que queira usufruir dos jogos que aplicarem a nova funcionalidade deve usar uma dessas tecnologias. Apesar de o Adobe Flash ser um software proprietário, o Flex SDK 4 tem sua distribuição gratuita (Adobe Products, 2010).

5.2 Action Script 3.0

Action Script é a linguagem de programação da plataforma Adobe Flash. Em sua versão mais atual 3.0 (AS3) ela se tornou uma linguagem orientada a objetos baseada em ECMAScript (ECMAScript, 2010). A linguagem é executada na AVM (Action Script Virtual Machine), a qual é encontrada tanto no Flash Player quanto no ambiente Adobe AIR. (Action Script Wikipedia Article, 2010)

Dentre as características da linguagem, cita-se:

- Sistema unificado de tratamento de eventos baseado no Document Object Model (Document Object Model (DOM), 2009);
- Acesso direto à lista de componentes do Adobe Flash em tempo de execução, permitindo controle completo em Action Script;
- Suporte a pacotes (*package*), como em Java, *namespaces*.
- Classes seladas: o comportamento padrão é que as classes não podem ser estendidas em tempo de execução, melhorando a alocação de memória e mantendo a aplicação mais segura. Classes dinâmicas podem ser utilizadas se definidas com o modificador *dynamic*.

5.3 IDE Flash Develop

Visando um melhor desenvolvimento e maior produção a empresa Decadium Studios testou alguns editores de código AS3 para medir o que melhor se encaixava no perfil de seus funcionários. A ferramenta escolhida foi o Flash Develop (Flash Develop, 2010).

Flash Develop é uma IDE para edição de código gratuita e de código aberto. Ela contem marcação de sintaxe em diferentes cores, sistema de auto-completar robusto, marcações de tarefas (TODO) e macros editáveis para facilitar a edição do código. Características comuns a editores, mas muito importantes para o desenvolvimento rápido e de qualidade.

5.4 Cirrus

Cirrus, que também já foi chamado Stratus, é a biblioteca do Adobe Flash em AS3 que permite que os aplicativos possam se comunicar através do Real Time Media Protocol (RTMFP) (Real Time Media Flow Protocol Wikipedia Article, 2010). Essa biblioteca se diferencia do protocolo anterior de comunicação dos aplicativos Flash na maneira em como ele se comunica. O protocolo anterior de comunicação Adobe Flash era restrito a comunicação cliente-servidor, onde os aplicativos eram tratados como clientes e deveria existir um servidor de endereço estático como o Smart Fox. O novo protocolo RTMFP permite que os aplicativos se comuniquem direto de cliente para cliente, como em sistemas P2P.

Essa tecnologia está funcional desde 2008 em Flash Players versão 10.0. Mas apenas na versão 10.1 e com a 2ª versão do Cirrus se permitiu a criação de grupos P2P que se comunicam com mensagens em *broadcast*. A Cirrus se torna uma melhor opção de comunicação considerando a capacidade limitada de banda de um servidor e seu preço. Em aplicações que não exigem sincronia de seus usuários ou não apresenta muitos usuários a Cirrus apresenta uma qualidade equivalente a de um servidor precisando apenas de um *tracker* para que haja as identificações dos grupos para novos membros.

6 IMPLEMENTAÇÃO DO PROJETO

Esse capítulo descreve a implementação do projeto conceitual descrito no capítulo 4 com as ferramentas escolhidas descritas no capítulo 5. A criação de tal ferramenta desencadeou certas dificuldades que também estão descritas nesse capítulo. Ao final são apresentados os resultados obtidos.

6.1 Projeto de implementação

Visando uma implementação consistente e eficiente houve uma fase de planejamento de como essa ferramenta deveria ser desenvolvida para os fins aqui citados. Além da concepção de um projeto conceitual de como ela deveria funcionar, foram definidos módulos funcionais os quais nos quais o programa estaria organizado. Também foi criada uma hierarquia de controle das classes que deveriam ser programadas em AS3. Uma hierarquia de controle é importante a um projeto de desenvolvimento visto que esta define como os objetos e classes devem se comportar para que esses desempenhem os seus papéis de maneira organizada e objetiva.

6.1.1 Módulos funcionais

Os módulos criados para o projeto foram quatro. O modulo contendo o código referente à organização da ferramenta foi batizado de `DecadiumMultiplayer`. O módulo referente à abstração da comunicação em mais baixo nível é chamado de `P2PConnection`. O módulo que deve interagir com o sistema nos quesitos relevantes a interface é referenciado como `MuliplayerInterface`. A figura 6.1 mostra como esses módulos se organizam entre si.

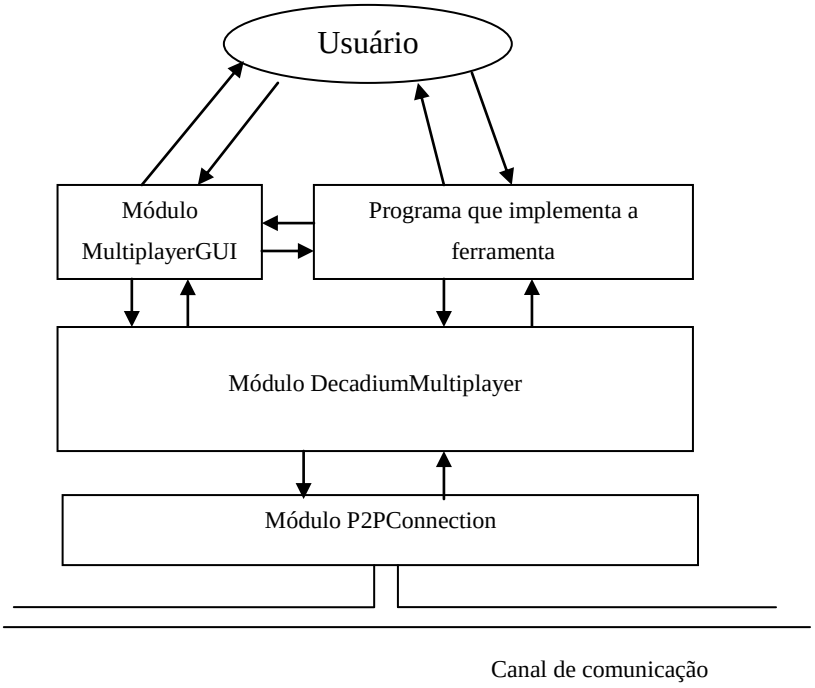

Figura 6.1 : Estrutura em módulos do jogo

6.1.2 Hierarquia de controle

A hierarquia aqui desenvolvida é encabeçada por um objeto que deve controlar a ordenação dos outros módulos nos passos executados pelo sistema *multiplayer*. Essa classe central foi balizada como DecadiumMultiplayer visto que ela seria uma implementação de projeto conceitual feito para a empresa parceira ao projeto. Esse objeto, visto que conterá grande parte da informação relevante a ferramenta e será de uma instância única, é um *singleton* (Singleton Artigo da Wikipédia, 2010).

O primeiro passo da classe DecadiumMultiplayer no seu processo é instanciar, caso esteja habilitada pra isso, uma LoginScene, que é uma classe que deve fazer sua interface com o sistema de login e retornar a validação do mesmo ao *singleton* central. Junto a validação a `LoginScene` deve retornar as informações relevantes ao login para serem armazenadas junto a `DecadiumMultiplayer`, para que assim todas as outras classes tenham acesso a essa. Essas informações devem estar expressas numa instância da classe `LoginInfo`.

A classe `DecadiumMultiplayer` contem instâncias públicas da classe `MainRoom`, que é a sala onde o jogador deve escolher sua sala de partida, e da classe `Room`,

a sala onde o jogador vai poder visualizar os outros jogadores e entrar em contato com esses. A `Room` por padrão já implementa um `Chat` para essa comunicação entre os jogadores.

Tanto a classe `Room` quanto a `MainRoom` para se comunicarem com os seus devidos grupos de usuários contêm uma instancia da classe `P2PConnection` que efetiva a conexão com o tracker e depois a conexão com o grupo. A hierarquia de controle descrita até então se apresenta no diagrama da figura 6.1.

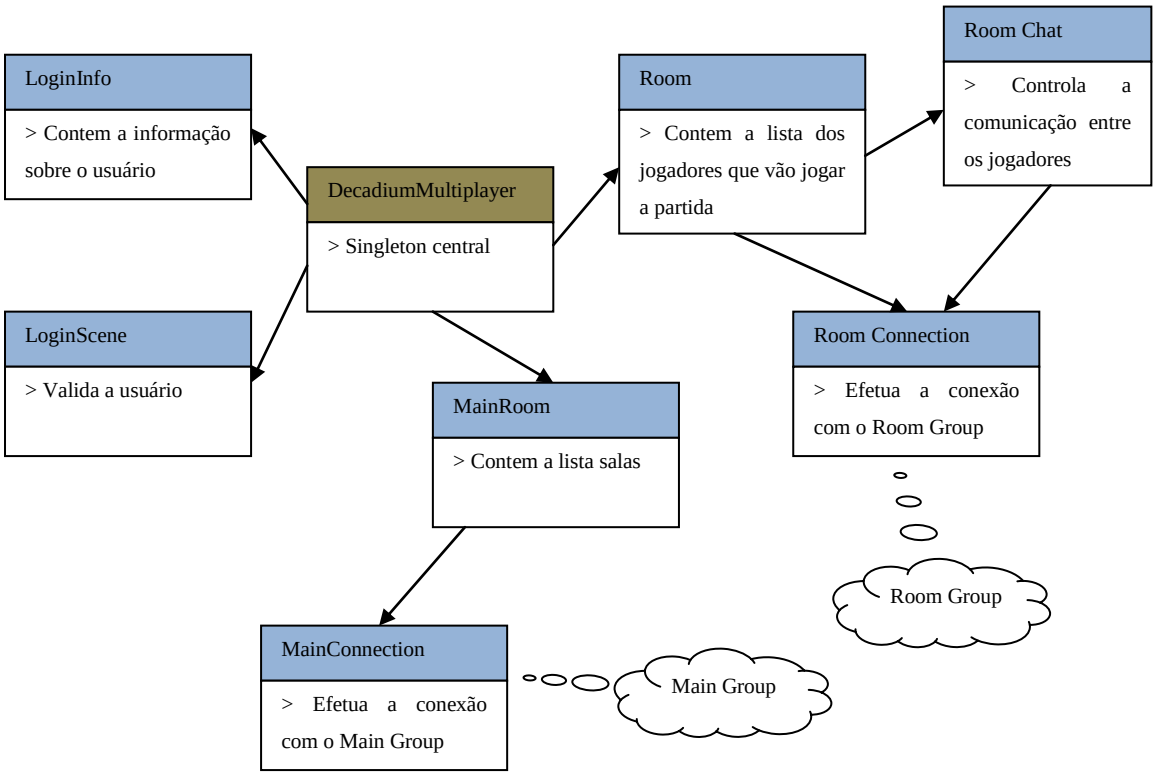

Figura 6.2: Hierarquia de controle da ferramenta

6.1.3 Classe para conexão P2P

Com o objetivo de abstrair a conexão com o *tracker* e a comunicação com o grupo, desenvolveu-se uma classe em AS3 que disponibiliza todas as funções e atributos referentes a essa abstração. Essa classe, rotulada `P2PConnection`, responde a uma série de eventos que podem ocorrer com as conexões, tanto a com o tracker quanto a conexão entre o grupo P2P.

Entre os métodos mais importantes implementados pela classe estão:

- `connect()`: método que efetiva a conexão com o *tracker-server*, cuja chave de acesso ao servidor Cirrus é uma chave única requisitada à Adobe pela

Decadium e o endereço é uma constante estática da classe `P2PConnection` (isto visto que o Cirrus deve ser o *tracker* do sistema até o desenvolvimento de um *tracker* próprio da Decadium, algo previsto em "trabalhos futuros"). Após se efetivar a conexão, o agora cliente requisita ao *tracker* a referência ao grupo com o nome que lhe foi passada na construção do objeto `P2PConnection`. Caso o grupo já tenha sido construído pelo Cirrus ele retorna uma conexão com um dos membros da nuvem de usuários. Caso o grupo não exista ainda para aquela chave de acesso do Cirrus esse cria e retorna o grupo vazio ao usuário, o qual será seu primeiro membro.

- `post(theMsg:String)`: esse método envia uma mensagem a todos os membros do grupo (*broadcast*) quando o `connect()` já tiver sido efetivado com sucesso. Caso não esteja conectado ainda o objeto gera uma exceção.
- `send(theMsg:String,theReciver:GroupMember)`: envia uma mensagem a um membro específico do grupo P2P. Para especificar o destinatário, se já não for um membro conhecido, basta acessar a lista de membros do objeto `P2PConnection memberList:Array`. Por padrão essa lista não é ordenada igualmente a todos os membros do grupo, mas para mudar essa propriedade visando uma unanimidade nessa ordenação basta ajustar para verdadeira a propriedade booleana `arrangeMembersList:Boolean`.

Para ser possível escutar os eventos que a classe `P2PConnection` invoca, pode-se adicionar os devidos ouvidores (*listeners*) ao objeto instanciado com o método `addEventListener(type:String,listener:Function)` na qual o parâmetro `type:String` pode ser escolhido dentre as constantes estáticas que a classe `P2PconnectionEvent` carrega consigo.

6.2 Prototipação

Após planejada a implementação criou-se dois diferentes protótipos para se testar a ferramenta distribuída. O primeiro protótipo foi referente a classe `P2PConnection`, um sistema simples que faz a conexão com os outros jogadores de modo semelhante ao sistema desenvolvido para Sim. O outro protótipo exemplifica todo o sistema de pré-partida do sistema completo Decadium-Multiplayer, e tem toda base para a implementação de um dos

jogos desenvolvidos em flash. Visto o âmbito do trabalho e tempo hábil não foi desenvolvido um protótipo para exemplificar algum dos jogos desenvolvidos com a Simskeleton.

6.2.1 Sistema *Fake-Sim*

O protótipo desenvolvido para testar a classe de conexão P2PConnection foi batizado de *Fake-Sim*, visto que ele fazia um protocolo de pré-partida diferente do usual e muito parecido com o desenvolvido para a empresa Sim. O funcionamento desse protocolo está descrito no capítulo 3.1.1.

O sistema da Fake-Sim se é uma implementação de testes que não utiliza todo sistema da ferramenta desenvolvida, somente utiliza a classe `P2PConnection` para abstrair uma conexão única entre as execuções desse protótipo. O protótipo inicia a conexão e pede ao tracker para juntar-se a uma sala com id constante para todas as instâncias, ou seja, esse protótipo se assemelha ao sistema completo, mas com apenas uma sala. Após isso se contabiliza o número de usuários conectados ao grupo. Caso o valor seja igual ao da constante que define o numero de jogadores necessário para se iniciar uma partida, o sistema inicia a partida. A saída desse sistema se limita a um simples *log*. O *log* com os passos de conexão pode ser visualizado na figura 6.2. No exemplo o programa espera até localizar quatro outras execuções para dar início ao jogo, assim, são mostradas os ID dos outros jogadores no grupo e se escreve a mensagem de “começou o jogo” no *log* ao momento que o evento `GameStart` é disparado.

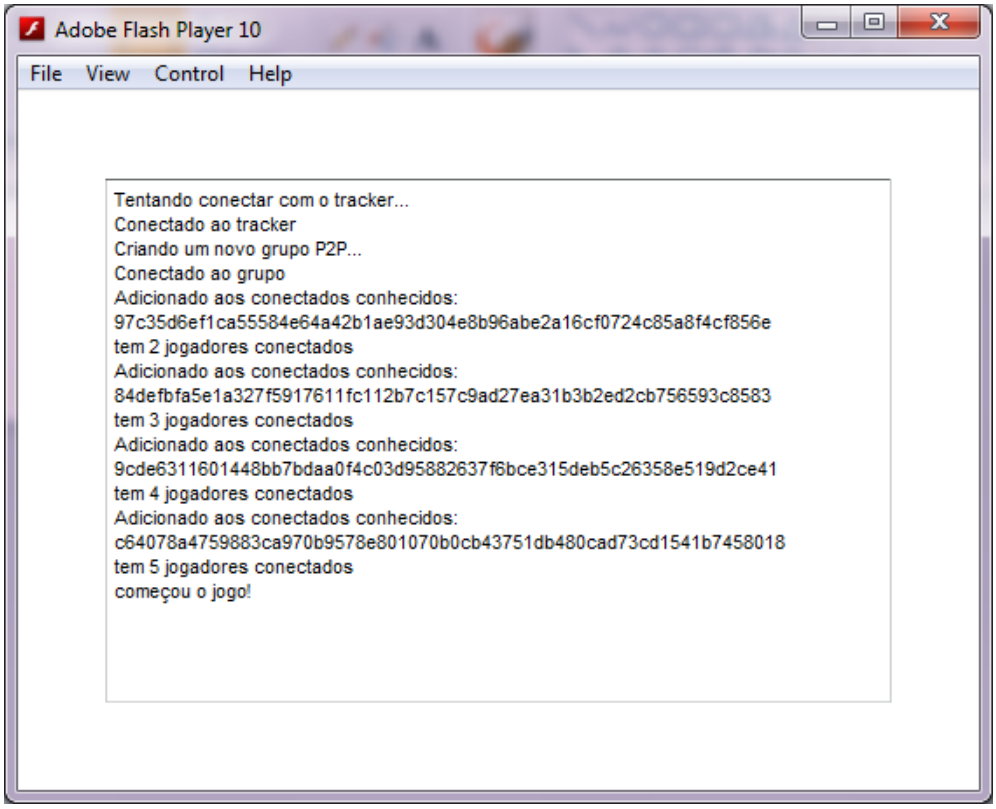

Figura 6.3: Imagem do protótipo fake Sim em execução.

6.2.2 Exemplo da *Decadium-Multiplayer*

O protótipo do exemplo do sistema completo da Decadium-Multiplayer nada mais é do que uma série de interfaces gráficas que acessam as propriedades do sistema e responde aos seus eventos. Esse protótipo não responde a inicialização do jogo visto que esse não foi desenvolvido para nenhum dos projetos de jogo, mas essa interface pode ser implementada por qualquer outro programa em AS3 e Flash que assim poderá responder corretamente a tal evento.

Da figura 6.4 à 6.7 são mostradas as principais cenas da interface de exemplo implementada. A figura 6.4 mostra um exemplo de interface para o LoginScene, onde o usuário pode inserir seus dados para serem conferidos com o servidor de Login, caso exista. A figura 6.4 contém um exemplo de interface de criação de sala nova. A opção *private* refere-se à possibilidade de ser feita uma sala de jogo privada a qual só poderão entrar aqueles que souberem uma senha definida pelo criador do jogo. Após o jogador criar uma sala seu sistema multiplayer começará a responder comandos *refresh*. A figura 6.5 refere-se à interface para a `ChooseRoom`, classe que armazenará a lista de jogos disponíveis ao jogador. O botão *refresh*

dessa interface manda ao grupo um novo comando *refresh* e as resposta são armazenadas visualmente na lista com título *Open Games*. A figura 6.6 mostra um exemplo de uma sala de jogo com três jogadores onde deles um é o host, outro está pronto para o jogo e outro ainda não está pronto. Como dito posteriormente a classe `Room` por padrão já implementa um chat para conversa entre os jogadores, então junto a interface da sala há também uma interface para o chat.

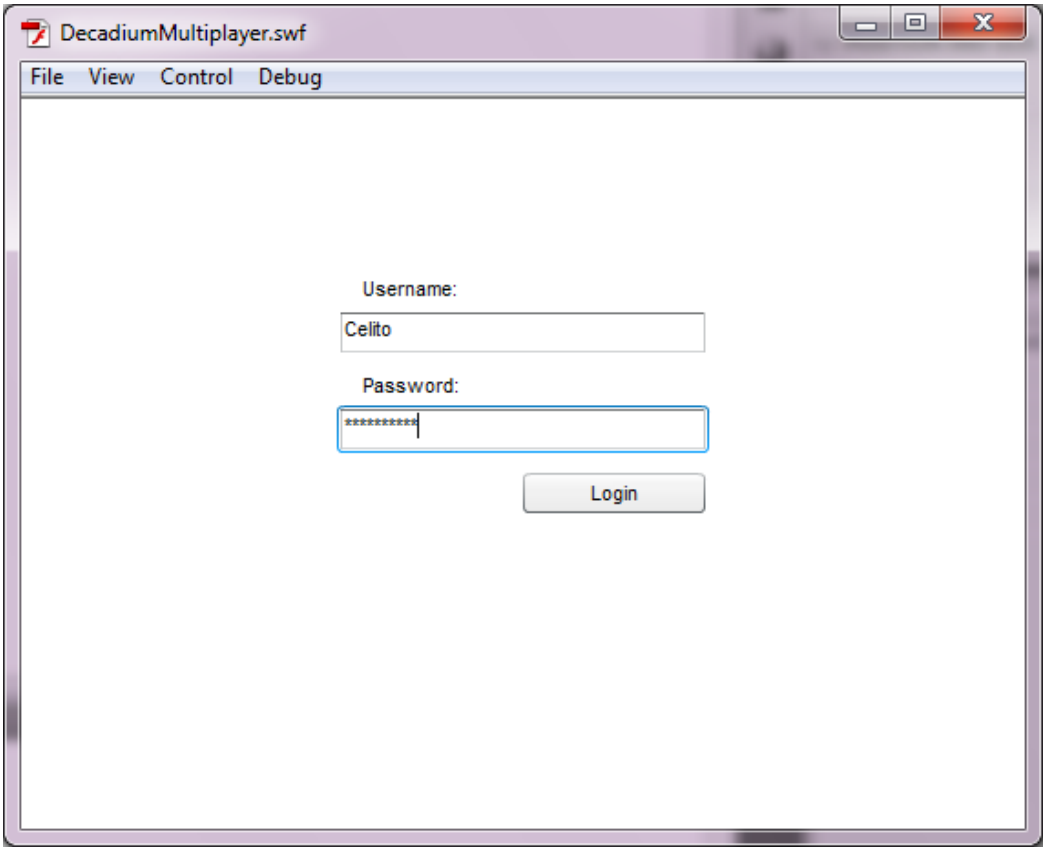

Figura 6.4: Exemplo de interface para a classe LoginScene

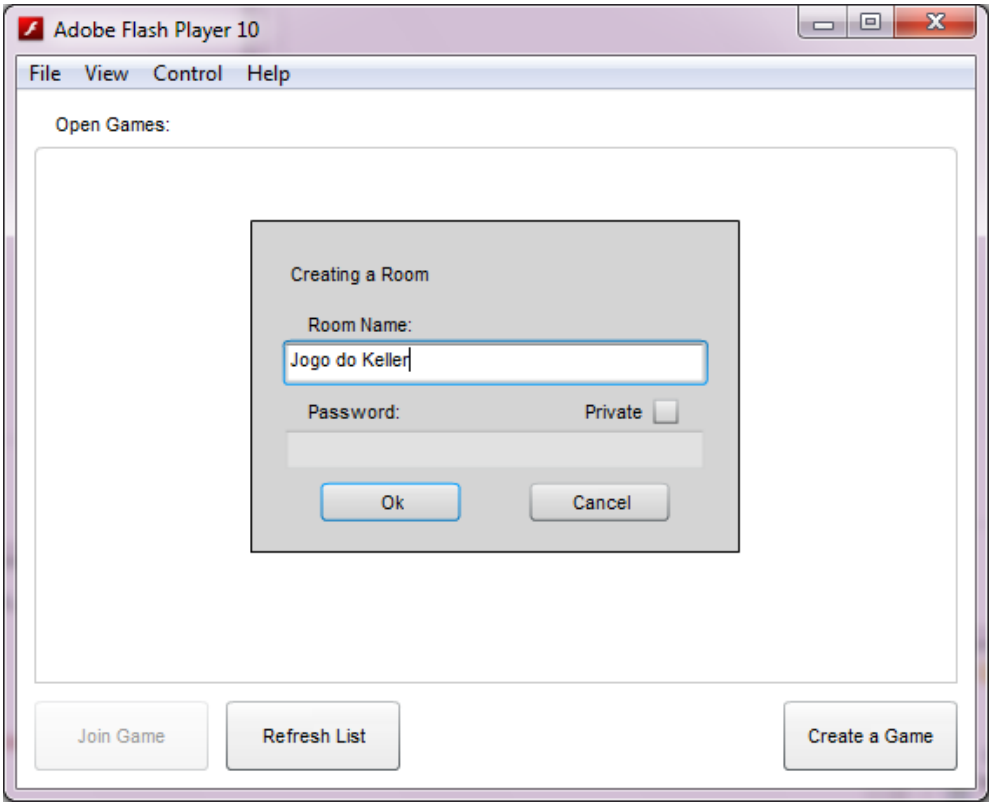

Figura 6.5: Exemplo de interface para a criação de salas de pré-partida

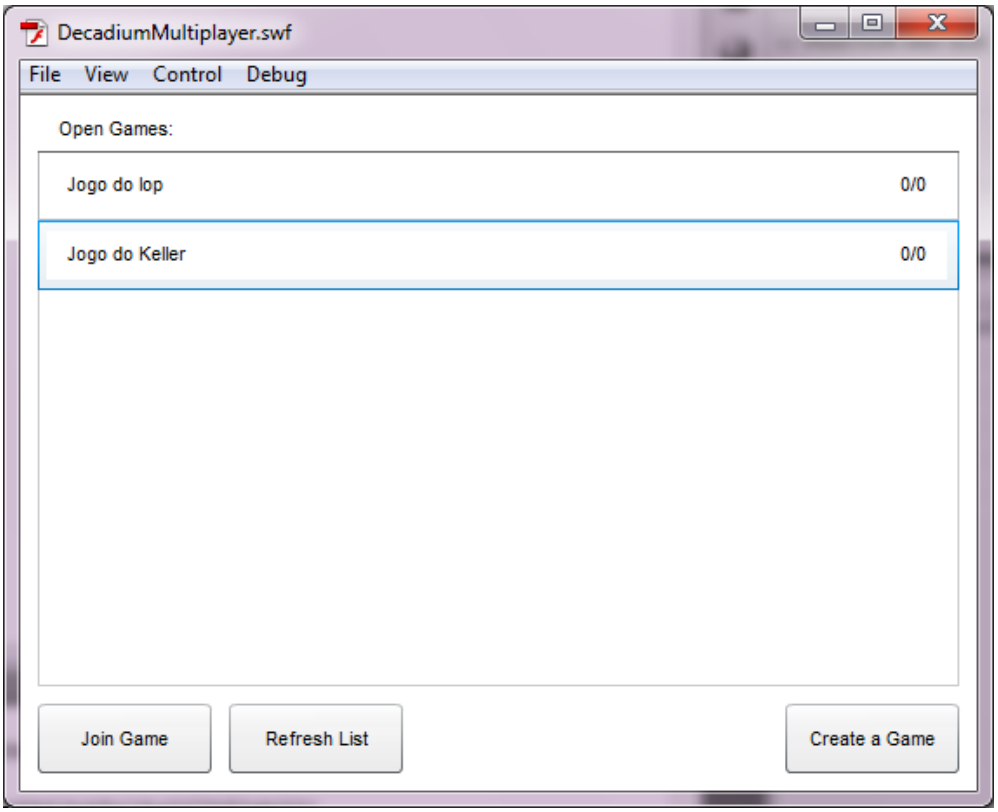

Figura 6.6: Exemplo de interface para a classe ChooseRoom

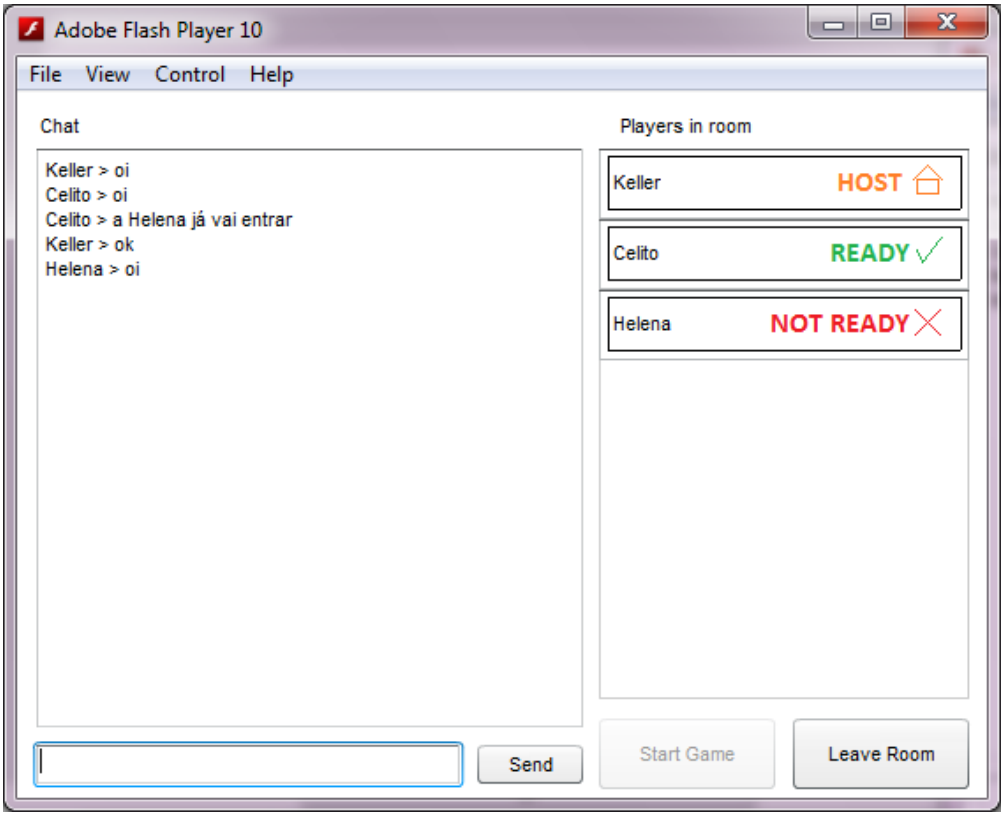

Figura 6.7: Exemplo de interface para a classe Room

6.3 Resultados

Para conclusões mais claras da eficiência da criação desse trabalho, os protótipos foram testados em diferentes ambientes. Os testes ocorreram em diferentes ambientes: em instâncias de mesma máquina e em computadores em uma rede local. Não houve tempo hábil para testar em computadores conectados a internet. A tabela 7.1 compara os diferentes tempos de *ping* entre aplicações tanto no sistema antigo quanto no Decadium-Multiplayer. O *ping* medido aqui considera uma média aproximada de *pings* durante 30 segundos de execução onde o *ping* é o maior tempo de enviar uma mensagem e receber uma resposta de um jogador a outro qualquer.

Tabela 6.1 : Resultados obtidos na média do *ping* do antigo e do novo sistema

Espécie de experimento	2 jogadores	4 jogadores	8 jogadores
Sistema da Simskeleton para comunicação através do SmartFox	276 ms	305 ms	317 ms
Sistema da Decadium-Multiplayer em um único computador	150 ms	158 ms	243 ms
Sistema da Decadium-Multiplayer em uma rede local	241 ms	406 ms	633 ms

6.4 Dificuldades encontradas

Certas dificuldades ocorreram no processo de criação da ferramenta *middleware* aqui desenvolvida. Dentre essas dificuldades serão descritas nesse capítulo as mais relevantes.

6.4.1 Sistema em camadas mal definidas

No decorrer do trabalho criou-se a interface gráfica paralelamente ao desenvolvimentoda estrutura do sistema, de modo que o programa resultante ficou completamente dependente da interface. Isso gerou uma complexidade desnecessária a ferramenta e uma intercambialidade da mesma entre diferentes interfaces. A complexidade desnecessária é gerada visto que o código de comportamento da interface estava misturado ao de execução das funções da ferramenta. Do mesmo modo a intercambialidade de interface é algo que fazia o sistema ser difícil de ser adicionado a diferentes projetos. Visando a correção nessa falha replanejou-se a organização geral do sistema para que esse fosse modular e os módulos funcionassem independentes uns dos outros como explicado na seção 6.1.1.

6.4.2 Bloqueio de pacotes UDP

Depois de desenvolvidos os protótipos, para fins de testes esses deveriam ser rodados em diferentes máquinas. Mas na maioria das redes testadas os protótipos não conseguiam se comunicar visto que suas portas deveriam estar abertas para receber pacotes UDP. Essa incapacidade de comunicação obriga que cada rede seja configurada para que a porta

especificada para comunicação entre os membros esteja habilitada. Apesar dessa dificuldade técnica o protótipo teve o comportamento esperado em redes depois dessa reconfiguração.

7 CONCLUSÃO

O trabalho então desenvolvido concluí-se e seus resultados ainda não tiveram utilização prática para seu objetivo primário: a função de middleware em um projeto de jogo. Os protótipos, a arquitetura e a ferramenta gerada por esse trabalho são o seu resultado de programação. Esse capítulo faz uma análise dos resultados obtidos pelo trabalho aqui desenvolvido e trata dos possíveis trabalhos futuros a serem feitos para complementar esse.

O sistema aqui planejado e construído foi concluído e foi testado com a elaboração dos descritos protótipos. Esses desempenham o papel que os projetos de jogos devem desempenhar interagindo com a ferramenta. Apesar de problemas técnicos com a configuração de determinadas redes e interface gráfica, os resultados podem ser visualizados nos testes descritos no capítulo 6.3.

Os testes efetivados comprovaram a funcionalidade da ferramenta. Seu desempenho foi menor do que o da comunicação da Simskeleton em certas situações, principalmente com o número de jogadores mais elevado. Apesar disso a ferramenta conta com a vantagem da não utilização de servidor para troca de mensagens como planejado. Mesmo assim ela ainda possui a falha de ser dependente do Cirrus, ou seja, a ferramenta não pode ser utilizada *offline* por ter de se conectar ao seu tracker visto que o sistema P2P do Flash ainda não suporta outros *trackers*.

7.1 Trabalhos futuros

O trabalho desenvolvido obteve resultados funcionais, mas ainda existem melhorias e expansões a serem feitas. Algumas melhorias não foram feitas por falta de tempo hábil para isso. Algumas funcionalidades não foram incluídas visto que as ferramentas Cirrus e P2P para Flash ainda estão em construção e para total integração desse trabalho com essas ferramentas é necessário que elas tenham suas versões finais lançadas.

Dentre os possíveis trabalhos futuros referentes a esse projeto de middleware planeja-se primeiramente a integração desse com a Simskeleton para que essa, corretamente configurada, possa compilar jogos com a capacidade de se comunicarem através da nova ferramenta. A Simskeleton com uma ferramenta para criação de jogos P2P poderá então produzir jogos independentes de um servidor SmartFox. Após a integração será possível a inclusão do

sistema em um jogo multiplayer para resultados mais conclusivos. Além da integração com a biblioteca, propõe-se também a criação de um arquivo em formato SWC, arquivo padrão de biblioteca AS3, para que assim o sistema possa ser usado em qualquer outro programa desenvolvido na mesma linguagem.

Devem ser planejados e executados testes pela internet para resultados mais conclusivos referentes ao consumo de banda de rede do sistema. Com esses resultados será possível se redefinir quais as possíveis utilizações para a ferramenta.

A Decadium-Multiplayer também deve ter sua programação ampliada para que essa não necessite de servidor *tracker*. Essa ampliação de implementação só será possível quando a ferramenta Flash adicionar seu sistema de conexão direta com um grupo (KRCHA, 2010). Junto a tal ampliação deve surgir o projeto de um servidor tracker substituto ao Cirrus, para que esse identifique unicamente os grupos instanciados pela Decadium-Multiplayer.

REFERENCIAS

ACTION Script Wikipedia Article. **Wikipedia:** The Free Encyclopedia, 2010. Disponível em: <http://en.wikipedia.org/wiki/ActionScript>. Acesso em: 23 Outubro 2010.

ADOBE AIR. **Adobe Official Website**, 2010. Disponível em: <http://www.adobe.com/products/air/>. Acesso em: 12 Dezembro 2010.

ADOBE Flash Wikipedia Article. **Wikipédia:** The Free Encyclopedia, 2010. Disponível em: <http://en.wikipedia.org/wiki/Adobe_Flash>. Acesso em: 22 outubro 2010.

ADOBE Products. **Adobe Official Website**, 2010. Disponível em: <http://tryit.adobe.com/br/products/flash/?sdid=FGUBM&>. Acesso em: 20 Novemrbro 2010.

BLIZZARD Entertainment. **Blizzard Official Website**, 2010. Disponível em: <http://us.blizzard.com/pt-br/?->. Acesso em: 20 Novembro 2010.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas Distribuídos:** Conceitos e Projeto. Tradução de João Tortello. 4ª. ed. Porto Alegre: Bookman, 2007.

DAVISON, A. **Killer Game Programming in Java**. Sebastopol: O'Reilly, 2005.

DOCUMENT Object Model (DOM). **Site da W3C**, 2009. Disponível em: <http://www.w3.org/DOM/>. Acesso em: 23 outubro 2010.

ECMASCRIPT. **ECMAScript Official Webssite**, 2010. Disponível em: <http://www.ecmascript.org/>. Acesso em: 18 Novembro 2010.

FLASH Develop. **Flash Develop Wiki**, 2010. Disponível em: <http://www.flashdevelop.org/wikidocs/index.php?title=Main_Page>. Acesso em: 20 Novembro 2010.

GAME Engine Wikipedia Article. **Wikipedia:** the Free Encyclopedia, 2010. Disponível em: <http://en.wikipedia.org/wiki/Game_engine>. Acesso em: 16 Novembro 2010.

GAME Spy. **Game Spy Official Website**, 2010. Disponível em: <http://www.gamespy.com/>. Acesso em: 20 Novembro 2010.

GARENA. **Garena Official Website**, 2010. Disponivel em: <http://www.garena.com/>. Acesso em: 20 Novembro 2010.

GOTOANDPLAY. **gotoAndPlay Official Website**, 2008. Disponível em: <http://www.gotoandplay.it/>. Acesso em: 18 Novembro 2010.

HARRIS, T. Como Funciona a Formatação de Videos. **HowStuffWorks**, 2000. Disponível em: <http://lazer.hsw.uol.com.br/formatacao-de-videos.htm>. Acesso em: 18 Novembro 2010.

HISTORY Of Multiplayer Games. **SSAGSg:** Society of Simulation and Gaming of Singapore, 2007. Disponível em: <http://www.ssagsg.org/LearningSpace/EntertainmentGaming/HistoryMPG.htm>. Acesso em: 25 outubro 2010.

JOSEPH Plateau Wikipedia Article. **Wikipedia:** the Free Encyclopedia, 2010. Disponível em: <http://en.wikipedia.org/wiki/Joseph_Plateau>. Acesso em: 19 Novembro 2010.

KRCHA, T. Local Flash peer-to-peer comunication over lan without cirrus/stratus. **Flash Realtime.com**, 2010. Disponível em: <http://www.flashrealtime.com/local-flash-peer-to-peer-communication-over-lan-without-cirrus/>. Acesso em: 23 Novembro 2010.

MARSHALL, P. S. Internet Usage Statistics. **Internet World Stats:** Usage and Population Statics, 2010. Disponível em: <http://www.internetworldstats.com/stats.htm>. Acesso em: 20 Outubro 2010.

MATCH Making Article. **Team Liquid:** Starcraft Programing News, 2010. Disponível em: <http://www.teamliquid.net/forum/viewmessage.php?topic_id=118212>. Acesso em: 20 Novembro 2010.

MMORPG Artigo da Wikipédia. **Wikipédia:** A Enciclopédia Livre., 2010. Disponível em: <http://pt.wikipedia.org/wiki/MMORPG>. Acesso em: 20 Novembro 2010.

OPENNAP. **OpenNap:** Open Source Napster Server, 2001. Disponível em: <http://opennap.sourceforge.net/>. Acesso em: 20 Outubro 2010.

PING Artigo da Wikipédia. **Wikipédia:** a Enciclopédia Livre, 2010. Disponível em: <http://pt.wikipedia.org/wiki/Ping>. Acesso em: 12 Dezembro 2010.

POSTEL, J. RFC 768. **Internet Engineering Task Force**, 1980. Disponível em: <http://tools.ietf.org/html/rfc768>. Acesso em: 26 Novembro 2010.

REAL Time Media Flow Protocol - RTMFP. **Ask Me Flash.com beta**, 2009. Disponível em: <http://askmeflash.com/article/3/real-time-media-flow-protocol-rtmfp>. Acesso em: 26 Novembro 2010.

REAL Time Media Flow Protocol Wikipedia Article. **Wikepedia:** The Free Encyclopedia, 2010. Disponível em: <http://en.wikipedia.org/wiki/Real_Time_Media_Flow_Protocol>. Acesso em: 20 Novembro 2010.

SEVEN Games. **SevenOne Games Website**, 2010. Disponível em: <http://www.sevengames.de/>. Acesso em: 16 Novemebro 2010.

SEVENONE. **SevenOne Intermedia Website**, 2010. Disponível em: <http://www.sevenoneintermedia.de/>. Acesso em: 16 Novembro 2010.

SHIRKY, C. OpenP2P. **What's P2P and what's not**, 2000. Disponível em: <http://openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html>. Acesso em: 20 Outubro 2010.

SINGLETON Artigo da Wikipédia. **Wikipédia:** a Enciclopédia Livre, 2010. Disponível em: <http://pt.wikipedia.org/wiki/Singleton>. Acesso em: 12 Dezembro 2010.

SMARTFOX. **Smart Fox Server:** Massive Multiplayer Server, 2010. Disponível em: <http://www.smartfoxserver.com/>. Acesso em: 22 Outubro 2010.

STARCRAFT 2. **Starcraft 2 Official Website**, 2010. Disponível em: <http://us.battle.net/sc2/pt/>. Acesso em: 20 Novembro 2010.

STEAM. **Steam Official Webstore**, 2010. Disponível em: <http://store.steampowered.com/>. Acesso em: 20 Novembro 2010.

TANENBAUM, A. S.; STEEN, M. V. **Distributed System:** Principles and Paradigms. Upper Saddle River: Prentice Hall, 2002.

TCP/IP Artigo da Wikipédia. **Wikipédia:** A Enciclopédia Livre, 2010. Disponivel em: <http://pt.wikipedia.org/wiki/TCP/IP>. Acesso em: 22 Novembro 2010.

TOTAL War. **Total War Official Website**, 2010. Disponivel em: <http://www.totalwar.com/>. Acesso em: 16 Novembro 2010.

USER Datagram Protocol Artigo da Wikipedia. **Wikipédia:** A Enciclopédia Livre, 2010. Disponivel em: <http://pt.wikipedia.org/wiki/Protocolo_UDP>. Acesso em: 26 Novembro 2010.