Universidade Federal de Pernambuco
Graduação em Ciência da Computação
Centro de Informática

Trabalho de Graduação

JOGOS UBÍQUOS COM BLUETOOTH

Daniel Arraes Pereira

Orientador: Carlos André Guimarães Ferraz
Co-orientador: Fernando Antônio Mota Trinta

Recife
Fevereiro de 2006

0 dia mais belo? Hoje
A coisa mais fácil? Equivocar-se
O obstáculo maior? 0 medo
0 erro maior? Abandonar-se
A raiz de todos os males? 0 egoísmo
A distração mais bela? 0 trabalho
A pior derrota? 0 desalento
Os melhores professores? As crianças
A primeira necessidade? Comunicar-se
0 que mais faz feliz? Ser útil aos demais
0 mistério maior? A morte
0 pior defeito? 0 mau humor
A coisa mais perigosa? A mentira
0 sentimento pior? 0 rancor
0 presente mais belo? 0 perdão
0 mais imprescindível? 0 lar
A estrada mais rápida? 0 caminho correto
A sensação mais grata? A paz interior
0 resguardo mais eficaz? 0 sorriso
0 melhor remédio? 0 otimismo
A maior satisfação? 0 dever cumprido
A força mais potente do mundo? A fé
As pessoas mais necessárias? Os pais
A coisa mais bela de todas? 0 amor

—MADRE TEREZA DE CALCUTÁ (Poema da Paz)

A Glauber, meu irmão.

AGRADECIMENTOS

"Make it a habit to tell people thank you. To express your appreciation, sincerely and without the expectation of anything in return. Truly appreciate those around you, and you'll soon find many others around you. Truly appreciate life, and you'll find that you have more of it".

—RALPH MARSTON

A meus pais, simplesmente por serem quem são.

A meus avós, por estarem presentes em todos os momentos da minha vida.

À Marina, minha namorada, pelo apoio, pela dedicação...

A meus amigos de todas as horas, em especial, Gerlando, Silvio, Rômulo e Fábio, por serem meus irmãos de consideração.

A meus orientadores, Carlos e Fernando, pelos importantes ensinamentos.

A meus colegas de trabalho, pela ajuda e companherismo.

Ao Dr. Eraldo Ramos, por realizar uma cirurgia perfeita em meu braço.

Ao Dr. Alexandre Arraes, tio, por um excelente e rígido acompanhamento da minha recuperação.

E a todos que contribuíram, direta ou indiretamente, para a conclusão deste trabalho, meus sinceros agradecimentos. Seria impossível sem vocês.

RESUMO

Jogos Massivamente Multiusuário são aplicações cada vez mais populares no mundo do entretenimento digital. Sua principal característica é o regime de funcionamento contínuo (24x7), onde eventos podem ocorrer sem que o usuário esteja conectado. Neste sentido, dispositivos móveis, especialmente telefones celulares, aparecem como uma interessante solução para permitir acesso a um jogo massivo a qualquer hora e de qualquer lugar, criando o chamado MMUG - *Massively Multiplayer Ubiquitous Game*. Neste contexto, este trabalho propõe uma arquitetura que suporte o cenário de pequenos jogos para dispositivos móveis, jogados via *Bluetooth* e cujos resultados são integrados a um jogo Massivo, o MMUG.

Palavras-chave: MMOG, Bluetooth, Ubíquo, Jogos Massivos.

ABSTRACT

Massively Multiplayer Games are applications that are becoming more and more popular in the digital entertainment world. Its main feature is the full time working period (24x7), where events could happen to a player even if he is off-line. Therefore, mobile devices, especially cell phones, appear as an interesting solution to allow access to a massive game anytime and anywhere, creating what is called MMUG - Massively Multiplayer Ubiquitous Game. In this context, this project proposes an architecture that supports the scenario of small multiplayer games played through Bluetooth and whose results are integrated with a massive game, the MMUG.

Keywords: MMOG, Bluetooth, Ubiquitous, Massive Games.

SUMÁRIO

LISTA DE FIGURAS

CAPÍTULO 1

INTRODUÇÃO

Imagination is more important than knowledge.

—ISAAC NEWTON

O mercado de jogos para dispositivos móveis tem se tornado bastante promissor nesses últimos anos, principalmente para telefones celulares. Um dos fatores que mais contribuiu para isto foi o surgimento de plataformas como J2ME[Sun05b, Muc04] e BREW[QUA05], que, associadas ao aumento do poder computacional desses dispositivos, permitiu o desenvolvimento de jogos monousuário mais complexos. Jogos multiusuário móveis são uma evolução deste estágio inicial, onde a maturidade de tecnologias de comunicação sem fio como *Bluetooth*[Blu04] e Wi-Fi[IEE05] facilita o desenvolvimento de tais aplicações. Esta tendência é consolidada pelo lançamento de produtos voltados para estes jogos, como o Nokia NGage[Nok06a].

Um cenário secundário é o de jogos *on-line* massivamente multiusuário - MMOGs, onde milhares de jogadores interagem concorrentemente em um mundo virtual persistente. Estes jogos ocorrem continuamente e sem interrupções de forma que parte dos jogadores podem se ausentar enquanto outros permanecem jogando. Alguns dos mais populares MMOGs como *Ultima Online*[1] e *Lineage*[2] possuem mais de 500 mil assinantes, com registros de até 180 mil jogadores simultâneos em uma noite [Tri06].

A união entre MMOGs e jogos móveis cria um novo cenário onde usuários interagem, em jogos massivos, através celulares, criando os chamados *Massively Mobile Multiplayer Online Games* - MMMOG[FTF05]. Atualmente, a grande maioria dos jogos disponíveis para celulares no mercado mundial ainda são aplicações monousuário e *off-line*, porém ao analisar os mercados mais avançados, como Ásia e Europa, nota-se que há um consenso em torno do potencial comercial de se produzir jogos MMMOGs[Nok06b]. Neste sentido, as primeiras experiências foram realizadas, através de jogos como TibiaME[3] e Undercover[4], recentemente lançados na Europa, e Samurai Romanesque[5],

[1] http://www.uo.com
[2] http://www.lineage2.com
[3] http://www.tibiame.com
[4] http://www.undercover2.com/main.php
[5] http://www.theshiftedlibrarian.com/2002/02/20.html

no Japão.

Indo um pouco mais além, pode-se pensar em jogos massivamente multi-usuário ubíquos, MMUG - *Massively Multiplayer Ubiquitous Game*[Tri06], os quais, derivados da união entre MMOGs e MMMOGs, abrangem um conjunto de cenários onde jogadores podem interagir entre si e com o jogo massivo a qualquer momento, de qualquer lugar e através de diferentes dispositivos, como *Desktops* ou aparelhos celulares, criando uma idéia do usuário nunca estar completamente desconectado do jogo. Ainda não existe registro de um jogo massivo ubíquo.

1.1 OBJETIVOS

O objetivo deste trabalho é modelar, em UML[OMG03], parte de uma plataforma para desenvolvimento de jogos MMUG. Mais especificamente, modelar uma arquitetura que suporte o cenário em que jogadores competem entre si através da tecnologia *Bluetooth* e enviam os resultados das partidas ao servidor do jogo massivo via HTTP.

1.2 VISÃO GERAL

Os capítulos 2 e 3 dão uma visão geral a respeito de *Bluetooth* e *J2ME* respectivamente. O capítulo 4 fala os conceitos sobre jogos massivos ubíquos e, finalmente, o capítulo 5 fala sobre a arquitetura citada na seção anterior, contribuição deste trabalho.

CAPÍTULO 2

BLUETOOTH

Bluetooth é uma tecnologia de rádio de curto alcance criada pela Ericsson em meados da década de 90 e desenvolvida hoje por diversas companhias. Esta tecnologia sem fio possibilita a transmissão de dados em curtas distâncias entre telefones celulares, computadores e outros aparelhos eletroeletrônicos. Projetada para ser simples, ela visa cumprir com três restrições fundamentais: um pequeno consumo de energia, baixo custo e a possibilidade de uso em dispositivos pequenos.

O alcance dessa tecnologia depende da classe do rádio empregada no dispositivo. Existem três classes:

- Classe 1: suportam até cem metros.
- Classe 2: suportam até dez metros.
- Classe 3: suportam até dez centímetros.

A maioria dos dispositivos utilizam rádios da classe 2, provendo o alcance de dez metros, em um ambiente livre de obstáculos.

Mais do que somente uma substituição de cabos, a tecnologia sem fio *Bluetooth* provê uma conexão universal para redes de dados existentes, possibilitando a formação de pequenos grupos privados de aparelhos conectados entre si. Ela usa um sistema de frequência de sinal que provê um link seguro e robusto, mesmo em ambientes com alto ruído e de grande interferência

2.1 REDES BLUETOOTH

2.1.1 Piconet e Scatternet

Uma rede *Bluetooth* é chamada de *piconet*. No caso mais simples, significa que dois dispositivos estão conectados (veja Figura 2.1 a). O dispositivo que inicia a conexão é chamado de *master* e os outros dispositivos são chamados de *slaves*. Conexões *Bluetooth* são tipicamente *ad-hoc*, ou seja, são estabelecidas apenas para a tarefa corrente e desfeitas ao termino de tal tarefa.

Um *master* pode possuir, simultaneamente, até 7 *slaves* (veja Figura 2.1 b), entretanto a taxa de transmissão fica limitada. Um dispositivo pode estar conectado em mais de uma *piconet* ao mesmo tempo, esse cenário é chamado de *scatternet* (veja Figura 2.1 c), porém só poderá ser *master* de uma.

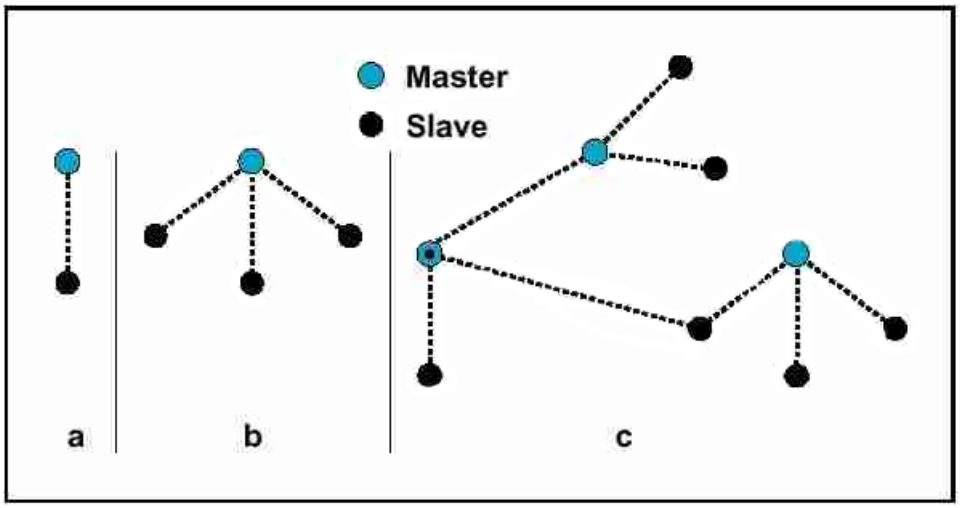

Figura 2.1. Cenários piconet e scatternet.

2.1.2 Frequency Hopping

Bluetooth utiliza uma técnica chamada *Frequency Hopping*, o que significa que cada pacote é transmitido em uma frequência diferente que varia baseada em uma sequência pseudorandômica conhecida tanto pelo remetente quanto pelo destinatário. Esta técnica garante que, dada uma alta taxa de variação de frequência, se consegue uma boa resistencia a interferência e, caso existam duas ou mais *piconets* ativas na mesma área, pacotes que se chocarem em um dado momento serão reenviados em outras frequências seguindo a sequência de cada *master*. A Figura 2.2 mostra duas *piconets* enviando pacotes ao longo do tempo.

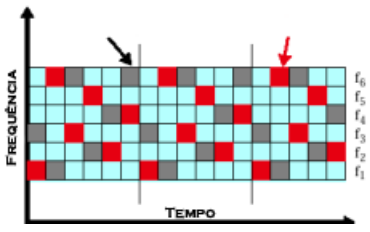

Figura 2.2. Frequency hopping.

2.1.3 Estabelecendo conexões

De uma forma geral, a descoberta de dispositivos é feita pelo seguinte processo: periodicamente, dispositivos que podem ser descobertos entram em um estado chamado *inquiry scan substate*. Nesse estado, o dispositivo fica escutando diversas frequências até que receba uma mensagem de inquérito (*inquiry message*) de um outro aparelho que esteja a procura de um "servidor" *Bluetooth*. Ao receber tal mensagem o dispositivo servidor, antes no estado de *scan*, entra em um estado de resposta (*inquiry response substate*), respondendo com uma mensagem, a qual contém seu endereço e seu *clock*. Só depois de enviado o endereço, os dispositivos podem abrir uma conexão, estabelecer a sequência a ser usada no *Frequency Hopping* para, finalmente, começar a troca de dados.

Depois de conectada, a unidade *Bluetooth* pode entrar em diversos modos de operação com as finalidades de economizar energia e liberar a capacidade das *piconet´s*. Esses modos são listados a seguir:

- **Active mode**: A unidade *Bluetooth* participa normalmente no canal.
- **Sniff mode**: O ciclo de escuta de atividade do *slave* pode ser reduzido, fazendo com que o *master* só possa iniciar uma transmissão em espaços específicos de tempo.
- **Hold mode**: Nesse modo, o *slave* pode fazer outras coisas como *inquiring*, *scanning*...
- **Park mode**: Caso o *slave* não precise participar da *piconet*, mas ainda queira permanecer sincronizado (para participar futuramente).

2.2 A PILHA DE PROTOCOLOS BLUETOOTH

A Figura 2.3 mostra uma visão em alto-nível da arquitetura da pilha de protocolos *Bluetooth*.

As responsabilidades de cada camada na pilha são:

- Camada de Rádio: É a conexão física. O dispositivo *Bluetooth* divide a banda de frequência de 2.4 GHz em 79 canais distantes um do outro em 1 MHz (de 2.402 a 2.480 GHz), e utiliza o seu espectro(*spread spectrum*) para pular de um canal a outro, por mais de 1600 vezes por segundo. O comprimento de onda padrão varia de 10cm a 10m e pode ser estendido para 100m através do aumento do poder de transmissão.

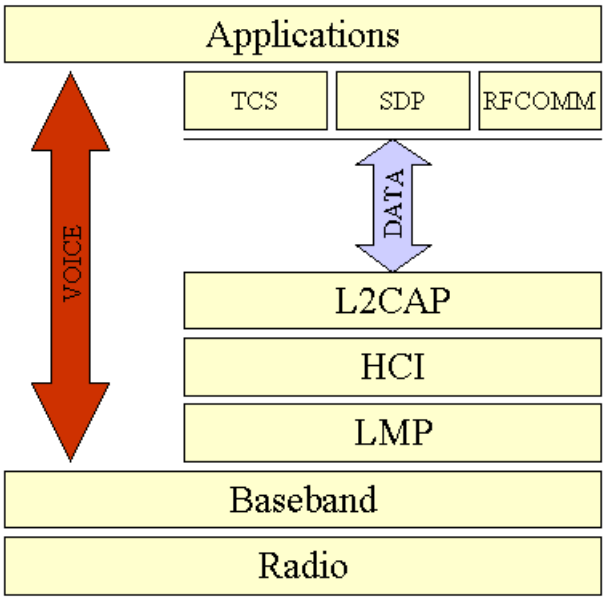

Figura 2.3. Pilha de protocolos Bluetooth.

- Camada *Baseband*: É responsável por controlar e enviar pacotes de dados através do *link* de rádio. Esta camada provê canais de transmissão tanto para dados quanto para voz. A camada *Baseband* mantém *links* síncronos e orientados a conexão (*SCO - Synchronous Connection-Oriented*) para voz e links assíncronos não orientados a conexão(*ACL - Asynchronous Connectionless*) para dados. Pacotes *SCO* nunca são retransmitidos enquanto os pacotes ACL são, para garantir a integridade dos dados.

- LMP - *Link Manager Protocol*: Este protocolo usa os *links* configurados pela camada *Baseband* para estabelecer conexões e gerenciar *piconets*. Entre as responsabilidades do LMP também estão incluídos os serviços de autenticação e de segurança, e o monitoramento da qualidade do serviço.

- *Host Controller Interface* (HCI): É a linha que divide software e hardware. Esta camada e as camadas acima são implementadas em software, a camada LMP a as camadas abaixo são implementadas em hardware. O HCI é a interface do *driver* para o barramento físico que conecta estes dois componentes. Esta camada pode não ser necessária já que a L2CAP pode ser acessada diretamente pela aplicação.

- *Logical Link Control and Application Protocol* (L2CAP): Recebe dados das aplicações e os adapta ao formato *Bluetooth*. Os parâmetros de qualidade de serviço (QoS - *Quality of Service*) são trocados nessa camada.

2.3 PERFIS

O Grupo Especial de Interesse em *Bluetooth* (abreviadamente SIG, do inglês *Special Interest Group*) definiu vários modelos de uso para *Bluetooth*. Eles descreveram as aplicações mais importantes e seus dispositivos alvos.

Perfis especificam como as soluções para as funções dos modelos de uso são disponibilizadas. Em outras palavras, um perfil define os protocolos e suas características para suportar um dado modelo de uso. Os perfis são mostrados na figura Figura 2.4. Alguns perfis são dependentes de outros, por exemplo, três perfis (*File Transfer Profile, Object Push Profile e Synchronization Profile*) são dependentes do *Generic Object Exchage Profile*. Todos os perfis são dependentes do *Generic Access Profile*. Como uma visão mais detalhada de *Bluetooth* foge do escopo deste trabalho, apenas os perfis mais gerais serão apresentados a seguir.

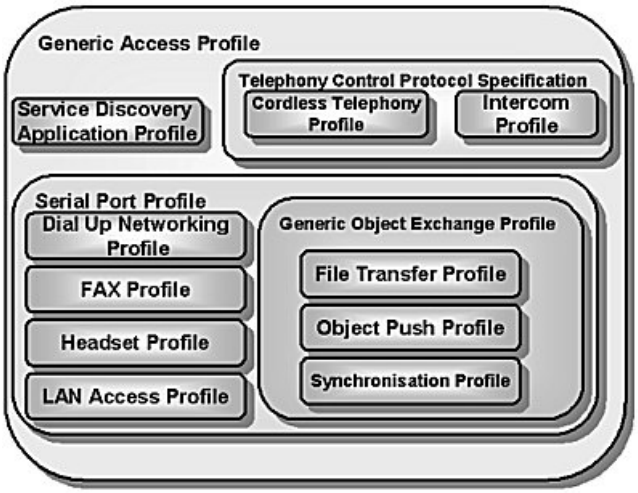

Figura 2.4. Bluetooth profiles.

2.3.1 Os quatro perfis gerais

Generic Access Profile - Esse perfil define os procedimentos genéricos relacionados com o descobrimento de dispositivos *Bluetooth* e aspectos do

gerenciamento de *links* para conexões com os mesmos. Esse perfil também define procedimentos relacionados com o uso de diferentes níveis de segurança.

Service Discovery Application Profile - Define as características e os procedimentos para uma aplicação em um dispositivo *Bluetooth* descobrir serviços de outro dispositivo.

Serial Port Profile - Define os requisitos necessários para configurar conexões de cabo serial emuladas RFCOMM entre dois dispositivos.

Generic Object Exchange Profile - Define os procedimentos necessários ao suporte do protocolo *OBEX*. *OBEX* é um protocolo de comunicação que facilita a troca de objetos binários entre dispositivos.

2.4 SEGURANÇA BLUETOOTH

No *Bluetooth*, a segurança é disponibilizada de três formas: *frequency hopping* pseudo-randômica, autenticação e encriptação. Através de *Frequency Hopping*, torna-se difícil a captura de pacotes. A autenticação permite ao usuário limitar a conectividade com dispositivos específicos. A encriptação usa chaves secretas para tornar os dados visíveis apenas aos dispositivos autorizados.

Todos os dispositivos habilitados com *Bluetooth* devem implementar o *Generic Access Profile*, que contém todos os protocolos. Este profile define um modelo de segurança que inclui três modos:

- Modo 1: É um modo de operação inseguro. Nenhum procedimento de segurança é inicializado.
- Modo 2: É conhecido como segurança reforçada a nível de serviço. Quando os dispositivos operam neste modo, nenhum procedimento de segurança é iniciado antes que o canal seja estabelecido. Este modo permite que aplicações tenham diferentes políticas de acesso e sejam executadas em paralelo.
- Modo 3: É conhecido como segurança reforçada a nível de *link*. Neste modo, os procedimentos de segurança são inicializados antes da configuração do *link* ser completada.

CAPÍTULO 3

J2ME

A plataforma J2ME - *Java 2 Micro Edition*, é uma edição da plataforma Java focada em dispositivos móveis. Esta tecnologia consiste na especificação de um máquina virtual e um conjunto de APIs adequadas às restrições dos dispositivos móveis atuais.

Desde sua criação, a plataforma J2ME vem ganhando cada vez mais espaço no mercado. Suas aplicações podem ser facilmente instaladas nos dispositivos móveis via cabo de dados ou internet, o que fez surgir um novo modelo de negócio, no qual as operadoras de telefonia celular, através de contratos com desenvolvedores, comercializam jogos e aplicações, que podem ser baixados a qualquer momento pelos seus clientes.

3.1 A ARQUITETURA J2ME

A arquitetura J2ME é constituída de configurações, perfis e pacotes opcionais. Através da combinação destes três elementos são construídos ambientes de execução completos e que preenchem os requisitos de diversas classes de dispositivos. Cada um desses elementos é otimizado levando em conta as restrições de memória, armazenamento e processamento de uma determinada categoria de dispositivos. O resultado é uma plataforma comum, que é compatível com a maioria dos dispositivos móveis do mercado. Na Figura 3.1 é mostrada arquitetura geral de J2ME.

3.1.1 Configurações

As configurações definem dois itens importantes para uma classe de dispositivos: uma máquina virtual adequada às necessidades desta classe e a especificação de um conjunto de bibliotecas básicas. Estas bibliotecas disponibilizam funcionalidades básicas a dispositivos que apresentam características similares como conectividade, memória e armazenamento. A implementação destas bibliotecas fica a cargo de cada fabricante. Atualmente existem duas configurações em J2ME: CLDC e CDC.

Connected Limited Device Configuration (CLDC) - É a configuração que especifica os dispositivos mais restritos. A *Sun Microsystems* classifica os dispositivos CLDC como "aqueles que você carrega na

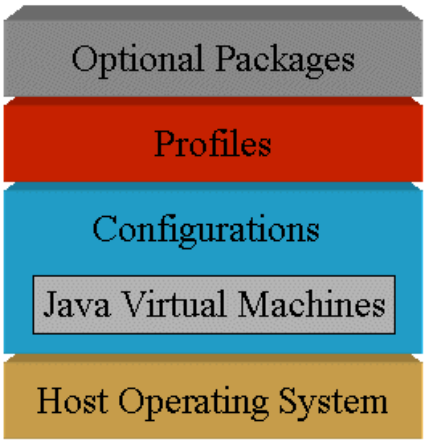

Figura 3.1. Arquitetura J2ME.

mão"[Sun05b]. CLDC foi projetada para os dispositivos que possuem conexões de rede intermitentes, baixo poder de processamento, display pequeno e memória limitada. Em maioria, estes dispositivos são telefones celulares, pagers e PDAs. Um dispositivo que implementa a CLDC deve possuir no mínimo 128 kilobytes de memória para executar a plataforma Java. A CLDC também especifica uma série de classes Java responsáveis pelos tipos de dados, entrada/saída, exceções, coleções e classes do sistema.

Connected Device Configuration (CDC) - A CDC é focada em dispositivos com maior poder de processamento, mais recursos de rede, mais memória e interface com o usuário mais sofisticada. Segundo a *Sun*, "São os aparelhos que você pluga na parede"[Sun05b]. *Gateways*[1] residenciais, *set-top-boxes* de tv digital etc. Fazem parte desta configuração. CDC especifica como máquina virtual a JVM, a máquina virtual utilizada na versão padrão do Java, *J2SE - Java 2 Standard Edition*.

3.1.2 Perfis

Os perfis são extensões das configurações. Através deles dispositivos de uma mesma configuração podem ser utilizados em segmentos de indústria diferentes. Por exemplo, celulares e PDAs, apesar de classificados na mesma configuração (CLDC), ambos possuem diferentes restrições (PDAs possuem

[1] É uma máquina intermédia geralmente destinado a interligar redes, separar domínios de colisão, ou mesmo traduzir protocolos.

telas maiores, maior processamento, interface mais aprimorada). Os Perfis definem um conjunto de APIs de alto nível, que quando combinadas com a API básica das configurações, fornecem um ambiente Java completo, contendo modelo de ciclo de vida da aplicação, interface com o usuário, acesso à propriedades do dispositivo, etc. A seguir serão explanados os perfis existentes no J2ME.

Mobile Information Device Profile (MIDP) - O MIDP é o perfil mais utilizado atualmente e mais aceito no mercado. Designado para telefones celulares e PDAs, este perfil especifica uma interface com o usuário simples, persistência de dados, conexão HTTP e ciclo de vida da aplicação. O MIDP é bastante utilizado no desenvolvimento de jogos para telefones celulares onde possuem grande aceitação no mercado, o que levou à inclusão de uma API para jogos e multimídia em sua segunda versão. O MIDP é normalmente utilizado em conjundo com a CLDC.

Information Module Profile (IMP) - Quando utilizado com a CLDC, o IMP, fornece um ambiente para sistemas embarcados conectados que não possuem rica capacidade gráfica, ou que seus recursos são limitados de alguma outra forma. Caixas de chamada de emergência, parquímetros, módulos sem fio em sistemas de alarme residenciais, são exemplos de dispositivos atendidos por este perfil.

Foundation Profile (FP) - Os perfis CDCs são divididos, ou seja, esses perfis podem ser adicionados, quando necessários, para provê a funcionalidade à diferentes tipos de dispositivos. O *Foundation Profile* é o nível mais baixo de perfil do CDC. Ele fornece uma implementação de rede do CDC que pode ser usada para construir aplicações sem interface com o usuário. Ele pode também ser combinado com o PBP e o PP para dispositivos que necessitem de uma interface com o usuário.

Personal Profile (PP) - O *Personal Profile* é o perfil CDC utilizado em dispositivos que necessitam de um suporte completo para interface, como PDAs e consoles para jogos. Ele inclui a biblioteca AWT completa e é fiel ao ambiente web.

Personal Basis Profile (PBP) - Este Profile é uma divisão do PP que fornece um ambiente para dispositivos conectados que suportem um nível básico de apresentação gráfica ou necessitem do uso de *toolkits* específicos para aplicações. Ambos, PP e PBP, são as camadas superiores do CDC e FP.

3.1.3 Pacotes Opcionais

A plataforma J2ME pode ser extendida com a combinação entre vários pacotes opcionais, as configurações CDC e CLDC, e seus perfis correspondentes. Criados para atender às várias restrições e os requisitos específicos do mercado, os pacotes opcionais oferecem APIs padrões para o uso de tecnologias existentes e também as emergentes como *Bluetooth*, *Web Services*, *wireless messaging*, multimídia, e conectividade com banco de dados. Por serem modulares, os fabricantes podem incluir os pacotes opcionais quando preciso para explorar as funcionalidades de cada dispositivo específico.

3.2 MIDP - MOBILE INFORMATION DEVICE PROFILE

O MIDP, combinado com o CLDC é o ambiente mais utilizado nos telefones celulares atualmente. A especificação do MIDP foi definida através do *Java Community Process*[Pro05], onde são definidas todas as especificações de APIs Java. Este perfil provê diversas funcionalidades importantes para os dispositivos celulares e PDAs mais limitados, incluindo uma simples e leve interface com usuário, armazenamento de dados persistentes, conectividade, segurança, recursos multimídia, e recuros para jogos.

3.2.1 Interface com o usuário

O MIDP possui uma interface com o usuário de alto nível, que facilita o desenvolvimento de aplicações portáveis. A API de interface de alto nível possui implementações de telas para visualização de itens, edição de texto, exibição de alertas, etc. Através das telas de formulário, podem-se incluir imagens, campos de texto, campos de data e hora, tabelas, etc. Todas as telas e itens são dependentes do dispositivo e possuem suporte nativo à entrada de dados, tamanho de tela e capacidade de navegação. Uma API de interface de baixo nível complementa a interface de alto nível, dando aos desenvolvedores maior controle sobre os gráficos e sobre a entrada de dados do usuário quando necessário.

3.2.2 Recursos Multimídia para jogos

O MIDP possui facilidades para o desenvolvimento de jogos portáveis e aplicações multimídia. A API de jogos adiciona recursos comuns à maioria dos jogos, como *sprites*[2], utilizando as capacidades nativas dos dispositivos.

[2]Uma imagem bitmap pequena, usada frequentemente em animações de jogos.

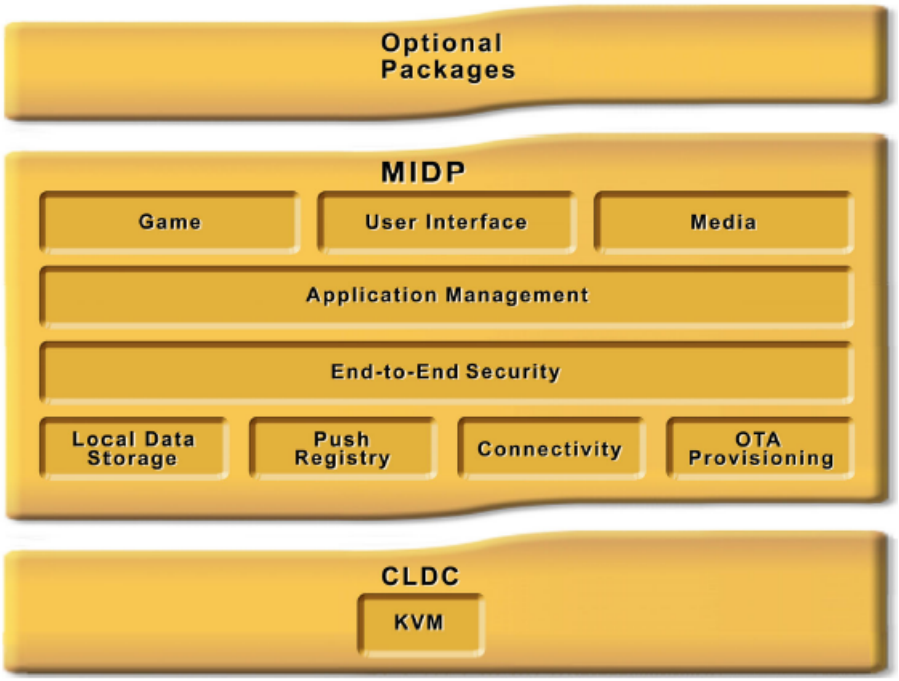

Figura 3.2. MIDP - Visão geral.

O MIDP possui suporte a audio, através de tons, sequências de tons e arquivos WAV. Os desenvolvedores podem utilizar também a *Mobile Media* API (MMAPI), um pacote adicional para o MIDP que adiciona capacidade de exibir vídeos e outros conteúdos multimídia.

3.2.3 Conectividade

O MIDP permite aos desenvolvedores a utilização dos recursos nativos de conectividade e envio de mensagens. Existe suporte a padrões de conectividade como HTTP, HTTPS, datagrama, *sockets*, *server sockets*, e comunicação serial. O MIDP também suporta SMS, o serviço de envio de mensagens de texto. Através do *Push Registry*, aplicações podem ser registradas para receber informações de forma assíncrona. Quando a informação chega, o dispositivo decide como iniciar a aplicação, baseado nas preferências do usuário. Esta arquitetura permite que os desenvolvedores incluam alertas, mensagens assíncronas, e *broadcasts* em aplicações MIDP.

3.2.4 OTA - Over The Air

Um grande benefício do MIDP é a habilidade de instalar e atualizar aplicações através do ar (*Over the air*). A especificação do MIDP define como as aplicações são descobertas, instaladas, atualizadas e removidas dos dispositivos. O MIDP também proporciona um provedor de serviços (OTA *provisioning*) para identificar que aplicação MIDP irá funcionar em um determinado dispositivo e obtém relatórios sobre as instalações, atualizações e remoções.

3.2.5 Segurança

O MIDP provê um modelo de segurança robusto, construído em cima de padrões abertos, que protegem a rede, as aplicações e os dispositivos móveis. O uso do HTTPS, que utiliza SSL e WTLS, permite a transmissão de dados encriptados. Domínios de segurança protegem contra acesso não-autorizado a dados, aplicações e outros recursos da rede feito por outras aplicações MIDP. Por padrão, aplicações MIDP não são confiáveis, e são designadas a domínios não-confiáveis, que previnem o acesso a qualquer funcionalidade privilegiada. Para ganhar acesso privilegiado, uma aplicação MIDP deve ser designada a domínios específicos que são definidos no dispositivo móvel, e são assinados usando o padrão X.509 PKI.

3.3 A API DE JAVA PARA BLUETOOTH - JSR 82

A JSR 82 é a primeira API aberta, não proprietária para o desenvolvimento de aplicações com *Bluetooth* usando Java. A API esconde toda a complexidade da pilha de protocolos, o que permite o foco no desenvolvimento da aplicação sem se preocupar com os detalhes de baixo nível de *Bluetooth*. A JSR 82 é baseada na versao 1.1 da especificação *Bluetooth*. Ressaltando que esta JSR não implementa essa especificação, mas sim provê um conjunto de classes para acessar e controlar dispositivos habilitados com *Bluetooth*.

3.3.1 Funcionalidades

A API tem como propósito provê as seguintes funcionalidades:

- Registrar Serviços.
- Descobrir dispositivos e serviços.
- Estabelecer conexões RFCOMM, L2CAP e OBEX entre dispositivos.

- Através destas conexões, enviar e receber dados (comunicação de voz não é suportada).
- Gerenciar e controlar as conexões de comunicação.
- Prover segurança para estas atividades.

3.3.2 Arquitetura

A JSR 82 foi projetada utilizando as APIs padrões de J2ME e o *framework* de conexão genérica do CLDC/MIDP. Eis algumas características importantes:

- Provê suporte básico aos protocolos e perfis *Bluetooth*. Ela não fornece APIs específicas para todos os perfis, simplesmente por que o número de perfis existentes é muito vasto e cresce cada vez mais.
- Incorpora os protocolos de comunicação OBEX, L2CAP e RFCOMM nas APIs primariamente por que atualmente todos os perfis *Bluetooth* são projetados para utilizar estes protocolos de comunicação.
- *Generic Access Profile, Service Discovery Application Profile, Serial Port Profile e Generic Object Exchange Profile* possuem APIs.
- O protocolo de Descobrimento de Serviços (*Service Discovery*) também é suportado. A JSR 82 define o serviço de registros em detalhes visando padronizar o processo de registro para os programadores de aplicações *Bluetooth*.

A JSR 82 requer que a pilha de protocolos *Bluetooth* abaixo de uma implementação seja qualificada para os *Generic Access Profile, Service Discovery Application Profile e o Serial Port Profile*. A pilha de protocolos também deve provê acesso ao seu protocolo de descobrimento de serviços e às camadas RFCOMM e L2CAP. As APIs são projetadas de maneira que os desenvolvedores possam utilizar a linguagem Java para criar novos perfis no topo da API, contanto que a camada original da especificação não mude. A Figura 3.3 mostra onde a API definida nessa especificação se junta com a arquitetura MIDP/CLDC do J2ME.

3.3.3 Programação de aplicações com a JSR 82

A anatomia de uma aplicação *Bluetooth* possui cinco partes: Inicialização da pilha de protocolos, gerenciamento de dispositivos, descobrimento de dispositivos, descobrimento de serviços e comunicação. A seguir será detalhada cada uma destas partes.

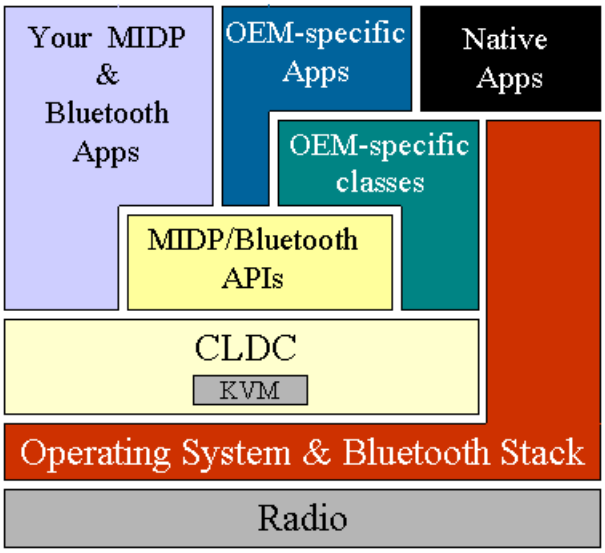

Figura 3.3. Arquitetura de alto nível de J2ME CLDC/MIDP e Bluetooth.

Inicialização da pilha de protocolos - A pilha *Bluetooth* é responsável por controlar o dispositivo. Portanto é preciso que se inicialize a pilha antes de se fazer qualquer outra coisa. O processo de inicialização é composto de uma série de passos cujo propósito é tornar o dispositivo pronto para a comunicação sem fio. Infelizmente, a JSR 82 deixa a implementação do controlador de inicialização livre para os fabricantes, e cada fabricante trata a inicialização da pilha de forma diferente.

Gerenciamento de dispositivos - As APIs de *Bluetooth* contêm as classes *LocalDevice* e *RemoteDevice* que provêem as capacidades de gerenciamento dos dispositivos definidos no *Generic Access Profile*. A classe *LocalDevice* extrai o tipo do dispositivo e os tipos de serviços que ele oferece. A classe *RemoteDevice* representa um dispositivo remoto (que esteja no alcance) e provê métodos para extrair informações sobre o dispositivo, incluindo o seu endereço *Bluetooth* e nome. A classe *RemoteDevice* também disponibiliza métodos para autenticar, autorizar ou encriptar dados transferidos entre dispositivos locais e remotos.

Descobrimento de dispositivos - Pelo fato dos dispositivos sem fio serem móveis, eles necessitam de um mecanismo que os permitem encontrar outros dispositivos e ter acesso as suas funcionalidades.

A classe *DiscoveryAgent* e a interface *DiscoveryListener* do pacote *javax.bluetooth* provêem os serviços de descobrimento. Um dispositivo *Bluetooth* pode utilizar um objeto *DiscoveyAgent* para obter uma lista de dispositivos acessíveis

através de alguma destas três maneiras:

O método *DiscoveryAgent.startInquiry* coloca o dispositivo no modo de procura. Para se tirar vantagem deste modo, a aplicação deve especificar um *DiscoveryListener* que irá responder a eventos relacionados com a procura. O método *DiscoveryListener.deviceDiscovered* é chamado cada vez que um dispositivo for encontrado. Quando a procura é completada ou cancelada, *DiscoveryListener.inquityCompleted* é invocado.

Descobrimento de serviços - Uma vez que o dispositivo local descobriu pelo menos um dispositivo remoto, ele pode iniciar a procura por serviços disponíveis, que nada mais são do que aplicações *Bluetooth* que podem ser utilizadas para cumprir determinadas tarefas. Por ser muito semelhante ao descobrimento de dispositivos, o descobrimento de serviços também utiliza a classe *DiscoveryAgent*, que possui métodos para descobrir serviços em dispositivos *Bluetooth*.

Registro de serviços - Antes que um serviço possa ser descoberto, ele deve ser registrado, tornando-se disponível em um dispositivo. Este é responsável por:

- Criar um registro que descreve o serviço oferecido.
- Adicionar o registro do serviço no banco de dados do descobrimento de serviços (*Service Discovery Database* - SDDB), sendo então visível e disponível para clientes em potencial.
- Registrar as medidas de segurança associadas ao serviço.
- Aceitar conexões dos clientes
- Atualizar o registro do serviço no SDDB cada vez que os atributos dos serviços se modificam.
- Remoção ou desativação de um registro de serviço no SDDB quando o este não está mais disponível.

Comunicação - Para um dispositivo local utilizar um serviço de um dispositivo remoto, os dois dispositivos devem compartilhar um protocolo de comunicação comum. Através deste protocolo, as aplicações poderão acessar uma grande variedade de serviços. A API de *Bluetooth* provê mecanismos que permitem conexões a qualquer serviço que utilize RFCOMM, L2CAP ou OBEX como protocolo. Se um serviço utiliza algum outro protocolo (ex: TCP/IP) acima destes protocolos citados, a aplicação poderá acessá-lo, mas apenas se o protocolo adicional for implementado na aplicação, usando o CLDC.

CAPÍTULO 4

JOGOS MASSIVAMENTE MULTIUSUÁRIO UBÍQUOS

Como citado, o surgimento de plataformas abertas como J2ME e Brew em paralelo a maturação de tecnologias de comunicação sem fio, a exemplo do *Bluetooth*, despertou um grande interesse no mercado pelo desenvolvimento de jogos multi-usuário em dispositivos móveis. A possibilidade de acessar jogos massivos (MMGs) através de dispositivos móveis provê ao jogador o poder de interagir com o jogo a qualquer momento e de qualquer lugar, criando o chamado MMUG - do inglês *Massively Multiplayer Ubiquitous Game*[Tri06].

Idealmente, o jogador deveria ter acesso a qualquer parte do mundo do jogo independente do tipo de dispositivo utilizado, mas esse requisito gera problemas de adaptação do jogo devido as diferentes características de cada dispositivo, como memória, poder de processamento, latência de conexão.

4.1 CENÁRIOS DE UM JOGO MASSIVO UBÍQUO

Considere um jogador que participe de um jogo massivamente multi-usuário ubíquo. Neste jogo, o jogador tem por objetivo principal a evolução de seu personagem em diferentes critérios como, por exemplo, força, agilidade, inteligência e riquezas materiais. Quando o jogador está em casa, em um cybercafé ou na casa de algum amigo (cenário 4 da Figura 4.1), ele pode acessar o jogo utilizando computadores pessoais com banda larga, alto poder de processamento etc. Este cenário permite que o jogador trave batalhas e tenha uma interação em tempo real com o jogo.

Entretanto, em certas situações torna-se inviável ao jogador ter acesso a computadores pessoais. Situações como esta podem acontecer quando o jogador está em um consultório médico, na fila do banco ou no ônibus. Nestas situações o jogador pode querer consultar informações a respeito de seu personagem ou sobre o mundo do jogo. Para isto o ele pode acessar o jogo através de seu telefone celular utilzando a rede de telefonia (cenário 1 da Figura 4.1). Neste cenário não se torna viável a interação com o mundo do jogo da mesma maneira que no cenário 4, mas o jogador poderia efetuar ações gerenciais no jogo, por exemplo fazer com que seu jogador tomasse posição defensiva no jogo ou fosse descansar.

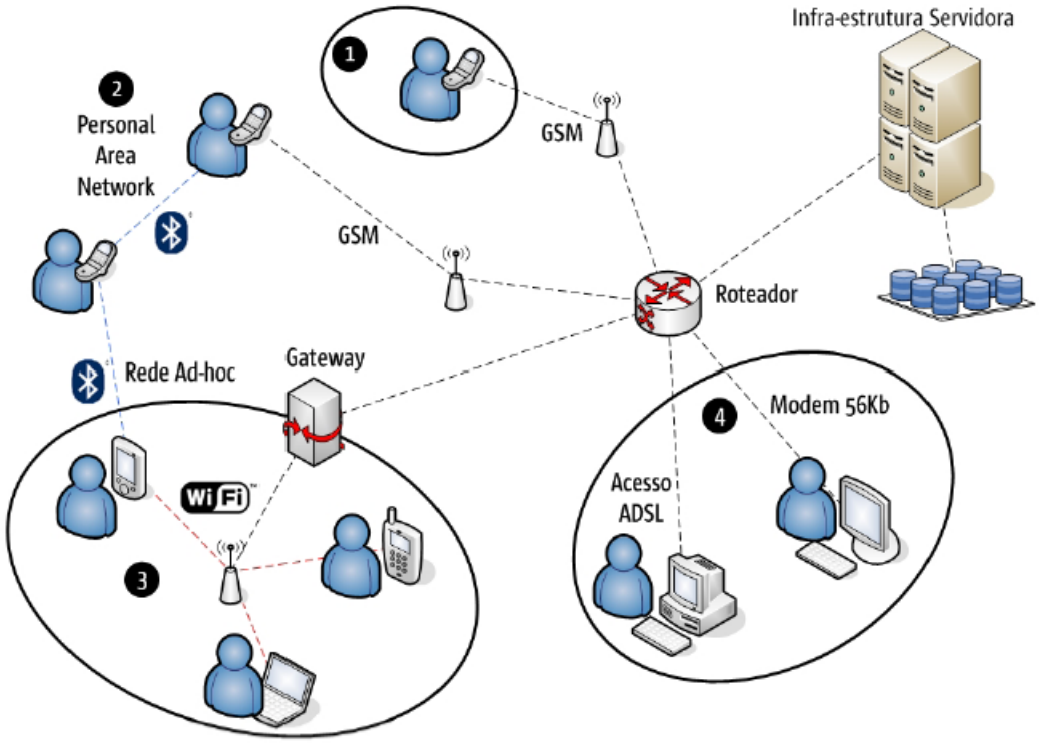

Figura 4.1. Cenário de um jogo massivamente multi-usuário ubíquo.

O cenário 4 é apropriado a uma funcionalidade de localização geográfica de jogadores nas proximidades pois, através de, por exemplo, *Bluetooth*, estes jogadores podem formar uma rede *ad-hoc* e disputar partidas desconectadas do servidor do jogo ubíquo. Ao fim da batalha o vencedor recebe bonificações e os atributos dos personagens são sincronizados com os personagens existentes no mundo do jogo massivamente multi-usuário através da rede de telefonia celular (cenário 2 da Figura 4.1).

Em uma situação onde o usuário se encontra em um local, o qual disponibiliza uma rede através de um ponto de acesso Wi-Fi (802.11), o jogador, através de um dispositivo habilitado com esta tecnologia, pode realizar ações no jogo com uma maior interatividade em relação ao acesso através da rede de telefonia celular. Nesse modelo, os jogadores podem tanto interagir diretamente com o servidor como disputar partidas locais com submissão de resultados, dependendo das características do dispositivo utilizado. Esse modelo é mostrado no cenário 3 da Figura 4.1.

4.2 MODELO DE APLICAÇÃO MMUG

Esta seção fala sobre um modelo para jogos ubíquos baseado nos padrões utilizados atualmente pela maioria dos jogos massivamente multi-usuário. A Figura 4.2 apresenta alguns destes conceitos onde maiores detalhes serão expostos a seguir.

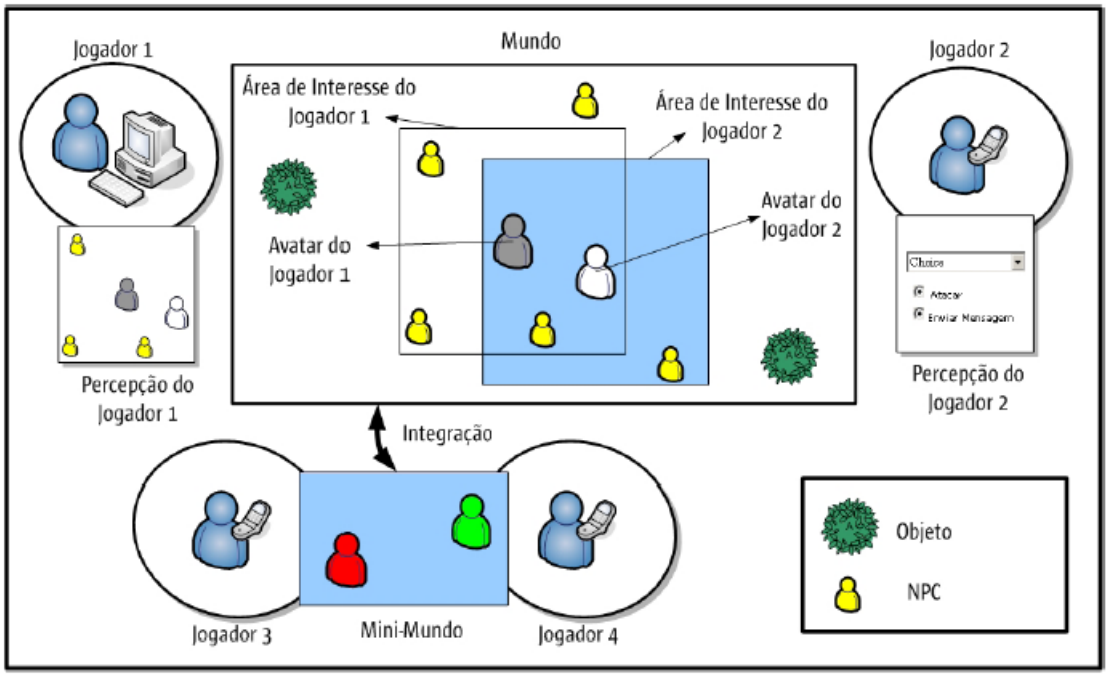

Figura 4.2. Modelo de aplicação MMUG.

- *Terreno*: são as informações imutáveis que compõem a paisagem do mundo virtual, influenciando o deslocamento dos jogadores. Exemplos deste elementos são rios, montanhas ou depressões.

- *Objeto*: qualquer elemento que pode ser inserido, copiado, alterado e removido do *mundo virtual*.

- *Mundo virtual*: é o espaço compartilhado pelos jogadores e objetos submetidos a um sistema de regras e/ou leis. Objetos em um mesmo mundo virtual podem interagir entre si (troca de informações, colisão, etc). Por tratar-se de um jogo massivo, o mundo virtual costuma ser grande, o que torna necessária a utilização de *Áreas de interesse*.

- *Áreas de interesse*: subconjuntos do mundo virtual de forma que jogadores possam interagir apenas com um desses subconjunto em um dado instante.

- *Avatar*: É o personagem do jogador no mundo virtual.

- *NPC - Non player Character*: é todo elemento do jogo controlado por inteligência artificial.

- *Contexto de execução*: está relacionado com a infra-estrutura de acesso de um jogador ao mundo virtual, incluindo o tipo de rede e o dispositivo utilizado para acessar o jogo.

- *Percepção*: indica como a área de interesse é apresentada ao jogador.

- *Interação*: é a forma como o jogador realiza ações no mundo virtual.

- *Mini-mundo*: são cenários paralelos de interação direta entre jogadores móveis utilizando redes espontâneas (*ad-hoc*) ou redes sem fio estruturadas (WLAN). Estes cenários são constituídos de partidas isoladas, mas influenciam no mundo do jogo. Um exemplo poderia ser um jogo de temática medieval onde um Mini-mundo poderia representar um duelo de espada entre jogadores em tempo real via *Bluetooth*, cujo resultado da batalha é enviado e sincronizado com o mundo do jogo. A vantagem destes cenários é que eles geram formas de interação sem as limitações impostas pela rede de telecomunicação. Ao contrário do mundo virtual, Mini-mundos são partidas de curta duração sem os requisitos de persistência de estado de jogo. Para ser feita a integração entre os Mini-mundos e o estado global do jogo, necessita-se que o último seja notificado da existência destes Mini-mundos e também dos resultados ao final de cada partida. A formação de Mini-mundos é facilitada pelos serviços de localizaçao, como citado anteriormente.

Em resumo, o jogo acontece pela interação dos jogadores, através de seus avatares, com NPCs, outros avatares e/ou qualquer objeto do mundo virtual. Esta interação pode ser diretamente, por *desktops*, ou indiretamente, por Mini-mundos. Vale ressaltar que diferentes dispositivos provêem diferentes percepões ao jogador.

4.3 MODELO DE SUPORTE MMUG

A partir do modelo de aplicação MMUG surge uma série de requisitos funcionais e não funcionais . Tais requisitos serão tratados por um modelo

de suporte para jogos massivos ubíquos. Entre os requisitos não funcionais estão escalabilidade e interoperabilidade. O modelo de suporte MMUG deve permitir a comunicação entre dispositivos de forma transparente, onde a utilização de dispositivos móveis por certos jogadores não deve trazer impacto negativo a jogadores que possuam infra-estrutura mais adequada e vice-versa.

Em adicional, o modelo de suporte MMUG estende os requisitos funcionais dos jogos massivos tradicionais acrescentando uma série de serviços. A Figura 4.3 mostra estes serviços agrupados de acordo com o tipo de jogo a que o serviço pertence.

Figura 4.3. Serviços MMUG.

4.3.1 Serviço de Gerenciamento de Contexto

Realiza o monitoramento reativo ao contexto de execução de um jogador para que as adaptações da interação e percepção do jogador possam ser realizadas.

4.3.2 Serviço de Adaptação de Conteúdo

Transforma as informações sobre as áreas de interesse de um jogador, para que a mesma esteja em um formato adequado ao seu contexto de execução, seguindo o modelo de percepção do modelo de aplicação do jogo. O formato da exibição de uma área de interesse, dado o contexto, é especificado pelo desenvolvedor do jogo, sendo possível a existência de diferentes formatos

para modelos distintos de uma mesma categoria de dispositivos, por exemplo, telefones celulares com menores ou maiores recursos de apresentação.

4.3.3 Serviço de Adaptação de Interação

Realiza as transformações das diferentes formas em que os jogadores podem interagir com o jogo em contextos diferentes. Este serviço cria um conjunto de ações padronizadas para cada área de interesse.

4.3.4 Serviço de Notificação de Eventos

Este serviço permite que os usuários sejam informados sobre eventos do jogo. As notificações podem ser feitas através de SMS (*Short Message Service*), *e-mail* ou aplicativos de mensagens instantâneas. Este serviço permite que o jogador possa ser encontrado e notificado através de diferentes meios de comunicação.

4.3.5 Serviço de Integração de Mini-mundos

Este serviço é responsável pela integração entre o mundo do jogo massivo e os cenários paralelos ao jogo (Mini-mundos). Este serviço deve disponibilizar, dentre outras, as seguintes operações:

- Permitir que os jogadores possam criar novas partidas (instâncias de Mini-mundo), informando também sua localização.
- Notificar o jogo sobre o início de um Mini-mundo.
- Indicar a localização física aproximada dos jogadores que fazem parte de instâncias de Mini-mundos.
- Permitir a recuperação dos dados persistentes do jogador para o contexto do Mini-mundo.
- Sincronizar os resultados dos Mini-mundos terminados com o mundo do jogo global.

CAPÍTULO 5

JOGOS UBÍQUOS COM BLUETOOTH

O capítulo anterior fala de um novo cenário de jogos online: Jogos Massivamente Multiusuário Ubíquos - MMUG (do inglês *Massively Multiplayer Ubiquitous Game*. Tais jogos possuem como grande novidade o atributo da pervasividade[1], permitindo que seus usuários possam jogá-lo de qualquer lugar a qualquer hora, desde que possuam algum dispositivo, que se conecte a *Internet*.

Devido a sua inerente ubiquidade, esses jogos devem possuir suporte à diversas tecnologias e modelos de jogo[2]. Este capítulo apresenta um modelo de jogo que utiliza conexões *Bluetooth* entre aparelhos celulares para simular pequenas partidas multiusuário e HTTP para submissão dos resultados dessas partidas ao contexto maior, o MMUG. Este modelo de jogo representa o *Serviço de Integração de Mini-mundos*, citado no capítulo anterior, com duas peculiaridades: ele não suporta localização geográfica de jogadores e adiciona ao serviço as primitivas lógicas de um jogo multiplayer (conexão, sincronização e troca de mensagens). Aqui são apresentados as restrições necessárias a esse modelo bem como uma pequena arquitetura que suporte o mesmo. Por convenção, as figuras deste capítulo estão em inglês.

5.1 RESTRIÇÕES

Esta seção apresenta as restrições necessárias em um jogo MMUG para que haja suporte a cenários de batalha via *Bluetooth*. Há um enfoque maior em problemas inerentes a telefones móveis.

5.1.1 Aparelhos celulares

Os jogos MMUG deverão ser em tempo real, geralmente de RPG [ECJ03]. Entretanto, os aparelhos celulares possuem uma latência de conexão muito alta [PM04], impossibilitando a interatividade necessária para jogos deste tipo. Eis o motivo para que nosso modelo de jogo gere resultados *off-line* para posteriormente enviar estes a um servidor MMUG.

Um outro problema no desenvolvimento de sistemas para celulares são as restrições de tamanho de tais aplicações. Um jogo construído para um

[1] Mesmo que ubiquidade.

[2] Forma como usuários interagem e modificam o estado do jogo.

aparelho da série 40 não deve ultrapassar 64kb [Sun05a], o que, por exemplo, restringe o tamanho de uma possível API para o nosso modelo de jogo em 15kb [PM04]. Como resultado, o desenvolvedor do jogo pode ser obrigado a acrescentar as funcionalidades propostas pela nossa arquitetura diretamente no código do jogo ao invés de incorporá-las via reutilização de um componente.

5.1.2 Integração com um MMUG

A integração dos resultados das partidas com o servidor MMUG deve ser explicitamente ordenada por dois participante da mesma, para garantir a corretude dos resultados. A necessidade dessa dupla submissão é derivada da facilidade de se trapacear em programas residentes em telefones móveis, pois o código do programa reside nos telefones, bastando apenas que o trapaceiro conheça técnicas de engenharia reversa para ter acesso a esse código e subsequentemente alterá-lo.

5.2 UMA ARQUITETURA BLUETOOTH PARA JOGOS MMUG

Dadas as restrições, esta seção apresenta um projeto de uma arquitetura para J2ME, a qual suporte nosso modelo de jogo. Esse projeto se concentra mais nos aspectos comportamentais da arquitetura, priorizando responder as perguntas "O que?"e "Quando?"executar uma tarefa ao invés de "Como?".

5.2.1 Modelagem Estática

Essa arquitetura só dará suporte a *piconets* e deve seguir as restrições citadas anteriormente. Em adicional, ela deve possuir uma quantidade mínima de classes e não deve possuir pacotes, para melhor performance[PM04].

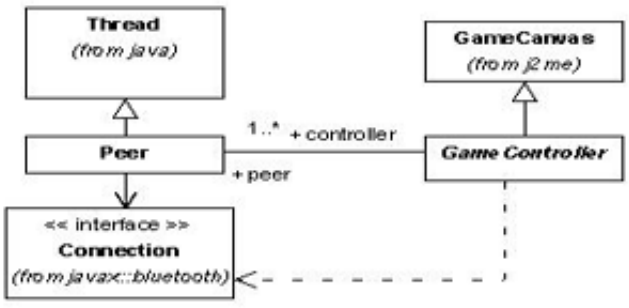

Figura 5.1. Diagrama de classes da API.

A Figura 5.1 apresenta a modelagem estática da arquitetura em um diagrama de classes UML. As funções das classes *GameController* e *Peer*, e seus relacionamentos serão explicados a medida que a dinâmica do sistema for sendo apresentada. Aqui, parte-se do pré-suposto que o leitor conheça a interface *Connection* e a classe *GameCanvas*, ambos do MIDP 2.0; e a classe *Thread* de *java.lang*. A partir de agora convencionamos que o termo *controlador* se refere a qualquer objeto do tipo *GameController* independente de seu papel na rede.

5.2.2 Modelagem Dinâmica

A modelagem dinâmica da arquitetura será apresentada a partir da máquina de estados do controlador (*GameController*). Esta, por sua vez, será explicada na ordem temporal que os eventos de um jogo acontecem. A Figura 5.2 mostra a máquina de estados em UML deste controlador completa e dividida em áreas. O objetivo desta divisão é facilitar a visualização e entendimento desta máquina de estados através de uma visualisação mais detalhada de cada uma destas áreas ao longo deste capítulo.

Figura 5.2. Máquina de estados do controlador.

Atualização de Avatares - A partir deste ponto, vamos reestringir a definição de *Mini-mundo*, citada no capítulo 4, à partidas *off-line* jogadas em aparelhos celulares via conexões *Bluetooth*.

Quando se inicia uma nova partida, o estado do avatar do jogador no celular deve ser sincronizado com o estado do MMUG. Isto é realizado através de uma requisição ao servidor MMUG. Tal servidor envia o novo estado, caso esse exista. Se nenhuma atualização for necessária, o servidor retorna uma mensagem comunicando tal fato.

A Figura 5.3 mostra os estados que modelam esse comportamento. No estado *asking the server for updates*, o controlador pergunta ao servidor se existe alguma atualização para o personagem do usuário, caso haja, ele vai para um estado no qual realiza essa atualização - *updating* - segue para o estado no qual se escolhe a política do jogo, *waiting for game policy selection* (políticas de jogo serão explicadas mais a frente); e finalmente vai para o estado *waiting for type selection*, o qual vai dar inicio a criação do Mini-mundo; caso não haja atualizações, o controlador pula o estado *updating* e segue normalmente.

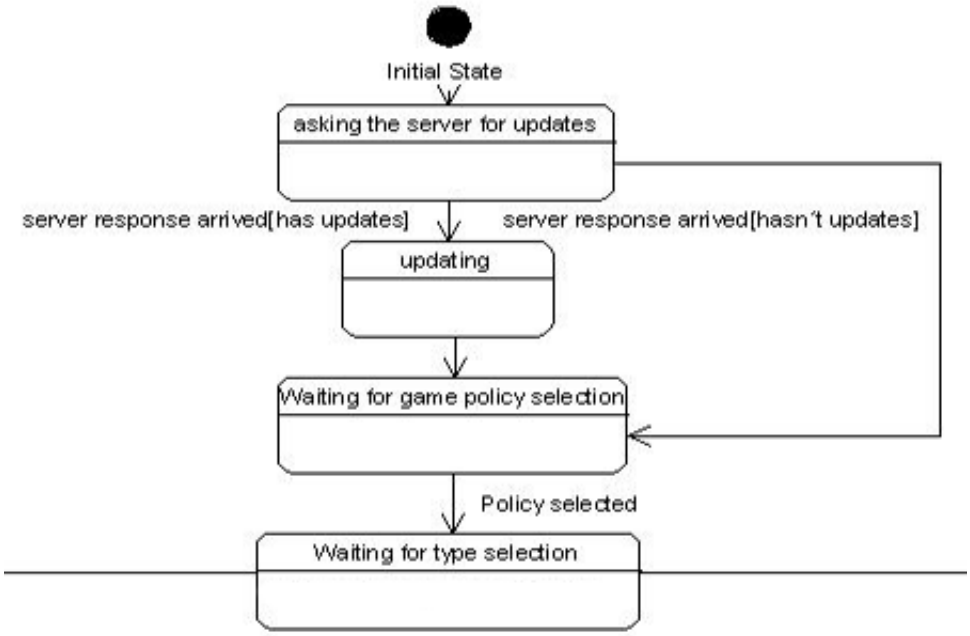

Figura 5.3. Estados relativos à atualização de avatares. (Área A da Figura 5.2)

O servidor do jogo MMUG deve dar suporte às diferentes requisições dos Mini-mundos. Esse suporte é dado através de um componente residente no servidor. Os estados desse componente são mostrados na Figura 5.4.

As requisiçoes de atualização do jogo são atendidas pelo estado *waiting for avatar update requests*, que envia o novo estado do avatar, caso este exista. Este estado é mostrado da linha de concorrência 3 (as outras linhas serão explicadas no decorrer deste capítulo).

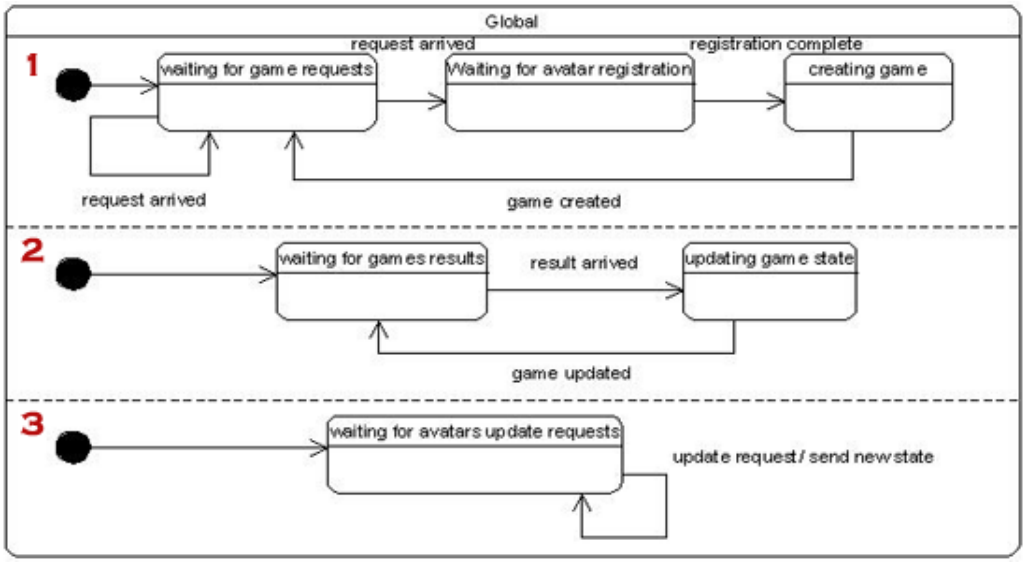

Figura 5.4. Estados do servidor MMMUG.

Criação da Partida e Tratamento de Conexões - Depois de atualizar o estado do avatar, o controlador se encontra no estado *waiting for type selection*, que representa a escolha entre conectar-se a um Mini-mundo previamente criado ou criar tal Mini-mundo.

No caso da criação do jogo, o controlador assume o papel de *master* da rede e vai para o estado *aquiring game id*, onde solicita ao servidor um identificador único para o Mini-mundo. O servidor atende essa requisição pelo seu estado *waiting for game requests* da linha de concorrência 1, Figura 5.4. Depois de enviar o identificador, o servidor fica a espera dos *id´s* dos avatares que participarão desse Mini-mundo para, só então, criar o jogo. Os estados *waiting for avatar registration* e *creating game* respondem por esse comportamento. A partir daqui convencionamos como *master* o objeto *GameController* que responde pelo dispositivo no papel mestre da rede.

Com o *id* estabelecido, o *master* passa a esperar conexões no estado *waiting for connections* (Figura 5.5). Toda vez que uma nova conexão é estabelecida, um objeto da classe *Peer*, Figura 5.1, é criado e associado ao *master*. Esse objeto *Peer* recebe as responsabilidades de sincronização e troca de mensagens pela rede com o dispositivo recém conectado: o dispositivo

slave. A associação entre um *Peer* e um *slave* é de um pra um. O *master*, em paralelo, continua esperando novas conexões.

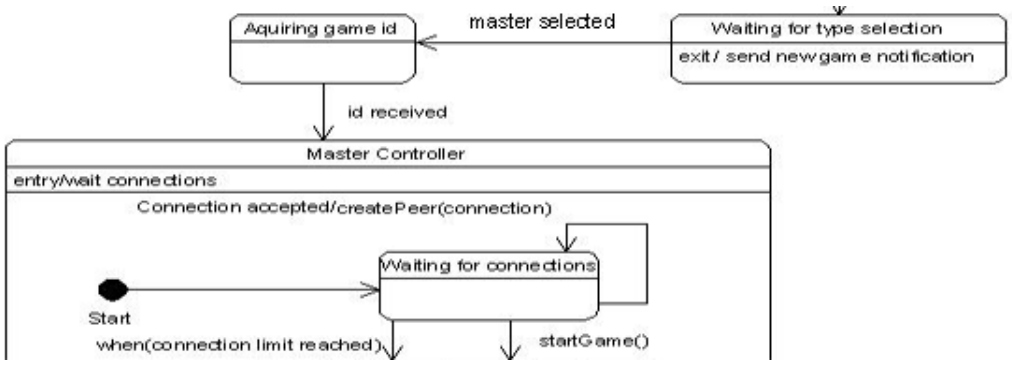

Figura 5.5. Criação de um Mini-mundo. (Área B da Figura 5.2)

O jogo inicia por comando explícito do usuário que o criou ou quando o limite de 7 conexões, limite para *piconets*, é atingido. Este comportamento é mostrado na Figura 5.5.

Localização de Jogos e Sincronização - Como dito anteriormente, depois de atualizar o estado do avatar, o controlador pode conectar-se a um Mini-mundo previamente criado ou criar tal Mini-mundo. Caso a escolha de conectar-se a um Mini-mundo seja feita, o controlador, agora no papel de *slave*, procura e se conecta a tal Mini-mundo seguindo os padrões *Bluetooth* citados no capítulo 2. Essas atividades são representadas nos estados *looking for game* e *connecting to game* respectivamente; como mostra a Figura 5.6. A partir daqui convencionamos como *slave* o objeto *GameController* que responde por um dispositivo no papel escravo da rede.

A Figura 5.6 mostra também o estado *synchronizing*, no qual as atividades de inicialização necessárias aos *slaves* participantes da rede são realizadas. Essa inicialização acontece em paralelo, através da troca de mensagens, com as atividades de inicialização do objeto *Peer* criado no *master* para esta conexão específica. Essas atividades estão dispostas na Figura 5.7.

O processo de inicialização começa com o *slave* enviando o identificador ao seu *Peer* para que o *master* possa manter registro dos participantes do Mini-mundo. Em seguida, o processo de sincronização é iniciado. O *Peer* envia uma mensagem específica a esse processo para o *slave*, o qual retorna imediatamente uma mensagem vazia (NOP). O *Peer* usa o intervalo de tempo entre o envio e o recebimento dessas mensagens para calcular o atraso da rede: atraso=tempo/2. Isso pode ser realizado n vezes até um atraso médio

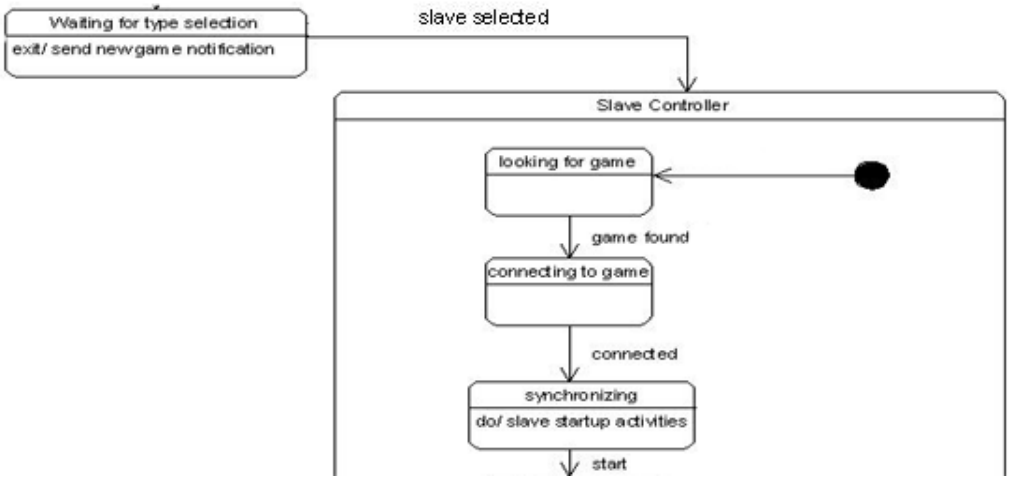

Figura 5.6. Localização, conexão e inicialização de escravos. (Área D da Figura 5.2)

seja calculado. Com essa informação o *Peer* envia o relógio lógico do *master* acrescido do atraso da rede ao *slave*, terminando assim o processo de sincronização. Por fim, o identificador do Mini-mundo é enviado ao *slave*.

O relógio lógico, R.L, é calculado através da subtração do relógio real por uma constante. O *master* cria a constante arbritariamente, por exemplo 1000, então quando o relógio real dele for 1250ms, o R.L será 250. Para um atraso da rede de 4ms, o *Peer* deve enviar ao *slave* o valor 254, pois esse será o valor do R.L do primeiro quando a mensagem atingir o último. Recebida a mensagem, o *slave* deve subtrair o valor 254 do seu relógio real, digamos 1264ms, chegando sua a constante, neste caso 1010, terminando assim o processo de sincronização. Note que no momento da recepção da mensagem pelo *slave* o relógio real do *master* é 1254ms, o qual subtraído do valor 1000 possui o mesmo resultado da subtração do atual relógio real do *slave*, 1264, pela sua recém calculada constante, 1010.

O estado *Synchronizing as master* da classe *Peer* responde pelas atividades de sincronização descritas anteriormente. (Figura 5.8).

Decorrer do Jogo - Os *Peer´s*, depois de sincronizados, ficam em um estado de espera pelo inicio do jogo, *waiting gaming start signal*, Figura 5.8. No caso dos *slaves*, eles permanecem no estado de sincronização, mas sem realizar atividade nenhuma. Ficam apenas na espera, também, do sinal de início, Figura 5.9.

O *master* pode, a qualquer momento, iniciar o jogo através de um comando explícito ou quando o limite de conexões for atingido. Em ambos

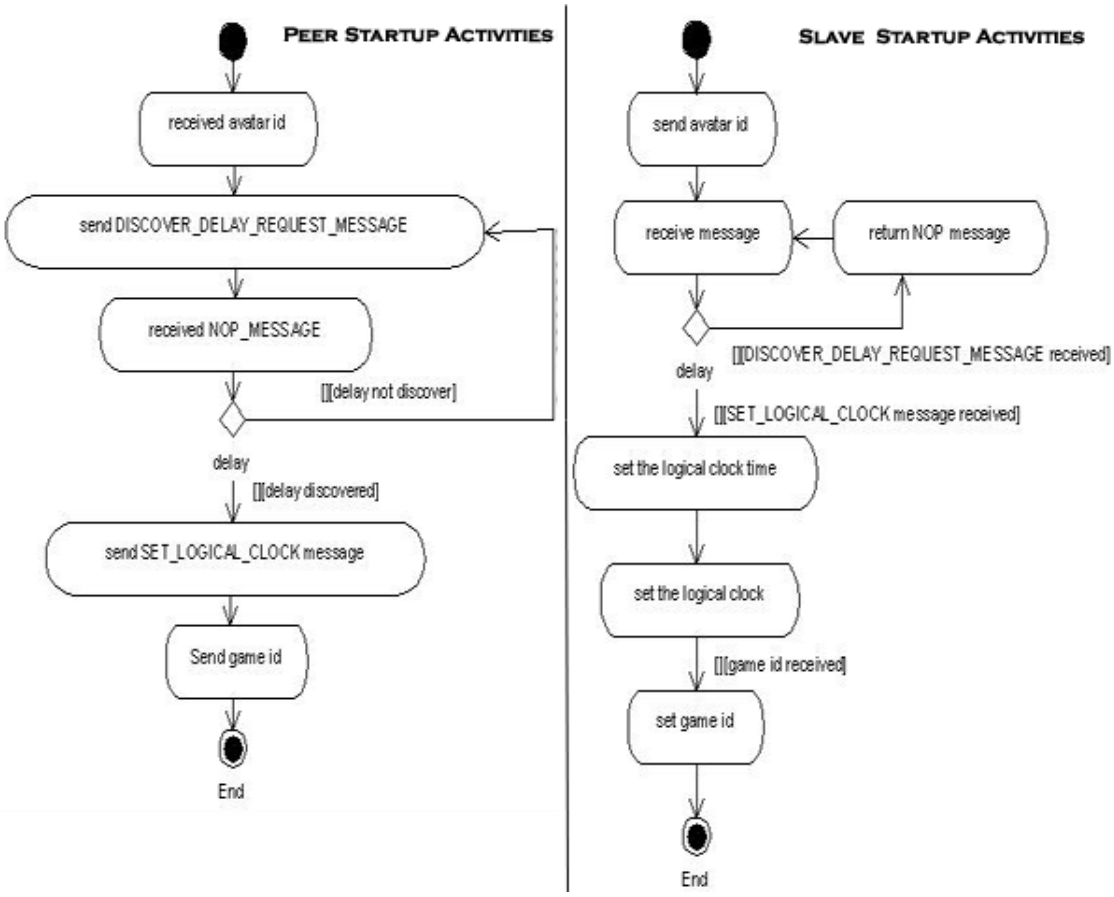

Figura 5.7. Atividades paralelas de inicialização.

os casos ele vai para o estado *registering players avatars in server* no qual avisa ao servidor quais são os participantes do Mini-mundo (utilizando os identificadores previamente adquiridos) através da linha de concorrência 1 do estado do servidor, Figura 5.4. Posteriormente, o *master* vai para o estado *Preparing for gaming*, onde envia a todos os *Peer´s* o sinal de *start*. Esse sinal possui a hora lógica que o jogo deve começar e o id de todos os participantes do Mini-mundo. Esses estados estão ilustrados na Figura 5.10.

Ao receber o sinal de início, cada *Peer* o envia para seu respectivo *slave* e todos, *slaves* e *Peer´s*, ficam no estado *waiting for game time*, no qual ficam esperando chegar a hora lógica, passada através desse sinal, para iniciar o jogo.

A idéia central do funcionamento desses Mini-mundos é que seu estado seja replicado em todos os seus participantes e cada comando, executado

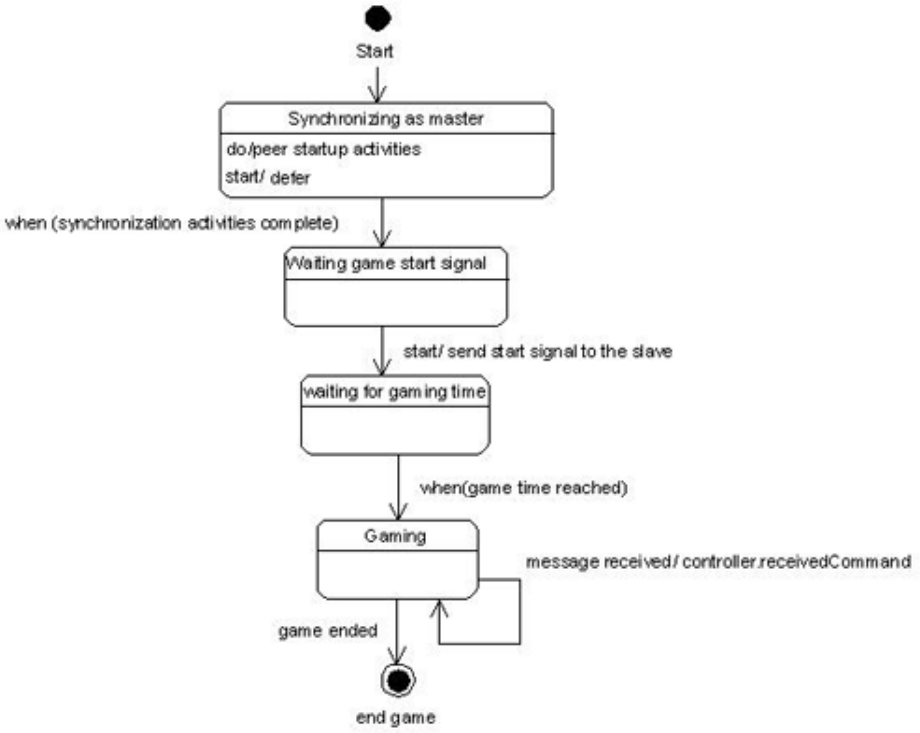

Figura 5.8. Máquina de estados da classe *Peer*.

por qualquer um desses participantes, seja enviado a todos os outros. Esses comandos possuem a hora lógica que foram criados para que os controladores saibam quando executá-los. A estratégia para executar os comandos utiliza o *bucket synchronization algorithm* [DG99], o qual mantém registro apenas o comando mais recente enviado por cada participante e, de tempos em tempos, executa esses comandos e os descarta. Esse procedimento garante que os estados replicados permanecam consistentes.

Iniciado o jogo, todos os participantes podem começar a enviar comandos, seguindo as restrições da política do jogo, escolhida previamente no estado *waiting for game policy selection*, Figura 5.3. Existem 3 políticas de jogo:

- Baseados em turno (*Turn based games*): Nos jogos baseados em turno, cada participante deve esperar sua vez para poder executar algum comando. Esse, quando enviado pelo jogador da vez, é executado normalmente, encadeando a passagem da vez para o próximo participante desse Mini-mundo. Qualquer comando enviado por outro jogador é descartado. É importante que o cálculo da ordem dos participantes seja determinístico, visto que ela será caculada, independentemente,

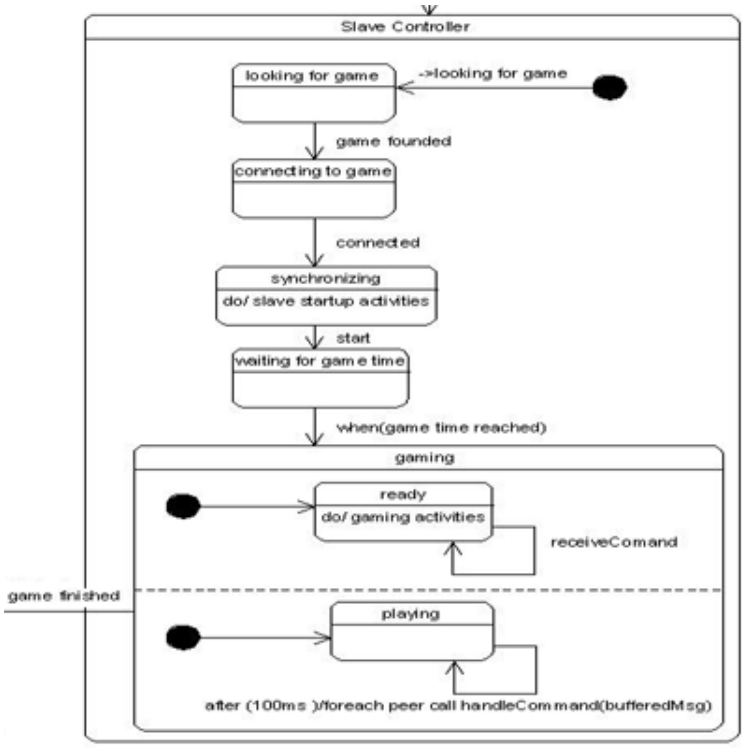

Figura 5.9. Decorrer do jogo do escravo. (Áreas D e E da Figura 5.2)

em cada controlador.

- Simultâneos (*Simultaneous games*): Nesse tipo de jogo, os participantes devem enviar seus comandos previamente. Só após a chegada de um comando de cada participante, o controlador os executa como um bloco atômico. Apenas após a execução de tal bloco, os participantes podem enviar outros comandos.

- Tempo Real (*Real time games*): Os comandos podem ser enviados a qualquer momento e serão executados na ordem que chegarem.

Os estados, responsáveis pelo decorrer do jogo, do *master* e do *slave* são mostrados na Figura 5.10 e na Figura 5.9 respectivamente. Em ambos, o controlador permanece, concorrentemente, em dois estados. O estado *ready* é responsável pelo recebimento, validação e armazenamento dos comandos e o estado *playing* fica responsável pela execução de comandos a cada 100ms seguindo as regras da política de jogo em questão. Para as 3 políticas de jogo citadas, a dinâmica da troca de comandos é a mesma, exceto quanto as regras de validação dos comandos, pelo estado *ready*, que deverão, também, ir

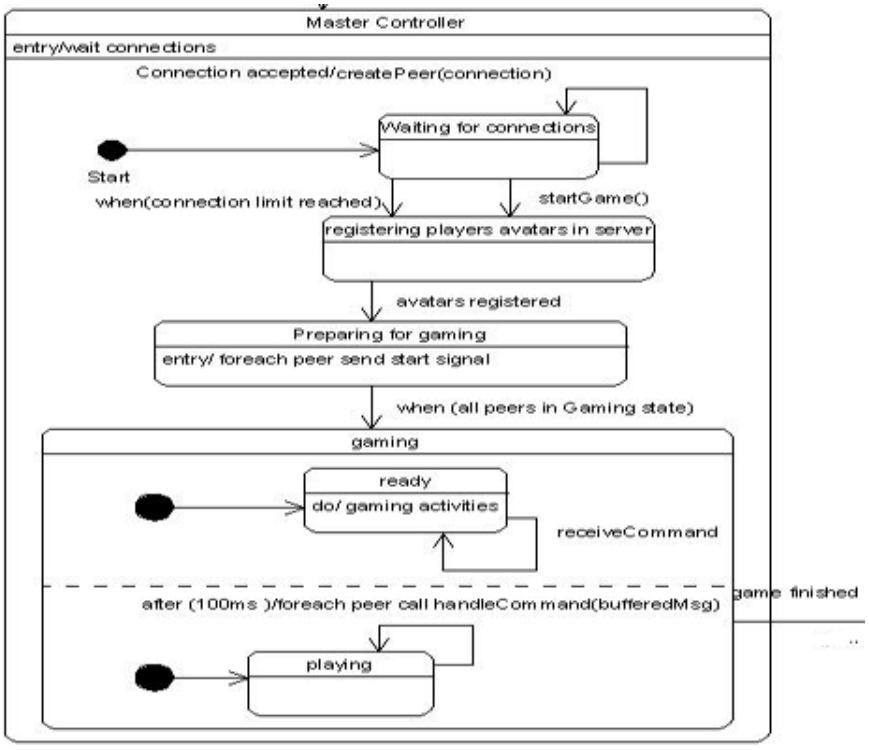

Figura 5.10. Decorrer do jogo do mestre. (Áreas B e C da Figura 5.2)

de acordo com a política em questão. Nesse ponto surgem duas observações importantes: o evento de recebimento de comando é o mesmo para comandos gerados pela interação do usuário com o dispositivio local e comandos recebidos pela rede; e o envio de mensagens não gera evento, por isso não está especificado explicitamente.

A Figura 5.11 mostra o fluxo de atividades de um jogo de turnos. Inicialmente, a sequência em que os participantes vão jogar é definida. A partir desse ponto, o controlador fica esperando os comandos. Se um comando, do jogador da vez, chegar, ele é salvo e, caso o controlador tiver papel de *master* da rede, ele faz um *broadcast* do comando para finalmente trocar o jogador da vez. Se o controlador tiver papel de *slave*, ele pula apenas o passo de *broadcast* e executa os outros normalmente. Se o comando recebido não for do jogador da vez, ele é descartado.

A Figura 5.12 mostra o fluxo de atividades de um jogo simultâneo. Para esse tipo de jogo, o controlador deve ser capaz de identificar quais participantes do Mini-mundo já enviaram seus comandos. Isso é feito através de um conjunto que contém os usuários que já enviaram seu comando. Ini-

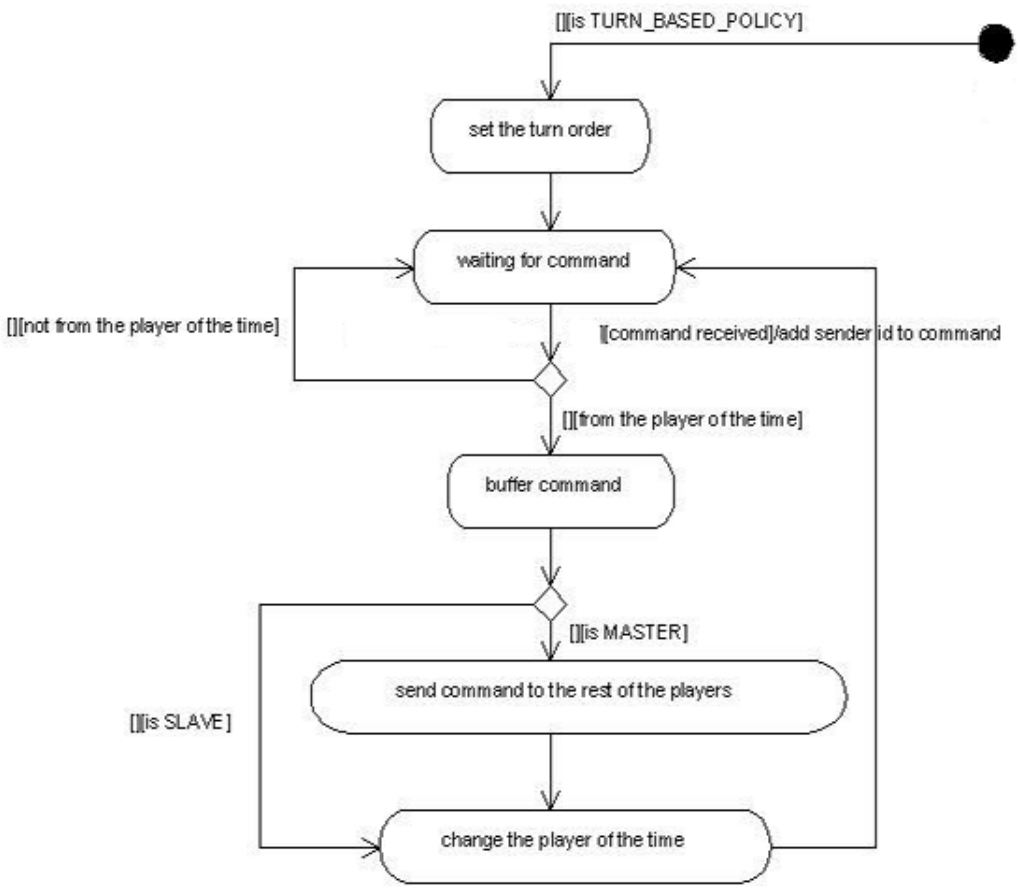

Figura 5.11. Fluxo de atividades de jogos de turno.

cialmente esse conjunto é esvaziado. Quando um comando, de alguém que ainda não está no conjunto, chega; ele é salvo, seu dono é adicionado ao conjunto e, caso o controlador for *master*, ele envia o comando a todos os outros participantes da rede e volta a esperar por comandos; caso for *slave* ele simplesmente volta a esperar por comandos. Quando todos os participantes enviarem seus comandos, eles serão executados e o fluxo reinicia.

A Figura 5.13 mostra o fluxo de atividades de um jogo de tempo real. Esse fluxo é o mais simples. Qualquer comando recebido é salvo. Caso o controlador seja *master*, ele além de salvar, envia um *broadcast* com o comando a todos os integrantes do Mini-mundo.

Finalização e Submição de resultados - Por fim, ao término da partida, os resultados devem ser processados e submetidos ao servidor. Essa submissão deve ser feita por dois participantes do Mini-mundo para evitar

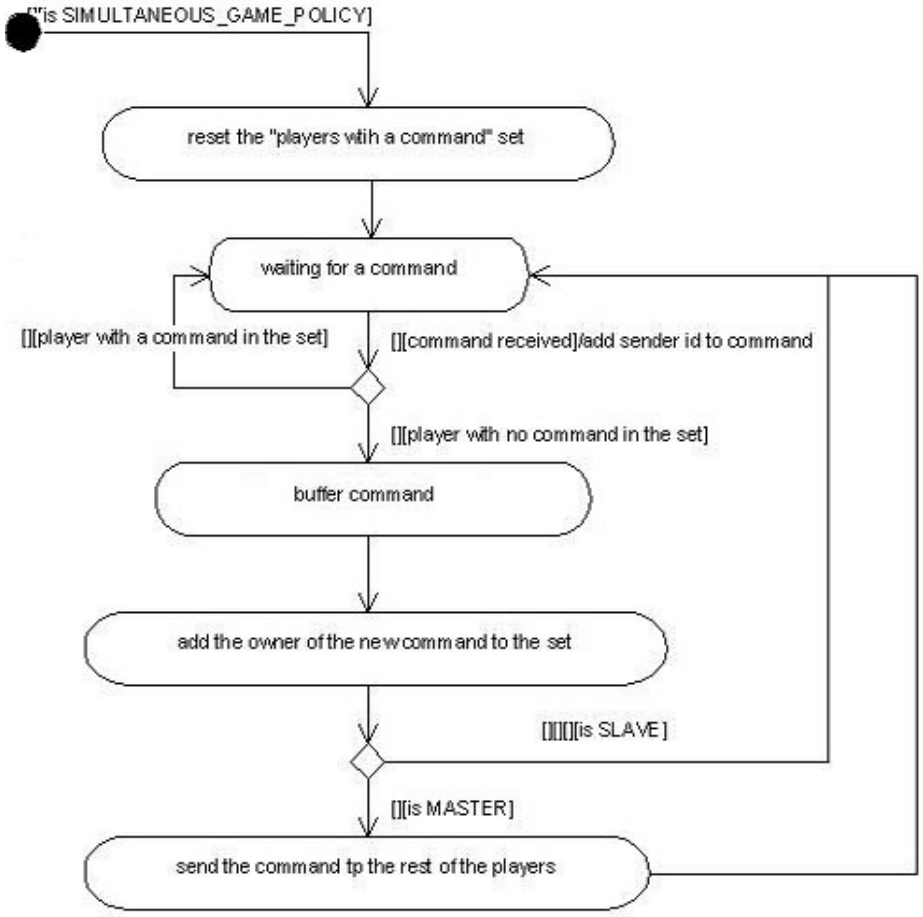

Figura 5.12. Fluxos de atividades de jogos simultâneos.

trapaça, visto que J2ME é facilmente sujeito a *patchs*[3]. Se o servidor receber os dois resultados consistentes, ele atualiza o estado de todos os personagens, que participaram do Mini-mundo em questão, encerrando todo o fluxo de um Mini-mundo. Qualquer participante da rede pode submeter esse resultado, visto que todos eles possuem o conhecimento dos identificadores do Mini-mundo e de cada participante, adquiridos durante o processo de sincronização e pelo sinal de *start* respectivamente; e do resultado propriamente dito.

A Figura 5.14 mostra os estados que contêm essas atividades. Inicialmente o controlador entra no estado *process result*, onde processa o resultado; e depois vai para o estado *waiting for result submition selection* no

[3]É uma maneira simples de prover atualizações no software quando seu código fonte está disponível

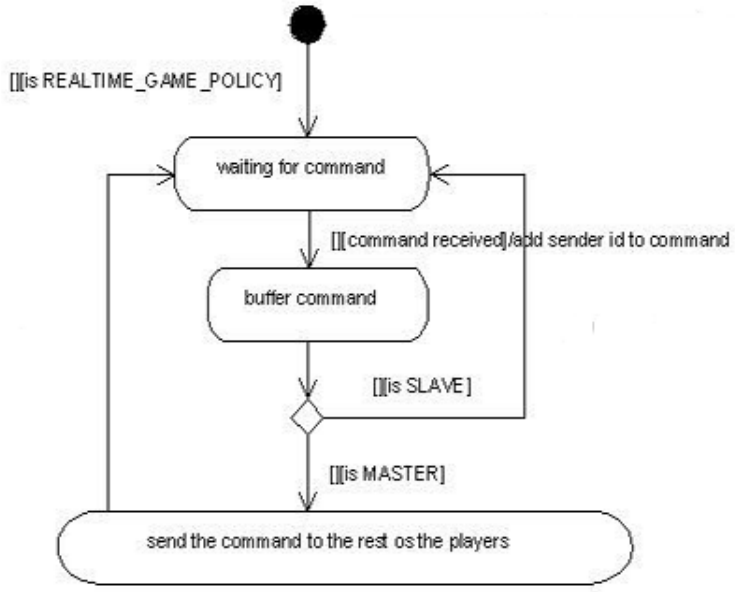

Figura 5.13. Fluxos de atividades de jogos de tempo real.

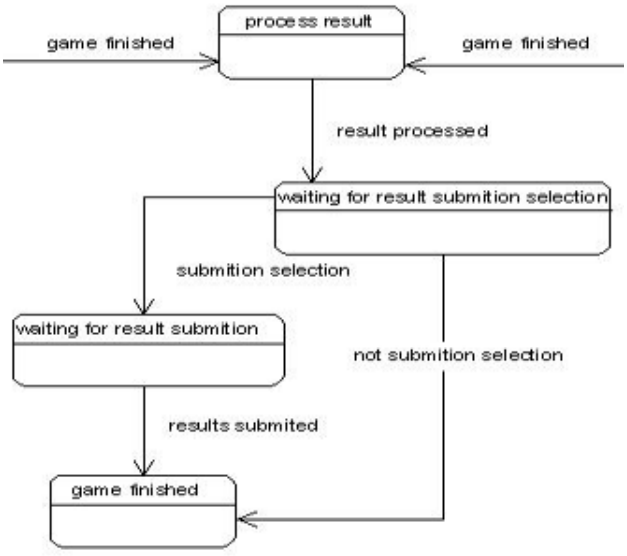

Figura 5.14. Subimissão de resultados. (Área F da Figura 5.2)

qual o jogador optará se fará a submissão. Em caso positivo, o controlador vai para o estado de submissão e segue para finalização do Mini-mundo; em

caso negativo, ele irá direto para finalização do Mini-mundo.

O servidor recebe os resultados e atualiza o estado dos personagens nos estados *waiting for game results* e *updating game state* respectivamete. A linha de concorrência 2, mostrada na Figura 5.4, demonstra tal dinâmica.

CAPÍTULO 6

CONCLUSÕES

Neste trabalho consideramos a arquitetura de uma plataforma para jogos multiusuário, em dispositivos móveis, cuja comunicação é realizada através de *Bluetooth* e cujos resultados são integrados a um jogo massivo.

Inicialmente, foi dada uma visão geral sobre as tecnologias *Bluetooth* e J2ME; e sobre os conceitos gerais de jogos Ubíquos Massivamente Multiusuário. O trabalho segue apresentando a idéia central do funcionamento da nossa arquitetura e as restrições sob as quais ela deve deve operar. Finalmente a modelagem estática e a modelagem dinâmica são apresentados.

No decorrer deste trabalho explanamos o funcionamento básico da API de *Bluetooth* para J2ME (JSR 82), mostrando as classes principais que a compõe e suas funcionalidades padrões. Essa explanação pode ser vista como um pequeno tutorial introdutório a tal JSR.

As grandes contribuições deste trabalho foram as modelagens estática e dinâmica da arquitetura que suporta o cenário em que jogadores competem entre si, em aparelhos celulares, através da tecnologia *Bluetooth* e enviam os resultados dessas competições a um servidor de um jogo massivo. Esta modelagem compreende desde a localização, conexão e sincronização dos telefones até a submissão de resultados ao servidor. Tal modelagem serve, principalmente como *guideline* para projetos desse tipo de jogo pois especifica todos os estados e as atividades necessárias em uma arquitetura simples. Além disso, esse projeto define o fluxo de atividades de 3 modelos chave de jogos: baseados em turno, simultâneos e de tempo real. Fluxos estes que podem ser tomados como referência para especificação de qualquer jogo.

6.1 TRABALHOS FUTUROS

Diversos são os trabalhos futuros que este trabalho pode dar origem; a grande parte deles está em especificar o comportamento de outros modelos de jogos integrados com um jogo MMUG, por exemplo um modelo similar ao nosso, mas que utilize SMS[1] para conectar dispositivos distintos ao invés de *Bluetooth*.

[1]Serviço de mensagens curtas ou Short message service (SMS) é um serviço disponível em telefones celulares (telemóveis) digitais que permite o envio de mensagens curtas entre estes equipamentos.

Indo um pouco mais além, este trabalho pode ser anexado ao projeto de uma plataforma para jogos massivos ubíquos completa, a qual modela todos os cenários de comunicação descritos no capítulo 4. Tal projeto seria pioneiro.

6.2 CONSIDERAÇÕES FINAIS

Neste capítulo apresentamos nossas contribuições e os possíveis trabalhos futuros. A originalidade dos jogos MMUG, principalmente seu cenário via *Bluetooth* é um ponto chave deste trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

[Blu04] Bluetooth.org. The official bluetooth membership site. disponível em http://www.bluetooth.org/spec/, 2004.

[DG99] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the internet, March 1999.

[ECJ03] B. Filstrup E. Cronin and S. Jamin. An Efficient Synchronization Mechanism for Mirrored Game Architectures. *International Conference on Application and Development of Computer Games*, 2003.

[FTF05] G. Ramalho F. Trinta and C. Ferraz. Serviços de Middleware para Jogos Massivos Ubíquos. *WJogos*, 2005.

[IEE05] IEEE. Ieee 802.11, the working group setting the standards for wireless lans. http://grouper.ieee.org/groups/802/11/, 2005.

[Muc04] J. Muchow. *Core J2ME*. Pearson Brasil, 2004.

[Nok06a] Nokia. The official nokia ngage site. disponível em http://www.ngage.com/, 2006.

[Nok06b] Forum Nokia. The official forum nokia site. disponível em http://www.forum.nokia.com/main.html, 2006.

[OMG03] OMG. Unified modeling language. Specification v1.5, Object Management Group, March 2003. http://www.omg.org/cgi-bin/doc?formal/03-03-01.

[PM04] I. Zanforlin P. Macêdo. Meantime Game Arquitecture. 2004.

[Pro05] Java Community Process. The java community process site. disponível em http://jcp.org/en/home/index, 2005.

[QUA05] Incorporated. QUALCOMM. Qualcomm brew home. disponível em http://brew.qualcomm.com/, 2005.

[Sun05a] Microsystems. Sun. Games on the java platform for mobile information. Whitepaper, Sun Developers Network. Disponível em http://java.sun.com/j2me/reference/whitepapers/, 2005.

[Sun05b] Microsystems. Sun. Java 2 platform, micro edition. disponível em http://java.sun.com/j2me/, 2005.

[Tri06] F. Trinta. *Jogos Ubíquos Massivamente Multi-usuário*. PhD thesis, Dept. de Informática, Univesidade Federal de Pernambuco, 2006.