



Universidade Federal de Pernambuco  
Centro de Informática

Pós-graduação em Ciência da Computação

**Definindo e Provendo Serviços de Suporte  
a Jogos Multiusuário e Multiplataforma:  
Rumo à Pervasividade**

Fernando Antonio Mota Trinta

TESE DE DOUTORADO

Recife  
3 de maio de 2007

Universidade Federal de Pernambuco  
Centro de Informática

Fernando Antonio Mota Trinta

**Definindo e Provendo Serviços de Suporte a Jogos  
Multiusuário e Multiplataforma: Rumo à Pervasividade**

*Trabalho apresentado ao Programa de Pós-graduação em  
Ciência da Computação do Centro de Informática da Uni-  
versidade Federal de Pernambuco como requisito parcial  
para obtenção do grau de Doutor em Ciência da Compu-  
tação.*

Orientador: *Carlos André Guimarães Ferraz*  
Co-orientador: *Geber Lisboa Ramalho*

Recife  
3 de maio de 2007

*À minha mãe*

# Agradecimentos

A realização de uma pesquisa de doutorado é um processo longo, extenuante, e por vezes, solitário. Particularmente, no meu caso, por vários momentos pensei em desistir. Porém, o suporte oferecido e incentivo de várias pessoas me deram força extra para a concretização deste trabalho. A estas pessoas, meu mais sincero agradecimento. Em particular, agradeço:

Aos meus orientadores, Carlos e Geber, pelo suporte, confiança e amizade oferecidos a mim ao longo dos anos de pesquisa... sem falar nas caronas para Boa Viagem. Ao professor Alex Sandro, pelo incentivo na reta final do trabalho.

A todos os funcionários do Centro de Informática da Universidade Federal de Pernambuco, por me mostrarem como uma instituição pública deveria funcionar no Brasil.

A todos os amigos feitos ao longo desta pesquisa, sejam estes do Radix, da Mobile S/A, C.E.S.A.R. (Projeto 3MOG) e do CIn, pela ajuda no trabalho e pelos momentos de descontração.

A Davi Pedrosa, pela ajuda na implementação de parte do protótipo de validação.

Ao meu sogro, Ivan, e sogra, Maria José, pelo suporte dado aos meus filhos e esposa quando da necessidade de me ausentar da presença deles, no intuito de adiantar ao máximo meus estudos.

Ao meu irmão João, que atuou como um segundo pai, fornecendo suporte para minha manutenção em Recife, bem como da minha família.

À minha mãe, pois sem seus incentivos, e as vezes sermões, jamais teria chegado ao final deste trabalho.

Mais que um agradecimento, cabe aqui um pedido de desculpas a três pessoas, pela ausência, mau-humor e angústia de um aluno de doutorado. Ivana, Ian e Ananda, obrigado por serem minha família. Amo voces.

*Não chores, meu filho;  
Não chores, que a vida  
É luta renhida: Viver é lutar.  
A vida é combate,  
Que os fracos abate,  
Que os fortes, os bravos  
Só pode exaltar.*

— GONÇALVES DIAS (Canção do Tamoio, 1857)

# Resumo

Ao longo dos últimos anos, jogos multiusuário distribuídos vêm se tornando um tópico em grande evidência tanto no meio acadêmico quanto industrial. Tradicionalmente, sua aplicação foi voltada para computadores pessoais de mesa. Recentemente, avanços na área de computação móvel impulsionaram a aplicação de jogos multiusuário também em dispositivos móveis. Um conceito ainda mais recente é o de jogos pervasivos ou ubíquos, onde o jogo é definido como uma aplicação sempre presente, disponível aos jogadores através de diferentes dispositivos e redes. A concepção destes jogos, bem como a necessidade de infra-estrutura de suporte à implementação e execução destas aplicações são questões que têm despertado grande interesse na comunidade mundial de jogos. O trabalho apresentado nesta tese contribui para solução de parte destas questões, particularmente em relação à adaptação da participação do jogador quando do uso de diferentes dispositivos. São apresentados cenários, onde o acesso ao jogo é irrestrito, criando situações onde jogadores utilizando diferentes dispositivos possam estar lado a lado dentro do mundo virtual, competindo ou cooperando de acordo com os objetivos do jogo. Ao mesmo tempo, estes jogos aproveitam situações específicas da computação móvel, como a formação de redes espontâneas, com objetivo de aumentar a imersão do usuário no jogo. Neste documento, estes jogos são denominados Jogos Multiusuário Multiplataforma Pervasivos (PM2G – do inglês, *Pervasive Multiplayer Multiplatform Games*). A partir de cenários definidos, são apresentados dois modelos. O primeiro, chamado de Modelo de Aplicação, diz respeito à visão de elementos que compõem o jogo PM2G, enquanto o segundo, chamado Modelo de Utilização, ressalta a forma como jogadores interagem. A partir destes cenários e modelos, foi especificada uma arquitetura baseada em uma camada de *middleware* que amplia os serviços de um jogo multiusuário tradicional, agregando serviços de suporte relativos aos conceitos específicos dos modelos apresentados. Para validação, foram implementados cenários de um típico jogo de representação de papéis (RPG – do inglês, *Role Playing Games*) chamado *Pervasive Wizards*, utilizando os serviços definidos.

**Palavras-chave:** Jogos Multiusuário Distribuídos, Jogos Pervasivos, Arquitetura de *Software*

# Abstract

Distributed Multiplayer Games have become widely popular in the last few years. Traditionally, these applications have been designed for personal computers connected over wired networks. Recently, advances in mobile computing and wireless communication have pushed multiplayer games into mobile handheld devices. An even more recent concept is called pervasive gaming. A pervasive game is a game that is always present, available to the player through different devices and networks. Pervasive Games rise many questions related to their software design and infra-structure support to their development and deployment. This thesis addresses part of these problems, specially those related with the playability adaptation due to device heterogeneity. Scenarios are presented, where players are almost always connected to the virtual world. Players using different devices can be side by side inside the virtual world, competing or cooperating, according their objectives. These games also take advantage from specific features from mobile computing, such as spontaneous (ad-hoc) networks. In this document, a game with such features is called Pervasive Multiplatform Multiplayer Game (PM2G). Based on these scenarios, two models have been defined: an Application Model and an Usage Model. The former creates a terminology for PM2G elements. The latter shows how users must interact with the game. Finally, a service oriented architecture based on a middleware layer is presented. This architecture adds new services to traditional multiplayer games middleware, offering support to concepts introduced by the PM2G scenarios and models. In order to validate this architecture, a simple Role Playing Game was prototyped, and its results are presented.

**Keywords:** Distributed Multiplayer Games, Pervasive Gaming, Software Architecture

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Tema	1
1.2	Motivação	3
1.2.1	Cenários de um jogo multiusuário futuro	4
1.3	Problema e Desafios	6
1.4	Solução	9
1.5	Estrutura do Documento	10
<b>2</b>	<b>Computação Pervasiva</b>	<b>11</b>
2.1	Introdução	11
2.2	Ambientes de Computação Pervasiva	13
2.3	Desafios para Computação Móvel e Pervasiva	14
2.3.1	Restrições de Recursos dos Dispositivos Móveis	14
2.3.2	Sensibilidade ao Contexto	15
2.3.2.1	Contexto	15
2.3.2.2	Classificações de Contexto	16
2.3.2.3	Rastreamento de Mobilidade Física	17
2.3.3	Adaptabilidade ao Contexto	18
2.3.3.1	Adaptação de Conteúdo	19
2.3.4	Escalabilidade	21
2.3.5	Descoberta de serviços	21
2.3.6	Segurança	22
2.4	Considerações sobre o Capítulo	22
<b>3</b>	<b>Suporte a Jogos Multiusuário Distribuídos</b>	<b>24</b>
3.1	Introdução	24
3.1.1	Estilos de Jogos Multiusuário	25
3.2	Cenários Atuais para Jogos Multiusuário	26
3.2.1	Jogos Multiusuário em Pequena Escala	27

3.2.2	Jogos Multiusuário Massivos (MMG)	27
3.2.3	Jogos Móveis Multiusuário	29
3.2.4	Jogos Pervasivos	31
3.2.5	Jogos Massivos Multiusuário Multiplataforma	32
3.3	Fatores limitantes para jogos multiusuário	33
3.3.1	Plataforma Física	34
3.3.2	Plataforma Lógica	35
3.3.3	Segurança e Trapaça	35
3.4	Soluções Clássicas para Suporte a Jogos Multiusuário	36
3.4.1	Distribuição de mensagens	37
3.4.2	Arquiteturas de Comunicação	38
3.5	Considerações sobre o Capítulo	40
<b>4</b>	<b>Trabalhos Relacionados</b>	<b>42</b>
4.1	Introdução	42
4.2	<i>Middleware</i> para Computação Pervasiva	42
4.2.1	<i>Middleware</i> Gaia	43
4.2.2	<i>Middleware</i> EXEHDA	44
4.2.3	MoCA	47
4.3	<i>Middleware</i> para Jogos Multiusuário	49
4.3.1	DirectPlay	49
4.3.2	TeraZona	50
4.3.3	Butterfly Grid	53
4.3.4	GASP	54
4.3.5	Plataforma Aberta 3MOG	56
4.4	<i>Middleware</i> para Jogos Pervasivos Multiplataforma Multiusuário	59
4.4.1	Proposta de David Fox	59
4.4.2	Grupos Focais realizados por Koivisto e Wenninger	60
4.4.3	<i>Middleware</i> sensível à situação aplicado a jogos	62
4.4.4	Iperg	64
4.5	Considerações Finais	66
<b>5</b>	<b>Modelos para Jogos Multiusuário Multiplataforma Pervasivos</b>	<b>68</b>
5.1	Introdução	68
5.2	Modelo de Jogo-Alvo PM2G	69
5.2.1	Jogos de Interpretação de Papéis (RPGs)	70

5.2.2	Análise de RPGs como temática para aplicações PM2G	71
5.3	Modelo de Aplicação PM2G	72
5.4	Modelo de Utilização	77
5.5	Considerações sobre o Capítulo	79
<b>6</b>	<b>Serviços de <i>Middleware</i> para Jogos Multiplataforma Pervasivos</b>	<b>80</b>
6.1	Introdução	80
6.2	Visão Geral da Arquitetura PM2G	82
6.3	<i>Framework</i> de Comunicação PM2G	84
6.3.1	Arquitetura	86
6.4	<i>Framework</i> de Simulação PM2G	87
6.5	Serviços de <i>Middleware</i> para Jogos PM2G	91
6.5.1	Serviço de Gerenciamento de Contexto	91
6.5.1.1	Modelagem	92
6.5.1.2	Arquitetura	95
6.5.2	Serviço de Adaptação de Conteúdo	98
6.5.2.1	Modelagem	99
6.5.2.2	Arquitetura Interna	102
6.5.3	Serviço de Adaptação de Interação	104
6.5.3.1	Modelagem	105
6.5.3.2	Arquitetura Interna	109
6.5.4	Serviço de Notificação de Mensagens	112
6.5.4.1	Modelagem	113
6.5.4.2	Arquitetura Interna	114
6.5.5	Serviço de Integração de Mini-mundos	117
6.5.5.1	Modelagem Estática de um Mini-Mundo	118
6.5.5.2	Modelagem Dinâmica – Interação Mini-Mundo e Serviço de Integração	118
6.5.5.3	Atualização de Avatares	121
6.5.5.4	Criação do Mini-Mundo e Tratamento de Conexões	122
6.5.5.5	Localização de Jogos e Sincronização	123
6.5.5.6	Início e Decorrer do Jogo	124
6.5.5.7	Finalização de Resultados	129
6.5.5.8	Modelagem Estática do Serviço	131
6.6	Considerações Finais	133

<b>7</b>	<b>Experimentos</b>	<b>134</b>
7.1	Introdução	134
7.2	Projeto do Jogo <i>Pervasive Wizards</i>	135
7.3	Prototipação	137
7.3.1	Definição das Entidades do Jogo	137
7.3.2	Representação da Área de Interesse	139
7.3.3	Implementação dos clientes de acesso ao Jogo	139
7.3.3.1	Clientes para computadores pessoais de mesa	140
7.3.3.2	Clientes para dispositivos móveis	142
7.3.3.3	Envio de informações de contexto	143
7.3.4	Configuração dos Serviços	144
7.3.4.1	Serviço de Adaptação de Conteúdo	144
7.3.4.2	Serviço de Adaptação de Interação	144
7.3.4.3	Serviço de Notificação de Mensagens	145
7.3.4.4	Serviço de Integração de Mini-Mundos	147
7.4	Descrição dos experimentos	148
7.4.1	Obtenção do estado do jogo	149
7.4.2	Movimentação do avatar	150
7.4.3	Ataque a um adversário	151
7.4.4	Coleta de item virtual	151
7.4.5	Simulação de Mini-Mundos	152
7.5	<i>Avaliação</i> do Middleware	153
<b>8</b>	<b>Conclusões</b>	<b>156</b>
8.1	Contribuições	156
8.2	Trabalhos Futuros	157

# Lista de Figuras

1.1	Cenários propostos para um Jogo Multiusuário Pervasivo.	4
2.1	Evolução de cenários para formação do conceito de computação Ubíqua [Aug04]	12
3.1	Exemplos de jogos (a) de Primeira Pessoa e (b) de Estratégia em tempo-real.	26
3.2	Crescimento do Número de Assinantes de MMGs atuais.	28
3.3	Plataformas de Jogos Móveis Multiusuário	29
3.4	O Jogo <i>Human PacMan</i> .	32
3.5	Técnicas para transmissão de mensagens. (a) <i>Broadcast</i> , (b) <i>Unicast</i> e (c) <i>Multicast</i> .	37
3.6	Arquiteturas de comunicação para jogos. (a) Ponto a Ponto, (b) Cliente/Servidor, (c) Replicação de servidores e (d) <i>Grid</i> de servidores.	38
4.1	Arquitetura <i>Gaia</i> [Rom03]	43
4.2	ISAM Pervasive Environment [Yam04]	45
4.3	Arquitetura de <i>Software</i> do Projeto ISAM [Yam04]	46
4.4	Arquitetura MoCA	47
4.5	Arquitetura DirectPlay	51
4.6	Arquitetura Terazona	52
4.7	Visão Simplificada da plataforma “Butterfly Grid”[Cec05]	53
4.8	Arquitetura GASP [PDGSS05]	56
4.9	Arquitetura 3MOG	57
4.10	Arquitetura da instância de um servidor 3MOG	58
4.11	Uso de um <i>middleware</i> sensível a situação no cenário de um jogo ubíquo (Adaptado de [HIW04]).	63
4.12	Arquitetura de Referência IPerg para uma Plataforma de Suporte a Jogos Pervasivos	65
5.1	Posicionamento de jogo PM2G em função de pervasividade, distribuição e heterogeneidade.	69

5.2	Divisão de Mercado de Jogos <i>Online</i> por Temática.	70
5.3	Modelo de Aplicação PM2G.	73
5.4	Mudança de Sessões em um PM2G.	78
6.1	Serviços de <i>Middleware</i> x Serviços de Aplicação	81
6.2	Arquitetura de Serviços PM2G	82
6.3	Fluxo de Comunicação Cliente/Servidor, adaptado de [BBV03]	85
6.4	Representação em UML de um Evento do Jogo – <i>GameEvent</i> .	86
6.5	Arquitetura PM2G – <i>Framework</i> de Comunicação.	87
6.6	Classes <i>GameObject</i> , <i>Performer</i> e <i>Actor</i> – Objetos do Jogos, Atuadores e Atores.	88
6.7	Classes <i>GameObjectFactory</i> e <i>GameObjectManager</i> – Gestores de Objetos	89
6.8	Área de Interesse e Gerenciador de áreas de Interesse.	90
6.9	Classe <i>SchedulerManager</i> – Escalonador da Simulação.	90
6.10	Classe <i>ServerSimulation</i> , responsável pelo controle da simulação servidora.	91
6.11	Modelo de Contexto PM2G	92
6.12	Notificação de mudança de contexto PM2G	93
6.13	Diagrama de Classes para informações de Contexto.	94
6.14	Diagrama de Classes – Serviço de Gerenciamento de Contexto	95
6.15	Diagrama de Classes – Classes relativas ao modelo publicador/assinante do Serviço de Gerenciamento de Contexto	96
6.16	Diagrama de Seqüência – Notificação de Eventos pelo Serviço de Gerenciamento de Contexto	97
6.17	Diagrama de Seqüência – Serviço de Adaptação de Conteúdo.	98
6.18	Diagrama de Classes – Estratégia de Adaptação.	100
6.19	Prioridades entre processamentos lógicos do Serviço de Adaptação de Conteúdo.	101
6.20	Diagrama de Classes – Arquitetura Interna do Serviço de Adaptação de Conteúdo.	103
6.21	Diagrama de Seqüência – Operação <i>adaptContent</i> .	104
6.22	Diagrama de Seqüência – Operação <i>adaptStrategy</i> .	105
6.23	Exemplo da relação entre estados e ações de um avatar.	106
6.24	Diagrama de Classes – Ação de um Jogador.	107
6.25	Exemplo de mapeamento entre estados relacionados a diferentes contextos.	108
6.26	Diagrama de Classes – Arquitetura Interna do Serviço de Adaptação de Interação	109
6.27	Diagrama de Seqüência – operação <i>getActions</i> .	110
6.28	Diagrama de Seqüência – operação <i>decodeAction</i> .	111
6.29	Diagrama de Seqüência – operação <i>getNewState</i> .	112

6.30	Diagrama de Classes – Evento PM2G e Criadores de Mensagens.	114
6.31	Diagrama de Classes – Emissores de Mensagens.	115
6.32	Diagrama de Classes – Arquitetura Interna do Serviço de Notificação de Mensagens	115
6.33	Configuração entre Eventos e <i>ObjectFactories</i> .	116
6.34	Criação e envio de uma mensagem a partir de um evento.	116
6.35	Diagrama de classes – Arquitetura de um Mini-Mundo PM2G.	118
6.36	Máquina de estados do controlador de um Mini-Mundo.	120
6.37	Máquina de Estados do Controlador de Mini-Mundo – atualização de avatares	121
6.38	Máquina de Estados do Serviço de Integração de Mini-Mundos – atualização de avatares.	122
6.39	Máquina de Estados do Controlador de Mini-Mundo – Processamento de pedidos de criação, registro e atualização de estado.	122
6.40	Máquina de Estados do Serviço de Integração de Mini-Mundos – Processamento de pedidos de criação, registro e atualização de estado.	124
6.41	Máquina de Estados do Controlador de Mini-Mundo – Localização, conexão e inicialização de escravos.	125
6.42	Máquina de Estados do Controlador de Mini-Mundo – Espera pelo início do jogos por parte dos controladores escravos.	125
6.43	Máquina de Estados do Controlador de Mini-Mundo – Início do jogos pelo controlador mestre.	126
6.44	Máquina de Estados do Controlador de Mini-Mundo – Decorrer do jogo/Sinalização de término (Controlador Mestre).	127
6.45	Fluxo de atividades de jogos de turno.	128
6.46	Fluxos de atividades de jogos simultâneos.	129
6.47	Fluxos de atividades de jogos de tempo-real.	130
6.48	Máquina de Estados do Controlador de Mini-Mundo – Decorrer do jogo/Sinalização de término (Controlador Escravo).	131
6.49	Máquina de Estados do Controlador de Mini-Mundo – Submissão de Resultados.	131
6.50	Submissão de resultados pelo controlador de um Mini-Mundo.	132
6.51	Diagrama de Classes – Arquitetura Interna do Serviço de Integração de Mini-Mundos	133
7.1	Exemplo do estado da <i>interface</i> de um jogador, durante uma sessão do jogo FreeWizards [Cec05]	136

7.2	Diagrama de Classes – Subclasses criadas para entidades do Jogo <i>Pervasive Wizards</i>	138
7.3	Diagrama de Classes – Aplicação-cliente para computadores pessoais.	141
7.4	Diagrama de Classes – Aplicação-cliente para dispositivos móveis.	143
7.5	Possíveis transições de estados em diferentes contextos de execução.	145
7.6	Diagrama de Classes dos Eventos concebidos para o jogo <i>Pervasive Wizards</i> .	145
7.7	Configuração do Serviço de Notificação de Eventos para o jogo <i>Pervasive Wizards</i>	147
7.8	Configuração entre Eventos e <i>ObjectFactories</i> para o jogo <i>Pervasive Wizards</i>	148
7.9	Diferentes visões do estado de uma região do jogo <i>Pervasive Wizards</i> .	150
7.10	Ação de ataque a um jogador no jogo <i>Pervasive Wizards</i> , utilizando um dispositivo móvel com interface gráfica	151
7.11	Coleta de um item no jogo <i>Pervasive Wizards</i> , utilizando um dispositivo móvel com interface textual	152
7.12	Amostra do acompanhamento da execução do serviço de integração de mini-mundos	153
7.13	Avaliação do Middleware PM2G: Quantidade de Classes e Linhas de Código	155

# Lista de Tabelas

1.1	Desafios para Jogos Multiusuário Multiplataforma Pervasivos.	7
4.1	Possíveis transformações de elementos de um jogo para diferentes plataformas (Adaptado de [Fox03])	60
5.1	Análise de jogos de representação de papéis, de acordo com os requisitos desejáveis para aplicações PM2G.	72
6.1	Repositório de Dispositivos.	94
6.2	Estados de um jogador	94
7.1	Relação entre elementos do Jogo <i>Pervasive Wizards</i> e o modelo de aplicação PM2G	137
7.2	Estrutura de Dados da Representação de uma área de interesse no Jogo <i>Pervasive Wizards</i>	140
7.3	Lista de Comandos disponíveis para um jogador, via cliente para computadores pessoais.	142
7.4	Comandos aplicação de Mini-Mundos	149
7.5	Avaliação do <i>Middleware</i>	154

# Abreviaturas

3MG	Massively Multiplayer Mobile Game
API	Appliaction Programming Interface
COM	Component Object Model
CSCW	Computer Supported Collaborative Work
FPS	First Person Shooter
GPRS	General Packet Radio Service
GPS	Global Positioning System
IPX	Internetwork Packet Exchange
J2ME	Java2 Micro Edition
JMS	Java Message Service
MGIF	Mobile Games Interoperability Forum
MIDP	Mobile Information Device Profile
MMG	Massively Multiplayer Game
NPC	Non-player Character
OMA	Open Mobile Alliance
P2P	Peer-to-Peer

PDA	Personal Digital Assistant
PM2G	Pervasive Multiplayer Multiplatform Game
RPG	Role Playing Game
RTS	Real-Time Strategy
SDK	Software Development Kit
SMS	Short Message Service
SOA	Service Oriented Architecture
UDP	User Datagram Protocol
UML	Unified Modelling Language
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network

# CAPÍTULO 1

## Introdução

*Um problema exposto com clareza já fica meio resolvido.*

— CHARLES F. KETTERING

### 1.1 Tema

Jogos Multiusuário Distribuídos vêm se tornando um tópico de pesquisa em grande evidência. Estas aplicações são essencialmente simulações interativas em tempo-real que permitem que uma pessoa chamada jogador, utilize um computador conectado à alguma rede para interagir, em um ambiente virtual, com outros jogadores [Cec05]. Em sua fase inicial, a aplicação de jogos multiusuário distribuídos era limitada a redes locais com poucos usuários, geralmente na ordem de dois a oito jogadores. Com o advento da Internet e melhoria da qualidade das conexões à rede mundial de computadores, jogos multiusuário distribuídos permitiram que jogadores geograficamente espalhados pudessem interagir em uma mesma partida. Segundo dados publicados pelo DFC Intelligence Institute (<http://www.dfcint.com/>), instituto de pesquisa especializado em mercado de entretenimento digital, o segmento de jogos *on-line* deverá atingir US\$ 13 bilhões ao final do ano de 2011.

O sucesso de jogos multiusuário fundamenta-se em certos fatores, sendo que dois pontos merecem destaque: (i) a competição entre humanos mostra-se mais atraente e desafiadora que as mais avançadas técnicas de Inteligência Artificial utilizadas em jogos monousuário [Cen01] e (ii) estes promovem aspectos sociais interessantes ao permitir a interação entre participantes, mesmo sem um conhecimento prévio entre os mesmos.

O mercado de jogos multiusuário tem passado por profundas transformações nestes últimos anos. A evolução das tecnologias de comunicação permitiu que estas aplicações evoluíssem de jogos com poucos usuários e curtas sessões de duração, para ambientes virtuais persistentes de grande escala com milhares de jogadores concorrentes, os chamados Jogos Massivamente Multiusuário, do inglês, *Massively Multiplayer Games (MMG)*. Estes jogos são geralmente disponibilizados por empresas especializadas como serviços, onde o acesso ao jogo por jogadores

assinantes é garantido mediante um pagamento periódico.

Tradicionalmente, jogos multiusuário têm sido projetados para computadores pessoais sobre redes fixas. Com a evolução da computação móvel, um novo cenário tem sido proposto. Avanços nas tecnologias de comunicação sem fio e o surgimento de plataformas abertas de desenvolvimento, como Java2 Micro Edition (J2ME) [SM05], possibilitaram a aplicação de jogos multiusuário distribuídos também em dispositivos móveis, como telefones celulares e *Personal Digital Assistants (PDA)*, criando um campo promissor para estas aplicações. Dispositivos móveis contam com um número crescente de tecnologias de comunicação sem fio, como bluetooth [Blu04], Wi-Fi (802.11) [IEE05] e *General Packet Radio Service (GPRS)* [GL00], dentre outras, permitindo conexão tanto entre si quanto via servidores na Internet. Estes jogos permitem que jogadores possam jogar, mesmo enquanto estejam se deslocando.

O uso de dispositivos móveis permite também que informações do mundo real do jogador, em especial sua localização física, possam ser utilizadas como elementos integrantes da dinâmica do jogo. Em sua maioria, jogos baseados em localização permitem que milhares de jogadores interajam em um mundo virtual persistente utilizando dispositivos móveis, principalmente telefones celulares, onde a localização física do jogador é o principal elemento da jogabilidade<sup>1</sup> do jogo. Devido a também apresentarem suporte a um alto número de jogadores, estes jogos são também comumente chamados de Jogos Móveis Massivamente Multiusuário, do inglês, *Massively Multiplayer Mobile Game (3MG)*.

Na realidade, as características advindas da computação móvel têm criado um cenário ainda mais inovador, o de jogos pervasivos ou jogos ubíquos. Estes jogos tiram proveito da visão de Computação Pervasiva ou Ubíqua introduzida por Mark Weiser [WGB99]. Este novo paradigma computacional procura promover uma computação melhor integrada ao cotidiano de seus usuários. Sistemas de computação pervasiva propõem o acesso constante do usuário às suas aplicações e dados via diferentes dispositivos e diferentes redes de comunicação sem fio, onde dados contextuais são monitorados pelas aplicações ou por sistemas de suporte, que modificam seu comportamento em função de mudanças relevantes nestas informações, para melhor atender as requisições do usuário. Baseado nestas características, um jogo pervasivo é definido como “um jogo que está sempre presente, disponível aos seus usuários. Os jogadores utilizam diferentes dispositivos e diferentes redes de acesso. O jogo utiliza informações contextuais para aumentar a experiência do usuário em relação à participação no jogo” [BHLM02]. Esta tese propõe o suporte ao desenvolvimento e execução deste tipo de jogo.

---

<sup>1</sup>Jogabilidade relaciona-se com a atratividade do jogo para o jogador e o incentivo que o mesmo tem de permanecer jogando, ou voltar a jogar após ter finalizado uma partida.

## 1.2 Motivação

Baseado nas idéias de outros trabalhos na área [KW05] [Wal04] e na experiência adquirida, os cenários apresentados na seção anterior indicam uma tendência para jogos multiusuário futuros, onde estas aplicações serão ambientes virtuais persistentes, acessados por um grande número de jogadores, através de diferentes dispositivos que variam em características de comunicação e processamento, e que levam também em conta informações contextuais. Estes jogos unirão características de todos os cenários anteriores, oferecendo um ambiente altamente inovador e interativo para seus usuários. O acesso a estes jogos será irrestrito, criando situações onde jogadores utilizando diferentes dispositivos possam estar lado a lado no mundo virtual, competindo ou cooperando, de acordo com seus objetivos.

Nesta visão proposta, a união entre jogos multiusuário e dispositivos móveis traria uma série de benefícios. Como mundos virtuais persistentes, estes jogos mantêm-se em evolução contínua. Eventos importantes como a devastação de uma cidade aliada ou ataques ao clã de um jogador podem acontecer a qualquer instante. O uso de serviços específicos da computação móvel, como mensagens *Short Message Service (SMS)*, permitiria, por exemplo, que o jogador pudesse ser avisado quando ocorressem certos eventos no jogo. De acordo com o evento existente, o uso de dispositivos móveis permitiria ainda que o jogador pudesse interagir com o jogo, realizando alguma contramedida ao evento ocorrido. Isto poderia ser feito enquanto o usuário se desloca, através de seu próprio telefone celular.

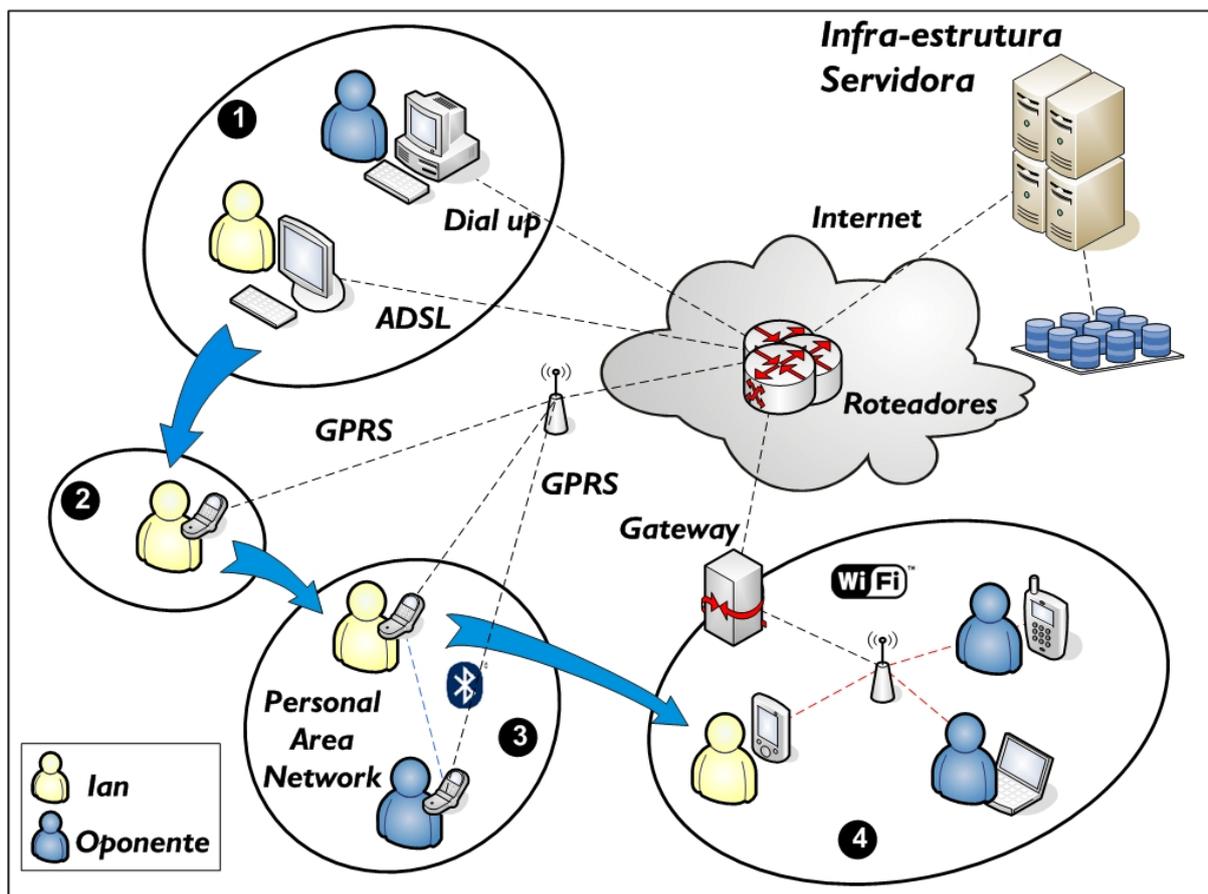
O uso de computadores pessoais de mesa permite que usuários usufruam de uma *interface* gráfica rica em detalhes multimídia e facilidades para interação com o jogo, usando de diferentes recursos de entrada, como apontadores (*mouse*) ou *joysticks*, recursos estes escassos ou inexistentes para dispositivos móveis em geral. Idealmente, o leque de ações possíveis para um jogador deveria ser independente do dispositivo ou do meio de comunicação utilizado, permitindo interações entre jogadores utilizando tanto computadores pessoais como dispositivos móveis, sem prejuízo de nenhuma das partes. Porém, partindo do princípio que os dispositivos são diferentes em termos de interfaces de apresentação, usabilidade, características de comunicação e capacidade de processamento, haveria também a necessidade que as interações e as formas de apresentação do jogo fossem adaptadas às características do acesso do jogador em um dado instante.

O uso de dispositivos móveis permitiria ainda a introdução de novas formas de interação entre jogadores, utilizando oportunidades e serviços específicos da computação móvel. A formação de redes espontâneas (*ad-hoc*) permitiria que jogadores se engajassem em batalhas interativas de tempo-real, cujos resultados como pontos ou bonificações pudessem ser integrados

no perfil do jogador, e utilizados quando de seu acesso via outros dispositivos. O uso de serviços baseados em localização facilitaria a formação destas redes, permitindo que jogadores pudessem localizar adversários nas proximidades de sua posição física. Para melhor ilustrar esta idéia, a subseção seguinte apresenta cenários típicos de um jogo multiusuário futuro, segundo visão defendida neste trabalho.

### 1.2.1 Cenários de um jogo multiusuário futuro

Tome-se como exemplo, um jogador chamado Ian, que participa de um jogo multiusuário distribuído, onde a infra-estrutura do jogo é formada por um conjunto de servidores acessados via Internet, como na Figura 1.1.



**Figura 1.1** Cenários propostos para um Jogo Multiusuário Pervasivo.

Neste jogo, o principal objetivo de um jogador é a evolução de seu personagem em diferentes critérios, como inteligência, agilidade, experiência ou riquezas materiais. Quando Ian

encontra-se em sua casa (cenário um), ele joga através de seu computador pessoal que possui acesso banda larga à Internet, com vastos recursos multimídia, como um monitor de alta definição e caixas de som amplificadas, permitindo que ele possa participar de batalhas em tempo-real altamente interativas com outros jogadores.

Porém, em certos intervalos de seu dia, Ian não possui estas mesmas condições de acesso ao jogo. Por exemplo, ele pode necessitar se deslocar para uma consulta médica. Neste ínterim, ele gostaria de receber informações sobre como está o mundo virtual do jogo em que participa. Para isto, ele pode utilizar seu telefone celular para acessar o jogo via rede de sua operadora de telefonia, de modo a obter informações sobre seu personagem (cenário dois). Porém, Ian não deseja apenas receber informações, mas também realizar ações que o possibilitem evoluir dentro do jogo. Por sua vez, tais ações devem ser mais gerenciais do que o efetivo controle do personagem do jogador, como fazer com que este assuma uma posição defensiva em determinado lugar do mundo virtual. Este modo de interação adequa-se ao contexto (dispositivo e conectividade) que o jogador possui naquele momento. Esta possibilidade permite que o jogador diminua a frequência da comunicação com o mundo virtual, bem como a necessidade de respostas rápidas, típicas das batalhas realizadas quando do acesso via computadores pessoais e redes fixas tradicionais.

Ao chegar ao consultório de seu médico (cenário três), Ian utiliza seu celular para descobrir se existem outros jogadores próximos que estejam disponíveis para jogar com ele. Como Ian possui um celular habilitado para comunicação bluetooth, ele busca e encontra outro jogador também com um celular habilitado com bluetooth, permitindo que os dois possam formar uma *Wireless Personal Area Network (WPAN)* para disputar uma batalha interativa, enquanto esperam seu atendimento. O vencedor receberá bonificações para atributos de seu personagem, que serão transmitidas pela rede da operadora de telefonia móvel de cada jogador e serão incorporadas aos atributos de seus personagens. Estes atributos poderão também ser utilizados quando de seu acesso através de outros dispositivos.

Após sair de sua consulta, Ian vai a um *shopping center*, que disponibiliza acesso à Internet através de um ponto de acesso Wi-Fi. Ian possui um PDA habilitado com uma interface de comunicação, que lhe permite acesso à rede, e conseqüentemente ao jogo (cenário quatro). Através desta conexão, ele pode realizar tarefas mais interativas que via a rede da operadora de seu celular. Mais que isso, o *shopping center* pode hospedar um servidor onde parte do mundo virtual seja simulado, permitindo aos jogadores realizar partidas entre aqueles que utilizem tal rede. Novamente, os resultados obtidos pelos jogadores nestas partidas terão efeito sobre o mundo virtual habitado por outros jogadores que não participaram destas batalhas.

### 1.3 Problema e Desafios

Os cenários descritos na seção anterior representam neste trabalho um jogo “onipresente”, chamado então de Jogo Multiusuário Multiplataforma Pervasivo, do inglês, *Pervasive Multiplayer Multiplatform Game (PM2G)*. De forma sucinta, um PM2G é um jogo multiusuário distribuído que pode ser jogado de diferentes formas por um mesmo jogador, utilizando diferentes dispositivos como telefones celulares, PDAs, *laptops* ou computadores pessoais. O jogador interage de acordo com a infra-estrutura disponível em um dado instante e local, preservando sua sessão enquanto o mesmo migra entre dispositivos. PM2Gs aproveitam ainda situações e serviços específicos do cenário móvel para permitir interações mais intensas entre jogadores móveis, como a formação de redes espontâneas através de diferentes tecnologias de comunicação. Neste jogo, sua principal motivação é possibilitar a idéia de que, seguindo a definição de um jogo pervasivo, o jogador sempre tenha a possibilidade de participar do jogo, a qualquer hora, em quase todo lugar, com qualquer dispositivo [Fox03].

Esta visão vai ao encontro de uma tendência apontada por especialistas, onde no futuro próximo, o suporte à mobilidade será um fator crucial para o sucesso de um jogo [Wal04]. Esta tendência começa a se confirmar com o lançamento de jogos como *HinterWars*<sup>2</sup>, que pode ser jogado tanto no computador pessoal quanto nos celulares Nokia N-Gage™.

A realização de jogos com estas características apresenta uma série de desafios a serem resolvidos. Estes desafios vão desde problemas técnicos, como o suporte a um modelo de integração entre diferentes tecnologias, até questões não-técnicas, como a elaboração de um projeto de jogabilidade adequado a estes jogos. Estes desafios englobam problemas relativos às diversas áreas relacionadas com a efetiva concepção de um jogo PM2G. Tais desafios podem então ser categorizados, segundo a Tabela 1.1.

Em relação à computação pervasiva, o suporte a plataformas heterogêneas (computadores pessoais, PDAs e telefones celulares) gera o problema de interoperabilidade para PM2Gs, onde estes precisam oferecer suporte adequado a diferentes contextos de execução de seus usuários, de acordo com a plataforma utilizada. Contexto para uma aplicação pervasiva representa qualquer informação relevante à aplicação [JS04], como dados técnicos de um dispositivo ou informações do próprio usuário. Monitorar este contexto, fazendo com que o jogo adapte-se às mudanças decorrentes da troca de dispositivo ou mesmo às preferências pessoais do jogador constitui uma questão a ser resolvida em um jogo PM2G.

O uso de aplicações mais sofisticadas em dispositivos móveis traz consigo o problema da autonomia do tempo da bateria destes dispositivos. O processamento da lógica do jogo, e

---

<sup>2</sup>(<http://www.hinterwars.com/>)

**Tabela 1.1** Desafios para Jogos Multiusuário Multiplataforma Pervasivos.

Categoria	Problema	Sub-problema
Computação Pervasiva	Uso de Dispositivos Móveis	Consumo de Energia em dispositivos baseados em bateria
		Baixo poder de processamento*
		Intermitência das tecnologias de comunicação sem fio
		Limitação da interface homem-máquina*
	Heterogeneidade de Plataformas	Gerência de Informações Contextuais*
		Adaptação de Conteúdo em diferentes contextos de execução*
Sistemas Distribuídos	Escalabilidade em face ao alto número de jogadores	Custos Operacionais de uma infra-estrutura servidora de suporte
Jogos Multiusuário	Projeto do Jogo ( <i>Game Design</i> )	Competição Justa entre jogadores ( <i>Fairness</i> )
		Diferenças comportamentais entre jogadores via dispositivos móveis e computadores pessoais*
	Combate ao uso de trapaça por jogadores mal-intencionados	

\*Questões estudadas neste trabalho.

principalmente a comunicação em rede, provocam um maior consumo da bateria do dispositivo. PM2Gs devem procurar formas de atenuar este consumo.

Ainda em relação à computação pervasiva, apesar de uma sensível melhoria nos últimos anos, dispositivos móveis ainda apresentam limitações de processamento, usabilidade e de conectividade que precisam ser levadas em conta para o efetivo projeto de um jogo PM2G. Como sistemas distribuídos, o alto número de usuários levanta questões em relações à escalabilidade e custos de uma infra-estrutura de suporte necessária a um jogo PM2G.

Já em relação a jogos multiusuário, o grande problema a ser enfrentado diz respeito ao projeto de um jogo que seja viável, interessante e estimulante para o usuário, utilizando as características pretendidas para um PM2G. Este projeto deve considerar questões relativas à uma competição justa (*fairness*) entre todos os jogadores. Jogos multiusuário tradicionais reforçam a idéia que todo jogador deve ter a mesma chance de sair vencedor em uma partida, independente de qualidade de sua conexão ou do computador utilizado. Em um PM2G, isto se torna crítico. Dispositivos móveis e computadores pessoais apresentam características discrepantes

de interface homem-máquina e de propriedades de conexão. Visto que na visão proposta é possível interação entre jogadores usando diferentes dispositivos, mecanismos de nivelamento devem ser projetados no intuito de garantir interações justas entre jogadores em diferentes contextos.

O projeto de um jogo PM2G deve também levar em consideração o próprio comportamento dos jogadores quando do uso de diferentes dispositivos. Jogadores em dispositivos móveis tem um comportamento mais esporádico ou casual, com curtas sessões de uso de suas aplicações, enquanto o acesso via computadores pessoais tende a ser mais duradouro. Agrega-se a estes problemas também o combate a jogadores trapaceiros, que procuram utilizar meios antiéticos para conseguir vantagens em relação aos demais jogadores.

Cada um destes desafios representa um problema fundamental para a visão de PM2Gs. Porém, não é objetivo deste trabalho atacar todos estes problemas diretamente, uma vez que muitos desses desafios estão sendo tratados individualmente por diversos grupos de pesquisa, cujos resultados podem ser aproveitados neste trabalho. Apesar da importância de cada um dos itens citados, o foco deste trabalho relaciona-se com aqueles destacados(\*) na Tabela 1.1, que estão mais diretamente relacionadas à adaptação da jogabilidade ao se utilizar diferentes dispositivos. Este trabalho apresenta soluções para integração de dispositivos diversos com jogo multiusuário, permitindo acesso ao mundo virtual de forma coerente apesar da grande diferença entre os contextos dos dispositivos usados (computador pessoal e dispositivos móveis) e incorporação de cenários específicos da computação móvel.

As idéias propostas por Mark Weiser prometem causar um profundo impacto no modo como a computação é percebida pelos usuários. Uma idéia particular está mais diretamente ligada a este trabalho. O acesso em todo momento e todo lugar tem uma grande valia para jogos futuros, dada a execução contínua destas aplicações. Porém, para utilização dos conceitos de computação pervasiva em jogos multiusuário futuros, algumas ressalvas são feitas. As atuais aplicações de computação pervasiva utilizam a idéia de espaços inteligentes, com infra-estrutura apropriada através de sensores e serviços aos usuários do ambiente. Dados os cenários apresentados, a limitação do conceito de espaços inteligentes, porém restritos a uma região física específica (prédio ou salas) não se adequa à proposta PM2G. Como apresentado nos cenários descritos (Seção 1.2.1), usuários devem ter acessos em diferentes locais de acordo com seu cotidiano.

Nesta proposta, a infra-estrutura utilizada para incorporação de pervasividade é aquela já presente e disponível, por exemplo, por operadoras de telefonia ou prestadores de serviço que oferecem acesso à Internet em locais públicos. Sendo assim, restringem-se as possibilidades de criação de um ambiente de computação pervasiva com todas as possibilidades que a mesma

oferece. As características pervasivas limitam-se a (i) permitir que os usuários e dispositivos possam ter acesso à informação, de (quase) qualquer local, aproveitando a oportunidade de utilizar diferentes dispositivos e suas possibilidades de comunicação e (ii) adequar a apresentação do conteúdo do jogo de acordo com o contexto do jogador.

## 1.4 Solução

A concepção de um *Pervasive Multiplayer Multiplatform Game (PM2G)* envolve uma série de desafios, o que a princípio, exigirá um grande esforço para desenvolvedores de tais aplicações. Desta forma, faz-se necessário o surgimento de infra-estruturas como *frameworks* ou plataformas de *middleware*, que amenizem o esforço requerido para a implementação destes jogos, reduzindo seus custos e tempo de desenvolvimento.

Existem várias definições para *middleware* [Ber96] [Emm00] [CDK05b] que convergem para a idéia de uma camada de *software* que facilite o desenvolvimento de aplicações distribuídas, abstraindo do desenvolvedor problemas relativos à distribuição de seus componentes. Para alcançar este objetivo, sistemas de *middleware* oferecem um modelo de execução composto por serviços que realizam tarefas comuns em um sistema distribuído, como localização de componentes, distribuição de eventos, dentre outros; e um modelo de programação na forma de uma *Application Programming Interface (API)*, para utilização dos desenvolvedores.

Após uma fase de consolidação, plataformas de *middleware* têm se especializado para atender requisitos específicos de certos domínios de aplicação, como computação em tempo-real [Sch02], computação móvel [Cap03] ou computação pervasiva [Yam04]. Plataformas de *middleware* também têm sido utilizadas para facilitar o desenvolvimento de jogos multiusuário. Exemplos destas plataformas são o TeraZona [Ter02] e o Microsoft DirectPlay [Mic02]. Estas plataformas facilitam a manutenção de estado compartilhado e comunicação entre jogadores. Sistemas de *middleware* específicos para jogos massivos incluem ainda serviços para suporte a tarifação, persistência e balanceamento de carga entre vários servidores.

Neste contexto, este trabalho mostra uma arquitetura [TFR06b] [TPFR07] [TFR06a] [TFR05] projetada para oferecer o suporte a PM2Gs através da introdução de uma camada de serviços especializados na forma de um *middleware*, seguindo os cenários descritos previamente (Seção 1.2.1). A partir destes cenários, foram definidos dois modelos, um de aplicação e outro de utilização, que determinam, respectivamente, os elementos que compõem um PM2G e como são realizadas as interações do jogador com o jogo. Estes modelos ajudam a definir os requisitos necessários para a camada de serviços responsável pelo suporte aos conceitos então

definidos. Cinco serviços são definidos: o serviço de *Gerenciamento de Contexto* gerencia as informações contextuais sobre o acesso de cada jogador, como dispositivo utilizado e preferências pessoais de cada usuário. Estas informações têm papel central na arquitetura, modificando o comportamento dos demais serviços presentes na mesma. Os serviços de *Adaptação de Conteúdo* e *Adaptação de Interação* atendem os requisitos de adaptação da jogabilidade de um jogador, facilitando a forma como o usuário acessa e interage com o mundo virtual [PTFR06]. O serviço de *Notificação de Mensagens* permite que o jogador possa ser notificado através de alertas sobre diferentes eventos de interesse do jogador que ocorrem no jogo. O serviço de *Integração de Mini-Mundos* permite que jogadores utilizando dispositivos móveis possam interagir diretamente em jogos paralelos e cujos resultados são integrados ao perfil do jogador.

## 1.5 Estrutura do Documento

O corpo deste trabalho é dividido em oito capítulos. Após esta introdução, o Capítulo 2 apresenta as principais questões relacionadas à área de computação móvel e pervasiva. O Capítulo 3 apresenta os conceitos e principais problemas técnicos que influenciam jogos multiusuário, bem como as principais soluções para estas questões. No Capítulo 4 são apresentadas as principais pesquisas relacionadas com a proposta PM2G, encontrados no levantamento bibliográfico realizado durante o desenvolvimento desta pesquisa. No Capítulo 5 são apresentados os modelos de aplicação e utilização PM2G, bem como é feita uma reflexão sobre o estilo de jogo que melhor se adequa aos cenários propostos. No Capítulo 6 apresenta-se a arquitetura de serviços, bem como outros elementos necessários para o suporte às aplicações PM2G. Os experimentos realizados para caracterizar a viabilidade do modelo PM2G são apresentados no Capítulo 7. Por fim, as conclusões e os trabalhos futuros são discutidos no Capítulo 8.

# Computação Pervasiva

*As tecnologias mais profundas são aquelas que desaparecem, dissipam-se nas coisas do cotidiano e em pouco tempo confundem-se com elas.*

— MARK WEISER (Xerox Parc)

As pesquisas na área de computação móvel ainda estão em fase de amadurecimento. Devido a este estágio inicial, a literatura apresenta diferentes visões dos conceitos de computação móvel, computação pervasiva e computação ubíqua. Este capítulo procura apresentar a visão destes conceitos de acordo com a literatura pesquisada e os principais desafios enfrentados por cada um deles. É dada maior ênfase àqueles que estão mais relacionados com o tema deste trabalho.

## 2.1 Introdução

Computação Móvel, Computação Pervasiva e Computação Ubíqua são paradigmas computacionais em crescente expansão, motivados principalmente pela miniaturização de componentes de *hardware* e pelo amadurecimento de tecnologias de comunicação sem fio. A computação móvel possibilita que usuários utilizem dispositivos portáteis, como *notebooks*, PDAs e telefones celulares, enquanto se deslocam entre diferentes pontos, mantendo um nível de conectividade com aplicações remotas.

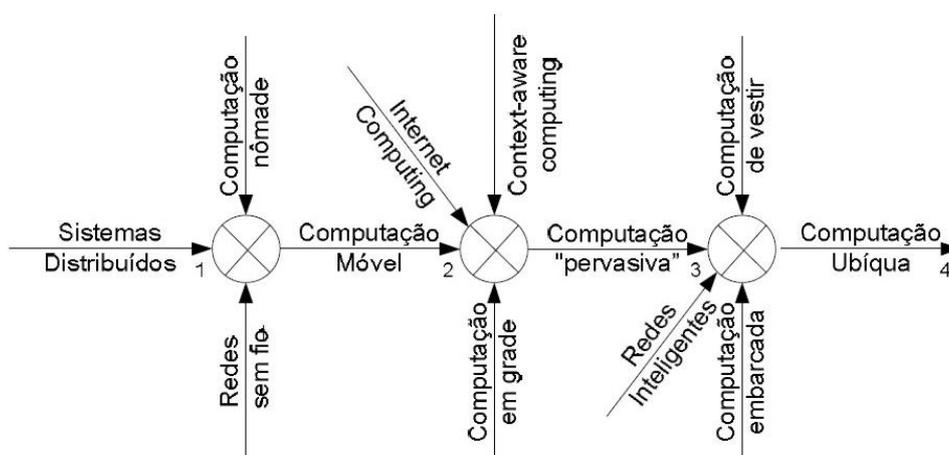
Computação Ubíqua (*Ubiquitous Computing*, de onipresente, existente ou estando em todo lugar ao mesmo tempo, constantemente encontrado<sup>1</sup>) representa uma proposta iniciada pelas pesquisas de Mark Weiser em 1988 [WGB99], que procura ampliar a utilização de dispositivos portáteis para promover uma computação melhor integrada à vida do usuário. Neste sentido também é utilizado o termo Computação Pervasiva<sup>2</sup> – *Pervasive Computing* (computação espalhada), promovido pela IBM [IBM99], embora não possuam exatamente o mesmo significado. Ambas são consideradas evoluções futuras do atual momento da computação móvel.

---

<sup>1</sup>Dicionário Michaelis.

<sup>2</sup>*Pervasive* significa difundido inteiramente por toda parte. Embora não exista na língua portuguesa o termo *pervasivo*, boa parte das pesquisas no Brasil utiliza este termo como uma tradução do termo *Pervasive*.

Não há hoje um consenso sobre quais seriam as diferenças entre Computação Ubíqua e Computação Pervasiva. Certos autores tratam os dois termos de forma unívoca [Sat01], enquanto outros apontam a visão de computação pervasiva como um momento mais atual em virtude das tecnologias necessárias para a materialização da visão de Weiser [Aug04] [Yam04]. Nesta segunda visão, a associação da Computação Pervasiva a outras tecnologias como a computação de vestir (*Wearable computing*), redes inteligentes e computação embarcada (*Embedded Computing*), levará à formação da Computação Ubíqua, como representado pela Figura 2.1. Neste trabalho, por concordar com esta segunda visão, será utilizado o termo “pervasivo” para caracterizar o cenário do jogo pretendido neste trabalho.



**Figura 2.1** Evolução de cenários para formação do conceito de computação Ubíqua [Aug04]

Pela visão de Weiser, através da integração de dispositivos computacionais em elementos do cotidiano dos usuários, um grande número de computadores imperceptíveis passará a existir em ambientes físicos. Estes computadores farão atividades automáticas em prol dos usuários, interagindo entre si e com outros computadores para alcançar seus objetivos. Cria-se uma nova relação entre usuários e dispositivos computacionais, onde cada usuário possui vários computadores. Esta visão contrasta com a era dos computadores pessoais, onde quase todo usuário tinha apenas um computador.

Ressalta-se que esta relação é diferente da situação comum atualmente, onde um usuário tem vários computadores semelhantes, um em casa, outro no trabalho ou um computador portátil. Na Computação Ubíqua, dispositivos multiplicam-se em forma e funcionalidade, não apenas em números, adequando-se a diferentes tarefas [CDK05a]. De certa forma, vê-se a concretização desta afirmação como uma consequência da aglutinação de funções que dispositivos portáteis estão sofrendo nos últimos anos, como a tendência de integração entre PDAs, telefones celulares e câmeras digitais.

## 2.2 Ambientes de Computação Pervasiva

Um dos principais objetivos da Computação Pervasiva é fornecer uma visão onde o poder computacional desejado pelos usuários estará disponível em qualquer lugar, a qualquer momento [SM03]. O usuário herda da computação móvel a liberdade para se deslocar enquanto mantém o acesso aos recursos (aplicações, dados, dispositivos) via dispositivos portáteis e redes de comunicação sem fio. Neste momento, este cenário em escala global ainda é hipotético, tanto em termos de infra-estrutura física (*hardware* e rede), quanto de *software* (*middleware* de suporte). As experiências realizadas restringem-se à espaços inteligentes (*smart spaces*), auto contidos em escopos físicos finitos e bem definidos, como prédios ou salas especiais.

Um ambiente de computação pervasiva deve ser formado por um espaço físico finito que disponibilize serviços embutidos, providos somente ou preferencialmente dentro deste espaço físico [CDK05a]. Estes espaços inteligentes formam ambientes de execução adequados para sistemas pervasivos, disponibilizando um aparato tecnológico necessário para a execução de suas aplicações. Tipicamente são formados por uma infra-estrutura computacional relativamente estável, contendo servidores convencionais, sensores de diversos fins, e suporte a uma rede de comunicação sem fio.

Esses ambientes devem também considerar as mobilidades física e lógica. A primeira diz respeito ao deslocamento de *hardware* e usuários nestes ambientes, enquanto a segunda caracteriza a movimentação de artefatos de *software* (processos ou agentes móveis) e seu contexto, para fora e para dentro dos dispositivos que integram o ambiente pervasivo [Yam04].

Devido a estes requisitos, os ambientes de computação pervasiva devem [Joh05]:

- Identificar e perceber usuários espontaneamente, suas ações e objetivos;
- Interagir com fontes de informações;
- Habilitar os dispositivos trazidos ao ambiente a se integrarem com outros dispositivos já presentes, coordenando suas interações;
- Fornecer extensiva capacidade adaptativa de apresentação de informações, quando formatos de dados não são compatíveis com os dispositivos-alvo;
- Entender e antecipar as necessidades dos usuários durante a execução de tarefas;
- Ajudar dispositivos fracos (de capacidade simples).

## 2.3 Desafios para Computação Móvel e Pervasiva

Para os pesquisadores, muitas das questões envolvendo sistemas móveis e pervasivos são semelhantes às de sistemas distribuídos tradicionais, mas as soluções são diferentes [Yam04] [Aug04] [CDK05a]. A principal diferença apontada é a necessidade de transparência (localização, acesso, concorrência) das aplicações distribuídas, em contrapartida à necessidade de ciência do ambiente (elementos, entidades, usuários, situações) que caracterizam uma aplicação móvel ou pervasiva.

Para Collouris [CDK05a], as propriedades que caracterizam as aplicações móveis e pervasivas criam um modelo de sistema chamado então de *volátil*, devido principalmente à configuração altamente dinâmica destas aplicações, onde usuários, elementos de *hardware* e *software* aparecem e desaparecem, de forma espontânea e não determinística. Esta volatilidade se manifesta através de (i) falhas de dispositivos e ligações de comunicação, (ii) mudanças nas características de comunicação e (iii) criação e destruição de associações – relacionamentos lógicos entre componentes de *software* residentes nos dispositivos.

Devido a esta volatilidade, sistemas pervasivos levantam uma série de questões sobre seu funcionamento. Dentre várias questões, destacam-se vários desafios [Joh05] [CDK05a], apresentados nas subseções a seguir.

### 2.3.1 Restrições de Recursos dos Dispositivos Móveis

Os dispositivos portáteis utilizados em aplicações móveis ou pervasivas apresentam restrições de processamento, comunicação e apresentação. A necessidade de serem leves e pequenos faz também com que suas baterias sejam pequenas e com pouca autonomia de vida. A computação, acesso a memória e outros tipos de processamento consomem a energia das baterias destes dispositivos, e em particular, a comunicação sem fio é uma atividade que requer um alto consumo de energia. Conseqüentemente, certos tipos de processamento entre dispositivos devem ser evitados, simplificados ou deslocados para nodos com melhor capacidade de processamento.

Portanto, em um ambiente pervasivo faz-se necessário mecanismos que gerenciem a execução de tarefas, levando em consideração as restrições: (i) de processamento, (ii) de memória, (iii) de conectividade e (iv) de consumo de energia dos dispositivos envolvidos.

### 2.3.2 Sensibilidade ao Contexto

Computação Sensível ao Contexto é o paradigma no qual aplicações podem descobrir e tirar vantagem de informações contextuais (como localização de usuário, hora do dia, pessoas próximas e atividade de usuários e dispositivos) [CK00]. O uso de informações contextuais por essas aplicações reduz a quantidade de atenção humana que uma aplicação necessita para atender às requisições dos usuários. A essência desta afirmação está na crença de que para a concretização da computação pervasiva, computadores do futuro interagirão automaticamente para realizar as tarefas do usuário sem a necessidade de sua interferência.

#### 2.3.2.1 Contexto

Ainda não existe um consenso a respeito da definição de contexto, devido a uma série de fatores, e principalmente devido à abrangência do uso do termo “contexto” em diferentes áreas de pesquisas. Certas áreas se preocupam mais com detalhes técnicos do acesso do usuário, enquanto outras prendem a informações relativas a aspectos sociais.

Existem definições que procuram listar elementos que são relevantes a partir de exemplos e aplicações específicas, citando dados relevantes apenas ao domínio destas aplicações. Goularte, em sua tese de doutorado [Gou03], lista algumas destas definições:

- Contexto pode se referir a informações como localização, pessoas e objetos em suas cercanias, situações sociais e até condições ambientais como iluminação e barulho. Destes elementos extraem-se três aspectos importantes de contexto: onde o usuário está, com quem o usuário está, e quais os recursos próximos ao usuário [ST94];
- Ryan e Morse limitam contexto a informações sobre localização, ambiente, identificação e tempo [MD97];
- Brown segue a mesma linha e sugere que contexto pode ser usado para descrever o ambiente, pessoas ao redor do usuário, situação, estado, temperatura, entre outros [Bro98];
- Saber, Dey e Abowd [SDA98] definem contexto como informações sobre pessoas ou dispositivos que podem ser usados para transformar o modo como um sistema fornece serviços. Exemplos de tais informações são: dados emocionais, dados históricos, dados de localização, dados de intenção e de foco de atenção do usuário;
- Pascoe [Pas98] define contexto como o subconjunto de estados físico e conceitual de interesse para uma entidade particular.

Baseada nestas definições, Dey [Dey01] formulou uma definição mais genérica, considerando como contexto tudo aquilo que seja relevante à aplicação e seu conjunto de usuários, sem enumerar explicitamente quais aspectos formam este contexto, uma vez que estes irão variar de

acordo com a situação. Sua definição é apresentada a seguir:

*Contexto é qualquer informação que pode ser utilizada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar, ou objeto considerado relevante para a interação entre um usuário e uma aplicação, incluindo o usuário e a aplicação em si.*

Por tal definição, cabe a cada projetista determinar quais elementos compõem o contexto relevante de sua aplicação. Baseando-se nesta definição, resume-se que qualquer informação que possa ser utilizada para caracterizar a participação de entidade (pessoa, *software* ou hardware) em uma interação, esta informação é contexto.

### 2.3.2.2 Classificações de Contexto

Para melhor entender as informações sobre contexto, existem trabalhos [DRD<sup>+</sup>00] [Cha02] [CCRR02] que dividem informações contextuais em quatro diferentes tipos: Infra-Estrutura, Sistema, Domínio e Ambiente.

O contexto da Infra-Estrutura representa o estado da plataforma de comunicação entre a aplicações e dispositivos, permitindo que possa haver notificações sobre saída (mesmo através de falhas) ou entrada de dispositivos em um ambiente pervasivo.

O chamado contexto de sistema é uma extensão do contexto de infra-estrutura, incluindo informações sobre os usuários no ambiente. A argumentação para sua existência é a necessidade de agrupar informações que um dispositivo fique ciente dos outros dispositivos em suas cercanias, e conseqüentemente de serviços oferecidos pelas aplicações. Porém, esta argumentação não deixa clara a separação entre contextos do Sistema e da Infra-Estrutura, uma vez que ambos tratam com aplicações e dispositivos [Gou03].

O contexto do Domínio procura agregar informações sobre a semântica do domínio da aplicação. Para isto, são consideradas as associações entre os dispositivos e seus usuários de forma a se determinar a natureza da interface ou serviço sendo apresentado. São ainda consideradas informações sobre a segurança e privacidade de informações dos usuários, como localização e atividade realizada.

Por fim, o contexto do Ambiente reúne dados sobre informações sobre o local onde uma determinada entidade se encontra. Exemplos de dados relevantes seriam luminosidade, temperatura, localização, dentre outros.

Apesar de válidas, as categorias propostas na classificação anterior criam certas dúvidas sobre quais informações pertencem a cada categoria. Outra classificação [CK00], também com quatro categorias, onde o mapeamento das informações é mais claro. São elas:

- **Computacional:** Engloba informações relativas a dispositivos utilizados pelos usuários (Consumo de CPU, bateria, interface de apresentação), recursos de comunicação via rede (largura de banda, custos de conexão) e recursos próximos, como monitores, impressoras, estações de trabalho e pontos de acesso;
- **Usuário:** agrupa informações do perfil de usuário, localização, dados de mobilidade, pessoas próximas e até a situação social atual;
- **Físico:** representam informações como luminosidade, nível de ruído, condições de tráfego;
- **Tempo:** indicam informações temporais como hora do dia, dia da semana, mês, e estação do ano.

A manipulação destas informações levanta várias questões: (i) obtenção de dados de contexto a partir de sensores colocados no ambiente pervasivo em face a dinâmica (volatilidade) destes elementos, (ii) acuidade dos dados obtidos a partir destes sensores (abstração), (iii) representação e integração de dados de diferentes sensores (agregação).

### 2.3.2.3 Rastreamento de Mobilidade Física

Os trabalhos sobre computação sensível ao contexto sugerem que certos elementos contextuais são mais importantes que outros [ST94] [Bro98] [SDA98] [Dey01] [CDK05a]. Dentre estes, sem dúvida, o que tem recebido maior atenção é a localização de usuário e dispositivos. A mobilidade de usuários e dispositivos, juntamente com a necessidade de se ter sensibilidade ao contexto, torna essencial o rastreamento da localização física do dispositivo ou usuário em um ambiente de computação pervasiva [Joh05].

Existem vários métodos e tecnologias que podem ser aplicados para a determinação da posição física de pessoas ou dispositivos. Porém, em geral, nenhuma será auto-suficiente para atender de forma eficaz qualquer tipo de ambiente. Estas abordagens empregam diferentes meios como ótico, acústico ou rádio-frequência, divergindo em aspectos como:

- **Área de aplicação:** dependendo do mecanismo utilizado, há restrições sobre onde este mecanismo pode ser utilizado. Algumas técnicas podem ser aplicadas apenas em espaços fechados, enquanto outras apenas em espaços abertos;
- **Precisão:** também de acordo com o mecanismo, erros de estimativa da posição de uma entidade podem variar de centímetros até quilômetros;
- **Tipo de dado de localização:** Dados obtidos podem representar coordenadas absolutas ou globais, como latitude, longitude e altitude; ou podem ser relativas a espaços fixos, como “Sala M1 do Prédio B do Centro de Informática”;

- **Privacidade do usuário:** Se alguma informação sobre a entidade a ser localizada deve ser fornecida para o rastreamento de sua posição;
- **Origem da informação de localização:** a diferença é se o usuário ou dispositivo é quem fornece os dados de sua posição, ou esta informação é calculada (também dita *rastreada*) por outro sistema ou dispositivo.

Dentre alguns dos sistemas e abordagens para localização, destacam-se aqueles baseados em satélite como o *Global Positioning System (GPS)*, o mais popular sistema de rastreamento atualmente. Sendo baseado por satélites, seu funcionamento é mais adequado apenas para áreas abertas, devido à atenuação do sinal dentro de locais fechados. Também utilizam sistemas de localização por satélite o GLONASS e o ainda em formação, Galileo.

Outros métodos procuram utilizar a infra-estrutura de redes baseando-se nas características de transmissão, utilizando informações sobre as características de estações-base de alcance limitado. Dispositivos podem comparar a força do sinal como uma medida para saber qual estação-base está mais próxima. Técnicas que se enquadram neste perfil são Diferença de Tempo de Chegada, Ângulo de Chegada e Diferença de Tempo Observada.

As principais questões são relacionadas a diversos fatores: primeiro, como ressaltado anteriormente, nenhum dos métodos utilizados será auto-suficiente em qualquer ambiente. Boa parte das tecnologias necessárias demandam um alto custo para sua implantação, principalmente em se tratando de espaços abertos. Por fim, para sua melhor utilização, os dados de localização devem ser associados com bases de dados em sistemas de informação geográficos (GIS), que possuem alto custo de utilização e manutenção.

### 2.3.3 Adaptabilidade ao Contexto

A volatilidade apontada por Collouris [CDK05a] traz uma série de conseqüências para sistemas pervasivos. A mobilidade de dispositivos e componentes de *software* faz com que a configuração de execução do sistema se modifique em função da entrada e saída de usuários, dispositivos e componentes de software. Por vezes, a saída de alguma destas entidades se dá de forma não espontânea devido a falhas de comunicação decorrentes das características do meio de comunicação sem fio (raio de alcance, oclusão devido a fatores físicos como paredes, dentre outros).

Como visto na subseção anterior, aplicações móveis e pervasivas precisam ter sensibilidade ao contexto (localização, situação, dentre outros) das entidades relevantes para que possam alterar seu funcionamento em virtude do estado atual dos elementos que cercam o sistema. A adaptabilidade ao contexto é apontada como o principal diferencial entre sistemas distribuídos

convencionais e sistemas móveis e pervasivos. Enquanto em sistemas distribuídos a aplicação pode ser adaptativa, em sistemas pervasivos ela é adaptativa [Aug04].

O objetivo de sistemas adaptativos é acomodar heterogeneidade através do reuso de *software* em diferentes contextos que variam como disponibilidade de recursos, preferências do usuário, adaptando o comportamento da aplicação sem sacrificar suas propriedades fundamentais, como a oferta do serviço. Esta tarefa mostra-se não trivial, uma vez que não há como se prever de antemão todas as situações em que aplicações serão utilizadas na fase de projeto.

A adaptação nos sistemas móveis é ainda discutível, podendo se referir a diversas noções, dependendo dos objetivos e domínio das aplicações. Adaptação pode ser feita pela modificação de componentes que compõem a aplicação, ou pela modificação dos dados trafegados entre os elementos interessados, como apresentado a seguir.

#### 2.3.3.1 Adaptação de Conteúdo

Técnicas de adaptação de conteúdo funcionam como um sistema inteligente que pode tomar decisões de acordo com a versão de resposta de conteúdo ou estratégias de adaptação com sensibilidade aos vários tipos de contexto [Joh05]. Estas técnicas compreendem redução, transformação, compressão ou *caching* de dados entre dispositivos.

Esta idéia opõe-se ao envio de dados em um formato único para todo e qualquer tipo de dispositivo presente no ambiente, o que pode vir a gerar custos inapropriados de largura de banda e dos recursos dos dispositivos [JS04].

Existem diferentes abordagens para adaptação de conteúdo. Uma primeira abordagem é através de reservas de recursos [Sat01]. Recursos englobam largura de banda, energia, ciclos de computação e memória. O cliente pode pedir ao ambiente que garanta um determinado nível de recursos, de acordo com a aplicação. Porém, obter garantia de recursos em um ambiente pervasivo pode ser impossível. Uma segunda abordagem é a notificação do sistema ao usuário sobre mudança nos níveis de recursos. O usuário pode então adaptar manualmente a aplicação para um melhor funcionamento face às novas condições de contexto. Por fim, a notificação do sistema é feita diretamente à aplicação que adequa-se automaticamente às mudanças.

Em relação às técnicas para a adaptação do conteúdo, uma classificação de quatro grupos é apresentada a seguir [LG01].

- **De acordo com contexto-alvo:**

- *Adaptação à infra-estrutura técnica:* quando a transformação de dados leva em conta apenas fatores relacionados às conexões de rede e capacidade dos dispositivos envolvidos na troca dos dados;

- *Adaptação às preferências do usuário*: quando o usuário tem diferentes requisitos em relação à apresentação de dados;
- **De acordo com o momento da criação do conteúdo**
  - *Adaptação estática*: quando os dados são pré-processados e armazenados em diferentes versões, de acordo com a qualidade de apresentação ou dados relevantes. A partir do contexto do usuário, a versão mais adequada é disponibilizada ao usuário;
  - *Adaptação dinâmica*: as informações são processadas e entregues sob demanda;
- **De acordo com os tipos de mídias**
  - *Adaptação de mídias simples*: quando os tipos de mídia do conteúdo original e conteúdo adaptado são os mesmos. Um exemplo é a transformação de uma imagem padrão bitmap BMP, para outra, padrão GIF, com menor qualidade;
  - *Adaptação de mídias cruzadas*: Quando um tipo de mídia é transformado em outro, mais adequado a um dispositivo particular. Exemplo desta adaptação é a transformação de um vídeo em um conjunto com as principais imagens do mesmo;
- **De acordo com o local da adaptação**
  - *Baseada no servidor*: Cabe ao dispositivo que atua como servidor do conteúdo, analisar o contexto existente entre ele e o dispositivo alvo, e realizar eventuais transformações (se necessárias) no conteúdo disponibilizado;
  - *Baseada no proxy*: nesta abordagem, a decisão e a efetiva realização da transformação do conteúdo é realizada por um terceiro componente, que atua como um *proxy* na comunicação entre cliente e servidor de conteúdo;
  - *Baseada no cliente*: O servidor repassa ao cliente uma lista de possíveis representações de um conteúdo. O cliente escolhe dentre as opções, aquela que melhor se adequa ao seu perfil e repassa ao servidor, que por sua vez, envia os dados ao cliente;
  - *Baseada no middleware*: Cabe ao *middleware* que interliga clientes e servidores realizar as decisões sobre transformações de conteúdo, de acordo com o contexto envolvido. Dado que estas infra-estruturas de *software* via de regra apresentam serviços potencialmente distribuídos em vários nodos, o serviço de adaptação pode ser realizado tanto no próprio servidor, quanto em outro nodo similar a um *proxy*.

### 2.3.4 Escalabilidade

Em um sistema de computação pervasiva, podem haver até milhares de dispositivos presentes em um dado instante, como no caso de sensores. O número de componentes de *software* presentes nestes dispositivos será ainda maior. Isto gera um problema de escalabilidade, visto que estes sistemas precisam se adaptar em virtude do aumento ou decréscimo destes componentes no seu dispositivo.

No caso de ambientes de computação pervasiva, salienta-se que da mesma forma que as cooperações entre dispositivos são formadas, a saída espontânea destes equipamentos, como também falhas nos mesmos ou nos meios de comunicação utilizados para comunicação determinam que associações são desfeitas de forma imprevisível.

### 2.3.5 Descoberta de serviços

Em um ambiente de computação pervasiva, usuários e dispositivos formam associações para cooperações dinâmicas, de acordo com seu deslocamento no ambiente. Para isto, cada dispositivo interage com os integrantes de ambiente pervasivo através do uso e oferta de funcionalidades. Faz-se então necessário que estes componentes tenham capacidade de descobrir, utilizar e oferecer tais funcionalidades às demais entidades presentes no ambiente.

Tais funcionalidades em um sistema pervasivo são vistas como serviços disponíveis e ofertados por cada dispositivo para potenciais clientes, que entram e saem do ambiente pervasivo. Esta característica impõe que dispositivos possam descobrir tais serviços em suas cercanias. Este problema pode ser solucionado através de um mecanismo específico de descoberta de serviços (*discovery service*). Este serviço funciona como uma espécie de serviço de diretório, onde ofertas de serviços são registradas para serem usados por potenciais clientes. Dentre suas funções estão a oferta, descoberta, escolha e uso de serviços oferecidos pelos diversos componentes presentes em um dado instante. Alguns exemplos de implementações deste serviço são o *Service Location Protocol* [GPVD99], Jini [Mic99] e *Universal Plug and Play* [For00].

No caso de aplicações pervasivas, as implementações destes serviços devem levar em conta as características destes sistemas, como: (i) dados relativos às consultas feitas pelos potenciais clientes são determinados em função do contexto do cliente, que por sua vez é altamente dinâmico; (ii) via de regra, não há uma infra-estrutura pré-determinada para hospedar um serviço de descoberta de serviços, como um nodo com endereço fixo; (iii) serviços registrados no serviço de descoberta podem desaparecer espontaneamente; (iv) e por fim, os protocolos para descoberta de serviços devem ser sensíveis às limitações de consumo de bateria e recursos de rede, como largura de banda.

### 2.3.6 Segurança

Sistemas pervasivos trazem muitas questões relativas à segurança. Primeiramente, muitas das informações que transitam nos ambientes pervasivos contêm dados privados dos usuários e, portanto, exigem a utilização de mecanismos que garantam confidencialidade e integridade destas informações. Soma-se a estes problemas, a facilidade de extravio de dispositivos portáteis, permitindo teoricamente acesso a pessoas não autorizadas através destes dispositivos [CDK05a].

Tomando por base os modelos convencionais de segurança, as soluções são baseadas na confiança ou conhecimento prévio entre seus componentes e usuários. Esta realidade não se aplica ao modelo utilizado na computação pervasiva, onde componentes podem interagir espontaneamente sem conhecimento prévio um do outro, ou presença de uma terceira entidade que possa atestar confiança de ambos.

Ressalta-se também que medidas necessárias para preservação de segurança em uma aplicação pervasiva devem proteger a espontaneidade das interações, reduzindo a necessidade de atenção humana [EHL01].

Por fim, as limitações de recursos dos dispositivos também devem ser consideradas. Mecanismos como criptografia demandam processamento adicional ou uso de recursos que são escassos nos dispositivos presentes em ambientes pervasivos. Sendo o consumo de energia também um agravante, protocolos e medidas de segurança devem procurar economizar a bateria de seus dispositivos [CDK05a].

## 2.4 Considerações sobre o Capítulo

Como visto neste capítulo, vários são os desafios para a concepção de aplicações pervasivas. A visão de jogos PM2G enfrenta as mesmas questões. Em relação às restrições dos dispositivos utilizados, a escolha de uma arquitetura cliente-servidor facilita a incorporação de dispositivos com baixos recursos. A simulação do jogo é realizada por uma máquina compatível tanto em realizar os procedimentos de simulação, quanto atender os pedidos dos clientes, quer sejam estes dispositivos móveis ou não. Os clientes atuam apenas como portais, recebendo informações sobre o estado corrente do jogo, apresentando o mesmo ao usuário, que envia seus comandos para serem executados pela simulação. A frequência de atualização é dependente do dispositivo e de sua rede de acesso. Jogadores via computadores pessoais podem ter uma taxa de atualização mais baixa, enquanto jogadores via dispositivos móveis mantêm um comportamento mais casual, requisitando informações sobre o jogo apenas quando explicitamente solicitado.

Em relação à sensibilidade ao contexto, um jogo PM2G permite que jogadores utilizem diferentes dispositivos para acesso ao jogo. Além disso, preferências pessoais do jogador também podem ser utilizadas para modificar a participação do usuário no jogo em um dado instante. Informações sobre localização do jogador também são relevantes. Segundo a idéia que contexto é definido de acordo com o domínio da aplicação em questão, um jogo PM2G utiliza tanto o contexto técnico (dispositivo), quanto do usuário (preferências pessoais e localização, quando possível) para guiar a participação do usuário. Porém, por não estar confinado a um espaço finito, o monitoramento deste contexto não é realizado de forma automática por sensores, mas informado diretamente pela própria aplicação.

A mudança nos dados de contexto ocasiona uma adaptação de conteúdo e de outros elementos necessários à adequação da jogabilidade do usuário, usando a estratégia de um *middleware*. Os serviços responsáveis por tais adaptações são endereçados através de computadores fixos, via *Internet*. Maiores detalhes sobre estratégias serão apresentadas no Capítulo 6.

## Suporte a Jogos Multiusuário Distribuídos

*Video-games não influenciam crianças. Se o Pac-Man tivesse influenciado a nossa geração, estaríamos todos correndo em salas escuras, comendo pílulas mágicas e escutando músicas eletrônicas repetitivas...*

— KRISTIAN WILSON (Nintendo Inc.)

Da implementação do primeiro jogo multiusuário até a idéia de inclusão de pervasividade nestas aplicações, os desafios encontrados por desenvolvedores são inúmeros. A abrangência de comunicação, latência, largura de banda, bem como a capacidade de processamento dos dispositivos envolvidos têm um impacto significativo sobre as técnicas existentes para implementação de tipos específicos de jogos multiusuário distribuídos. Neste capítulo são detalhados como estes aspectos influenciam a implementação de jogos multiusuário em geral, além de apresentar as técnicas mais utilizadas para atenuar o impacto destes problemas. Primeiramente, são apresentados conceitos gerais e cenários atuais de utilização de jogos multiusuário. Em seguida são apresentados seus desafios e as soluções clássicas a estes problemas.

### 3.1 Introdução

Jogos multiusuário distribuídos pertencem à categoria de aplicações de espaço compartilhado das quais fazem parte simulações militares, ambientes virtuais distribuídos e trabalho colaborativo apoiado por computador – do inglês, *Computer Supported Collaborative Work (CSCW)*. Jogos multiusuário são simulações de ambientes onde cada jogador busca atingir um certo objetivo através da interação com outros jogadores e com o ambiente [Cec05]. Este ambiente representa o espaço compartilhado entre jogadores, onde o jogo é realizado.

No entanto, mesmo classificados como aplicações de espaço compartilhado, jogos multiusuário apresentam características peculiares. Primeiramente, seu foco está na competição entre seus usuários, ao contrário das demais aplicações de CSCW, onde os usuários colaboram para a realização de alguma tarefa. Esta característica influencia diretamente um segundo aspecto peculiar a jogos multiusuário. Um sistema de suporte para jogos deve evitar que os jogado-

res possam alterar o estado do jogo de forma indevida, ou seja, de forma que as regras do jogo sejam burladas. De forma mais sucinta, jogos multissuário devem ser intolerantes à trapaça [Cec05]. Enquanto outros ambientes de espaço compartilhado geralmente utilizam redes privadas, com participantes confiáveis, jogos multissuário podem utilizar redes públicas, onde certos jogadores fazem uso de meios anti-éticos para obtenção de vantagens ilegais em relação a jogadores honestos.

### 3.1.1 Estilos de Jogos Multissuário

Jogos multissuário podem ser classificados sob diferentes aspectos. Em relação ao andamento da simulação, um *jogo de tempo-real* é aquele onde o jogador envia comandos de forma independente à passagem do tempo da simulação [Cec05]. Em outras palavras, os jogadores realizam ações sem uma ordem pré-definida, enviando dados continuamente e modificando concorrentemente o estado do jogo. Em contraste, um *jogo baseado em turnos* é um jogo onde o jogador envia comandos de acordo com o tempo da simulação. Neste caso, há uma alternância entre as ações de cada jogador, onde cada participante tem bem definido o seu momento de realizar uma jogada. Por exemplo, em um jogo de xadrez, a partida não avança até que o jogador da vez realize a sua jogada.

Jogos multissuário de tempo-real geralmente impõem restrições quanto ao atraso máximo entre o envio de um comando pelo jogador e a efetivação do comando no jogo. A influência deste atraso na percepção do andamento do jogo varia de pessoa para pessoa [PW02b], mas geralmente pode-se estabelecer um valor representativo para o atraso máximo tolerado em um jogo multissuário. Para jogos representados graficamente como ambientes bidimensionais ou tridimensionais, este atraso é dependente da *perspectiva do usuário* e da *reatividade* necessária para conservação da jogabilidade. Baseado nesta perspectiva, dois estilos de jogo são comumente encontrados na literatura: jogos em primeira pessoa e jogos de estratégia em tempo-real – do inglês, *Real-Time Strategy (RTS)*<sup>1</sup>.

Na primeira categoria, o jogador simula a visão do próprio personagem controlado no jogo. Boa parte destes jogos são jogos “de tiro”, chamados então *First Person Shooter (FPS)*, como visto no jogo *Return to Castle Wolfenstein* (2004) apresentado na Figura 3.1 (a). Esta perspectiva dá ao jogador uma intensa sensação de imersão no jogo, exigindo reflexos rápidos e tempo de respostas quase instantâneos.

Já em jogos de estratégia em tempo-real, cada jogador controla um conjunto de unidades independentes, tomando o papel de um comandante ou chefe de uma raça ou exército. O

---

<sup>1</sup>Também chamados de jogos *top-down view*.



**Figura 3.1** Exemplos de jogos (a) de Primeira Pessoa e (b) de Estratégia em tempo-real.

jogador instrui suas unidades (como por exemplo: soldados, veículos, dentre outros) nas ações que cada unidade deve realizar, como atacar inimigos ou deslocar-se para determinadas áreas do mapa. Os jogadores competem entre si para destruir o exército do inimigo ou capturar algum objeto vital no mundo virtual. Nestes jogos, o jogador visualiza o jogo como em uma tomada aérea do mundo virtual e suas ações não necessitam de tempos de resposta tão imediatos quanto jogos FPS. *Alexander* (2004), apresentado na Figura 3.1 (b), exemplifica um jogo RTS.

Mesmo em jogos de ação que exigem reflexos rápidos, alguns comandos podem possuir requisitos de tempo mais relaxados e podem ser processados com vários segundos de atrasos. Porém, quando se propõe sistemas de suporte para jogos de ação, a preocupação central geralmente é com os comandos que possuem fortes restrições de tempo.

### 3.2 Cenários Atuais para Jogos Multiusuário

O aspecto multiusuário em jogos eletrônicos apresenta diferentes cenários para sua aplicação. Estes cenários, não por coincidência, são uma consequência da evolução tecnológica vivida desde as primeiras experiências em redes locais até a utilização de dispositivos móveis como plataforma para jogos pervasivos nos dias atuais. A proposta Pervasive Multiplayer Multiplatform Game (PM2G) é uma consequência desta evolução. Esta seção apresenta os principais cenários para jogos multiusuário em uma linha de evolução tecnológica e temporal. O objetivo

desta seção é apresentar como estes diferentes cenários convergem para a proposta defendida neste trabalho.

### 3.2.1 Jogos Multiusuário em Pequena Escala

Estes jogos representam a primeira geração de jogos multiusuário para computadores pessoais. Seu surgimento aconteceu em meados da década de noventa, como consequência da popularização dos computadores pessoais, do surgimento da multimídia digital e da massificação das redes de computadores. Nestes jogos, a máquina de um jogador ou uma máquina dedicada atua como servidor onde a simulação é executada. Esta máquina é responsável por iniciar uma sessão de jogo, a partir da qual jogadores se conectam, permitindo sua interação. Uma característica destes jogos é que cada sessão tem um tempo limitado, determinado pelo objetivos específicos de cada jogo, como um limite de vinte voltas em uma corrida de carros. Típicas partidas duram alguns minutos ou algumas horas, e o resultado de uma partida não afeta o estado da partida seguinte.

Outra característica destes jogos é seu número de jogadores. Os primeiros jogos multiusuário permitiam de dois a oito jogadores em uma mesma sessão. Estes eram preferencialmente voltados para redes locais, devido ao atraso mínimo necessário para sua jogabilidade. Com a melhoria das tecnologias de comunicação e o aumento do poder de processamento dos computadores pessoais, este número elevou-se, possibilitando até 64 usuários em uma única sessão hospedada na máquina de um jogador. Exemplos destes jogos são *Counter-Strike* [Ste03] e *Starcraft* [BE05].

### 3.2.2 Jogos Multiusuário Massivos (MMG)

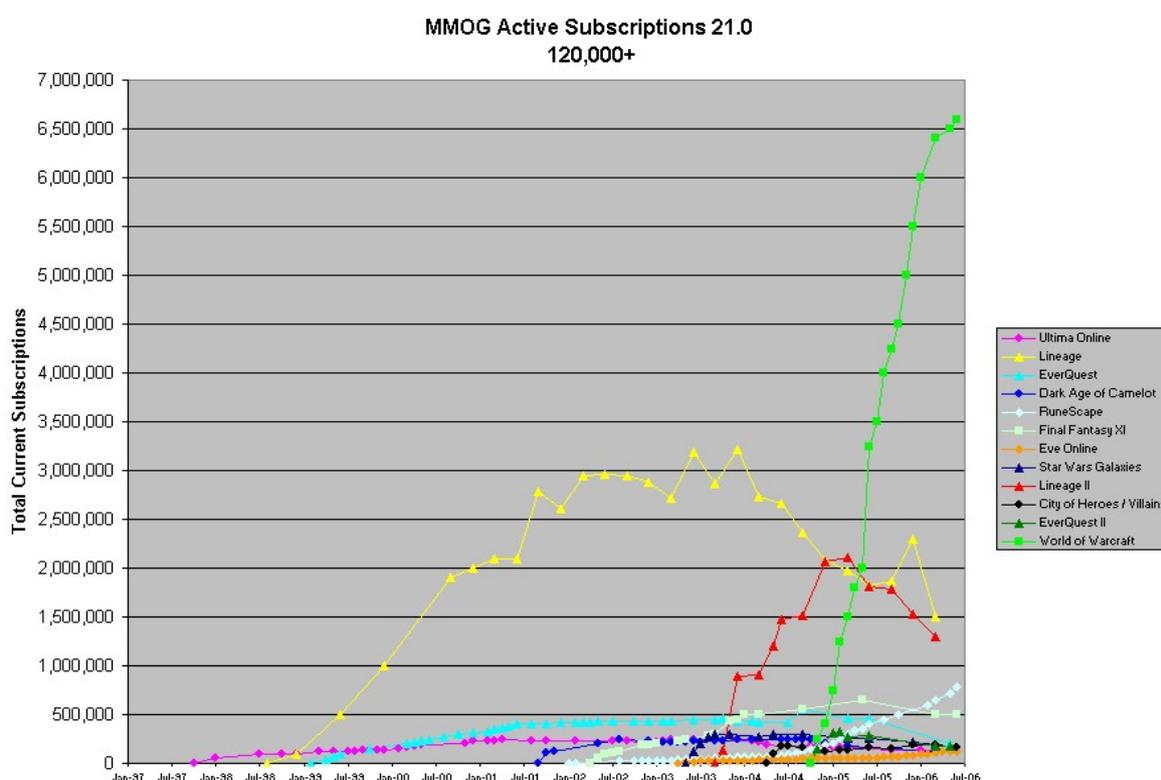
A popularização do acesso à Internet via banda larga permitiu um aumento considerável na escala de usuários em um jogo multiusuário distribuído, criando os Jogos Massivamente Multiusuário, do inglês, *Massively Multiplayer Games (MMG)*. Estes jogos diferem do cenário anterior por duas características adicionais. Primeiro, a magnitude do número de jogadores concorrentes é tipicamente na ordem de  $10^4$  ou ainda maior [CG04]. Segundo, MMGs dão suporte a uma simulação em constante execução<sup>2</sup>, independente da presença de jogadores, onde o estado do jogo é gravado em algum meio de persistência. Estes jogos, também chamados de mundos contínuos ou persistentes, indicam que ações realizadas por jogadores têm capacidade de moldar permanentemente o mundo virtual. Em outras palavras, um MMG pode continuar

---

<sup>2</sup>Acontecem paradas estratégicas para manutenção de servidores.

sem perspectiva de encerramento.

O modelo de negócios para Massively Multiplayer Games (MMG) difere dos jogos tradicionais. Um MMG é encarado como um serviço oferecido pela empresa mantenedora responsável pela infra-estrutura de suporte ao correto funcionamento do jogo, onde cada usuário é cobrado periodicamente pelo acesso a esta infra-estrutura. Alguns dos mais populares MMGs como *Everquest* [Eve06], *Ultima Online* [Ori02] e *Ragnarok* [Rag05] possuem mais de 200 mil assinantes. Exponentes como os jogos da série *Lineage* [NCs05b] [NCs05a] e o *World of Warcraft* [BE06] ultrapassam milhões de usuários [Woo06], como apresentado na Figura 3.2.



**Figura 3.2** Crescimento do Número de Assinantes de MMGs atuais.

Porém, apesar de seu sucesso e potencial, MMGs apresentam uma série de problemas ainda a serem solucionados, principalmente no que diz respeito às questões sobre escalabilidade e custos associados à manutenção da infra-estrutura necessária ao funcionamento do jogo. Estes problemas já fizeram inclusive que grandes empresas cancelassem o desenvolvimento de MMGs, como a própria Microsoft [MGS04].

### 3.2.3 Jogos Móveis Multiusuário

A computação móvel é o mais recente cenário de aplicação para jogos multiusuário. Este cenário é consequência de diversos fatores. A popularização de dispositivos móveis, o aumento do poder computacional destes dispositivos e o surgimento de plataformas abertas como J2ME [SM05] permitiu a criação de aplicações simples para tais dispositivos, principalmente jogos monousuário, chamados então “jogos móveis”. O número crescente de tecnologias de comunicação sem fio como *General Packet Radio Service (GPRS)*, *bluetooth* [Blu04] e *Wi-Fi (802.11)* [IEE05], aproximam cada vez mais dispositivos móveis das redes de computadores tradicionais. Seguindo esta evolução, a inclusão do aspecto multiusuário aparece como consequência natural para jogos móveis.

Este processo evolutivo segue os passos dos jogos em computadores pessoais, onde jogos multiusuário surgiram após o amadurecimento e proliferação das redes de computadores tradicionais. O surgimento de plataformas com o *Nokia N-Gage™* [Nok04], *NintendoDS™* [Nin05] e *Playstation Portátil™* [SE06] (Figura 3.3) reforça ainda mais esta afirmativa. Estas plataformas são projetadas para o mercado de entretenimento digital, oferecendo diversos jogos multiusuário em escalas que variam de 2 a 16 jogadores, de acordo com a tecnologia utilizada.



**Figura 3.3** Plataformas de Jogos Móveis Multiusuário

De fato, o uso da tecnologia de comunicação acaba sendo um fator determinante no projeto de um jogo multiusuário móvel. O uso das redes de operadoras de meio físico, sejam elas analógicas, de segunda (2G e 2.5G) ou terceira (3G) geração, utilizam tecnologias que apresentam latências elevadas e alto grau de intermitência na conexão do usuário. Além destes problemas técnicos, existem questões de mercado ainda em aberto, como o modelo de tarifação a ser utilizado entre empresas desenvolvedoras de jogos, distribuidores e operadoras de meio físico.

Por tais motivos, estas redes limitam sua utilização a jogos não tão interativos, como jogos em turno (Seção 3.1.1), exemplificado pelo jogo *Batalha Naval* [TdAdAL<sup>+</sup>03].

Uma alternativa para criação de jogos multiusuário mais interativos em dispositivos móveis, mesmo em face aos problemas de conexão de redes sem fio são os chamados Jogos Conectados (*Connected Games*). Nestes jogos, a jogabilidade básica não requer uma conexão permanente com outras entidades, como servidores ou outros jogadores. Porém, funcionalidades suplementares são acessadas através de uma rede. Exemplos de funções possíveis são a publicação da pontuação dos jogadores ao final de cada partida ou o *download* de novos níveis de dificuldade ou itens do jogo. Embora não haja uma interação direta entre jogadores, o que de certa forma acaba comprometendo o uso do termo “multiusuário” para tal cenário, a idéia de jogos conectados permite que jogadores possam competir indiretamente, como na publicação das maiores pontuações ou na ativação de níveis mais avançados. Um exemplo de jogo que segue esta idéia é *Meu Big Brother* [MG05].

Em relação às redes espontâneas (*ad-hoc*) ou redes locais sem fio, *Wireless Local Area Networks (WLAN)*, os problemas de latência e intermitência são atenuados, permitindo jogos mais interativos. Porém, estes jogos oferecem suporte a um número limitado de usuários, devido às restrições das tecnologias utilizadas e abrangência de alcance destas redes.

Apesar das deficiências de cada tecnologia, os resultados da implantação de jogos móveis multiusuário são promissores. Telefones celulares são conectados por natureza, estão sempre com seus usuários e podem ser utilizados em praticamente todo lugar e a todo momento. A penetração destes dispositivos na população mundial faz com que estas aplicações possam revolucionar a indústria do entretenimento digital, atingindo uma audiência até então inimaginável, com receitas igualmente proporcionais a esta audiência. Esta tendência se confirma com a concepção de jogos que unem características de jogos móveis e MMGs, os chamados Jogos Multiusuário Massivos Móveis, do inglês, *Massively Multiplayer Mobile Games (3MG)*. Estes jogos utilizam as redes das operadoras de telefonia celular de modo a permitir o suporte a um alto número de jogadores. De certa forma, as operadoras possuem um interesse especial nos jogos multiusuário, pelo potencial de geração de tráfego adicional de dados nas suas redes e conseqüente aumento da receita. Os problemas decorrentes das infra-estruturas de redes de telefonia são atenuados através de projetos adequados às limitações existentes, como o uso de batalhas por turnos ou retardo na execução das ações dos jogadores, bem como uso de recursos específicos das redes de telefonia móvel, como o uso de *Short Message Service (SMS)*. Exemplos de 3MGs são *Samurai Romanesque* [Kri03], *Tibia Micro Edition* [Nok03] e *Pocket Kingdom* [Nok].

### 3.2.4 Jogos Pervasivos

Estes jogos aproveitam a visão de Computação Pervasiva (vide Capítulo 2) para criar novas formas de interação e entretenimento que vão além daquelas realizadas em frente a computadores ou consoles de video-game, permitindo que jogadores desloquem-se e interajam de diferentes formas, através de diversos dispositivos e tecnologias de comunicação. Da mesma forma como concordamos que a computação pervasiva é uma evolução da computação móvel, podemos dizer que jogos pervasivos são inovações a partir dos atuais jogos móveis.

Estas aplicações combinam o real e o virtual em jogos que aproveitam os inovadores aparatos tecnológicos para utilizar informações do mundo real como elemento influente no mundo virtual que compõe o jogo. Estes jogos são concebidos com base em três tecnologias principais: (i) dispositivos móveis, (ii) comunicação sem fio e (iii) tecnologias capazes de capturar informações no contexto do mundo real dos jogadores, principalmente sua localização física.

Em realidade, é preciso fazer um ressalva ao uso do termo “jogo pervasivo” no contexto desta proposta. Vários jogos que não se baseiam nas tecnologias citadas anteriormente também são classificados como jogos pervasivos. Alguns deles dizem respeito a jogos que utilizam computadores pessoais, mas que usam meios de comunicação do cotidiano como *e-mail*, telefone e fax, para fornecer informações do jogo aos jogadores.

Outros são aqueles que utilizam modos não convencionais para interação homem-máquina entre jogador e jogo. Na proposta de Mark Weiser, o uso de interfaces naturais que facilitem a comunicação do usuário com seu ambiente computacional é apontado como um importante passo em direção à computação ubíqua [MI01]. Estas interfaces devem se aproximar da forma como seres humanos interagem com o mundo real. Esta idéia segue o conceito de Interfaces Tangíveis [IU97], sistemas relativos ao uso de artefatos físicos como representações e controles de sistemas de informação. Um exemplo recente deste conceito é o novo console da Nintendo, o Wii[Nin06]. Este *videogame* captura os movimentos do jogador e as interpreta como ações no jogo.

Neste texto, jogos pervasivos seguem a idéia de utilização de dispositivos móveis e informações do contexto do mundo real do jogador como elementos da dinâmica do jogo. Existem hoje pesquisadores que já realizam estudos na área, através de provas de conceito, protótipos e experiências comerciais [CFG<sup>+</sup>03] [FBA<sup>+</sup>03]. Suas possibilidades de aplicação em setores como educação e turismo são inúmeras. Alguns especialistas afirmam que jogos pervasivos têm inclusive o potencial de atrair pessoas que não sejam interessadas em jogos de computador. Jogos pervasivos oferecem novas oportunidades para projetistas, mas também demandam projetos diferentes dos tradicionais jogos eletrônicos, de maneira análoga a como um sistema

computacional pervasivo é diferente de sistemas tradicionais.

Boa parte das experiências com jogos pervasivos realizadas revive jogos infantis clássicos, no estilo “pega-pega” ou “capturar a bandeira”. Estes jogos exigem o deslocamento dos jogadores em algum ambiente físico, sendo chamados então de Jogos Baseados em Localização. Experiências com estes jogos incluem a utilização tanto de esquemas de posicionamento em redes locais sem fio, quanto o uso de serviços de localização oferecidos por operadoras de telefonia celular. No caso destes últimos, os jogos oferecem ainda suporte a um alto número de jogadores interagindo seus telefones celulares, em batalhas realizadas nas ruas e bairros de uma cidade. As oportunidades abertas por estes jogos são vastas, em termos de novas oportunidades de lazer, novos modelos de negócio e suporte à comunicação em grupo para aprendizado, socialização e atividades culturais.

Muitos experimentos relacionados com jogos pervasivos utilizam também conceitos de realidade virtual, como utilização de dispositivos como capacetes ou utensílios que permitem mesclar objetos do mundo virtual na visão dos jogadores. Algumas experiências com jogos pervasivos são o *Human Pacman* [CFG<sup>+</sup>03], apresentado na Figura 3.4, e *Can You See Me Now?* [FBA<sup>+</sup>03]. As primeiras experiências comerciais destes jogos datam do final da década de noventa, onde os casos de maior sucesso são relativos aos mercados europeu e asiático, onde a cultura de jogos é mais forte. Exemplos destes jogos são *Botfighters* [Day], *Undercover* [Tea] e *Alien Revolt* [MC05].



Figura 3.4 O Jogo *Human PacMan*.

### 3.2.5 Jogos Massivos Multiusuário Multiplataforma

Embora jogos móveis massivamente multiusuário e jogos baseados em localização ainda estejam em um estágio embrionário, cenários ainda mais inovadores já começam a ser propostos para a união entre jogos multiusuário e dispositivos móveis. Um deles seria a idéia de jogos multiplataforma que permitissem o acesso ao mundo virtual através de diferentes dispositivos. Enquanto os atuais 3MGs restringem seu uso a dispositivos móveis, jogos multiplataforma per-

mitiriam o acesso do jogador tanto via computadores pessoais, quanto via telefones celulares.

Porém, é preciso ressaltar que as diferentes condições de acesso de dispositivos móveis e computadores obrigam que o projeto destes jogos deva ser muito bem concebido, permitindo uma interação justa entre jogadores através de diferentes dispositivos. Uma idéia simples é permitir diferentes funcionalidades de acordo com o dispositivo de acesso do jogador. Por exemplo, um jogador poderia utilizar seu telefone celular para oferecer um artefato em algum mecanismo de troca disponibilizado pelo jogo, ou enviar mensagens para outros jogadores que estivessem conectados via computadores pessoais. Como frisado na introdução deste documento, o suporte à mobilidade é apontado como requisito imprescindível para o sucesso de jogos futuros [Wal04]. Neste sentido, esta visão começa a ser incorporada em alguns jogos, como *Lineage Mobile* [NCs05b], *Mogi*, *Item Hunt* [NG] e *HinterWars: The Aterian Invasion*.

### 3.3 Fatores limitantes para jogos multiusuário

Apesar de diferentes cenários de aplicações, todos possuem uma preocupação comum. Sendo aplicações de espaço compartilhado, é fundamental para jogos multiusuário manter o estado global consistente entre os jogadores. Para isto, as ações de cada jogador devem ser repassadas entre os jogadores de forma confiável e, possivelmente, de forma rápida e confiável. Além disso, tais ações precisam também ser ordenadas de modo a garantir a correta execução dos eventos no jogo, caso contrário, situações indesejadas podem acontecer como um jogador interagindo com outro já ausente do jogo ou coletando um objeto previamente retirado por outro jogador. Porém, a separação física dos componentes e jogadores decorrente da natureza distribuída de ambientes virtuais em rede impõem obstáculos para a manutenção de um estado global consistente entre os jogadores.

Além de trazer problemas inerentes de aplicações de espaço compartilhado, cada um dos diferentes cenários apresentados na Seção 3.2 incorpora ainda novos problemas específicos às suas características. Por exemplo, no caso de MMGs, a necessidade de escalabilidade face ao alto número de usuários cria mais um problema complexo a ser resolvido.

Helin [Hel03] aponta três grandes problemas enfrentados por jogos multiusuário em rede. Primeiramente, a infra-estrutura de rede afeta diretamente o andamento e consistência do jogo. Em seguida, a arquitetura de comunicação entre jogadores, servidores e demais componentes criam obstáculos para escalabilidade em jogos multiusuário. Por fim, a necessidade de garantia de um jogo justo através da prevenção de trapaça por certos jogadores constitui um problema relevante para jogo multiusuário. Cada um destes problemas é melhor abordado nas subseções

seguintes.

### 3.3.1 Plataforma Física

Smed [SH02] nomeia a infra-estrutura de rede utilizada como a *plataforma física* de suporte ao jogo. Esta plataforma apresenta limites com os quais jogos multiusuário devem conviver, como latência e largura de banda.

Largura de banda é definida como a capacidade de transmissão através de uma linha de comunicação. Para jogos multiusuário esta medida é de fundamental importância, uma vez que representa um limitador na quantidade de informação que pode ser transferida entre os jogadores. Em redes locais, a largura de banda atinge altas taxas de transmissão, diferentemente das redes geográficas, como a Internet. Além disso, a largura de banda necessária para um jogo multiusuário depende também do número de jogadores e das técnicas de distribuição de mensagens utilizadas no jogo a serem vistas a seguir (Seção 3.4).

Por sua vez, latência caracteriza-se pela medida de tempo entre o envio da mensagem e sua recepção pelo destinatário. Este atraso decorrente da inerente separação física entre os participantes existe em várias escalas e nunca poderá ser totalmente eliminado. Por exemplo, atrasos para a propagação e retransmissão dos sinais elétricos necessários para uma transmissão de uma mensagem entre Europa e Estados Unidos variam entre 25 a 30 ms. Adicionando-se o tempo decorrente das rotinas de roteamento, enfileiramento e manipulação de pacotes, o atraso médio desta transmissão eleva-se à faixa de 70 a 90 ms [SKH01].

Para jogos multiusuário, a latência na troca de informações afeta diretamente a percepção de andamento e continuidade do jogo, influenciando negativamente o nível de interatividade (ou reatividade) ao qual os jogadores estarão sujeitos. De fato, avanços em dispositivos multimídia como placas de vídeo e som mais apurados, bem como monitores de alta resolução, permitiram que a sensação de imersão nos jogos eletrônicos aumentasse cada vez mais. Porém, para jogos multiusuário, mesmo possuindo o melhor *hardware* disponível no mercado, a sensação de imersão será totalmente comprometida caso o andamento do jogo seja constantemente interrompido por atrasos na troca de informações entre os jogadores.

Para aplicações interativas distribuídas, pesquisas mostram que atrasos nas trocas de dados não devem ultrapassar 200 ms [SKH01], sob pena de afetar a percepção de continuidade para o usuário. Porém, para jogos multiusuário este número varia de acordo com o estilo do jogo. Em jogos de estratégia em tempo-real, latências de até 350 ms são aceitáveis, contanto que se mantenham estáveis. Entretanto, em jogos de alta interação e de maior imersão como aqueles em primeira pessoa, a latência média deve ficar próxima a 100 ms [SKH01] [PW02a]. Para

jogos em turno, o problema da reatividade é mais simples, uma vez que as ações de cada jogador acontecem em intervalos de tempo controlados e previsíveis. Porém, a consistência do estado do jogo é imprescindível. Para jogos em tempo-real, a consistência do estado global pode ser relaxada em benefício da reatividade de cada jogador. Como exemplo, jogadores podem utilizar estimativas (previsões) sobre a posição do avatar de um jogador para permitir um fluxo contínuo no andamento do jogo.

### 3.3.2 Plataforma Lógica

Além das limitações impostas pela infra-estrutura de rede, as arquiteturas de comunicação (cliente/servidor, ponto a ponto ou *clusters* de servidores), bem como arquiteturas de controle e de dados (centralizado, replicado ou distribuído) formam a *plataforma lógica*, criando outros fatores limitantes a serem considerados, principalmente em relação à escalabilidade (ou ausência dela) de um jogo. Enquanto não há muito a ser feito em relação à plataforma física (a não ser investir em *hardware* e rede), a plataforma lógica assume um papel fundamental para atenuar os efeitos prejudiciais que a plataforma física pode impor, como será visto na Seção 3.4.

### 3.3.3 Segurança e Trapaça

Questões sobre a segurança em jogos multiusuário vêm se tornando uma das maiores preocupações para os envolvidos no desenvolvimento e manutenção de tais aplicações. Enquanto para jogos monousuário, as maiores preocupações recaem sobre pirataria, para jogos multiusuário novos problemas surgem. Primeiramente, jogos *on-line* estão sujeitos a problemas como a segurança dos dados confidenciais de seus usuários, autenticidade e disponibilidade dos sistemas que hospedam o próprio jogo. Estes problemas são comuns a outros domínios de aplicação como comércio eletrônico ou agências bancárias na Internet, onde medidas já utilizadas, como criptografia, também se aplicam a jogos multiusuário.

Porém, um segundo problema de segurança é peculiar apenas a jogos *on-line*. Este relaciona-se com a possibilidade de jogadores trapaceiros utilizarem-se de recursos ou meios anti-éticos no intuito de obter vantagens ilegais sobre os demais jogadores. Para jogos multiusuário, jogadores trapaceiros criam desequilíbrio, desencorajando os demais jogadores, principalmente os iniciantes, a participarem do jogo. Em um mercado onde o jogador deve ser estimulado a permanecer o maior tempo possível no jogo, a facilidade de trapacear pode arruinar o sucesso de um jogo [Yan03].

A trapaça em um jogo multiusuário pode ocorrer de várias formas. Pritchard [Pri00] criou

uma primeira classificação para as formas de trapaça existentes em jogos multiusuário. Yan & Choi [YC02] estenderam esta classificação citando onze diferentes categorias. Certas categorias envolvem aspectos sociais e não se relacionam com alterações no *software* ou falhas em sua concepção. Para estes tipos de trapaça, é difícil imaginar medidas preventivas, como diagnosticar trapaça por convivência por administradores de jogos *on-line*. Porém, certos mecanismos podem ser utilizados para dificultar as ações de *hackers*. Mecanismos tradicionais de segurança como criptografia, assinaturas digitais e controle de integridade podem ser bem empregados para jogos *on-line*, embora individualmente não possam ser considerados como soluções definitivas.

Yan & Choi [YC02] apresentam o termo de “Mitigação de Trapaça”, através do qual é proposta uma abordagem sistemática para prevenção, detecção e gerenciamento de trapaça em jogos *on-line*. Nesta abordagem, ressaltam-se tanto medidas simples como o uso de políticas para definições de senhas dos jogadores, quanto mais complexas e controversas, como o uso de análises estatísticas para detecção de jogadores que sejam bons demais para não estarem trapaceando.

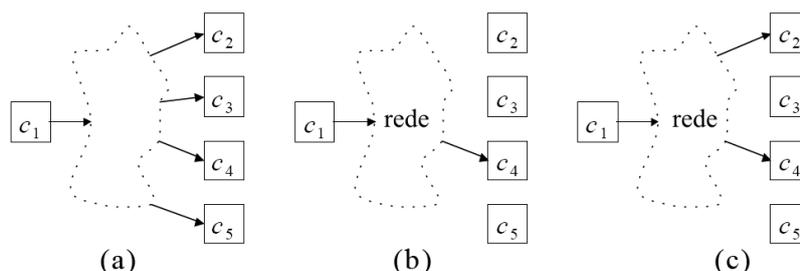
GauthierDickey *et al* [GZL05] categorizam trapaças de acordo com o nível onde as mesmas ocorrem: na rede, nos protocolos, nas regras do jogo ou na própria aplicação. No nível do jogo, trapaças ocorrem por ambiguidades ou por falhas ou manipulação nas regras do jogo para se obter resultados não planejados. A prevenção deste tipo de trapaça requer uma revisão do projeto do jogo de modo a resolver tais falhas. Trapaças no nível de rede ocorrem por problemas de segurança de redes, como ataques por Negação de Serviço (DoS). Trapaças nos níveis da aplicação e de protocolos ocorrem por alterações no código do *software* cliente do jogo, fazendo com que o jogador trapaceiro obtenha vantagens como: um tempo maior de reação ao jogador trapaceiro, ou acesso a informações que o mesmo não deveria ter.

### 3.4 Soluções Clássicas para Suporte a Jogos Multiusuário

De acordo com Smed [SKH01], os conceitos apresentados na Seção 3.3 impõem limites que devem ser tratados em níveis mais altos que aqueles que lidam com a plataforma física de um jogo multiusuário. O uso de diferentes topologias e técnicas de distribuição de mensagens permitem a definição de arquiteturas de comunicação, de controle e dados, que juntamente com técnicas compensatórias podem ajudar a atenuar os efeitos destes limitadores.

### 3.4.1 Distribuição de mensagens

A largura de banda necessária para um jogo multiusuário depende do número de jogadores e das técnicas de distribuição de mensagens utilizadas no jogo. A Figura 3.5 apresenta as principais formas de distribuição das mensagens entre os jogadores de um jogo multiusuário.



**Figura 3.5** Técnicas para transmissão de mensagens. (a) *Broadcast*, (b) *Unicast* e (c) *Multicast*.

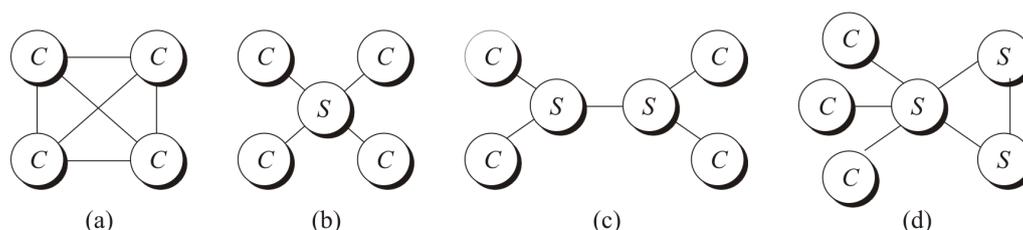
A primeira técnica, conhecida como *broadcast* (Figura 3.5 (a)), repassa a mensagem de um jogador para todos jogadores presentes na partida. Ao receber uma mensagem, cabe ao jogador utilizá-la ou descartá-la de acordo com seu conteúdo. Esta técnica gera uma má utilização da capacidade de transmissão da rede, visto que mensagens são enviadas para todos jogadores e podem ser irrelevantes para a maioria. Este problema acentua-se no caso do aumento de jogadores, criando um sério problema de escalabilidade e desempenho, devido ao congestionamento da rede para jogos que utilizem tal abordagem.

Uma segunda técnica, chamada *unicast*, representada pela Figura 3.5 (b), determina que cada mensagem possua um único emissor e um único receptor. Embora sendo mais eficiente que *broadcast*, torna-se desinteressante quando da necessidade de envio de uma única mensagem para múltiplos destinatários, caso comum na maioria dos jogos multiusuário. Neste cenário, *unicast* utiliza mensagens redundantes, fazendo com que novamente haja uma sobrecarga na rede.

A última técnica, chamada *multicast* e apresentada na Figura 3.5 (c), permite que uma única mensagem seja distribuída para um grupo de destinatários, permitindo uma melhor utilização da rede. Apesar de parecer a melhor solução para a comunicação entre os envolvidos no jogo, *multicast* apresenta como problemas a falta de suporte pela atual infra-estrutura da Internet para sua utilização, bem como questões de segurança relacionadas com implementação de *multicast* sobre uma rede TCP/IP.

### 3.4.2 Arquiteturas de Comunicação

A arquitetura de comunicação de um jogo multiusuário baseia-se nas diferentes formas como os computadores envolvidos em um jogo estão interconectados. A Figura 3.6 ilustra as mais conhecidas arquiteturas para jogos multiusuário [Hel03].



**Figura 3.6** Arquiteturas de comunicação para jogos. (a) Ponto a Ponto, (b) Cliente/Servidor, (c) Replicação de servidores e (d) *Grid* de servidores.

A Figura 3.6 (a) apresenta a arquitetura ponto a ponto, do inglês *Peer-to-Peer (P2P)*. Neste modelo, a topologia de comunicação entre jogadores é formada por um conjunto de nós, onde todos possuem características iguais em relação ao *software* necessário para participação no jogo. Desta forma, cada nó precisa se comunicar com todos os demais participantes para trocar informações. Como exemplo, para enviar uma mensagem aos demais jogadores, um jogador realiza um *broadcast* (ou *multicast*) de sua mensagem para cada participante do jogo. Cada participante possui uma cópia local do estado do jogo, fazendo com que seja essencial o uso de esquemas de sincronização para garantir a consistência do estado do jogo entre os jogadores. Devido a tal característica, arquiteturas ponto a ponto apresentam certos problemas de escalabilidade: primeiro, o andamento do jogo é determinado pelo nó com pior taxa de transmissão entre os usuários. No caso da Internet, é comum ter entre os jogadores, usuários com baixas taxas de transmissão, prejudicando o andamento do jogo como um todo. Um segundo fator é que não se espera que as máquinas dos jogadores tenham capacidade para manipular adequadamente um número muito alto de conexões, como por exemplo no caso de um jogo com centenas ou milhares de usuários[FRC03]. Mesmo com tais limitações, esta abordagem foi utilizada por jogos de estratégia em tempo-real, como *Age of Empires* [Mic04], onde o número de jogadores é limitado e o tempo de resposta não é tão restrito.

Na arquitetura cliente/servidor, representada pela Figura 3.6 (b), um nó da rede é promovido ao papel de servidor do jogo, responsável pela comunicação entre os jogadores. O servidor mantém o estado do jogo centralizado e recebe notificações sobre as atualizações de cada jogador. A partir destas informações, o servidor atualiza o estado do jogo e repassa as atualizações para os demais jogadores. Apesar de possuir implementações diferentes para os jogadores e

o servidor, tal modelo traz certas vantagens em relação ao modelo ponto a ponto. Primeiro, funcionalidades administrativas desejáveis, como controle de acesso ao jogo e tarifação de jogadores, são mais facilmente implementadas em uma topologia cliente/servidor. Outra vantagem é que a ação de um jogador só precisa ser comunicada ao servidor, fazendo com que seu tempo de resposta seja consideravelmente diminuído. Por estas características, jogos em primeira pessoa são tradicionalmente implementados usando a arquitetura cliente/servidor. Por fim, a manutenção do estado do jogo no servidor torna a arquitetura bem resistente a ações de jogadores trapaceiros.

No entanto, o modelo cliente/servidor também apresenta problemas. O primeiro e o mais observável, é a criação de um potencial risco representado pela possibilidade de falha ou indisponibilidade do servidor. Na arquitetura ponto a ponto, como não há centralização do estado do jogo, a queda de um nó não representa risco à continuidade do jogo. Um segundo problema é a latência adicional representada pelo processamento do estado do jogo pelo servidor, caso seu poder computacional não seja suficiente para tratar as requisições dos jogadores.

Dependendo da audiência, o custo do servidor será um problema. Jogos que utilizam servidores dedicados para suporte a milhares de usuários *on-line* apresentam custos anuais de manutenção de até US\$ 10 milhões em *hardware*, bem como alta demanda de recursos de rede para suportar tal escala [IDG02]. Apesar destes fatores, arquiteturas cliente/servidor são utilizadas na maioria dos jogos comerciais atuais ou então através de *clusters* de servidores, como veremos a seguir.

A Figura 3.6 (c) ilustra a arquitetura de replicação de servidores, que representa um modelo híbrido entre as arquiteturas anteriores. Nesta, a infraestrutura de comunicação dos jogadores pode ser vista como uma arquitetura ponto a ponto de servidores de arquiteturas cliente/servidor. Com isto, acaba-se com o potencial risco de se possuir um único ponto de falha como na arquitetura cliente/servidor convencional. No caso da queda de um servidor, jogadores podem ser realocados para outro servidor. Este mesmo princípio é válido para o balanceamento de carga entre servidores sobrecarregados ou subutilizados. A arquitetura de replicação de servidores reduz os requisitos computacionais impostos ao servidor, promovendo uma maior escalabilidade do jogo. Porém, como efeito colateral, a complexidade para a gerência do tráfego das informações entre jogadores e servidores é consideravelmente aumentada, além de na maioria dos casos, não permitir uma comunicação direta de jogadores em diferentes servidores.

Por fim, a Figura 3.6 (d) apresenta o modelo onde os servidores do jogo são organizados de modo a formar um *grid* computacional onde o estado do jogo é distribuído entre os vários servidores. De acordo com a alocação dos jogadores, mais servidores podem ser alocados de forma a dar suporte adequado à entrada de novos jogadores. Da mesma forma, se houver uma

saída de jogadores, os servidores podem ser realocados para outras aplicações ou outros jogos dinamicamente. O uso de *grid* computacional põe fim a um dilema onde a empresa hospedeira de um jogo acabe com poder computacional de sobra no caso de um jogo com poucos usuários, ou com uma sobrecarga de servidores quando do sucesso do jogo. Outra vantagem é que o uso de computação em grade elimina as fronteiras entre jogadores, deixando transparente aos desenvolvedores e jogadores as partições do mundo virtual. Porém, a utilização de *grids* não acaba com o alto consumo de banda do lado servidor do jogo.

Na realidade, as arquiteturas apresentadas por si só não são suficientes para reduzir os requisitos de comunicação e recursos necessários ao suporte de um jogo multiusuário. Algumas técnicas já existentes para ambientes virtuais em rede podem ser aplicadas como forma de atenuar a escassez de algum destes recursos. Dentre estas técnicas destacam-se a Compressão e/ou Agregação de Mensagens, o Gerenciamento de Áreas de Interesse e técnicas de previsão para diminuir o impacto da ausência de informações de atualização, comumente chamada *Dead Reckoning*<sup>3</sup>.

### 3.5 Considerações sobre o Capítulo

Os problemas decorrentes da separação física dos componentes envolvidos e os diferentes tipos de jogos multiusuário demandam requisitos que são tratados com arquiteturas e técnicas aplicadas em ambientes interativos distribuídos. Porém, no caso de PM2Gs, nem todas soluções apresentadas podem ser aplicadas diretamente, devido às suas peculiaridades.

Primeiramente, dependendo do número de usuários, a plataforma de suporte para PM2Gs pode vir a ser complexa e custosa. Embora soluções ponto a ponto tenham sido utilizadas com sucesso em certos jogos, a presença de dispositivos móveis inviabiliza seu emprego na proposta PM2G. Dispositivos móveis não possuem capacidade de processamento adequada para que parte da simulação do jogo lhes seja repassada. Além disso, a necessidade da manutenção de uma conexão rápida e confiável entre vários dispositivos é impraticável para redes sem fio, ainda mais através da Internet. Por estes fatores, jogos PM2G utilizam uma arquitetura cliente-servidor, que permite que tanto computadores pessoais quanto dispositivos móveis participem da mesma simulação, levando em consideração suas capacidades de processamento e conectividade. Embora esta solução apresente suas limitações, principalmente de escalabilidade, a mesma traz benefícios em relação à manutenção de um estado consistente do jogo e uma maior resistência à trapaça.

---

<sup>3</sup>Abreviatura de *Deduced Reckoning*

As altas latências, a intermitência e o custo associado à conexão de um dispositivo móvel via Internet dificultam a implementação de jogos multiusuário altamente interativos nas atuais infra-estruturas de redes de comunicação sem fio de longo alcance. Embora o surgimento de redes 3G traga melhorias de qualidade de serviço, testes mostram que os requisitos de jogos multiusuário altamente interativos ainda não são alcançados [BLME04] [FSR02]. Jogos PM2G propõem o aumento da interatividade entre jogadores via dispositivos móveis através da conexão direta entre jogadores, como no cenário 2 da Seção 1.2.1. O uso de tecnologias de comunicação sem fio de pequeno alcance permite que sejam construídos jogos paralelos mais interativos entre jogadores móveis, sem prejuízo para o andamento do mundo virtual global.

Por fim, jogadores móveis, mesmo em jogos monousuário, possuem um comportamento mais esporádico, com sessões curtas e espaçadas. Ao se pensar no acesso destes usuários a um jogo PM2G via Internet, tal comportamento é levado em conta, visto que as características de conexão e os modelos de negócio existentes não favorecem a criação de uma sessão semelhante a de um jogador via um computador pessoal. Para isto, em um jogo PM2G, o acesso de um jogador móvel ao mundo virtual é visto como assíncrono, onde o jogador dispara ações a serem realizadas por seu personagem, enquanto o usuário pode realizar outras tarefas. Este comportamento ainda permite poupar a bateria de dispositivos móveis, fator crítico quando estes dispositivos precisam realizar computações intensivas ou estabelecer conexões de rede constantes com o servidor do jogo, ou com outras entidades.

## Trabalhos Relacionados

*Um sábio tira mais proveito de seus inimigos, que um tolo de seus amigos.*

— BALTASAR GRACIAN

Aplicações de computação pervasiva e jogos multiusuário distribuídos tornaram-se uma área de pesquisa bastante ativa. Recentemente, a união das duas áreas têm sido proposta. Neste capítulo, apresenta-se um apanhado geral sobre as principais pesquisas relacionadas à proposta PM2G.

### 4.1 Introdução

Jogos multiusuário e computação pervasiva são dois campos recentes de pesquisa, que apresentam um ritmo acelerado de crescimento. Como apresentado nos capítulos anteriores, muitas são as questões a serem resolvidas em cada área. Em comum, ambas apresentam-se como aplicações complexas que demandam pesquisas em arquiteturas, *frameworks* e plataformas de *middleware* que auxiliem seu desenvolvimento e execução.

Este capítulo apresenta soluções propostas em ambas as áreas, no sentido de possibilitar e facilitar sua implementação. Primeiramente são apresentados soluções para computação pervasiva e em seguida, plataformas de *middleware* para jogos de diferentes gêneros: tradicionais, massivos, móveis ou ambos.

Pesquisas relacionadas a união entre computação pervasiva e jogos multiusuário são bem recentes. A Seção 4.4 apresenta a seleção de trabalhos encontrados que relacionam-se diretamente com a proposta PM2G. Ao final, são feitas considerações sobre estes trabalhos em relação à proposta PM2G.

### 4.2 *Middleware* para Computação Pervasiva

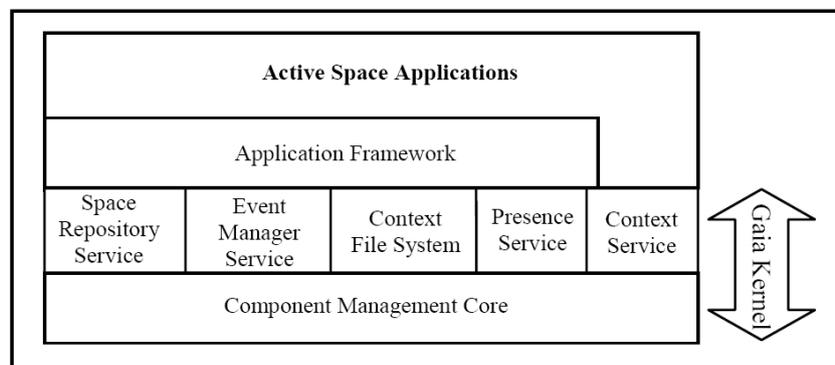
As plataformas de *middleware* para computação pervasiva apresentam como modelo comum de solução o monitoramento do contexto das aplicações e dos usuários e sua adaptação aos

mesmos. A seguir são apresentados algumas destas soluções.

#### 4.2.1 Middleware Gaia

O projeto *Gaia* (<http://www.cs.uiuc.edu/gaia>) sugere que no futuro, o espaço habitado pelas pessoas será interativo e programável. Este espaço, então chamado espaço ativo (*active space*) [Rom03], seguirá a idéia de computação pervasiva ao permitir aos seus usuários, beneficiar-se de recursos disponíveis para o acesso às informações. Para viabilizar esta visão, o projeto criou um sistema programável composto de uma coleção de dispositivos móveis e fixos que interagem por intermédio de um *middleware* chamado *Gaia*. Um espaço ativo oferece *frameworks* para a programação do sistema, adição de novos serviços dinamicamente e criação de aplicações multiplataforma. Este espaço pode ser aplicado em salas de aula, escritórios, domicílios, hospitais e ambulatórios, dentre outros espaços.

*Gaia* é um meta-sistema operacional que consiste de uma coleção de componentes distribuídos que representam um dispositivo, um serviço ou uma aplicação [RHC<sup>+</sup>02]. Este meta-sistema operacional coordena as entidades de *software* e dispositivos em rede dentro dos espaços ativos, exporta serviços para pesquisar e utilizar recursos, acessar e usar o contexto corrente, e também fornece um *framework* para desenvolver aplicações. A Figura 4.1 apresenta a arquitetura do *Gaia*.



**Figura 4.1** Arquitetura *Gaia* [Rom03]

Esta arquitetura possui três grandes blocos: *kernel*, *framework* e aplicações. O *kernel* é composto pelo gerenciamento dos componentes e por serviços fornecidos. Dentre os vários serviços oferecidos incluem-se tolerância a falhas, rastreamento de localização, segurança e manipulação de eventos. O gerenciamento de componentes distribuídos inclui carregar, descarregar, transferir, criar e destruir todos os componentes e aplicações *Gaia*. Os serviços são ativados de forma particular à localização, contexto, eventos e repositórios com informações

dos espaços ativos. O contexto refere-se a informações de presença de objetos e pessoas, bem como condições ambientais como temperatura e som.

Um *framework* baseado no padrão MVC (*model-view-controller*) permite que aplicações seja distribuídas em vários dispositivos heterogêneos. Os vários componentes de *Gaia* se comunicam via chamadas *CORBA*. O *framework* fornece mecanismos para construir, executar ou adaptar aplicações existentes em espaços ativos. Este é composto por uma infra-estrutura de objetos distribuídos, um mecanismo de mapeamento, e políticas de personalização das aplicações. Além de *CORBA*, a implementação de *Gaia* usa a linguagem de script *LuaORB* para configuração e criação de espaços ativos. São definidas quatro entidades básicas: pessoas, dispositivos, serviços e aplicações. Os recursos são descritos através de suas propriedades, expressas em XML. Para demonstrar a funcionalidade de *Gaia* foi desenvolvida uma aplicação de Gerenciamento de Apresentações.

Mais recentemente, uma nova versão do projeto *Gaia*, chamado *Super Spaces* [AMCRC04], foi proposta. Esta estende o conceito de espaços ativos original, criando agrupamento de espaços ainda maior. Dois modelos de serviços e aplicações são propostos: (i) um modelo recursivo hierárquico e (ii) um ponto a ponto. Ambos, porém, apresentam problemas de desempenho. *Super Spaces* possui um conjunto de serviços principais – nomes, descoberta, repositório de espaços, sistema de arquivos e gerência de contexto – que permite a execução de operações, sincronização de dados e atividades entre diferentes sub-espaços ativos. *Super Spaces* é um projeto em andamento, cujos resultados mais completos ainda não foram publicados.

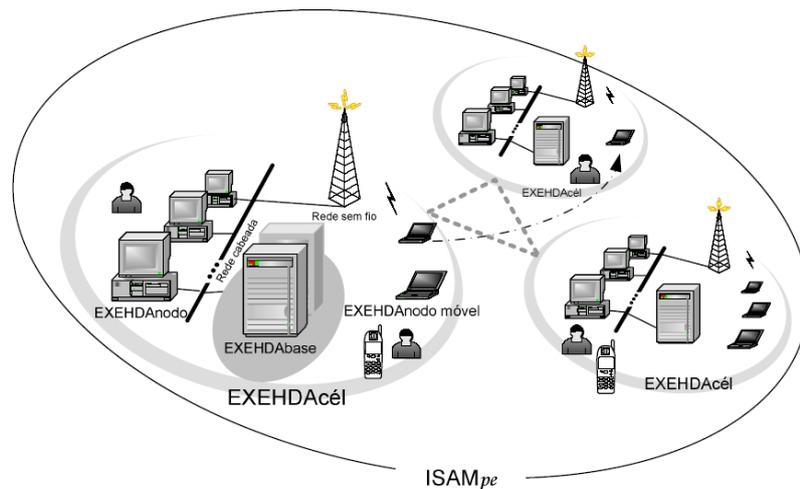
#### 4.2.2 Middleware EXEHDA

O projeto ISAM/UFRGS (<http://www.inf.ufrgs.br/isam>) tem por objetivo estudar alternativas para simplificar o projeto e execução de aplicações pervasivas de escala global [AYBG02]. Para isto, o projeto define uma chamada *semântica siga-me* para execução de aplicações, onde cada usuário de uma aplicação pervasiva possui um ambiente pessoal virtual que lhe segue em seu deslocamento, incluindo tanto sua mobilidade lógica (dados, código e processamento) quanto física (recursos e dispositivos utilizados).

O projeto define um ambiente computacional de escala global, chamado *ISAMpe* (*ISAM Pervasive Environment*), representado pela Figura 4.2.

O *ISAMpe* é formado por células compostas da união entre diversos dispositivos móveis e fixos em uma rede infra-estruturada. Este ambiente utiliza conceitos de computação em grade para organizar recursos do ambiente em uma forma hierárquica. São eles:

- *EXEHDAcel*: denota a área de atuação de uma *EXEHDAbase*, e é composta por esta e por



**Figura 4.2** ISAM Pervasive Environment [Yam04]

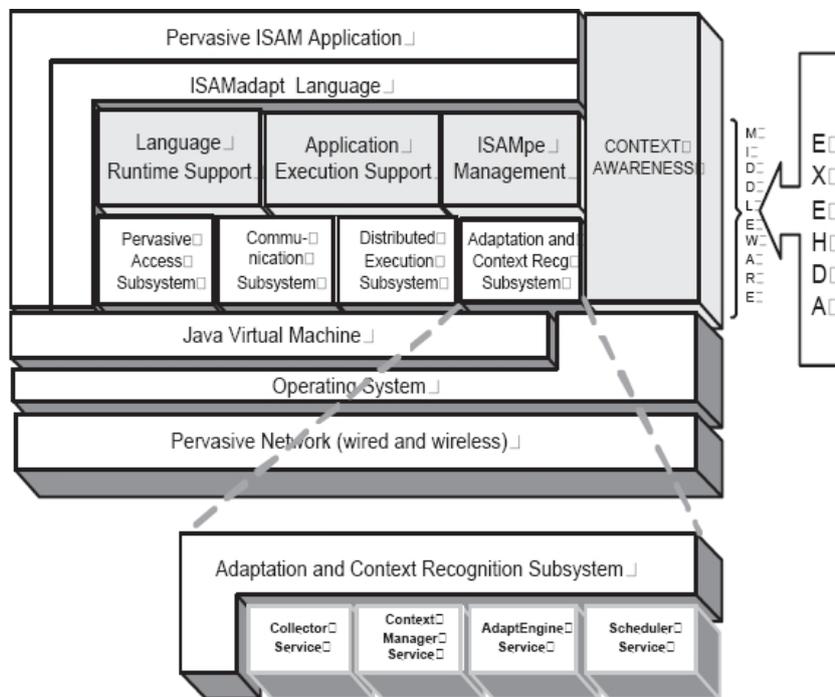
*EXEHDA nodos*. Os principais aspectos considerados na definição da abrangência de uma célula são o escopo institucional, a proximidade geográfica e o custo de comunicação;

- *EXEHDA base*: ponto de contato para os *EXEHDA nodos*. É responsável por todos os serviços básicos do ambiente de computação em grade e, embora constitua uma referência lógica única, seus serviços, sobretudo por aspectos de escalabilidade, poderão estar distribuídos entre vários equipamentos;
- *EXEHDA nodo*: são os equipamentos de processamento disponíveis no cenário, sendo responsáveis pela execução das aplicações. Um tipo específico deste recurso é o *EXEHDA nodo móvel*. Estes são nodos do sistema com elevada portabilidade, tipicamente dotados de interface de rede para operação sem fio e, neste caso, integram a célula a qual seu ponto-de-acesso está subordinado. São funcionalmente análogos aos *EXEHDA nodos*, porém eventualmente com uma capacidade mais restrita (por exemplo, PDAs).

No *ISAMpe*, dispositivos móveis não armazenam códigos ou dados, mas atuam como portais de acesso. Estes recebem código executável e podem transferir a execução de uma aplicação para outros dispositivos, de acordo com a proximidade ou disponibilidade de recursos. A aplicação é instalada sob demanda no dispositivo utilizado e sua instalação adapta-se ao contexto do mesmo. Os elementos deste ambiente computacional (código, dados, serviços, dispositivos) são dispersos e gerenciados pelo *middleware*, chamado de *EXEHDA* [Yam04], que lhes provê acesso pervasivo.

A arquitetura de *software* do ISAM foi projetada de modo a tornar o ambiente *ISAMpe* possível, possibilitando tanto o desenvolvimento quanto execução de aplicações pervasivas. Esta é organizada em camadas com níveis diferenciados de abstração, porém sem excluir os mecanis-

mos de adaptação nativos das tecnologias que compõem sua camada inferior. Esta arquitetura é apresentada na Figura 4.3. A camada superior corresponde as abstrações para o projetista da aplicação, que incluem uma linguagem para programação da aplicação, chamada *ISAMadapt* [Aug04]. A execução das aplicações ISAM são gerenciadas pelo *middleware EXEHDA*. Este componente baseia-se em serviços que colaboram entre si sobre três perspectivas: (i) gerenciamento *ISAMpe*, via serviços de controle do meio físico no qual o processamento se realizará; (ii) suporte a programação da aplicação via uma API; e (iii) suporte a execução da aplicação, com serviços e abstrações exigidas para implementação da semântica siga-me definida anteriormente. Para tal, o *EXEHDA* fornece suporte a adaptação no nível da aplicação, ao mesmo tempo que também é adaptativo.



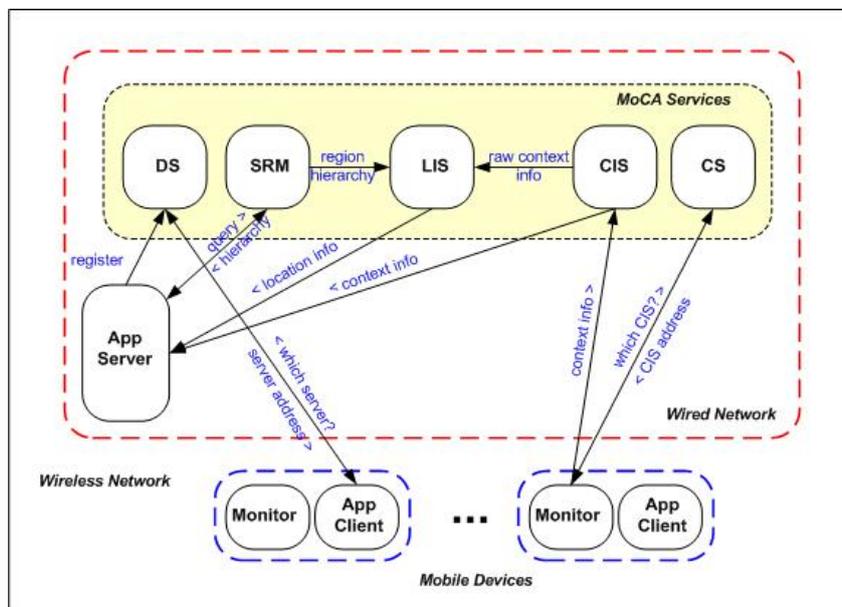
**Figura 4.3** Arquitetura de *Software* do Projeto ISAM [Yam04]

A chave para a semântica siga-me é a migração de componentes da aplicação cujo código pode ser adaptado para o estado atual do dispositivo alvo. Este estado, que reflete a idéia de contexto, engloba informações de qualquer entidade que força uma adaptação da aplicação. Informações de contexto são gerenciadas por um subsistema chamado *Context Recognition Subsystem*.

### 4.2.3 MoCA

MoCA (*Mobile Collaboration Architecture*) é um *middleware* que suporta o desenvolvimento e implantação de aplicações distribuídas sensíveis a contexto para redes locais sem fio (IEEE 802.11b/g WLANs), provendo meios para a coleta, armazenamento e processamento de informações contextuais de dispositivos móveis [SER<sup>+</sup>04]. Essencialmente, o MoCA consiste num conjunto de APIs que permite o desenvolvimento de aplicações de maneira mais simples e efetiva, provendo fácil acesso a serviços genéricos, úteis para uma grande variedade de aplicações. O MoCA não assume nenhuma arquitetura específica para a aplicação (por exemplo, cliente/servidor ou ponto a ponto). No entanto, requer que os dispositivos móveis tenham *interfaces* IEEE 802.11, e sejam capaz de se conectar a pontos de acesso (*access points*) IEEE 802.11. O MoCA envolve uma série de serviços que suportam a execução de aplicações sensíveis a contexto, que podem implantados em dispositivos utilizando Windows XP, Windows CE ou PalmOS.

Para executar sobre o MoCA, uma aplicação deve ser desenvolvida utilizando a *MoCA ClientAPI*. Se a aplicação possui uma arquitetura cliente/servidor, esta deve usar a *MoCA ServerAPI*. Estas APIs abstraem do programador muitos dos detalhes dos principais serviços MoCA. Enquanto um cliente MoCA geralmente executa em um cliente móvel, o servidor tipicamente é executado em uma rede fixa tradicional. A Figura 4.4 apresenta a arquitetura MoCA.



**Figura 4.4** Arquitetura MoCA

A *ClientAPI*, quando executada, automaticamente inicia o monitor MoCA, um *daemon* que

é executado em cada dispositivo móvel e é responsável pela coleta de dados contextuais do dispositivo e do ambiente que o cerca. Estes dados incluem: (i) a qualidade de sua conexão sem fio, (ii) carga de bateria restante, (iii) uso da CPU, (iv) memória disponível, (v) ponto de acesso ao qual o dispositivo esteja conectado em um dado instante e (vi) a lista de todos os pontos de acesso acessíveis ao dispositivo e suas respectivas forças de sinal. Todos estes dados são coletados e enviados a um serviço específico chamado Serviço de Informação de Contexto – *Context Information Service (CIS)*.

O *CIS* é um serviço distribuído onde cada Servidor *CIS*, recebe, armazena e processa os dados contextuais enviados por monitores dos dispositivos móveis. Este servidor tipicamente é executado em uma rede fixa. O servidor *CIS* também recebe pedidos para notificações (assinaturas) de aplicações-clientes e servidores, e envia eventos a estes assinantes sempre que há uma mudança no contexto de algum cliente, de acordo com o interesse do assinante. Tanto a *ClientAPI* e a *ServerAPI* disponibiliza formas de acesso aos dados de qualquer dispositivo que esteja sendo monitorado por um servidor *CIS*.

O serviço de configuração, *Configuration Server (CS)*, armazena e gerencia informações de configuração sobre cada dispositivo móvel, para que estes possam utilizar o *CIS*. O *CS* armazena as informações em um tabela *hash* persistente, onde cada entrada, indexada pelo endereço de controle de acesso ao meio (do inglês, *Media Access Control Address (MAC address)*) do dispositivo, guarda o endereço (IP e porta) do servidor *CIS* a ser utilizado pelo dispositivo. Outra informação guardada é a periodicidade que o monitor deve mandar informações do dispositivo para o *CIS*. Esta forma de endereçamento é essencial para que cada servidor *CIS* receba aproximadamente a mesma carga de processamento dos contextos dos dispositivos presentes na arquitetura MoCA.

O Serviço de Inferência de Localização, *Location Inference Service (LIS)*, é responsável por inferir a localização aproximada de um dispositivo móvel a partir das informações contextuais coletadas pelo *CIS* a partir do dado dispositivo. Este serviço compara o padrão atual do sinal de rádio frequência (RF) com padrões de sinal previamente medidos em pontos de referência pré-definidos em um determinado prédio ou área descoberta. Portanto, antes de realizar qualquer inferência, a base de dados do *LIS* deve ser criada com amostras dos sinais de RF para cada ponto de referência. Parâmetros de inferência devem ser escolhidos de acordo com características específicas da região. O número de pontos de referência determina a confiabilidade da inferência do serviço *LIS*.

*LIS* permite ao seu administrador definir regiões lógicas de tamanho arbitrário e de forma retangular, além de uma descrição hierárquica de regiões, auto-contidas ou não. Esta funcionalidade é implementada no Gerenciador de Regiões Simbólicas, *Symbolic Region Manager*

(SRM). Este componente provê uma interface para definir (criar, modificar e remover) e solicitar informações sobre hierarquias de regiões simbólicas, cujos nomes são associados a regiões físicas bem definidas (como salas ou edifícios) que podem ser de interesse para aplicações baseadas em localização. Todas estas regiões simbólicas devem ser baseadas em um conjunto de regiões atômicas definidas pelo administrador do serviço LIS.

Por fim, a *ServerAPI* automaticamente registra-se no serviço de descoberta, *Discovery Service (DS)*. Este é um servidor responsável por armazenar informações (nome, propriedade, endereço) de qualquer aplicação ou qualquer serviço registrado no *middleware* MoCA, de forma que a mesma possa ser acessada por possíveis clientes.

### 4.3 Middleware para Jogos Multiusuário

Plataformas de *middleware* para jogos multiusuário apresentam diferentes serviços de acordo com o estilo de jogo a ser implementado. Muitos destas plataformas aparecem principalmente como produtos comerciais, porém esforços para criação de *middleware* para jogos multiusuário de código aberto também já são encontrados. Alguns destes sistemas de *middleware* são apresentados nas subseções seguintes.

#### 4.3.1 DirectPlay

DirectPlay é parte integrante do Software Development Kit (SDK) DirectX [Sla03], uma solução da Microsoft para construção de aplicações multimídia sobre a plataforma Windows e Xbox. O DirectPlay é o componente do SDK responsável por abstrações para construções de aplicações multiusuário, como jogos. Este componente provê um modelo de comunicação independente de rede e dispositivo, através de uma API que disponibiliza serviços no nível das camadas de transporte e sessão. A comunicação utiliza o conceito de sessão, onde uma sessão representa uma instância de um jogo multiusuário em uma rede. Esta sessão estabelece o canal de comunicação entre vários usuários de um mesmo jogo.

A Application Programming Interface (API) DirectPlay não impõe uma arquitetura (ponto a ponto ou cliente/servidor) para uma sessão de jogo, mas disponibiliza um conjunto de funções alto nível para tratamento de conexões em uma sessão de comunicação, sobre a pilha de protocolos da plataforma Windows. DirectPlay provê uma série de funções que simplificam a implementação de vários aspectos de uma aplicação multiusuário, como:

- Criação e gerenciamento de sessões utilizando topologias cliente/servidor ou ponto a

ponto;

- Gerenciamento de usuários individuais ou em grupos dentro de uma sessão de comunicação;
- Gerenciamento da troca de mensagens entre membros de uma sessão, mesmo através de diferentes tipos de rede e sobre diferentes condições de tráfego, e;
- Uso de comunicação via voz.

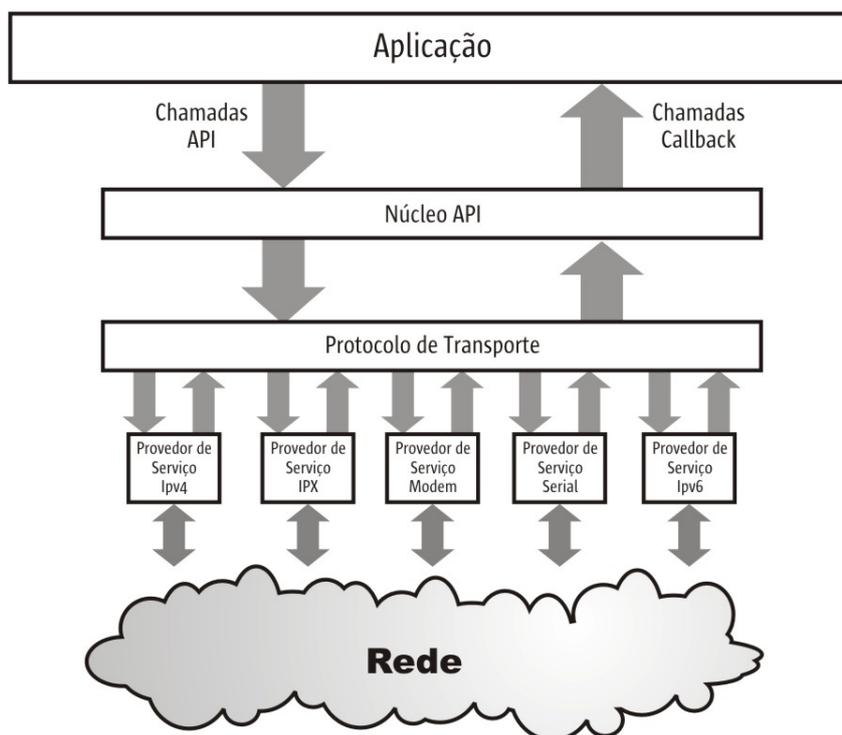
DirectPlay consiste de três camadas distintas: a camada núcleo, a camada de protocolo e a camada de provedores de serviços. A aplicação interage diretamente com a camada núcleo, através de chamadas às funções da API. A camada núcleo é responsável por tarefas como configuração da sessão de jogo e manutenção da lista de jogadores. DirectPlay utiliza um modelo *push* de troca de informação. Desta forma, toda aplicação deve registrar uma função de *callback*, que será chamada pela camada núcleo quando mensagens relevantes forem recebidas pela camada de protocolo. A camada de protocolo é responsável pela construção e interpretação de pacotes, sendo construída sobre o protocolo *User Datagram Protocol (UDP)*. A camada de provedores de serviço disponibiliza diferentes meios de comunicação para uso em uma sessão, como TCP/IP, *Internetwork Packet Exchange (IPX)* ou uso de portas seriais. O conceito de provedor de serviço encapsula o acesso destes diferentes meios de comunicação, de tal forma que o desenvolvedor não precisa lidar com problemas relativos a diferentes meios de comunicação existentes. A Figura 4.5 apresenta uma visão simplificada desta arquitetura.

DirectPlay é baseado fortemente na tecnologia Component Object Model (COM), consistindo de uma coleção de objetos COM. Cada objeto expõe uma ou mais interfaces que permitem ao desenvolvedor controlar vários aspectos de um jogo. Por exemplo, para enviar dados para outro usuário em um jogo Peer-to-Peer (P2P), o desenvolvedor deve utilizar o método `SendTo` do objeto `CLSID_DirectPlay8Peer`, fornecido pelo DirectPlay SDK. Sendo baseado fortemente na tecnologia COM, seu uso é restrito à plataforma Windows.

DirectPlay disponibiliza ainda avançados serviços no nível de transporte, como entrega garantida ou não de mensagens, controle (*throttling*) de tráfego para ligações de baixa qualidade e detecção de queda de conexão. São providos ainda serviços de controle de sessão como migração do gerenciador de uma sessão P2P.

### 4.3.2 TeraZona

Terazona[Ter02] é a solução desenvolvida pela empresa Zona Inc.® para o suporte a MMGs. Através de um *framework* de software, é fornecida uma camada abstrata de rede especialmente projetada para resolver questões de escalabilidade e confiabilidade de MMGs. Segundo dados



**Figura 4.5** Arquitetura DirectPlay

da empresa, a solução Terazona permite até 32 mil jogadores por aglomerado de servidores, enquanto as soluções concorrentes suportam em média, dois jogadores. Através da união deste aglomerados, a solução pode suportar centenas de milhares de jogadores.

A arquitetura Terazona é constituída de múltiplas camadas, cada uma com uma responsabilidade específica. Cada camada é formada por componentes extensíveis e reutilizáveis baseados em objetos distribuídos, agrupados por funcionalidades, oferecendo seus serviços via uma interface. Segurança é conseguida por criptografia de mensagens, e certos tipos de trapaça podem ser combatidos pelo monitoramento do comportamento do jogador. A Figura 4.6 ilustra o *framework* Terazona.

Este *framework* é dividido em duas partes: um relativo ao processo servidor do jogo e outro relativo aos jogadores. A API do lado servidor trata de questões necessárias para efetiva distribuição do estado do jogo entre os jogadores. O lado servidor do *framework* recebe os estados de cada jogador, e utiliza seus componentes para integrá-los ao jogo.

O componente de validação (*Validation Engine*) é responsável pela validação dos estados de cada jogador. Este componente garante que apenas estados que sejam validados pelo servidor, possam ser propagados para o jogo e para outros jogadores. Por exemplo, o estado de um jogador pode ser comparado com seu estado anterior para evitar trapaça.

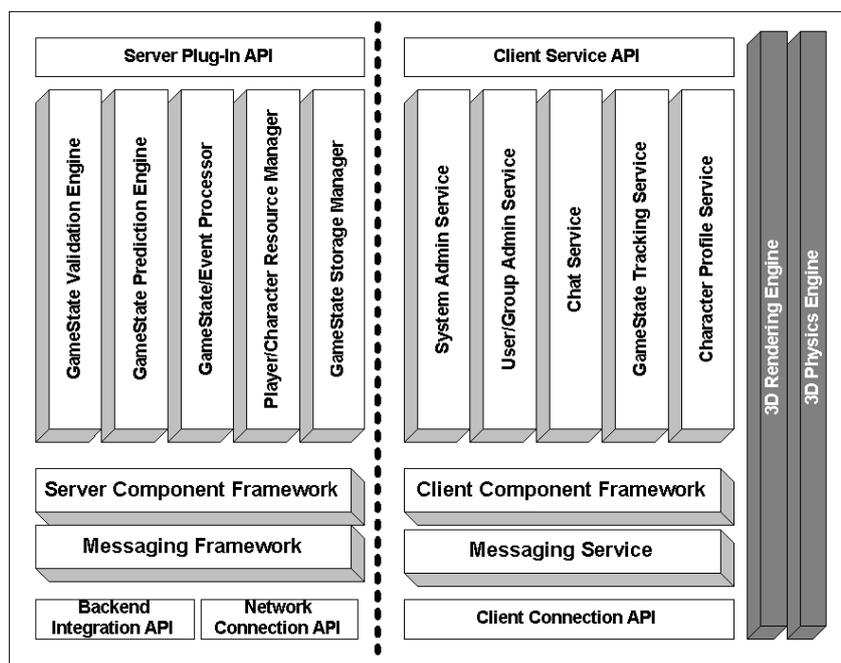


Figura 4.6 Arquitetura Terazona

O componente de estimativas (*Prediction Engine*) compensa eventuais atrasos no processamento das ações dos jogadores, através de interpolação e extrapolação de valores. O componente de eventos (*Event Engine*) é responsável pela propagação de eventos no jogo, bem como pela comunicação entre jogadores. O gerenciador de recursos de jogadores e personagens (*Player/Character Resource Manager*) gerencia os atributos do avatar de cada jogador, enquanto o *Storage Manager* gerencia a persistência dos estados dos jogadores.

O lado cliente do *framework* apresenta uma camada de funções para o desenvolvedor, que abstrai a distribuição de servidores, criando a idéia de que existe um único servidor central. As funcionalidades do *framework* são oferecidas aos clientes como serviços. O Serviço de Administração gerencia tarefas do servidor, como início ou parada da sessão de um jogo ou de servidores de conversas textuais (*chat*). O Serviço *Sysadmin* permite opcionalmente acesso a funções de controle remoto de todos servidores, mesmo através de *firewalls*.

O módulo Usuário/Grupo permite que usuários sejam monitorados, enquanto o Serviço de Chat (*Chat Service*) permite comunicação tanto entre jogadores, quanto entre jogadores e administradores. O Serviço de Acompanhamento do Estado do Jogo (*Game State Tracking service*) é utilizado para o envio e recebimento de mensagens do servidor, enquanto o Perfil de Entidade permite que jogadores criem ou atualizem novos personagens. Por fim, Terazona permite que desenvolvedores integrem motores de renderização e de modelagem física do jogo,

como elementos externos do *framework*.

A camada inferior do *framework* contém o serviço de mensagens e camada de integração. Estas entidades são responsáveis pela comunicação entre clientes, servidores e elementos de terceiros, como servidores de tarifação, bancos de dados ou de autenticação. A camada do serviço de mensagens (*Messaging Service Layer*) utiliza *Java Message Service (JMS)*, um *middleware* orientado a mensagens para comunicação entre os componentes do *framework*.

### 4.3.3 Butterfly Grid

Butterfly Grid<sup>®</sup> é uma proposta da empresa Butterfly.net [But] para hospedagem de MMGs, baseada no uso de computação em grade. Nesta solução, os servidores utilizados pelas empresas são substituídos pelo *Grid* da empresa. A empresa fornece dois serviços para as empresas que desenvolvem e comercializam os MMGs: Aluguel de uma infra-estrutura de computação em grade, dotada de centros de processamento de dados poderosos, interconectados por fibra ótica ou outras tecnologias de rede de alta velocidade, que por sua vez gerenciam uma rede de provedores de serviço distribuída pelo globo; ou Licença de um *middleware* que atua como uma interface entre o Grid e o MMG.

A Figura 4.7 mostra uma visão simplificada do Butterfly Grid. O Grid possui um “centro” onde ficam hospedadas as máquinas da Butterfly.net. Neste centro localizam-se servidores mestres de dados e outros serviços essenciais, além dos servidores de jogo propriamente ditos. O servidor de jogo é o que se conecta diretamente com os clientes dos vários MMGs que são servidos pelo Grid. Conectados ao “centro” do Grid, e entre si de forma dinâmica, estão servidores de jogo hospedados por provedores de serviço associados, por exemplo, provedores de acesso à Internet ou provedores de jogos e conteúdo.

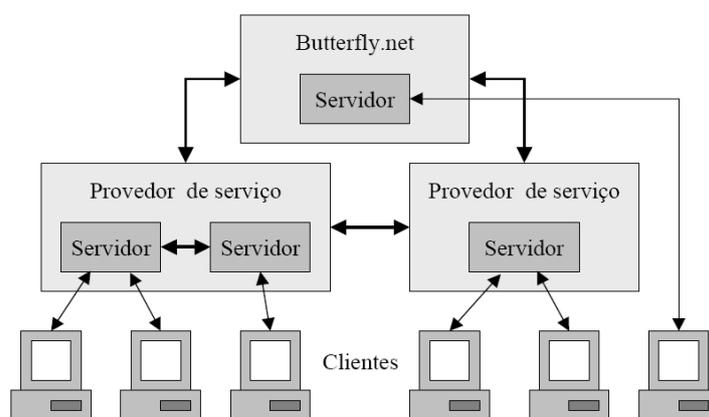


Figura 4.7 Visão Simplificada da plataforma “Butterfly Grid”[Cec05]

Esta solução oferece às empresas desenvolvedoras de MMGs duas vantagens principais em relação aos paradigmas fortemente centralizados ou puramente cliente-servidor:

1. Custo compatível com a demanda: a empresa só paga pela parcela de *hardware* que estiver utilizando. No caso do *Grid*, o *hardware* disponível se adapta à demanda, e a empresa não precisa se preocupar em “adivinhar” o número de usuários que terá inicialmente;
2. Eliminação de mundos virtuais fragmentados: com a distribuição do processo servidor, torna-se necessária a introdução de um rígido processamento de sincronização entre os vários processos simuladores do mundo virtual. Como um *Grid* possui *hardware* poderoso e, principalmente, uma rede de baixíssimo atraso e perdas entre os processos simuladores, a sincronização em tempo-real dos simuladores se torna bastante facilitada.

Porém, apesar de resolver em parte o problema da gerência de recursos relacionados ao provimento de um MMG, o *Butterfly Grid* ainda depende da existência de uma infra-estrutura especial de *hardware*. Além de redes de alta velocidade para conectar os vários provedores de serviço, o *Butterfly Grid* ainda conta com o centro de processamento de dados da empresa *Butterfly.net*. Este centro possui um papel-chave no funcionamento do *Butterfly Grid* e, portanto, atua como uma entidade centralizadora. Esta centralização indica que o consumo de rede total entre clientes e servidores deve continuar na mesma ordem de grandeza.

O uso de computação em grade traz ainda consigo o problema não-trivial do escalonamento das máquinas, para alocação de cada MMG hospedado pelo grid. Para cada máquina do *Grid*, um escalonador deve decidir quais máquinas “físicas” vão compor o servidor “lógico” do MMG.

#### 4.3.4 GASP

GASP (*G*Aming *S*ervices *P*latform) [PDGSS05] é um *middleware* desenvolvido pelo grupo ObjectWeb [Obj05], que permite a implementação de jogos multiusuário para dispositivos móveis baseados na topologia cliente/servidor, que utilizam a rede General Packet Radio Service (GPRS) de uma operadora de telefonia, como meio de comunicação do jogo. GASP é um projeto de *software* aberto baseado na especificação dos Serviços de Jogos do consórcio *Open Mobile Alliance (OMA)* [OMAOB] em conjunto com o *Mobile Games Interoperability Forum (MGIF)* [OMAOa].

O consórcio Open Mobile Alliance (OMA) foi formado em 2002 por uma associação de mais de 200 empresas de diversas áreas de atuação: operadores de telefonia móvel, empresas de manufatura de dispositivos móveis, empresas especializadas em redes de comunicação, nas tecnologias da informação e fornecedores de conteúdo, como Nokia, IBM, Motorola, SI-

EMENS, Intel e Sun. Este consórcio visa realizar esforços de padronização motivados pela vontade de homogeneização e interoperabilidade de tecnologias para computação móvel.

Por sua vez, Mobile Games Interoperability Forum (MGIF), grupo já existente formado por Ericsson, Motorola, Nokia e SIEMENS, tinha o objetivo de criar serviços específicos para os jogos móveis. Este grupo juntou-se ao consórcio OMA, criando o grupo de trabalho de Serviços de Jogos – *Games Services*. O grupo lançou uma especificação de uma arquitetura de jogos para dispositivos móveis baseada na topologia cliente/servidor, com dois objetivos prioritários: a interoperabilidade, a fim de compartilhar recursos ou serviços, e a mobilidade. Esta arquitetura OMA/MGIF é especificada na forma de uma API Java, com interfaces para um conjunto de serviços básicos. Dentre alguns serviços, pode-se citar: gerência de sessão, conectividade, tarifação, logging, temporização e gerência de pontuação. O GASP é uma implementação parcial desta arquitetura proposta.

Na proposta GASP, as interfaces de comunicação da especificação OMA relativos à lógica do jogo entre cliente e servidor são respeitadas. Porém, o modelo de funcionamento interno da arquitetura é estendido. Este modelo permite um esclarecimento dos papéis de cada classe e uma otimização do número de classes instanciadas na arquitetura. Uma visão da arquitetura proposta é apresentada na Figura 4.8. Esta arquitetura compõe-se de vários módulos interrelacionados. O Módulo de Representação da Plataforma segue o padrão OMA, com classes que definem a representação de sessões de jogadores. O Servlet Container é responsável pela comunicação da plataforma GASP, recebendo as mensagens dos jogadores móveis. Os Servidores do Jogo são responsáveis pela lógica específica de cada jogo. Há ainda uma base de dados para informações de dados de contas de usuários que pode ser compartilhada entre os jogos.

Os futuros serviços de GASP deverão permitir a concepção de jogos que integram as novas funções dos dispositivos móveis e os seus acessórios: *streaming* de vídeo, geolocalização, equipamentos Bluetooth.

- *Gerência de sessão*: principal serviço da API, provê identificadores que conectam as interações do usuário no conceito de um jogo. Provê acesso aos demais serviços e fornece uma interface na qual o ciclo de vida de entidades do jogo são gerenciadas.
- *Conectividade*: provê camadas de comunicação que abstraem do desenvolvedor, detalhes de baixo nível relacionado com o mecanismo de transporte utilizado.
- *Tarifação*: cria mecanismos pelos quais eventos de uma sessão podem ser tarifados.
- Gerência de pontuação provê meios pelos quais o jogo pode atualizar ou recuperar pontuações de jogadores.
- *logging*: serviço utilizado para criação de logs de acompanhamento do status de um jogo. Serviço importante especialmente para tarefas de depuração.

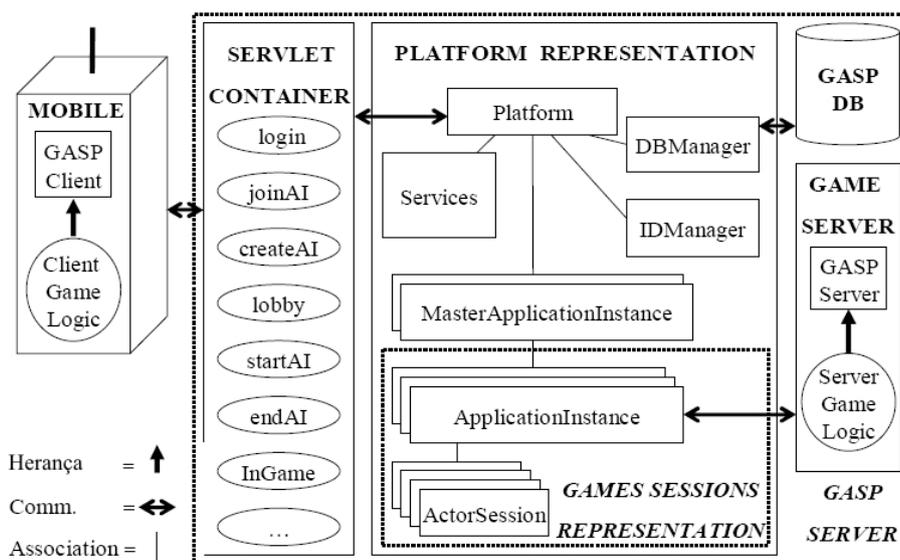


Figura 4.8 Arquitetura GASP [PDGSS05]

- *Temporizadores*: mecanismo pelo qual o jogo pode agendar ações baseados no tempo.
- *Gerência de pontuação*: provê meios pelos quais o jogo pode atualizar ou recuperar pontuações de jogadores.

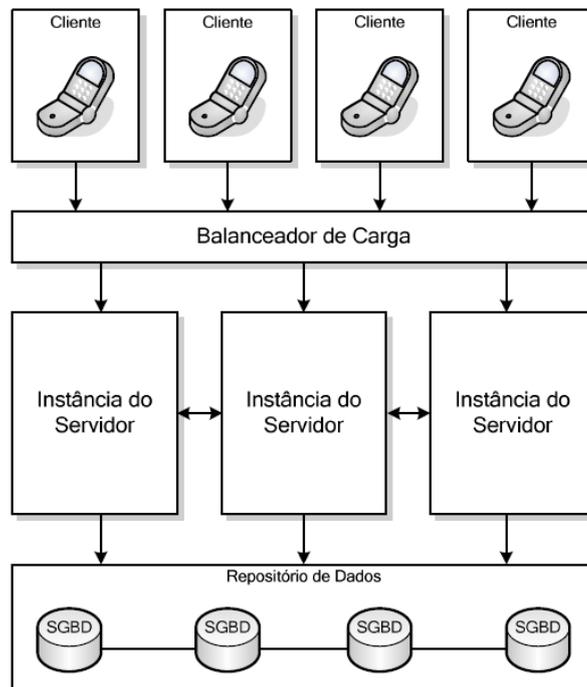
#### 4.3.5 Plataforma Aberta 3MOG

O projeto 3MOG é uma iniciativa inter-institucional realizada pela Empresa Meantime (<http://www.meantime.com.br>), O Centro de Estudos Avançados do Recife (<http://www.cesar.org.br>), Centro de Informática da Universidade Federal de Pernambuco (<http://www.cin.ufpe.br>), Universidade Estadual de Campinas (<http://www.unicamp.br>) e Intel (<http://www.intel.com.br>) para a especificação e implementação de uma plataforma aberta para desenvolvimento e execução de jogos móveis massivamente multiusuário.

A plataforma 3MOG [MSA<sup>+</sup>06] inclui serviços para manipulação de contas dos usuários, ranqueamento, pontuação e notícias, dentre outros, seguindo uma topologia cliente/servidor. Os serviços são categorizados em dois grupos: (i) os de propósito geral, inerentes a qualquer jogo, e (ii) os específicos a um jogo, como a lógica do mesmo.

As principais preocupações do projeto relacionam-se com questões de escalabilidade, distribuição de carga, tolerância a falhas, gerenciamento de transações e persistência de dados dos jogos executados sobre a plataforma 3MOG. Em relação a escalabilidade, esta foi obtida através da distribuição de computadores servidores do jogo. A aplicação de uma abordagem

simplificada de revezamento (*round-robin*) entre os servidores proporciona o balanceamento de carga entre os servidores escolhidos. Tolerância a falhas é alcançada via replicação dos dados entre os servidores, onde o estado do jogo é compartilhado em um meio de persistência. Persistência esta alcançada pela utilização de Sistemas Gerenciadores de Banco de Dados (SGBD) tradicionais, com um esquema de *cache* em memória volátil para dados acessados mais frequentemente. Por fim, transações são tratadas com a retenção (*locks*) de acesso de recursos compartilhados, enquanto os mesmos estiverem na execução de alguma transação. Como estas determinações, a Figura 4.9 apresenta a arquitetura da plataforma.

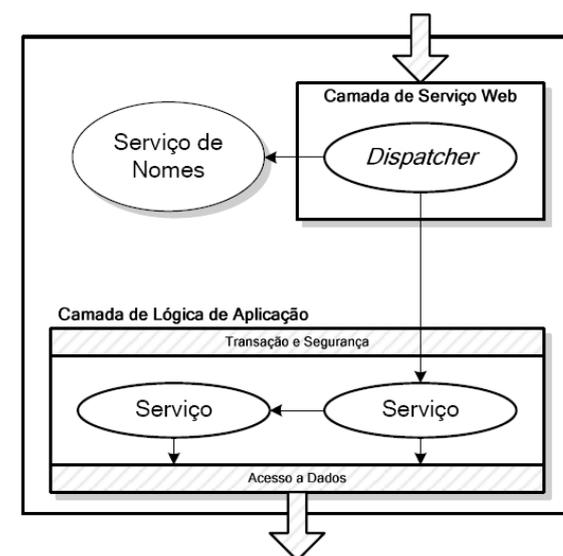


**Figura 4.9** Arquitetura 3MOG

A arquitetura 3MOG compõem-se de quatro partes:

- *clientes* que acessam remotamente a plataforma;
- Um *balanceador de carga*, responsável pela escolha de uma das instâncias do servidor para tratar a solicitação do cliente;
- Conjunto de *instâncias do servidor*, que processam as requisições, e se necessário, acessam o repositório de dados;
- *Repositórios de Dados*, que armazenam e recuperam informações do jogo.

Destes componentes, o servidor do jogo recebe uma particular atenção, por utilizar os serviços definidos pela plataforma. A Figura 4.10 apresenta a arquitetura interna de um servidor 3MOG.



**Figura 4.10** Arquitetura da instância de um servidor 3MOG

Cada instância de um servidor possui os seguintes componentes:

- O *Dispatcher*, responsável por receber a solicitação do balanceador de cargas, decodificar os parâmetros da mensagem, localizar (através do serviço de nomes) e invocar o serviço a ela atrelado e, por fim, codificar a resposta para o cliente;
- O *Serviço de Nomes*, que contém referências para os serviços disponíveis na plataforma;
- *Camada de Transação e Segurança*, que intercepta chamadas aos serviços, provendo à elas contexto transacional e de segurança;
- Os *Serviços*, que contêm a lógica de negócio do 3MOG. Vale salientar que estes devem estar preparados para lidar com esta situação de distribuição;
- *Camada de Acesso a Dados*, responsável por prover abstrações para os serviços que simplificam o acesso ao repositório de dados.

Para validar a plataforma foi projetado e implementado o jogo *CellMons*, uma batalha em turnos entre jogadores que devem evoluir seus mascotes (chamados *cellmons*) através de cuidados e treinamentos. Os desafios realizados por jogadores lhes proporcionam ganhar pontos e experiências, sendo que o jogador a ganhar mais pontos torna-se o campeão da liga mundial de *CellMons*.

A plataforma foi implementada em duas versões utilizando a linguagem Java: uma cuja implementação abrange toda a infra-estrutura de serviços e outra que adapta uma plataforma genérica às necessidades de um 3MOG, utilizando bibliotecas de código aberto. Enquanto a primeira apresentou um melhor desempenho em termos de tempo de resposta às requisições dos clientes, a segunda reduziu drasticamente o tempo de desenvolvimento da plataforma, sem

grandes penalidades no desempenho.

## **4.4 Middleware para Jogos Pervasivos Multiplataforma Multiusuário**

Como ressaltado anterior, a idéia de utilizar conceitos de computação pervasiva em jogos multiusuário é muito incipiente. Como resultado, poucos são os trabalhos encontrados na literatura que se relacionam mais diretamente com o tema. Estes trabalhos podem ser divididos em dois grupos. Primeiro, existem trabalhos motivadores que apresentam as vantagens da união entre dispositivos móveis e jogos multiusuário distribuídos, porém sem apresentar soluções concretas de sua realização. Segundo, há trabalhos que apresentam propostas de infra-estrutura para a realização de diferentes visões para jogos pervasivos. Esta seção procura apresentá-los, em uma ordem cronológica de publicação.

### **4.4.1 Proposta de David Fox**

Fox [Fox03] foi o primeiro a propor a integração entre MMGs e dispositivos móveis, onde o autor destaca os pontos positivos desta integração, como o acesso irrestrito a jogos persistentes e a notificação de eventos via serviços específicos da operadora. Dentre os principais pontos a serem estudados em seu artigo, o autor ressalta a necessidade da adaptação da jogabilidade de jogadores via diferentes dispositivos em um MMG. Esta jogabilidade englobaria não apenas a forma como o jogador visualiza o jogo, mas também como cada jogador interage com o mundo virtual.

Em relação ao envio do estado do jogo, o autor sugere o uso de diferentes representações de cada elemento do jogo (objetos, avatares), de tal forma que estes sejam personalizados de acordo com a plataforma alvo. Conceitos concretos dentro um MMG devem se tornar um tanto abstratos em um dispositivo móvel, como a idéia de ícones ou outras simplificações. Exemplos de possíveis transformações são apresentadas na Tabela 4.1.

Em relação ao tratamento da entrada do jogador, o autor sugere o estabelecimento de papéis de jogador e treinador, de acordo com a plataforma utilizada. Ao usar um dispositivo, o projeto adequado do jogo pode permitir que as interações de um jogador sejam feitas através de ações gerenciais (administrativas), onde o controle é feito através de ordens de “alto nível”, em vez do controle efetivo de seu(s) avatar(es). Após a ordem do jogador, técnicas de Inteligência Artificial podem ser utilizadas de modo simular o comportamento do avatar. Qualquer ação de jogabilidade que se baseie em reflexos rápidos precisa ser alterada.

**Tabela 4.1** Possíveis transformações de elementos de um jogo para diferentes plataformas (Adaptado de [Fox03])

	Console/PC	Dispositivo Móvel
<b>Personagens</b>	Representações Animadas	Ícones em um mapa
<b>Interações entre jogadores</b>	Precisas, de acordo com interfaces disponíveis	Textual ou através de Ícones
<b>Estratégias</b>	Sob demanda	Planejada através de diagramas, com possibilidade de atualização
<b>Locais Fechados</b>	Salas separadas por corredores com portas fechadas, etc	Representação plana do local, com ícones indicando salas, portas, etc
<b>Locais Abertos</b>	Representação gráfica de torres, recursos naturais, ruínas, etc	Mapa com ícones para locais importantes do mundo
<b>Inventário de itens</b>	Menu gráfico estilo <i>drag and drop</i>	Menu com caixas de seleção ( <i>checkboxes</i> )
<b>Conversação Narrativas</b>	Voz ou texto livre	Mensagens pré-definidas
<b>Tática do usuário</b>	Entradas baseadas no reflexo do usuário	Uso de Inteligência Artificial

Este artigo foi o principal motivador deste trabalho de doutorado, uma vez que suas soluções são apresentadas em um alto-nível, sem o detalhamento de como suas visões seriam de fato realizadas.

#### 4.4.2 Grupos Focais realizados por Koivisto e Wenninger

Koivisto e Wenninger [KW05] realizaram um levantamento sobre quais seriam as funcionalidades interessantes na integração da dispositivos móveis, especificamente telefones celulares, em jogos massivos tradicionais. Foram realizados seis grupos focais (*focus groups*), sendo cinco compostos por jogadores e um por desenvolvedores de jogos, onde foram avaliadas seis categorias de funcionalidades. O grupo de entrevistados possuía faixa etária entre 17 e 40 anos, sendo que 13% eram mulheres. As categorias foram definidas baseadas nas possíveis formas de como um jogador poderia interagir ou influenciar o mundo virtual ou outros jogadores utilizando seu telefone celular. As seis categorias são:

1. **Comunicação entre usuários fora do mundo virtual:** canais de comunicação permitiriam que usuários pudessem trocar mensagens via texto ou voz, enquanto utilizassem dispositivos móveis;
2. **Notificação de Eventos:** O uso de dispositivos móveis permitiria o envio de informações sobre o andamento do jogo, mesmo quando estes estivessem fora do mundo virtual;
3. **Possibilidade de jogabilidade assíncrona:** Realização de tarefas que não demandassem interação em tempo-real, como desenvolvimento de habilidades do personagem ou oferta ou compra de objetos virtuais em um possível sistema de leilões do jogo;
4. **Possibilidade de jogabilidade síncrona:** Permitir o usuário orientar seu personagem a realizar certas ações no mundo virtual;
5. **Participação Passiva:** Permitir que certas decisões de interesse global do mundo virtual possam ser tomadas através de um sistema de votação ou ranqueamento, onde usuários em dispositivos móveis também possam opinar;
6. **Realidade Paralela:** Fazer com certas ações do mundo real influenciem o mundo virtual.

De acordo com as entrevistas realizadas, os autores relatam que a primeira categoria foi bem aceita por jogadores e desenvolvedores, porém com ressalvas. Certos jogadores alegaram que o uso de voz poderia “quebrar” a imersão no jogo. Porém, a maioria dos jogadores alegou haveria uma nova possibilidade para discussões e aprendizado dos jogos a partir do contato com outros jogadores.

A segunda categoria recebeu críticas, indicando ser necessário algum tipo de controle sobre a funcionalidade. Jogadores argumentaram haver um aumento da pressão de estar sempre presente, recebendo informações e certas vezes não poderem fazer nada. Já desenvolvedores sentiram-se responsáveis em proteger o jogador de efeitos colaterais, como excesso de notificações. Segundo autores, esta categoria mostra-se mais adequada a jogadores que passam muito tempo no jogo, chamados de *hardcore gamers*.

A terceira categoria foi bem aceita, onde jogadores observaram dois potenciais uso da funcionalidade: (i) comércio ou troca de itens entre jogadores e a (ii) realização de tarefas então consideradas entediantes, que poderiam ser realizadas ou parcialmente iniciadas através do celular.

A quarta categoria ressalta o problema da competição justa (*fairness*) para interações síncronas. Jogadores em diferentes plataformas não podem ter grandes vantagens ou desvantagens apenas pelo uso de uma plataforma diferente. Uso de técnicas de inteligência artificial é constatado como uma forma de reduzir esta desigualdade. Outra alternativa é a adaptação do projeto do jogo a esta realidade, como forçar que ações críticas sejam projetadas para turno adequadas a latência de dispositivos móveis. O uso de inteligência artificial criou a preocupação entre os

desenvolvedores em não transformar o jogo em um mundo de caracteres automatizados. Entre os jogadores, certas ações, como deslocamentos entre partes distantes do mundo receberam avaliação positiva.

Sobre a quinta categoria, jogadores mostraram-se interessados em observar amigos ou jogadores mais experientes. Desenvolvedores, entretanto, argumentaram que um jogo deve sempre ser uma atividade interativa, e não passiva, colocando esta categoria como um aspecto menos importante. A idéia de votações foi bem aceita tanto por jogadores quanto desenvolvedores, com a ressalva que estas votações não afetem em muito, o mundo virtual.

Em relação a última categoria, jogadores, especialmente os mais velhos, não demonstraram muito interesse na mesma. Seus argumentos são que eles buscam esquecer a realidade enquanto jogam. Jogadores mais novos acharam a idéia interessante, mas futurística. Entre todos os jogadores, a maior preocupação é em relação a privacidade, questão relacionada a possibilidade de entrar em contato com pessoas desconhecidas enquanto jogam e misturam a ação real com a virtual. Enquanto jogadores mais novos mostraram-se entusiasmados com a idéia, jogadores mais velhos se mostraram mais reticentes à mesma.

#### 4.4.3 *Middleware* sensível à situação aplicado a jogos

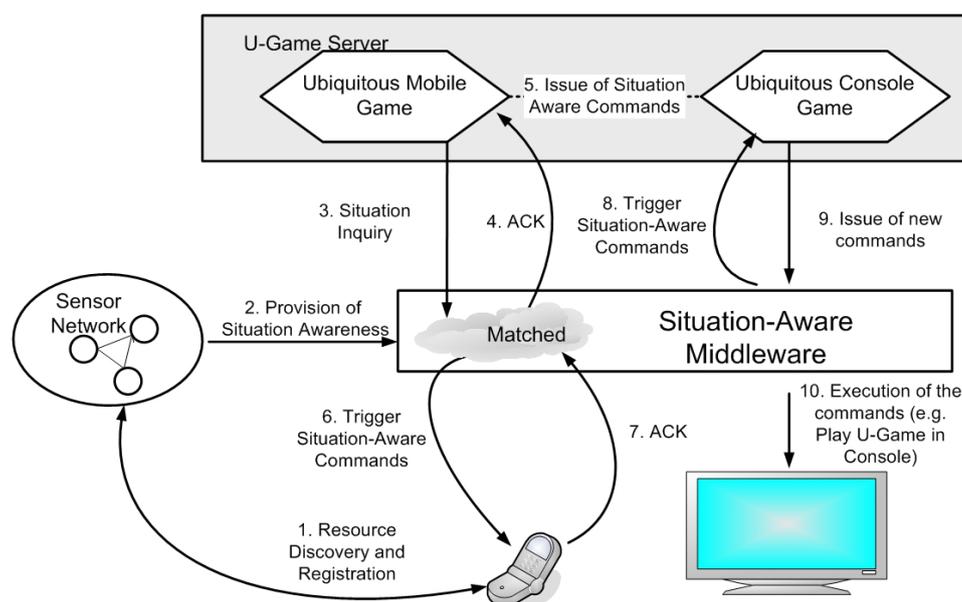
Han *et al* [HIW04] apresentam a proposta de utilização de um *middleware* sensível à situação (*situation-aware middleware*) para a execução de jogos multiplataforma. Cinco tipos de plataforma são suportadas: consoles, máquinas comerciais (*arcade machines*, ex: flipperama), computadores pessoais, PDAs e telefones celulares. As três primeiras são classificadas como plataformas *high-end*, enquanto as duas últimas, como *low-end*. Os jogadores de uma plataforma podem migrar o estado de seus jogos entre diferentes plataformas, de acordo com a situação do jogador. A arquitetura proposta por Han *et al* define a existência de três componentes principais: (i) clientes multimídia 3D, (ii) um servidor de jogo/multimídia multiplataforma e (iii) um *middleware* sensível a situação.

Os clientes das plataformas *high-end* utilizam as APIs OpenGL e DirectX para seu desenvolvimento, enquanto plataformas *low-end* utilizam OpenGL ES (*OpenGL for Embedded Systems*), um subconjunto de OpenGL projetada para dispositivos embarcados. O servidor multiplataforma do jogo consiste de um conjunto de componentes chamado *ubiquitous game framework*. Os principais componentes deste *framework* são: (i) o *Game Container*, que disponibiliza a API para o servidor do jogo sobre a qual a lógica do mesmo é implementada, (ii) um servidor que funciona como um *broker* pedido-resposta, e o (iii) servidor distribuído que dinamicamente aloca e desaloca recursos baseados em um rede virtual. Fazem parte ainda

como suporte ao servidor: um banco de dados do jogo, um servidor de tarifação e um servidor para Gerenciamento de Relacionamento com o Cliente, em inglês, *Customer Relationship Management (CRM)*.

O *middleware* utiliza informações de sensores que detectam situações (como presença de um dispositivo melhor adequado para o jogador) e comportamento associado a cada situação (por exemplo, transferência do estado entre dispositivos).

A Figura 4.11 apresenta a utilização do *middleware* sensível à situação aplicado a jogos. Dada a existência de um jogador utilizando um telefone celular que entra em uma sala e quer continuar seu jogo em um console que tenha uma melhor condição de jogabilidade, como um console de videogame. Uma rede de sensores descobre a entrada do telefone celular e registra informações sobre este dispositivo (Passo 1).



**Figura 4.11** Uso de um *middleware* sensível a situação no cenário de um jogo ubíquo (Adaptado de [HIW04]).

A rede de sensores informa ao *middleware* sensível a situação que o telefone celular entrou na sala. O servidor do jogo utiliza o *middleware* para saber se o console e o telefone estão próximos, de acordo com sua localização (Passo 3). O *middleware* em seguida, responde ao servidor do jogo (Passo 4), que por sua vez, solicita comandos ao telefone celular através do *middleware* (Passos 5 e 6). Um exemplo de um possível comando é arguir o usuário se o mesmo deseja transferir o jogo para o console. Caso o usuário concorde (Passo 7), o *middleware* aciona o servidor (Passo 8) e então o jogo é transferido para o console (Passos 9 e 10).

Para a implementação dos cenários descritos, uma linguagem de definição de interfaces

sensíveis a situação, *Situation-Aware Interface Definition Language (SA-IDL)*, foi definida para realizar a comunicação entre o servidor do jogo e o *middleware*. O *middleware* interpreta esta linguagem e gera código executável para várias plataformas. Os detalhes específicos de cada plataforma são escondidos dos desenvolvedores, que precisam apenas descrever as situações e suas respectivas ações. Um gerenciador sensível a situação foi desenvolvido, de modo a monitorar informações contextuais obtidas via rede de sensores e interpretá-las como “situações”. Este gerenciador notifica o servidor do jogo, repassando que a situação interpretada corresponde a alguma situação predefinida (Passos 2 a 4).

#### 4.4.4 Iperg

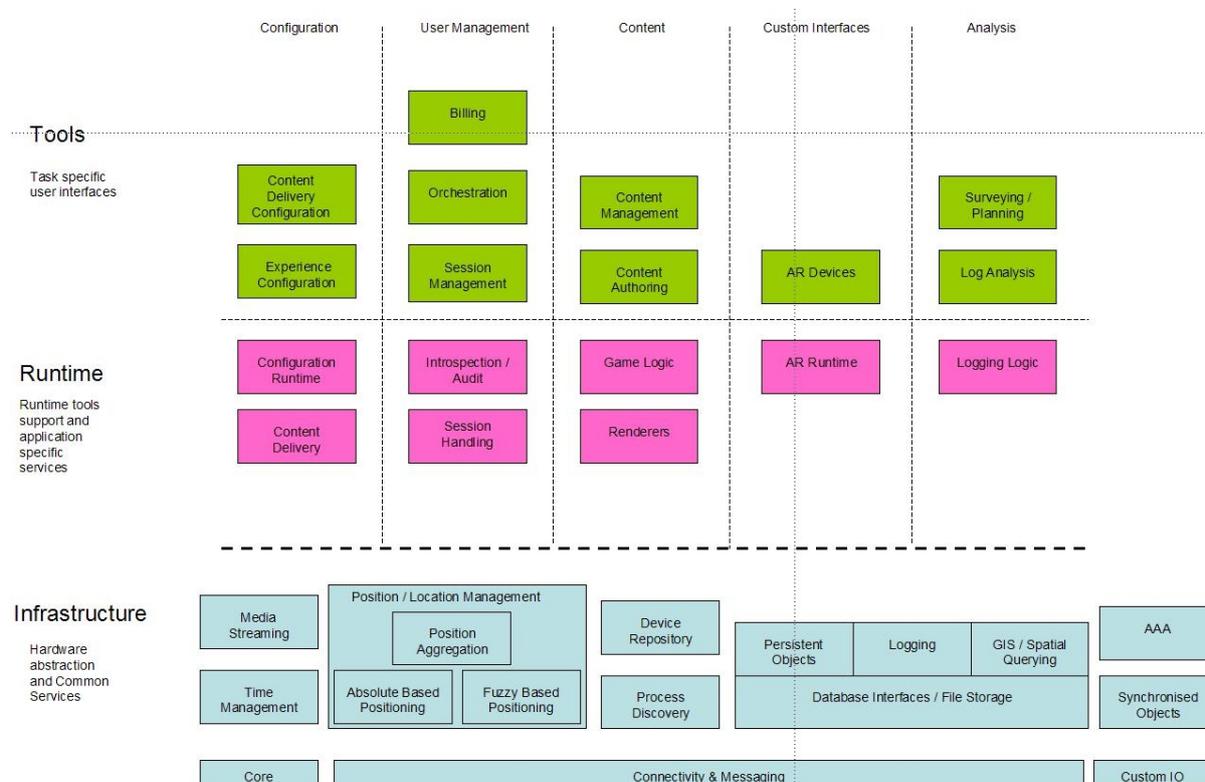
Mais recentemente, foi lançado o projeto IPerg (*EU Integrated Project on Pervasive Games* – <http://iperg.sics.se/>). Segundo as informações do *website* do projeto, o IPerg visa o estudo de novas e atrativas formas para implementação de jogos pervasivos, que estarão próximos ao cotidiano dos jogadores através de itens, dispositivos e pessoas comuns nos locais que habitamos. Para atingir este objeto, o projeto pesquisa aspectos técnicos e conceitos de projetos de jogos pervasivos. A partir destas pesquisas, espera-se a especificação e desenvolvimento de infra-estruturas, ferramentas e métodos de forma a:

- Permitir a rápida e factível criação e execução de jogos pervasivos;
- Criação de bons projetos destas aplicações;
- Entendimento da audiência pretendida;
- Estudos sobre o impacto social de jogos pervasivos.

O projeto divide-se em núcleos de pesquisa que estudam desde a análise de mercado, decisões de projeto, passando pela infraestrutura necessária para jogos pervasivos. O projeto estabelece diferentes gêneros e terminologias para jogos pervasivos. Um deles é o de *cross-media games*, jogos em que o conteúdo de uma única instância de jogo é disponibilizada para diferentes interfaces de acesso. Jogadores são equipados com uma variedade de interfaces de acesso ao jogo, como telefones celulares, sistemas de realidade aumentada móvel, mesas de jogo e estações de comunicação. A experiência de cada jogador difere pela combinação destes vários dispositivos e diferentes mídias. Jogos *crossmedia* são potencialmente acessíveis a qualquer hora em qualquer lugar, através destes diferentes dispositivos.

Em relação a infraestrutura, o foco é a compreensão e definição dos requisitos de uma plataforma para jogos pervasivos em termos de suporte a dispositivos, componentes de *software* e múltiplas redes de acesso. Esta plataforma é apresentada via uma arquitetura de referência, descrita por um diagrama de blocos que identifica categorias de componentes de *software* que

podem ser utilizados para construção e execução de um jogo pervasivo. A Figura 4.12 apresenta esta arquitetura.



**Figura 4.12** Arquitetura de Referência IPerg para uma Plataforma de Suporte a Jogos Pervasivos

O projeto da plataforma objetiva um mapeamento entre o mundo virtual do jogo e um espaço físico pré-determinado como ambientes ou “espaços” distintos, porém interconectados: um relativo ao mundo real e outro ao mundo virtual. Nesta abordagem, a estrutura de um jogo sugere que ações realizadas em diferentes sub-espaços (subconjunto do mundo virtual ou real), tem influência simultânea nos demais sub-espaços do jogo. Esta característica cria um estilo chamado então de trans-realidade (*trans-reality*), onde mundo virtual e real são sintetizados em um único ambiente. Por exemplo, o deslocamento do usuário no mundo real, representaria o deslocamento de seu avatar no mundo virtual.

Jogos aplicados nesta plataforma poderão se espalhar por vários diferentes espaços que necessitarão do suporte de um *middleware* extremamente flexível, que oferecerá um modelo compartilhado de posicionamento e contexto, através de um conjunto de serviços comuns. Um ponto chave neste *middleware* é a necessidade de adaptação a mudanças dinâmicas tanto nos contextos físicos quanto virtual.

Iperg é um projeto em andamento, com término previsto para Março de 2008. Como resultados, projetos de jogos e protótipos foram divulgados recentemente, porém sem a disponibilização da plataforma de utilização dos mesmos.

## 4.5 Considerações Finais

As soluções apresentadas neste capítulo apresentam a idéia comum de arquiteturas ou plataformas de *middleware* que buscam diminuir o esforço necessário para construção tanto de aplicações pervasivas, quanto jogos multiusuário. Através destas soluções, serviços são disponibilizados para resolução de problemas típicos do domínio de cada aplicação. A arquitetura PM2G segue esta idéia, através de serviços específicos para os cenários apresentados na Seção 1.2.1.

Em relação aos trabalhos diretamente relacionados a esta proposta, o artigo de David Fox [Fox03] apresenta idéias e interessantes e motivadoras para uma integração entre dispositivos específicos (telefones celulares), porém sem preocupações na infra-estrutura de suporte necessária às suas idéias. Os grupos focais realizados por Koivisto e Wenninger [KW05] aumentam e especificam melhor as funcionalidades possíveis nesta integração. Seus resultados apresentados indicam que as funcionalidades são factíveis atualmente, e são, em sua maioria, bem vistas por jogadores e desenvolvedores. Estes dois trabalhos são considerados motivadores da proposta PM2G, por buscarem funcionalidades semelhantes àquelas definidas nos cenários pretendidos para a visão de jogos multiusuário Seção 1.2.1. A proposta PM2G segue estes trabalhos, ao mesmo tempo que busca oferecer um suporte ao desenvolvimento destas aplicações. De fato, novas idéias são propostas, como as conexões diretas entre adversários para a melhoria da interatividade entre jogadores via dispositivos móveis.

Dos trabalhos que buscam oferecer suporte a jogos, o Han *et al* é o primeiro a utilizar um *middleware* para realização de jogos multiplataforma. O *middleware* baseia-se em um escopo físico limitado, monitorado por sensores que guiam a migração do jogo entre vários dispositivos. No caso da proposta PM2G, não há esta limitação física, e o usuário pode acessar o jogo em espaços abertos. Apesar de se mostrar bastante interessante, o conceito de situação e a falta de dados sobre experimentos realizados pela pesquisa não permite que seja feita uma análise mais profunda sobre os objetivos deste trabalho com a proposta PM2G.

A proposta mais relativa a nossa é o Iperg. As definições criadas por este projeto fazem de um jogo PM2G é um *crossmedia game*. Porém, o Iperg é um projeto mais amplo, que explora desde o projeto do jogo até questões de mercado para jogos pervasivos. A maior relação entre

os dois projetos está na infra-estrutura necessária para a realização dos jogos, embora o projeto IPerg estabeleça conceitos mais complexos, com a idéia de trans-realidade, o que possivelmente implicará em requisitos mais complexos para sua realização.

# Modelos para Jogos Multiusuário Multiplataforma Pervasivos

*A maneira como enxergamos o problema é o problema.*

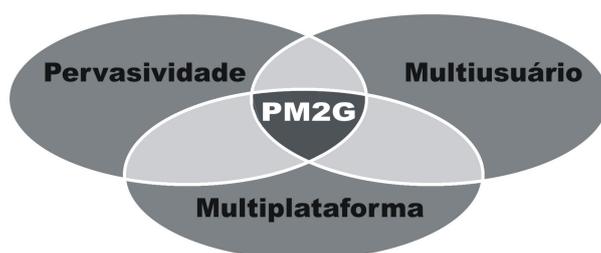
— STEPHEN R. COVEY (Do livro “Sete hábitos das pessoas altamente eficazes”)

As características pretendidas pela visão de Jogos Multiplataforma Pervasivos defendida nesta tese indicam a necessidade de um suporte adequado ao seu conjunto de funcionalidades. Para facilitar a especificação de uma plataforma que contemple este suporte, este capítulo apresenta modelos que refinam os cenários propostos previamente, identificando os elementos que compõem uma aplicação PM2G e a maneira como é realizada a interação do usuário com o jogo.

## 5.1 Introdução

A incorporação de dispositivos heterogêneos em jogos multiusuários apresenta várias possibilidades de novas experiências aos jogadores. No trabalho realizado por Koivisto e Wenninger [KW05] (Seção 4.4) são apresentadas algumas possíveis opções. Tomando por base estas alternativas, a visão defendida nesta tese baseia-se em três principais características, retratadas na Figura 5.1: (i) necessidade de suporte a vários jogadores, (ii) à mobilidade do usuário, permitindo que o mesmo jogue em diferentes momentos e locais, e (iii) à heterogeneidade de plataformas de *hardware* (Computadores Pessoais de Mesa (*Desktops*), Personal Digital Assistants (PDA), Telefones Celulares), sistemas operacionais e redes (Ethernet/IP, GPRS, Wi-Fi, *Bluetooth*). Nesta proposta, a junção destas características levará à pervasividade pretendida para aplicações PM2G – *Pervasive Multiplayer Multiplatform Game*.

A partir da junção destas características, faz-se necessário o estabelecimento de uma visão particular de onde possam ser extraídos conceitos que deixem mais claro o cenário de jogabilidade pretendido para um jogo PM2G. Neste trabalho, esta visão é apresentada através de dois modelos: um de aplicação e outro de utilização. O primeiro diz respeito à visão de elementos



**Figura 5.1** Posicionamento de jogo PM2G em função de pervasividade, distribuição e heterogeneidade.

que compõem o jogo PM2G, enquanto o segundo ressalta a forma como os usuários interagem com o jogo. Porém, antes da apresentação destes modelos, é necessário também entender quais são os estilos ou tipos de jogos em que os cenários definidos por este trabalho melhor se adequam. Este tópico é tratado na próxima seção, bem como as justificativas para esta limitação de escopo. As seções seguintes apresentam, respectivamente, os modelos de aplicação e utilização PM2G.

## 5.2 Modelo de Jogo-Alvo PM2G

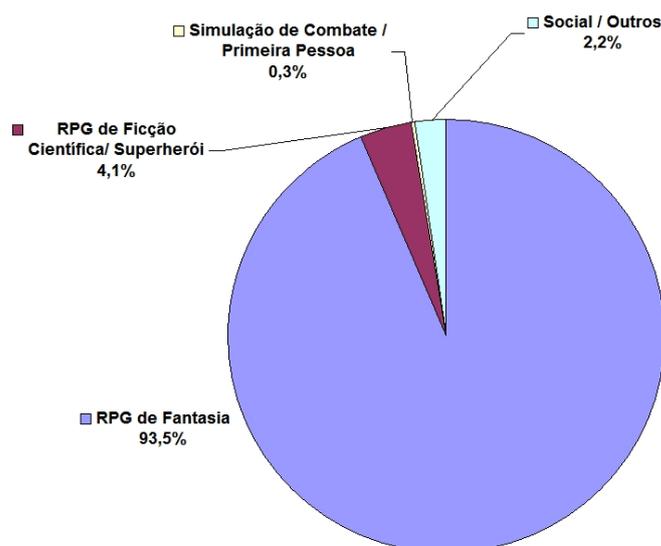
Jogos multiusuário podem apresentar diferentes temáticas, que variam de corridas de automóveis, batalhas entre exércitos até simulações de esportes. Cada uma destas temáticas apresenta requisitos peculiares, quer sejam técnicos ou de projeto do jogo (*game design*), que de certa forma inviabilizam um tratamento uniforme por parte de uma plataforma de suporte a tais aplicações.

Na verdade, certas temáticas de jogos não parecem ser adequadas à integração de dispositivos móveis, baseado nos cenários descritos na Seção 1.2.1. Como exemplo, é difícil imaginar a participação de jogador usando um dispositivo móvel na simulação de uma corrida de carros, onde é exigida a participação constante do jogador, efetivamente controlando seu carro. A participação deste jogador seria bastante comprometida, uma vez que uma conexão permanente com o jogo deve ser evitada, ou simplesmente, por não ser previsível ou garantida.

Sendo assim, a escolha do estilo de jogo adequado aos cenários indica que certos requisitos precisam ser contemplados. Primeiramente, uma aplicação PM2G precisa passar a idéia de uma (*i*) alta longevidade. Neste contexto, longevidade caracteriza o tempo necessário para um jogador completar todos os objetivos do jogo. Uma alta longevidade indica que o jogo não se completará em apenas alguns minutos ou horas. Quanto maior este tempo, maior o número de sessões que o jogador precisará para realizar suas tarefas, onde o usuário poderá então

utilizar diferentes dispositivos. Além disto, o jogo deve permitir também que um jogador tenha (ii) períodos esporádicos de interação. Estas sessões de interação podem ser utilizadas para acompanhamento ou realização de ações de curta duração, que permitam ao jogador evoluir no jogo. Em outras palavras, não é necessário que para cumprir todas tarefas, o jogador precise manter uma única duradoura conexão com o jogo. Devido às restrições de comunicação de dispositivos móveis, um jogo PM2G não deve ter sua (iii) jogabilidade baseada em ações ou movimentos rápidos, como no caso de jogos de tiro em primeira pessoa. O jogo deve focar mais na estratégia do jogador, que na sua agilidade com dispositivos de entrada.

Durante esta pesquisa, uma análise na divisão de temáticas [Woo06] em uso em jogos *online* foi de fundamental importância para decisão sobre qual estilo de jogo deveria ser focado. Através dados obtidos, verificou-se que há uma concentração de mais de 97% dos jogos em apenas um estilo de jogo, os jogos de representação de papéis, mais conhecidos como RPG (do inglês, *Role Playing Games*) – vide Figura 5.2.



**Figura 5.2** Divisão de Mercado de Jogos *Online* por Temática.

### 5.2.1 Jogos de Interpretação de Papéis (RPGs)

RPGs caracterizam-se por uma narrativa complexa, onde cada jogador assume o papel de um personagem em um mundo virtual formado por diferentes raças (humanos, elfos, etc.) e classes (bruxos, bárbaros, etc.) cumprindo missões, comumente chamadas *quests*, como encontrar objetos ou lugares e destruir inimigos. Estes focam no comportamento e na evolução de cada personagem, a partir do acúmulo de riquezas e experiências adquiridas durante o cumprimento

de suas missões.

Estes jogos são originalmente concebidos como jogos de tabuleiro, onde os próprios participantes dos jogos vão colaborativamente criando sua história e regras. Isso faz com que estes sejam fundamentalmente de outros estilos de jogos. Não há a noção de vencedores e vencidos. O prazer do participante relaciona-se com sua participação e o sentimento de imersão decorrente também da liberdade que o mesmo tem em participar da criação da história. Estes jogos possuem um forte apelo pelo uso constante da imaginação dos jogadores no desenrolar do jogo.

Em sua versão digital, a história do jogo é preconcebida, onde as missões são prédefinidas pelo projeto do jogo, sendo que a ordem que os jogadores as realizam é variável. Cada personagem é representado por um conjunto de *estatísticas*, onde geralmente há uma diferenciação entre dois tipos: *atributos* e *habilidades*.

O primeiro tipo indica propriedades que todo personagem possui. Atributos como força, agilidade e inteligência são exemplos comuns destas propriedades. Estas propriedades possuem uma escala numérica, onde o valor de cada atributo determina as ações que um personagem pode ou não realizar. Por exemplo, o valor do atributo “força” pode ser usado para determinar a quantidade de itens que um personagem pode carregar consigo, ou mesmo, o impacto que o ataque deste personagem tem sobre outros personagens do jogo. Por sua vez, habilidades são propriedades que apenas alguns personagens possuem, como habilidade de negociar itens, montar animais, curar outros personagens ou a si próprio.

A modificação dos atributos e habilidades de um personagem se dá através da realização das missões. O sucesso no cumprimento destas missões indica o aumento nos atributos de um personagem, e possivelmente o ganho de novas habilidades. Da mesma forma, o insucesso nestas missões traz penalizações ao jogador, como a diminuição dos valores de seus atributos ou na eficiência de suas habilidades.

### **5.2.2 Análise de RPGs como temática para aplicações PM2G**

De acordo com os requisitos levantados previamente, pode-se avaliar a temática RPG como uma temática ideal para os cenários propostos por este trabalho (vide Tabela 5.1).

Além disto, estabelecer RPGs como um modelo alvo para aplicações PM2G traz outros benefícios:

- A evolução baseada em atributos e habilidades facilita a idéia das informações que podem ser migradas pelos jogadores quando estiverem interagindo entre si via dispositivos móveis, como representado pelo cenário 2 (Seção 1.2.1). Estas informações representariam o conjunto de atributos do personagem de um jogador. De acordo com as interações

**Tabela 5.1** Análise de jogos de representação de papéis, de acordo com os requisitos desejáveis para aplicações PM2G.

Requisito	Atende Requisito?	Justificativa
Alta longevidade	Sim	As missões em RPG são tarefas de longa duração. Com o desenrolar do jogo, estas missões passam a ser cada vez mais difíceis, e conseqüentemente mais duradouras. O jogador poderá então continuar a realizar suas missões em diferentes plataformas;
Jogo Baseado em sessões	Sim	O jogo não é realizado em uma única sessão de interação do jogo com o mundo virtual. Uma única missão pode ser tão complexa que demanda do usuário horas para sua conclusão. Então, o usuário poderá realizá-la através de várias sessões de interação, mesmo através de diferentes dispositivos.
Jogabilidade não baseada em rápidos movimentos	Sim	Boa parte da interação do jogador não demanda respostas tão rápidas quanto outros jogos, como aqueles em primeira pessoa, por exemplo. As interações do usuário são mais trabalhadas e estudadas, de acordo com a estratégia do jogador para realizar determinada missão.

entre os jogadores, estas informações seriam modificadas, e posteriormente integradas ao jogador;

- O uso de elementos contextuais, como localização, pode inserir no projeto do jogo, missões que também envolvam o deslocamento físico do jogador. Esta possibilidade permitiria criar projetos inovadores e mais atrativos ao jogador. Mesmo para aqueles que não se sintam a vontade para este tipo de tarefa, estas missões poderiam ser definidas como facultativas.

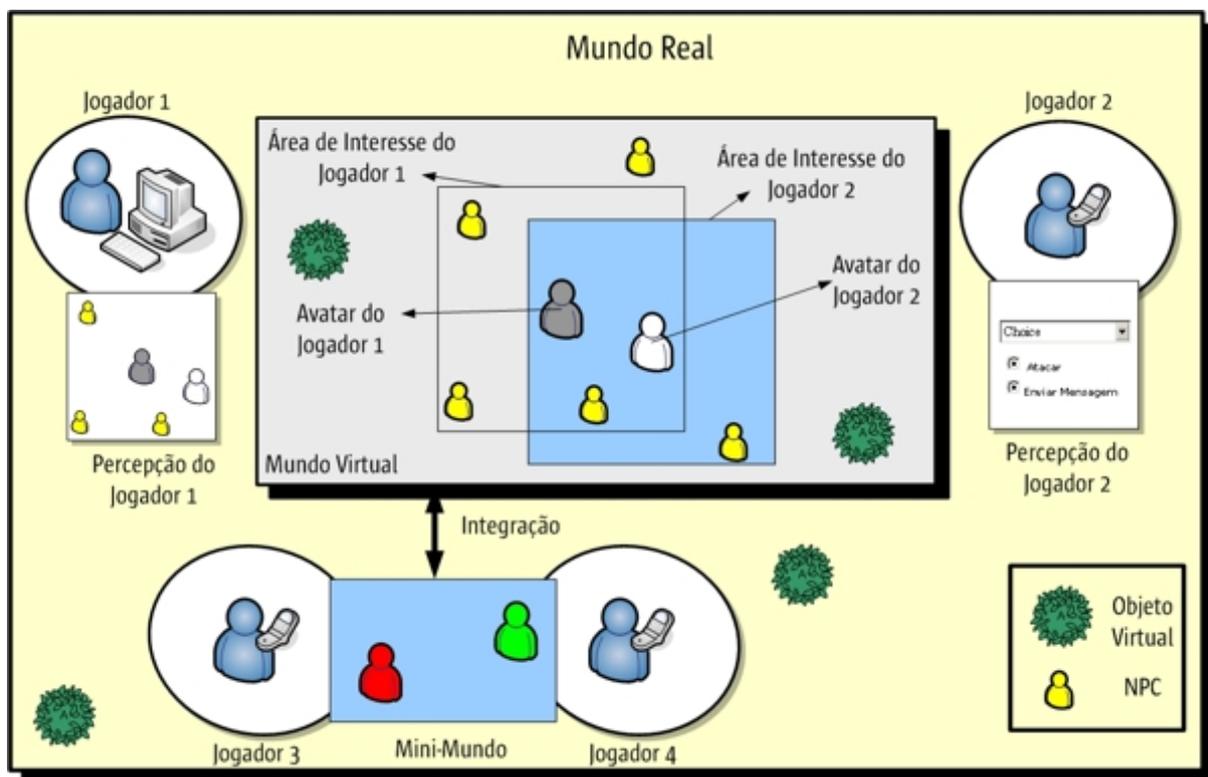
Sendo assim, de forma a diminuir a complexidade da especificação de uma plataforma que vise o suporte a um modelo genérico de jogos, e visto que há uma forte concentração de mercado na temática Role Playing Game (RPG), neste trabalho foca-se neste estilo de jogo.

### 5.3 Modelo de Aplicação PM2G

Sucintamente, uma aplicação PM2G pode ser definida como uma extensão de um RPG digital multiusuário, com o uso de dispositivos computacionais diversos (inclusive móveis) como

meio de acesso ao jogo. O uso destes dispositivos permite ainda que situações específicas da computação móvel sejam aproveitadas, criando novos cenários e gerando maior interatividade entre jogadores móveis. Ressalta-se ainda que a conotação de um jogador como “móvel” ou “estacionário” não é imutável, variando de acordo com o tempo.

Sendo assim, o modelo PM2G baseia-se nos padrões da maioria dos RPGs existentes, e em modelos de aplicação já definidos em outros trabalhos, no caso, o FreeMMG [Cec05] e o SimMUD [KLXH04]. A partir destes trabalhos são extraídos os seguintes conceitos-chaves para uma aplicação RPG: *terreno*, *objeto*, *mundo*, *jogador*, *personagens automatizados* e *áreas de interesse*. O modelo de aplicação PM2G estende este modelo criando quatro novos conceitos: *contexto de execução*, *percepção*, *interação* e *mini-mundo*. A Figura 5.3 apresenta alguns destes conceitos, onde maiores detalhes são expostos a seguir.



**Figura 5.3** Modelo de Aplicação PM2G.

Primeiramente, *terreno* representa o conjunto de informações imutáveis que compõem a paisagem do mundo virtual, influenciando o deslocamento dos jogadores. Exemplos destes elementos são rios, montanhas ou depressões.

*Objeto* é o elemento básico que compõe o conjunto de informações dinâmicas do mundo

virtual. Em outras palavras, um objeto é qualquer informação que pode ser adicionada, copiada, alterada ou removida do mundo virtual. Exemplos típicos de objetos seriam comida, armas, ferramentas e tesouros. Elementos que compõem a paisagem do mundo virtual também podem ser classificados como objetos, como uma árvore ou uma casa. No modelo PM2G, estes objetos virtuais também podem ser inseridos em locais do mundo real de cada jogador. Estes objetos representam elementos (armas, porções ou tesouros) que podem ser capturados por um jogador móvel para uso no mundo virtual, através de seu deslocamento aos pontos onde os objetos estejam representados.

*Mundo virtual* (ou simplesmente *mundo*) é o sistema de regras ou leis que implementam o espaço virtual onde estão localizados os objetos. Espera-se que o mundo simule um ambiente virtual típico, onde os objetos possuem uma posição relativa aos outros objetos, tamanho, velocidade, aceleração, etc. Espera-se também que quaisquer objetos localizados no mesmo mundo possam interagir entre si (colisão, troca de informações, etc) e que as interações entre objetos ocorram com maior frequência quando estes objetos estão próximos, dentro do mundo virtual. Opcionalmente, um objeto pode ser associado a um jogador, sendo este então considerado o *dono* deste objeto.

É de se esperar que o mundo virtual seja grande e que um jogador, em um dado instante, possa interagir apenas com um subconjunto deste mundo. Este subconjunto define a *área de interesse* de um jogador, abrangendo objetos, jogadores e demais elementos do mundo com os quais o jogador pode interagir em um dado instante. Esta área de interesse pode ser determinada através de regiões fixas, como no modelo FreeMMG [Cec05], ou variáveis de acordo com o deslocamento do jogador, como nas propostas que usam o diagrama de Voronoi [HL04] [TV05]. O modelo PM2G deve ser independente desta escolha, impondo apenas que haja uma limitação na área de interação de um jogador em um dado instante.

O *jogador* representa o usuário do jogo, que interage com o mundo ao exercer controle sobre objetos. Tipicamente, cada jogador controla diretamente uma parte dos objetos. Por exemplo, pode-se determinar que um jogador possa controlar apenas os objetos que são marcados como tendo o jogador em questão como *dono*. Cada jogador tem um estado associado, composto pela posição do jogador no mundo e informações de seu avatar, como habilidades, energia vital e posses. Este estado deve ser salvo no servidor, de modo a ser levado entre as sessões do jogador, e na migração do mesmo entre diferentes dispositivos. Jogadores ao se deslocarem pelo mundo virtual podem realizar três tipos de ação: mudança de sua posição, interação com objetos e interações com outros jogadores. As interações estão sujeitas às regras do jogo e modificam o estado dos jogadores bem como de objetos envolvidos nas interações. Por exemplo, se o mundo virtual simula uma batalha, a luta entre dois jogadores deve modificar

os atributos de “energia vital” dos dois jogadores.

Os *personagens automatizados* representam os *Non-player Characters (NPC)*, que podem ser mapeados como inimigos, aliados ou simplesmente espectadores (*bypassers*). Assim como os jogadores, estes elementos também possuem um estado e um avatar associado, e podem ter seu estado salvo.

Os próximos conceitos são diretamente ligados ao modelo PM2G. Primeiramente, *percepção* indica como a área de interesse é apresentada a cada jogador. *Contexto de execução* relaciona-se com a infra-estrutura de acesso de um jogo com o mundo virtual. Esta infra-estrutura engloba o dispositivo e o tipo de rede de acesso utilizado pelo jogador para sua comunicação com o jogo. A relação entre estes conceitos indica que a *percepção* da área de interesse é diretamente influenciada pelo *contexto de execução* do jogador. Por exemplo, na Figura 5.3, o jogador 1 e o jogador 2, mesmo próximos topologicamente no mundo e possuindo elementos em comum, têm percepções diferentes de suas áreas de interesse. Enquanto o jogador 1 tem como *percepção* um conjunto de imagens, o jogador 2 tem sua percepção resumida a uma informação textual que descreve os elementos próximos de seu avatar.

O *contexto de execução* e a *percepção* também influenciam a *interação* do jogador, ditando a forma como o jogador realiza suas ações no mundo. A limitação da interface homem-máquina de dispositivos móveis faz com que as ações típicas baseadas em movimentações ágeis do avatar sejam inadequadas em um contexto móvel. O conceito de *interação* visa criar um mecanismo que facilite a realização de certas tarefas a jogadores móveis, mesmo com suas inerentes limitações. A motivação para este conceito é a analogia proposta por Fox [Fox03], descrita na Seção 4.4. Tomando como exemplo a Figura 5.3, o jogador 1 pode atacar o jogador 2 usando o apontador para escolher algum tipo de arma, e depois clicar sobre o avatar do jogador 2. Por sua vez, o jogador 2 pode atacar o jogador 1 escolhendo-o dentre aqueles presentes na lista de jogadores de sua *percepção*, assim como a arma a ser utilizada para o ataque. O jogador 1 verá o deslocamento e ataques mútuos entre os dois avatares, enquanto o jogador 2 pode ser notificado através de um relatório também textual sobre o resultado da disputa. Esta disposição traz uma característica interessante ao modelo proposto. Para jogadores estacionários, é transparente o contexto de execução de seus adversários ou aliados. A interação é mantida em um nível mais alto, sobrepondo as limitações de jogadores móveis. Além disso, esta proposta facilita a mudança de perfil de um jogador estacionário para móvel e vice-versa. Para isto, um jogador ao acessar o mundo virtual utilizando um computador pessoal de mesa, pode assumir o controle efetivo de seu avatar a partir das ações de “alto nível” disparadas quando o mesmo estava em um contexto móvel. Ressalva-se entretanto que as diferenças de *percepção* e *interação* entre jogadores móveis e estacionários podem ser atenuadas, de acordo com o projeto do jogo

em questão. Por exemplo, a utilização de *áreas de interesse* pode determinar que certas regiões do mundo virtual sejam marcadas pela interação através de batalhas por turno, independente do *contexto de execução* de um jogador.

A idéia de *percepção e interação* adaptada ao *contexto de execução* de um jogador permite que o padrão de comportamento entre jogadores estacionários e móveis seja preservado. Em outras palavras, jogadores móveis podem realizar ações gerenciais que lhe permitam um acompanhamento esporádico de seu avatar, enquanto jogadores estacionários poderiam permanecer por longas sessões de conexão com o jogo.

Por fim, o conceito de *mini-mundo* está relacionado com o aproveitamento de novas oportunidades de interação, específicas para os jogadores móveis. *Mini-mundos* constituem-se como cenários paralelos com interação direta entre jogadores móveis, através de redes espontâneas ou *Wireless Local Area Networks (WLAN)*. Estes cenários podem ser compreendidos como partidas isoladas entre jogadores móveis, porém com conseqüências no jogo massivo, de forma semelhante às conseqüências das interações entre jogadores dentro do mundo virtual. Cabe ao projetista de um jogo PM2G fazer com que estes cenários estejam relacionados com o tema central que rege o mundo virtual. Como exemplo, em um jogo medieval, um mini-mundo pode representar um duelo de espadas entre dois jogadores em tempo-real via *bluetooth*. O que difere esta aplicação de um jogo multiusuário móvel tradicional (Seção 3.2.3) é que os jogadores envolvidos podem interagir com o mundo virtual para utilizar recursos (armas, objetos e itens do perfil de seu avatar) no contexto destas batalhas individuais.

Na mesma linha do exemplo anterior, cada um dos jogadores pode utilizar armas e outros elementos do jogo no mini-mundo. No modelo proposto, estes cenários contribuem para melhorar a dinâmica do jogo, permitindo que jogadores móveis participem de interações mais intensas, sem as limitações e custos adicionais impostos pelas tecnologias de comunicação via operadoras de telefonia ao jogador. Ao contrário do mundo virtual simulado, os mini-mundos são instâncias ou partidas de curta duração e não têm como objetivo suportar persistência do estado do jogo por longos períodos de tempo. Porém, ressalta-se que a existência de *mini-mundos* não é independente do jogo massivo convencional. A integração entre os mini-mundos e o mundo global demanda que este último seja notificado da existência destes cenários, assim como dos resultados ao término de suas sessões. Como exemplo, a vitória no duelo de espadas pode indicar o aumento de um determinado atributo (como força) do jogador vencedor, de acordo com o projeto do jogo.

Um aspecto importante ao conceito de mini-mundo diz respeito às diferentes maneiras como estas partidas podem ser realizadas. Uma primeira opção seria a existência de locais específicos com infra-estrutura adequada e disponível para jogadores móveis, como *hotspots*

Wi-Fi em aeroportos ou similares. De fato, esta configuração permite que o local real onde está montada tal infra-estrutura seja mapeada em uma determinada região do mundo virtual acessada apenas através do deslocamento do jogador a este local. Porém, isto se apresenta como uma questão de projeto do jogo. A existência destes locais serviria para facilitar o acesso de jogadores móveis através destas infra-estruturas. Uma segunda opção é a formação de redes espontâneas entre dispositivos móveis, onde jogadores procurariam por adversários em suas cercanias para realização dos mini-mundos.

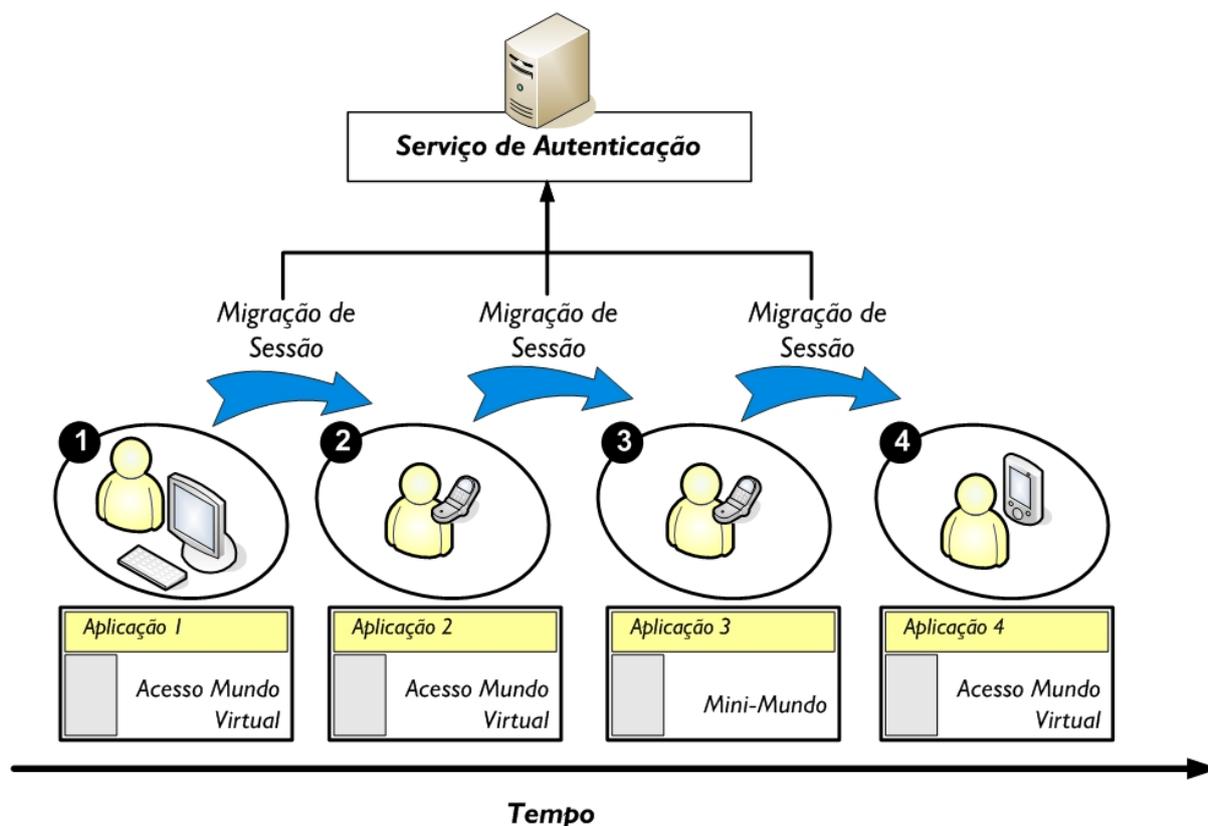
Em ambos os casos, a formação destes mini-mundos poderia ser facilitada através da utilização de serviços de localização. No primeiro caso, o serviço de localização permitiria que um determinado jogador móvel identificasse locais com facilidades de acesso para jogadores próximos de sua posição atual. No segundo caso, jogadores interessados na formação de novos *mini-mundos* poderiam expor sua localização para demais jogadores, ou ainda permitiria que o mesmo procurasse por instâncias de *mini-mundos* já em andamento na cercania de sua localização. Em suma, mini-mundo é caracterizado pela necessária proximidade física (e não apenas virtual) dos usuários – seja na área de abrangência de uma *Wireless Local Area Network (WLAN)* ou de uma rede espontânea (ad-hoc).

## 5.4 Modelo de Utilização

As participações dos jogadores ocorrem através de *sessões* de interação com o mundo virtual. Uma sessão caracteriza-se por um intervalo de tempo em que um jogador interage com o jogo a partir de um dispositivo específico, quer seja um computador pessoal de mesa ou um dispositivo móvel. Quando encerrada uma sessão, o jogador pode migrar para outro dispositivo, abrindo uma nova sessão e possivelmente continuando as ações a partir do ponto onde a última sessão foi finalizada. Esta visão de mudança de sessões entre dispositivos é representada pela Figura 5.4.

As sessões de jogo podem marcar a interação tanto através da realização de tarefas no mundo virtual, quanto através de mini-mundos. Toda sessão é iniciada através de um processo de autenticação do jogador com os dados de identificação do usuário e sua senha de acesso. Um jogador móvel ao iniciar um mini-mundo também realiza uma autenticação, para que o mesmo possa recuperar dados de seu perfil para uso no mini-mundo. Jogadores podem, ao encerrar uma sessão, emitir ordens que devem ser realizadas pelo(s) seu(s) avatar(es), já prevendo a continuidade do jogo em outro dispositivo.

Ainda em relação à participação do jogador, espera-se que sejam necessárias diferentes



**Figura 5.4** Mudança de Sessões em um PM2G.

aplicações, com diferentes propósitos e para diferentes dispositivos. Como exemplo, uma aplicação pode ser utilizada para acesso ao mundo virtual, enquanto outra pode implementar um Mini-Mundo do modelo de aplicação PM2G. Devido à limitação de recursos dos dispositivos móveis, a idéia de uma única aplicação multifuncional acarreta problemas de alocação de memória. Neste sentido, serão necessárias diferentes aplicações para diferentes fins, em contrapartida à idéia de uma única aplicação com todas as funcionalidades possíveis para o jogador. Os jogadores alternam as aplicações de acordo com a função desejada em um dado momento. Cada uma destas aplicações-cliente apresenta diferentes versões para diferentes plataformas, ressaltando-se porém que quando houver compatibilidade entre dispositivos, uma mesma aplicação-cliente pode ser utilizada entre os mesmos.

Para que se possa criar um relacionamento entre aplicações e dispositivos, faz-se necessário a identificação dos dispositivos suportados pelo jogo. Estas informações qualificam cada dispositivo, permitindo que decisões possam ser tomadas em função do dispositivo utilizado por um jogador. Na realidade, a preocupação com o dispositivo do jogador é maior para o uso de

dispositivos móveis, devido à maior heterogeneidade destes aparelhos e suas funcionalidades. A administração das informações sobre dispositivos suportados e aplicações existentes é de responsabilidade dos desenvolvedores e administradores da plataforma de suporte do jogo.

## **5.5 Considerações sobre o Capítulo**

A proposta PM2G é uma visão particular da integração de diversas tendências, através de modelos que ditam como estas coexistem. A partir de uma análise sobre as temáticas adequadas aos cenários propostos para PM2Gs, optou-se neste trabalho por reduzir o escopo dos jogos contemplados àqueles de representação de papéis, mais conhecidos como RPGs. Esta redução permitiu a elaboração de dois modelos, um de aplicação e outro de utilização, que permitem uma visão concreta da integração destas tendências.

A partir deste capítulo foi especificada uma plataforma de suporte aos cenários especificados, tomando como requisitos os conceitos e escopo definidos neste capítulo. O Capítulo 6 apresenta esta plataforma.

## Serviços de *Middleware* para Jogos Multiplataforma Pervasivos

Os cenários e modelos apresentados nos capítulos anteriores levantam requisitos necessários para uma infra-estrutura de suporte a jogos PM2G. Em resposta a estes requisitos, este capítulo apresenta uma arquitetura baseada em serviços de uma camada de *middleware* para estas aplicações, incluindo o desenvolvimento (modelo de programação – API) e execução (modelo de execução).

### 6.1 Introdução

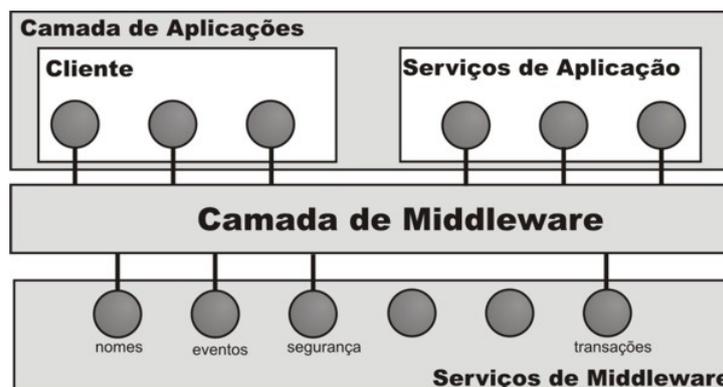
Os cenários e modelos descritos no Capítulo 5 impõem uma série de requisitos funcionais e não-funcionais a serem tratados por um modelo de suporte para jogos que possuam as características apresentadas para aplicações PM2G. Como apresentado na Seção 4.3, o modelo tradicional de suporte a jogos multiusuário utiliza uma camada de serviços na figura de um *middleware*. Os serviços destas plataformas de *middleware*, como aqueles descritos em trabalhos como [SSS03] [SSR<sup>+</sup>04], atacam questões comuns ao desenvolvimento e execução de jogos, tipicamente relacionados tanto a requisitos funcionais, como gerência de contas, autenticação, tarifação de jogadores, dentre outros, quanto requisitos não-funcionais, como escalabilidade e tolerância a falhas. Tais serviços são comumente construídos sobre uma base de serviços de suporte, como os de comunicação e persistência.

A proposta de suporte a jogos PM2G amplia os serviços de um jogo multiusuário tradicional, agregando serviços de suporte relativos aos conceitos específicos do modelo de aplicação descrito na Seção 5.3. São especificados cinco novos serviços:

- *Serviço de Gerenciamento de Contexto;*
- *Serviço de Adaptação de Conteúdo;*
- *Serviço de Adaptação de Interação;*
- *Serviço de Notificação de Mensagens;*
- *Serviço de Integração de Mini-mundos.*

Nesta proposta foi adotada uma arquitetura orientada a serviços, devido ao amadurecimento das ferramentas e do modelo de desenvolvimento baseado nos mesmos, conhecida como *Service Oriented Architecture (SOA)* [Pap03], que tem vantagens como transparência de localização e padronização de acesso e visa maximizar o desacoplamento e o reuso. A definição do termo “serviço” no contexto de arquiteturas orientadas a serviços pode ser descrita como: funções de negócio bem definidas e encapsuladas que recebem requisições de clientes, na forma de mensagens, e realizam algum processamento com base na requisição, podendo ou não gerar uma mensagem de resposta. Neste contexto, cliente é qualquer aplicação que interage com serviços através da troca de mensagens para solicitar a execução de alguma tarefa. Um serviço disponibiliza uma ou mais funcionalidades, através de sua interface pública, que podem ser acessadas remotamente, através de requisições síncronas ou assíncronas.

Adicionalmente, para a compreensão deste trabalho, faz-se necessária a diferenciação clara entre dois tipos de serviços, como ilustrado na Figura 6.1: os serviços de *middleware* e os serviços de aplicação. Os primeiros são os componentes que estão implementados e expostos, via API, na camada de *middleware*, enquanto os serviços de aplicação são componentes remotos, e estão na camada de aplicações e servem às requisições de clientes, não fazendo parte do *middleware*. Estes serviços são desenvolvidos e disponibilizados por provedores independentes e acessados pelas partes clientes das aplicações que são executadas nos dispositivos, as quais foram denominadas aplicações-clientes. Ao longo deste trabalho, quando a referência ao tipo de serviço não estiver explícita, deve-se entender por serviço de aplicação.

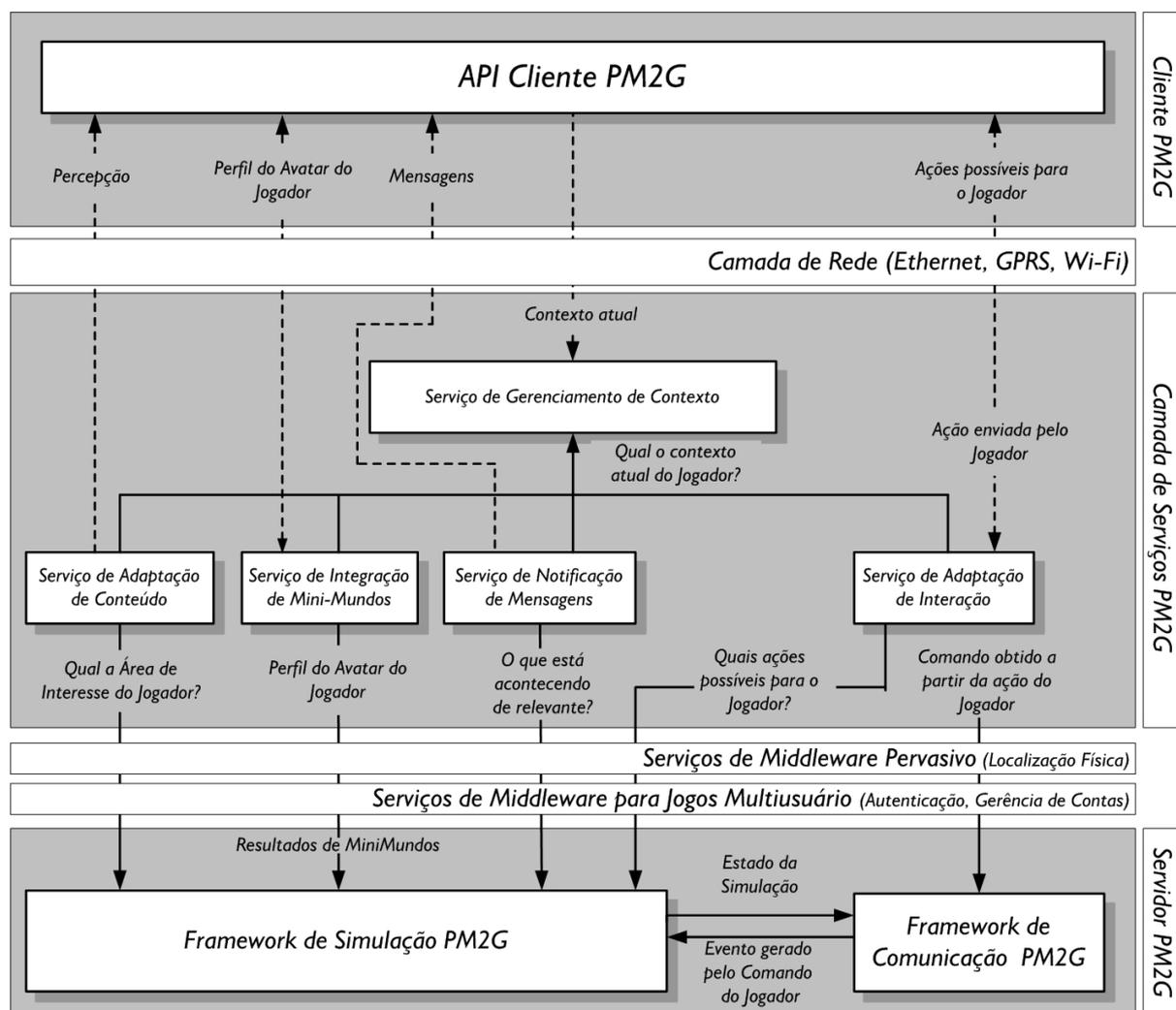


**Figura 6.1** Serviços de *Middleware* x Serviços de Aplicação

Nas seções seguintes deste capítulo são apresentadas (i) uma visão geral da arquitetura PM2G em uma visão de camadas, para em seguida, (ii) detalhar o funcionamento e componentes de cada serviço.

## 6.2 Visão Geral da Arquitetura PM2G

A Figura 6.2 apresenta a disposição dos serviços e demais componentes da arquitetura PM2G. A arquitetura segue o modelo Cliente /Servidor, onde no lado Servidor localizam-se a simulação servidora da aplicação e o conjunto de serviços responsáveis por diferentes funcionalidades. A decisão sobre o uso desta abordagem é influenciada por fatores como: (i) caso seja necessário melhorar a escalabilidade da solução, o uso de servidores replicados pode ser aplicado, (ii) melhor controle sobre o gerenciamento do estado do jogo, (iii) maior resistência à trapaça e (iv) facilidade de implementação da solução.



**Figura 6.2** Arquitetura de Serviços PM2G

Pelo diagrama, esta arquitetura é dividida em três blocos principais. O primeiro bloco, re-

ferenciado como Servidor PM2G, engloba dois *frameworks*: um para a simulação do jogo e outro para o tratamento da comunicação entre aplicações-cliente e a simulação servidora. Toda comunicação de jogadores com o mundo virtual simulado é feita através de eventos, que são recebidos pelo *framework* de comunicação e repassados ao *framework* de simulação. Por sua vez, o *framework* de simulação oferece aos clientes e demais serviços, o estado corrente do jogo em um dado momento. Este estado representa o conjunto de informações sobre os personagens de cada jogador e demais elementos do jogo (NPCs ou itens) em um dado instante. Este conjunto é repassado às aplicações-clientes, que atualizam a interface gráfica para visualização do jogo por cada usuário.

O segundo bloco apresenta o conjunto de cinco serviços que refletem o objeto maior deste trabalho. Estes serviços são resumidamente apresentados nesta seção, porém, a sua dinâmica de funcionamento, bem como suas arquiteturas internas são detalhadas nas seções seguintes. O primeiro deles é o *Serviço de Gerenciamento de Contexto*, que visa monitorar informações contextuais de acordo com o acesso dos jogadores. Estas informações contextuais incluem dados como dispositivo utilizado, preferências pessoais e possivelmente a localização física de cada jogador. Este serviço tem um papel central em relação aos demais, uma vez que as informações de contexto de cada jogador guiam o funcionamento interno de cada serviço.

Este é o caso do *Serviço de Adaptação de Conteúdo*, responsável por transformar o estado corrente do jogo, e repassá-lo ao jogador, de acordo com seu contexto atual. Como frisado no modelo de aplicação (Seção 5.3), cada jogador interaje apenas com sua área de interesse do mundo virtual. A partir desta área de interesse, o serviço realiza transformações que adequam esta informação desta região, de acordo com o atual contexto do jogador. O resultado destas transformações segue o conceito de percepção do modelo de aplicação PM2G.

Um terceiro serviço é de *Adaptação de Interação*. Este serviço tem por objetivo facilitar a realização de tarefas por parte de jogadores que utilizem dispositivos usando diferentes dispositivos, particularmente os móveis, uma vez que estes possuem limitações intrínsecas de conectividade, processamento e interface de entrada de ações. Para alcançar este objetivo, o serviço filtra a partir do estado da simulação e do atual contexto do jogador, quais são as ações possíveis para o usuário em dado momento do jogo. Estas ações refletem os comandos de alto nível também citados no modelo de aplicação PM2G. Estas ações e suas opções associadas são apresentadas ao jogador, que por sua vez, escolhe qual delas deseja realizar, repassado-as de volta ao serviço. Em um processo inverso, o serviço transforma a ação escolhida pelo jogador, em um ou mais comandos convencionais, que serão repassados ao *framework* de comunicação, e posteriormente ao *framework* de simulação para sua execução no mundo virtual.

Outro serviço presente na camada de serviços PM2G é o de *Notificação de Mensagens*. Este

serviço tem por finalidade, monitorar a execução da simulação do jogo em busca de eventos relevantes a cada jogador. Quando estes eventos ocorrerem, o serviço utiliza informações de contexto do jogador para enviar mensagens sobre estas ocorrências, de modo que o usuário possa realizar contra medidas a estes eventos. Estas mensagens utilizam diferentes tecnologias de comunicação, como *Short Message Service (SMS)* ou e-mail, baseadas no dispositivo e preferências pessoais de cada jogador.

O quinto serviço desta camada é o de *Integração de Mini-Mundos*. Este serviço visa dar suporte ao conceito de Mini-Mundos do modelo de aplicação PM2G. De forma sucinta, o serviço tem como principais funções (i) permitir que informações do perfil do personagem de um jogador, como armas, níveis de força ou magia possam ser migradas entre o mundo virtual simulado e os jogos paralelos que representam mini-mundos e (ii) bem como permitir que os resultados decorrentes destas partidas paralelas possam ser posteriormente integrados com o jogo e usadas no mundo virtual simulado.

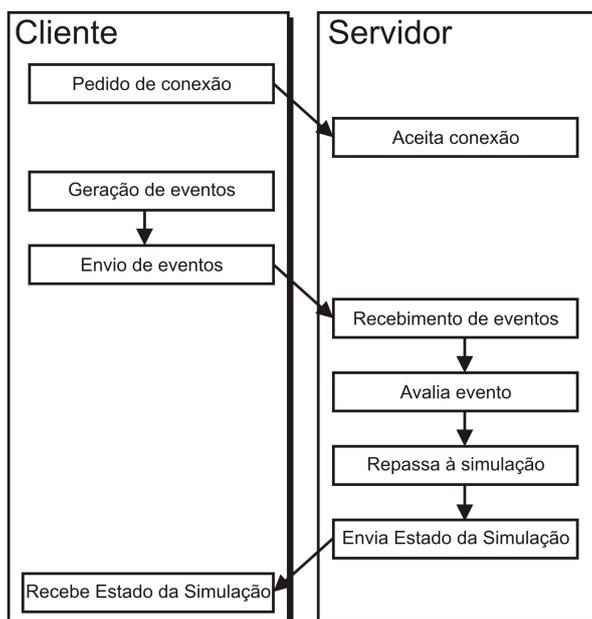
Os serviços anteriormente descritos foram concebidos de modo a tratar questões específicas do domínio da aplicação-alvo, no caso, as aplicações PM2G. Seguindo a abordagem de camadas de *middleware*, estes serviços são colocados acima de serviços já existentes, relacionados tanto com plataformas de *middleware* para computação pervasiva, quanto para jogos multiusuário (Vide Figura 6.2). Esta disposição indica que a arquitetura busca reusar estes serviços, como por exemplo, a utilização de um serviço de localização adequado a uma aplicação PM2G, como o *Location Inference Service (LIS)* de MoCA [SER<sup>+</sup>04].

No último bloco da arquitetura, o cliente PM2G representa a aplicação de acesso do jogador ao jogo. Esta aplicação recebe e envia informações tanto aos serviços quanto ao jogo, por meio de uma API. Ressalta-se que, embora não ilustrado no diagrama, o cliente também teria acesso às APIs das demais plataformas de *middleware*, que sejam integradas ao jogo, como MoCA [SER<sup>+</sup>04].

### 6.3 Framework de Comunicação PM2G

Como ressaltado na Seção 6.2, a arquitetura de serviços PM2G utiliza o modelo cliente/servidor para possibilitar o compartilhamento do estado do jogo entre jogadores. Nesta arquitetura, procurou-se uma forma de padronizar a comunicação entre jogadores e o servidor do jogo. Assim sendo, o fluxo de comunicação segue um modelo simplificado para aplicações de espaço compartilhado, apresentado na Figura 6.3.

A partir da conexão do cliente com o servidor, o cliente começa a gerar eventos de acordo



**Figura 6.3** Fluxo de Comunicação Cliente/Servidor, adaptado de [BBV03]

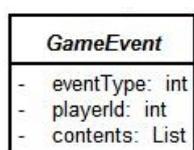
com a interação do usuário com a interface do jogo. Estes eventos são então repassados ao servidor do jogo. O servidor lê, realiza o *parsing* das mensagens recebidas e repassa cada evento ao *framework* de simulação. A simulação também é responsável pela geração de novos eventos, de acordo com a lógica do jogo, de forma independente das ações dos clientes. A partir da simulação, o estado do jogo é então repassado a cada jogador, de acordo com sua área de interesse. O cliente recebe esta informação, que então a apresenta ao usuário. A partir desta representação, o ciclo se repete.

Uma vantagem nesta abordagem para a proposta PM2G é que procura-se evitar (i) processamentos locais custosos nos dispositivos, (ii) a necessidade de uma constante comunicação para repasse dos eventos, bem como (iii) a necessidade de procedimentos mais complexos de sincronização entre clientes e o servidor. Em suma, os clientes atuam como interfaces de apresentação e interação com o servidor, sem processamento do estado do jogo localmente. Todo processamento relativo à simulação do mundo virtual é de responsabilidade do servidor. Em contrapartida, a quantidade de informações que representa o estado do jogo tende a ser maior que o repasse do conjunto de eventos enviados pelos jogadores. Em nossa proposta este problema é atenuado, pois o mundo virtual é dividido em áreas de interesse, e o jogador só recebe informações sobre a área com que interage em um dado instante.

Baseado nesta idéia, o *framework* de comunicação foi elaborado como uma adaptação da proposta de Bracken *et al*[BBV03]. Em sua proposta original, a comunicação utiliza a abor-

dagem de retransmissão dos eventos pelo servidor apenas para clientes que utilizem computadores pessoais. Nossa adaptação modificou este *framework* para que eventos sejam enviados apenas pelos jogadores (independente se estes utilizam PCs ou dispositivos móveis), enquanto o servidor repassa aos jogadores o estado completo da simulação.

Outro aspecto importante é que na proposta PM2G, toda a comunicação entre clientes e o jogo é encapsulada na forma de eventos de jogo (*Game Event*). Toda mensagem transmitida pelos clientes é transformada pelo *framework* de comunicação para um formato padronizado, antes de ser repassado ao *framework* de simulação. Este formato é representado pela Figura 6.4.



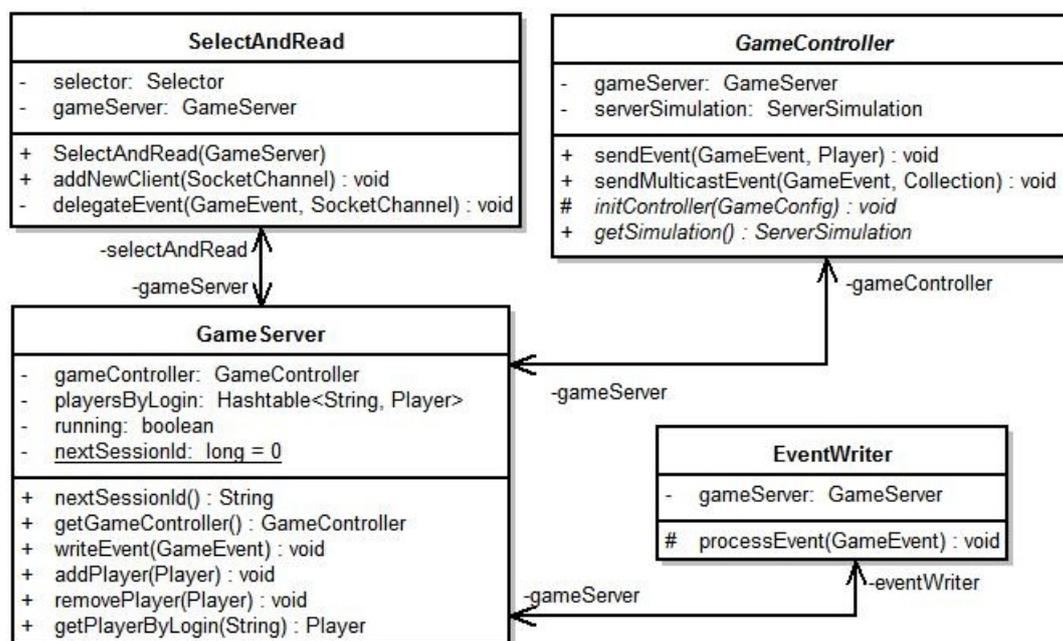
**Figura 6.4** Representação em UML de um Evento do Jogo – *GameEvent*.

Cada evento é gerado por um jogador (*playerId*), e possui um tipo que o identifica dentro do protocolo de comunicação com o servidor, e também com a simulação (*eventType*). Embora o formato de dados relacionado com cada evento seja dependente do projeto de cada jogo, o mesmo deve ser transparente para o *framework*, uma vez que é disponibilizada esta interface para eventos e seu tratamento pelo servidor. Cada jogo pode ter seu formato distinto. Para isto, cada evento utiliza uma lista de conteúdo que guarda o conteúdo de cada mensagem transmitida entre jogadores e o servidor, e vice-versa. Por exemplo, o evento relativo à movimentação do avatar do jogador encapsula as novas coordenadas do mundo virtual para onde o mesmo queira se deslocar.

### 6.3.1 Arquitetura

A Figura 6.5 apresenta uma versão simplificada da arquitetura do *framework* de comunicação PM2G. A classe *GameServer* representa o servidor do jogo. Este tem a função de aceitar as conexões dos clientes, gerenciar conexões ativas de jogadores em computadores pessoais, gerenciar a classe responsável pelo jogo em si, chamada *GameController*. O servidor lida com todo o processo *marshaling* e *unmarshaling* de mensagens transmitidas entre jogadores e o servidor do jogo, criando uma transparência no processo de comunicação.

A estrutura de comunicação do Servidor possui duas classes auxiliares: *SelectAndRead* e *EventWriter*. A primeira classe é responsável pelo processamento da mensagem recebida pelo servidor, sua transformação em um evento do jogo (*GameEvent*), e repasse ao *GameController*.



**Figura 6.5** Arquitetura PM2G – *Framework* de Comunicação.

A segunda é responsável por receber as informações do estado do jogo, encapsulado em um evento (*GameEvent*) e repassá-los aos jogadores.

A classe abstrata *GameController* representa o processo servidor do jogo PM2G. Um jogo deve estender esta classe, inserindo a lógica relativa a sua simulação. Esta classe possui uma referência à simulação do jogo, repassando-lhe todos os eventos recebidos pelo servidor.

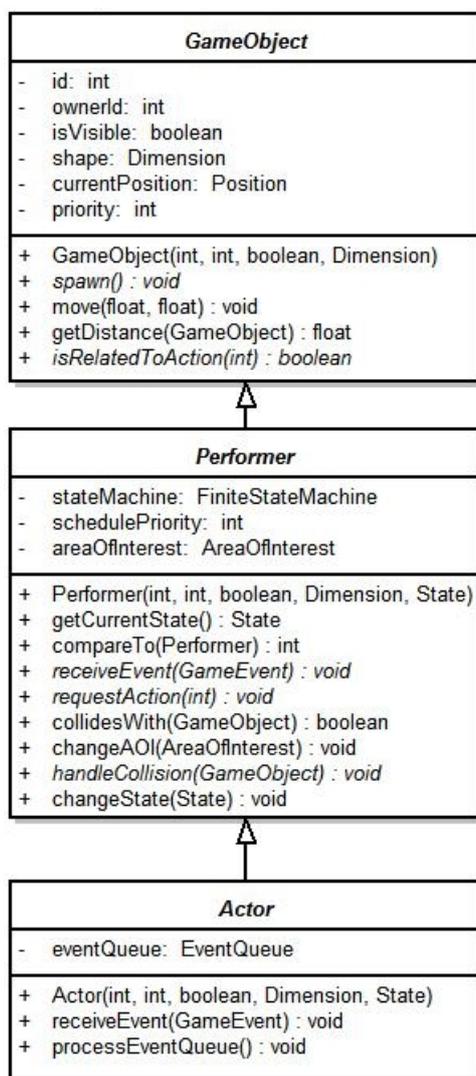
## 6.4 *Framework* de Simulação PM2G

O segundo componente do servidor PM2G é o *framework* de simulação. Este componente é o responsável por manter consistente o estado da simulação do jogo. Para isto, este componente recebe os eventos referentes aos comandos de cada jogador, que já foram processados pelo *framework* de comunicação. Estes eventos são postos em uma fila, onde o estado corrente do mundo é modificado de acordo com sua execução.

O estado do jogo engloba o estado de cada elemento que forma o mundo virtual, quer sejam estes objetos, NPCs ou personagens controlados por jogadores. Para modelar este componente, adaptou-se a proposta de Thor [Ale02] [Ale03a] [Ale03b] de um *framework* de simulação para MMGs. Nesta adaptação são incluídos aspectos específicos do modelo de aplicação PM2G.

Aqui, todo objeto do jogo é representado por uma classe *GameObject*, apresentado na Fi-

gura 6.6.

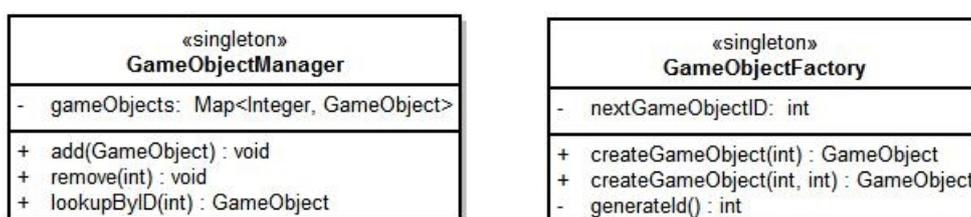


**Figura 6.6** Classes *GameObject*, *Performer* e *Actor* – Objetos do Jogos, Atuadores e Atores.

Estes objetos incluem desde itens até personagens do jogo, sendo que cada objeto possui um identificador único na simulação – atributo *id*. O *framework* diferencia certos tipos de objetos através de especializações. A classe abstrata *Performer* representa elementos do jogo que realizam ações (*atuadores*), como Non-player Characters (NPC). Cada atuador tem seu comportamento controlado por uma máquina de estados finitos, que indica qual o estado atual do objeto, bem como as transições entre os possíveis estados que o elemento pode assumir. Um atuador possui associada uma área de interesse, que representa o conjunto de objetos que o mesmo pode interagir em um dado instante. A classe abstrata *Actor* representa um personagem controlado por um jogador, aqui chamado de *ator*. Um ator também é capaz de interagir com

a simulação. Cada ator possui uma lista de eventos que acumula os comandos enviados pelo usuário, de acordo com sua interação com a aplicação-cliente do jogador.

Para gerência da criação e da referência aos objetos do jogo, são utilizadas as classes *GameObjectFactory* e *GameObjectManager*, apresentados na Figura 6.7. A primeira estabelece uma forma padronizada para criação dos objetos, que utilizando o padrão de projeto *Factory*. Este objeto é o responsável por atribuir um código único na simulação a cada objeto criado no jogo. A segunda classe encapsula a função de um dicionário de todos os objetos do jogo. Sua principal função é resolver referências a objetos do jogo, de acordo com o identificador do objeto.

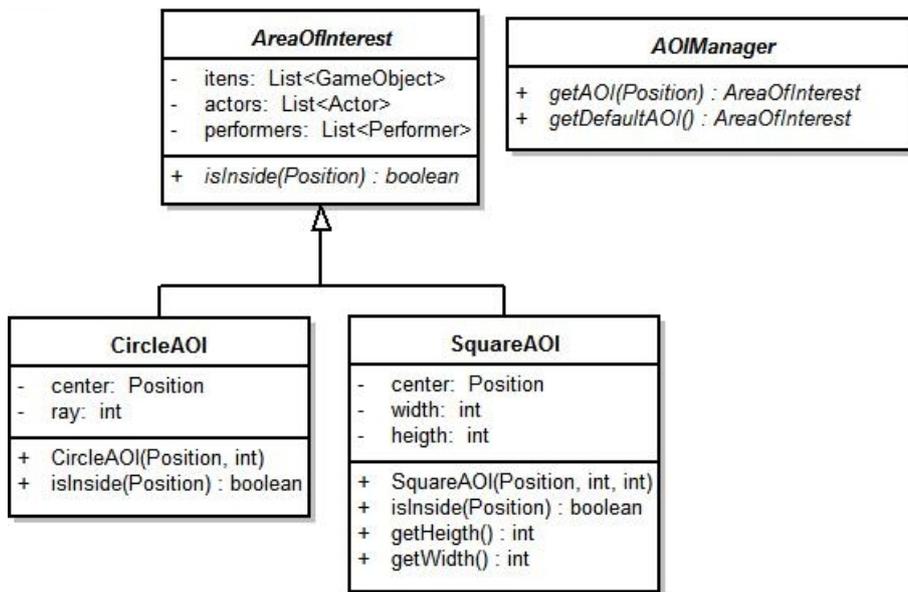


**Figura 6.7** Classes *GameObjectFactory* e *GameObjectManager* – Gestores de Objetos

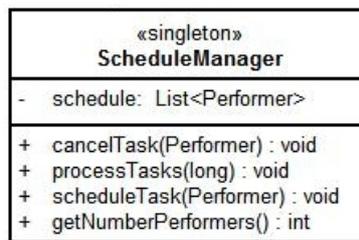
Como adaptação do *framework* original [Ale02] [Ale03a] [Ale03b], foi criada uma classe chamada *AreaOfInterest* – (Figura 6.8), que representa o conceito de área de interesse do modelo de aplicação PM2G. Esta classe possui um conjunto de objetos, NPCs e atores de uma área do mundo virtual. Esta área pode representar diferentes representações geométricas na topologia de um mundo virtual, como áreas retangulares ou circulares. Em sua principal função, é possível determinar se a partir de uma posição absoluta do mundo virtual, esta encontra-se dentro de uma determinada área de interesse. Na proposta PM2G, o mundo virtual será dividido em várias regiões. De modo a gerenciar estas áreas, criou-se um gerenciador de áreas de interesse, representado pela classe abstrata *AOIManager*. Sua principal função é descobrir qual a área de interesse, a partir de uma posição absoluta do mundo virtual.

O escalonamento de ações do jogo é realizado pela classe *SchedulerManager*, apresentado na Figura 6.9. Este objeto executa a ação do estado corrente de cada atuador e ator ativo na simulação. Este possui métodos para inserir (*scheduleTask*) ou remover (*cancelTask*) objetos na simulação do jogo. O método *processTasks* recebe como argumento um intervalo de tempo durante o qual, executa tarefas pendentes em todos os objetos que estejam ativos na simulação, de acordo com suas prioridades.

Para gerenciar todos estes elementos, o *framework* oferece uma classe chamada *ServerSimulation* (Figura 6.10) que possui referências a todos elementos gerenciadores da simulação, no caso, os gerenciadores de objetos, o escalonador e o gerenciador das áreas de interesse. Esta

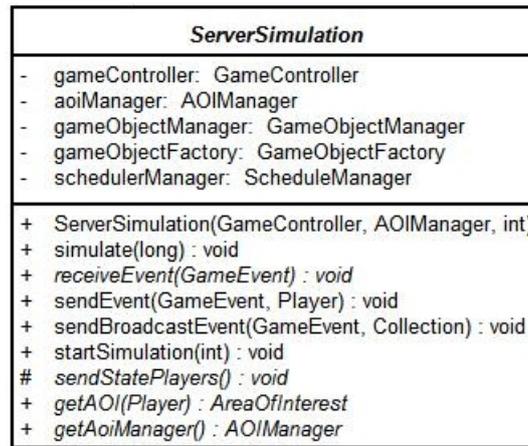


**Figura 6.8** Área de Interesse e Gerenciador de áreas de Interesse.



**Figura 6.9** Classe *SchedulerManager* – Escalonador da Simulação.

classe possui o método *simulate*, responsável por executar ações em todos os atuadores ativos no jogo em um dado instante. Este método basicamente delega a execução da simulação à classe *SchedulerManager*.



**Figura 6.10** Classe *ServerSimulation*, responsável pelo controle da simulação servidora.

## 6.5 Serviços de *Middleware* para Jogos PM2G

Nesta seção são apresentados os serviços da arquitetura descrita no início deste capítulo (vide Figura 6.2). Após uma explanação sobre o funcionamento de cada serviço, aspectos de modelagem de seus componentes são representados através de diagramas de seqüência e diagramas de classes em *Unified Modelling Language (UML)*.

### 6.5.1 Serviço de Gerenciamento de Contexto

Como ressaltado na Seção 2.3.2, contexto é um conceito-chave em uma aplicação pervasiva. Nestas aplicações, toda informação relevante e que caracterize a situação de uma entidade que compõe (*software/hardware*) ou interage (*usuário*) com a aplicação é chamada contexto. Aplicações pervasivas monitoram mudanças no contexto para adequar seu funcionamento em relação a estas entidades.

De acordo com o modelo de aplicação PM2G, o contexto de execução de um jogador tem papel fundamental na forma como este usuário percebe e interage com o jogo, norteando o funcionamento dos demais serviços e as possibilidades de interação entre jogadores. O ***Serviço de Gerenciamento de Contexto*** tem por finalidade realizar o monitoramento reativo do contexto de execução de um jogador, para que adaptações necessárias à participação do jogador no jogo possam ser realizadas pelos demais serviços.

## 6.5.1.1 Modelagem

A idéia de contexto apresentada no modelo de aplicação PM2G segue a idéia de um modelo informal de contexto. Neste modelo, as informações relevantes para determinar o modo de interação do jogador são: (i) o dispositivo utilizado para acesso ao jogo, (ii) preferências pessoais e (iii) a localização física de cada jogador, quando este utilizar um dispositivo móvel e puder ser rastreado ou enviar informações sobre sua localização – (Figura 6.11).

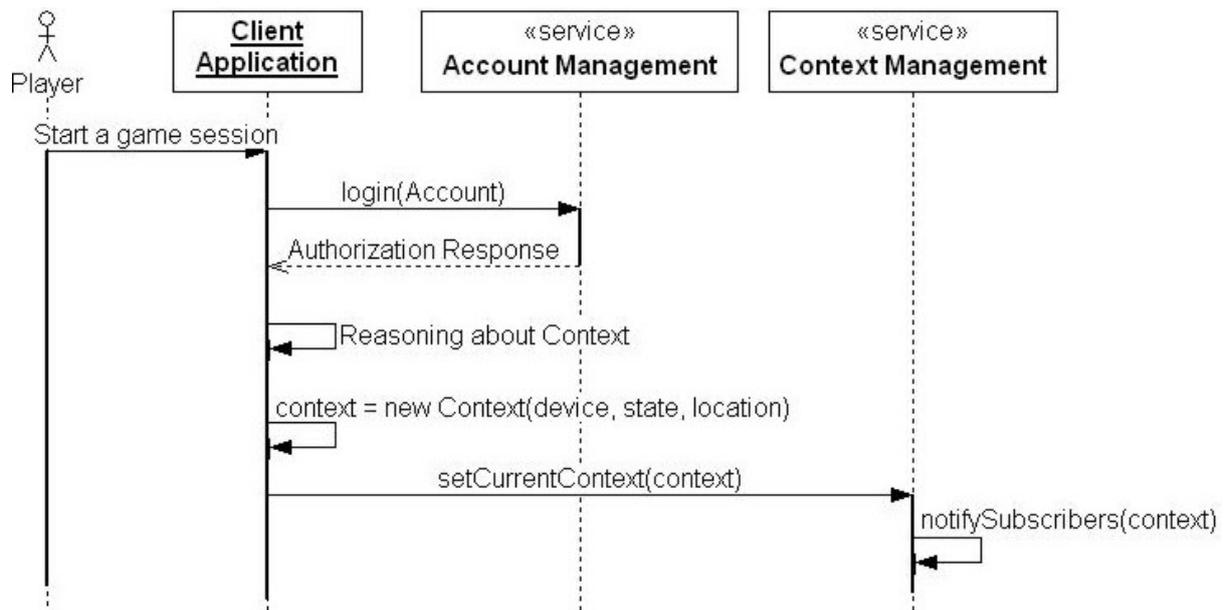


**Figura 6.11** Modelo de Contexto PM2G

O dispositivo é o elemento principal do contexto, pois indica como a informação do mundo virtual será apresentada ao jogador, quais são as possibilidades de conectividade do jogador, dentre outras funcionalidades do *middleware*. No caso do uso de preferências pessoais, cria-se a idéia de um estado ou comportamento associado a cada jogador, a ser visto a seguir (Tabela 6.2, página 94). Este estado visa evitar possíveis efeitos colaterais, como excesso de notificações ao jogador sobre eventos do jogo, pois apesar de uma das principais motivações para esta proposta seja proporcionar ao jogador a sensação de nunca estar desconectado do jogo, em certas ocasiões, isto será inevitável. Por fim, a localização física de um jogador permite que outros jogadores possam ter ciência (mesmo que apenas aproximada), de potenciais adversários para realização de mini-mundos, bem como para captura de objetos “virtuais” inseridos em posições do mundo real. A informação sobre a localização física pode ser atualizada de forma explícita ou através de um rastreamento periódico, de acordo com uma política de preferências do usuário, bem como das possíveis formas de obtenção da posição física do jogador, como *Global Positioning System (GPS)*, caso este esteja integrado ao dispositivo móvel.

A conotação reativa do monitoramento de contexto procura passar a idéia de que o fornecimento de informações sobre sua mudança de contexto é realizada pelo jogador, quando da migração de sessões entre dispositivos, mudança de localização física ou alteração de preferências pessoais. Cada aplicação utilizada por um jogador informa ao serviço informações sobre o contexto do jogador. Esta informação é repassada, inicialmente, após o processo de abertura de uma sessão com o jogo (Figura 6.12).

Como ressaltado na Seção 5.4, mudanças de dispositivos demandam um novo processo de autenticação com o *middleware*. A partir desta autenticação, seguem a informação do contexto



**Figura 6.12** Notificação de mudança de contexto PM2G

atual do jogador.

Para a identificação do dispositivo utilizado pelo jogador por parte do serviço, é necessária a criação de um Repositório de Dispositivos na plataforma servidora, onde são armazenadas informações sobre cada modelo de dispositivo suportado pelo jogo. Este repositório possibilita a criação de uma referência única para qualquer modelo de dispositivo utilizado por um jogador.

A identificação de cada tipo de dispositivo é feita de acordo com códigos atribuídos a cada modelo suportado pelo *middleware*. Quando da instalação de uma aplicação por um jogador, o código do dispositivo é atrelado à aplicação, permitindo que o mesmo seja repassado ao serviço quando do acesso do jogador através do dispositivo.

O dispositivo utilizado em um dado momento permite classificar um usuário como jogador estacionário ou móvel. Esta classificação é possível pois cada entrada no repositório de dispositivos identifica um dispositivo quanto ao contexto em que o mesmo é utilizado (estacionário ou móvel), e uma série de propriedades do dispositivo como: suas possibilidades de comunicação (*bluetooth*, Wi-Fi ou outras) e suporte a rastreamento via *Global Positioning System (GPS)*. A Tabela 6.1 apresenta uma visão simplificada deste repositório.

Em relação às preferências de um jogador, o serviço permite que cada jogador estabeleça o conceito de *estado* de execução para controle da participação do jogador no jogo. Os possíveis estados de um jogador são representados na Tabela 6.2. O gerenciamento deste estado é de responsabilidade do jogador, controlando-o de acordo com seu interesse.

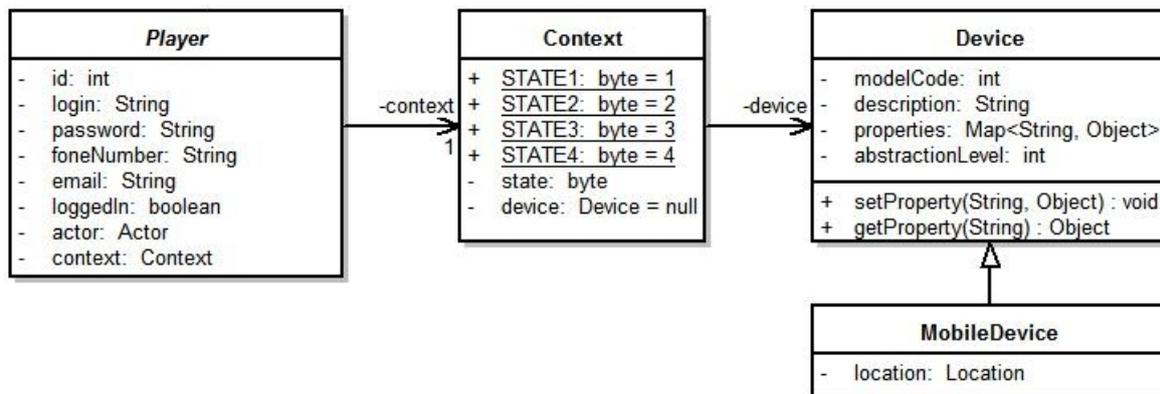
**Tabela 6.1** Repositório de Dispositivos.

Identificação do Dispositivo	Modelo	Contexto de Uso	Wi-Fi	Bluetooth	GPS
1	Celular Nokia N91	Móvel	Sim	Sim	Sim
2	PC	Estacionário	Não	Não	Não
3	Pocket	Móvel	Sim	Sim	Não

**Tabela 6.2** Estados de um jogador

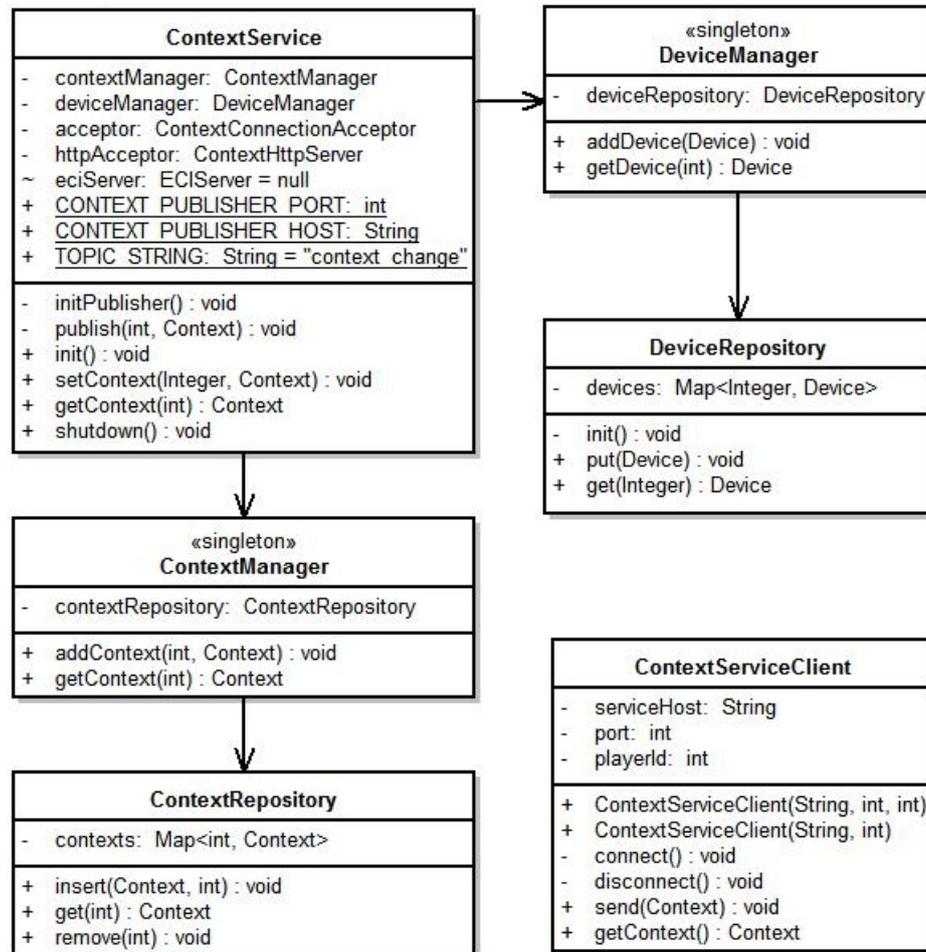
Estado	Notificações de Eventos	Mini-mundos
1	Disponível	Disponível
2	Disponível	Indisponível
3	Indisponível	Disponível
4	Indisponível	Indisponível

O serviço associa um contexto a cada jogador, de acordo com seu dispositivo, estado e localização. Para criar esta associação, o serviço utiliza a base de dados dos usuários do jogo, presumindo para isto que cada usuário do jogador pode ser identificado unicamente no *middleware* pelo login de acesso. A Figura 6.13 apresenta a relação entre jogador e contexto na visão de um diagrama de classes UML. Cada jogador (Classe *Player*) possui um contexto (Classe *Context*), cujos principais atributos são um estado (atributo *state*) e um dispositivo (Classe *Device*). No caso específico de jogadores com dispositivos móveis, é utilizada a classe *MobileDevice*, que além de um lista de propriedades como tamanho de tela, número de cores suportadas e número do telefone (no caso de celulares), possui uma localização associada. As informações do contexto de cada jogador são guardadas de forma persistente através de um Repositório de Contexto.

**Figura 6.13** Diagrama de Classes para informações de Contexto.

## 6.5.1.2 Arquitetura

Para o funcionamento do Serviço de Gerenciamento de Contexto, propõe-se a idéia de um Gerenciador de Contexto – *ContextManager* (Figura 6.14), que monitora a mudança de dispositivo ou alteração do estado do jogador. Este gerenciador é o núcleo do serviço, responsável pela sinalização de mudança do contexto para os demais serviços do *middleware*.



**Figura 6.14** Diagrama de Classes – Serviço de Gerenciamento de Contexto

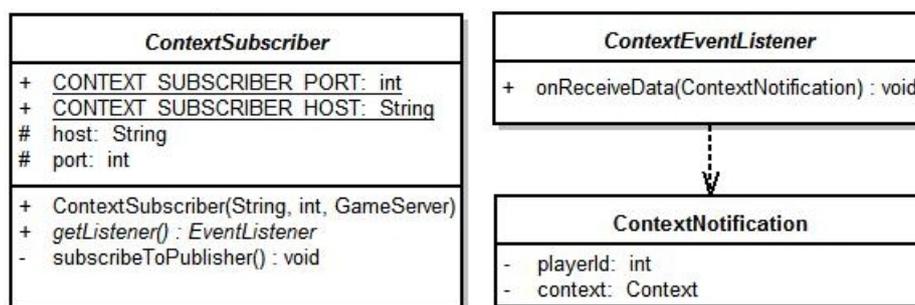
Para o seu correto funcionamento, o gerenciador interage com outros componentes do serviço. Primeiramente, o *DeviceRepository* encapsula as funções do repositório de dispositivos, permitindo busca e inserção de dispositivos suportados pelo jogo. O repositório de contexto, *ContextRepository*, guarda a informação sobre o contexto de todos os jogadores em algum meio de persistência, oferecendo funções de manipulação destes dados para o gerenciador.

O Gerenciador de Contexto é notificado após o acesso dos jogadores ao jogo, de acordo com a aplicação-cliente, que via API, utiliza a classe *ContextServiceClient*. O Gerenciador de

contexto também deve atender à mudança de estado, de acordo com a solicitação explícita de um jogador, segundo a Tabela 6.2. Para tanto, o Gerenciador oferece operações tanto de manipulação completa, bem como para alteração de dados específicos do contexto de um jogador.

O serviço possui dois atributos do tipo *ContextConnectionAcceptor* e *ContextHttpServer*, que encapsulam classes que seguem o padrão de projeto *Acceptor* [Sch97]. O segundo trata especificamente da troca de mensagens utilizando protocolo HTTP. Esta diferenciação se faz necessária, pois segundo a especificação *Mobile Information Device Profile (MIDP) 2.0*, o protocolo HTTP é o único obrigatoriamente suportado pela plataforma J2ME, a ser utilizado pelas aplicações móveis.

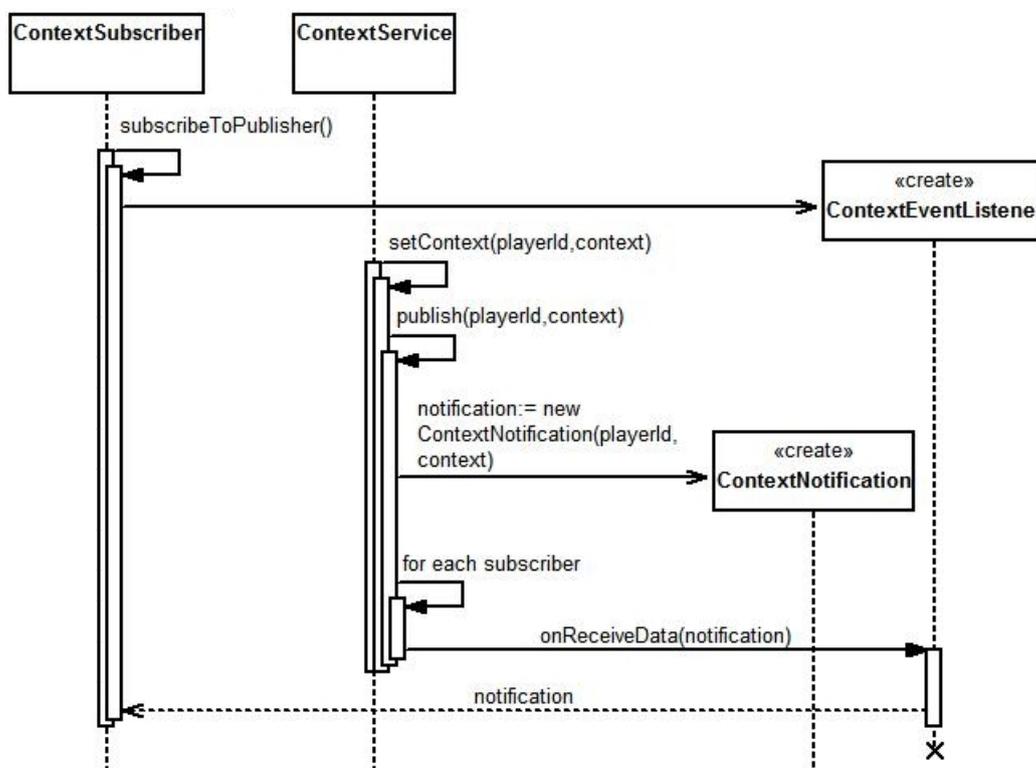
Devido ao papel central do Serviço de Gerenciamento de Contexto no funcionamento do *middleware*, este serviço deve dispor de um meio de indicar aos demais serviços interessados, mudanças no contexto dos jogadores. No caso, é proposta a idéia de um canal de comunicação entre o gerenciamento de contexto e os demais serviços. Com isso, este serviço segue a idéia de um fornecedor de eventos, onde serviços interessados cadastram-se no mesmo, sendo notificados quando da mudança no contexto de um jogador. Para modelar e implementar esta funcionalidade, o serviço de gerenciamento de contexto baseou-se na API *Event-based Communication Interface (ECI)* para comunicação baseada em eventos do *middleware* MoCA [LoAC05]. Esta API provê facilidades para implementar comunicação assíncrona usando o paradigma *publish/subscribe*, onde processos podem se inscrever em eventos relacionados a algum “assunto”, e/ou publicar eventos relacionados a este “assunto”. No nosso caso, o serviço de gerenciamento de eventos é o publicador de eventos onde o “assunto” será a mudança de contexto de um jogador. A Figura 6.15 apresenta as principais classes utilizadas neste modelo de comunicação.



**Figura 6.15** Diagrama de Classes – Classes relativas ao modelo publicador/assinante do Serviço de Gerenciamento de Contexto

Para isto, o serviço utiliza a classe *ECIServer*, a qual são delegadas as tarefas de gerenciar assinantes, bem como de notificar os interessados nas mudanças de contexto. Cada interes-

sado em mudanças de contexto utiliza a classe abstrata *ContextSubscriber* para se registrar como assinante do serviço de contexto – método *subscribeToPublisher*. Através do padrão *Observer*[GHJV95], cada assinante especifica o que deve ser realizado quando da mudança do contexto através de uma extensão da classe *ContextEventListener*. Esta classe tem como única operação o método *onReceiveData*, que recebe uma notificação de mudança de contexto (classe *ContextNotification*), onde são repassadas as informações sobre o jogador e seu novo contexto. O serviço de gerenciamento de contexto através de seu método *publish*, dispara esta notificação a todos assinantes interessados. A Figura 6.16 ilustra estes passos através de um diagrama de seqüência em UML.

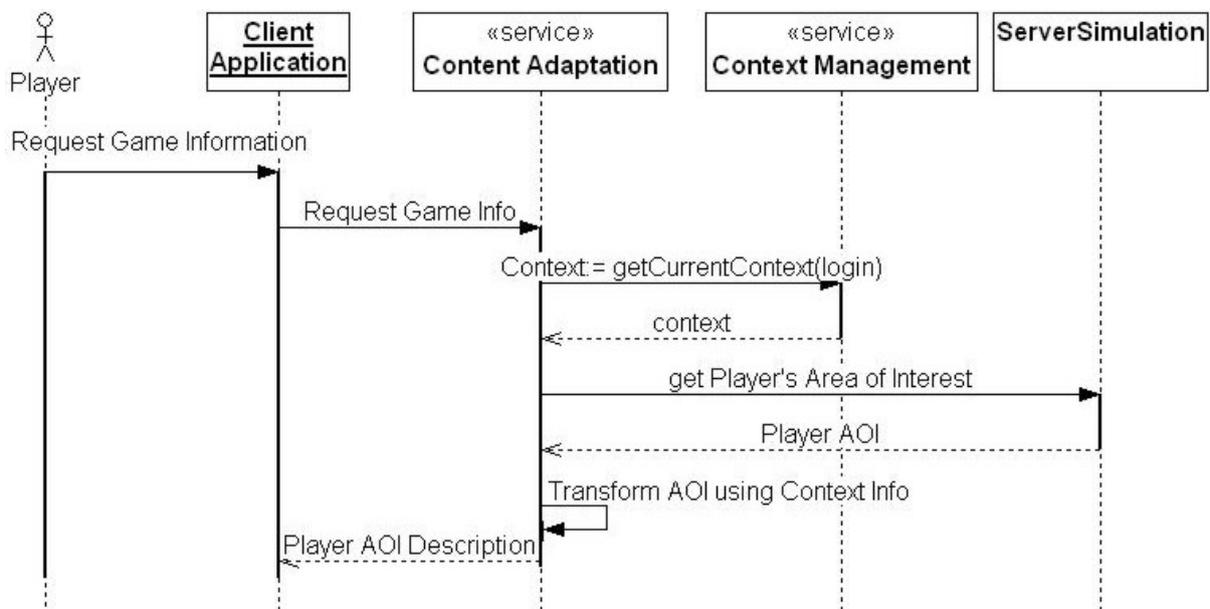


**Figura 6.16** Diagrama de Seqüência – Notificação de Eventos pelo Serviço de Gerenciamento de Contexto

### 6.5.2 Serviço de Adaptação de Conteúdo

A utilização de diferentes plataformas leva ao problema da heterogeneidade das características de cada uma delas. Diferentes tamanhos de tela, capacidade de memória e de processamento, largura de banda e latência da conexão com a rede são alguns exemplos. Este problema demanda que as informações enviadas ao jogador sejam adaptadas de acordo com o seu dispositivo utilizado, sendo apresentadas da forma mais adequada.

No modelo de aplicação PM2G, a informação relevante a cada jogador é composta por duas partes. A primeira refere-se à *área de interesse* simulada no mundo virtual, seguindo o modelo de aplicação PM2G. A segunda é composta por informações relativas ao seu mundo real, como proximidade de objetos virtuais ou de mini-mundos em sua cercania. Esta segunda informação é dependente do estado do jogador, que indica seu interesse ou não em receber tais informações. A função do Serviço de Adaptação de Conteúdo é transformar estas informações relevantes, para um formato adequado ao contexto atual de execução de cada jogador, seguindo o conceito de *percepção* do modelo de aplicação PM2G. A Figura 6.17 apresenta uma versão simplificada de um diagrama de seqüência que ilustra os passos realizados na adaptação de conteúdo em um jogo PM2G.



**Figura 6.17** Diagrama de Seqüência – Serviço de Adaptação de Conteúdo.

A utilização deste serviço evita que os dispositivos móveis sejam responsáveis por procedimentos de transformação de conteúdo, o que aconteceria no caso do estabelecimento de um

formato único independente de dispositivo. Esta situação exigiria a execução de uma tarefa inadequada para dispositivos com restrições de poder de processamento, memória e conectividade.

#### 6.5.2.1 Modelagem

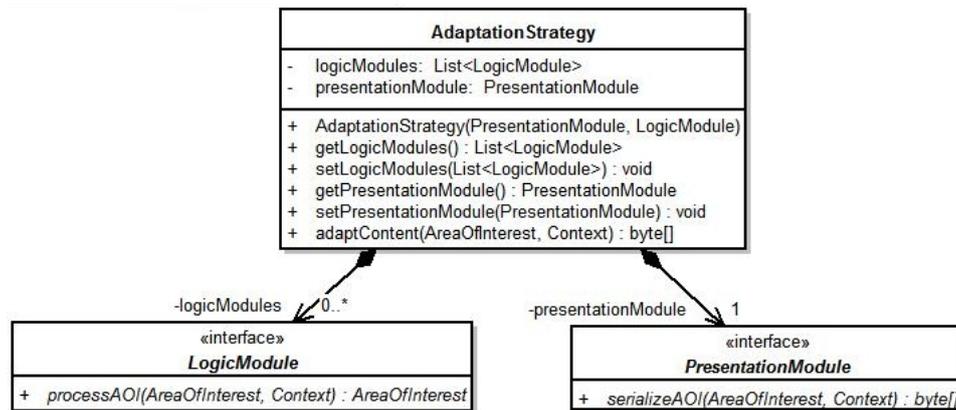
Para definir como cada dispositivo determina a adaptação de conteúdo a ser realizada, foi criado o conceito de *estratégia de adaptação*. Uma estratégia de adaptação é associada a cada dispositivo, e indica quais transformações são realizadas nas informações a serem enviadas ao jogador.

Para modelar este serviço, foi observado que o conjunto de dispositivos abordados pode ser ordenado no que diz respeito à quantidade de informações recebidas pelo jogador, característica que se relaciona com a sensação de imersão provocada no mesmo. Com isto em mente, foi criado um atributo na classe *Device* (Figura 6.13) que diz respeito ao nível de abstração do dispositivo. Quanto maior for este valor, menos informações o jogador que o utiliza irá receber. Um computador pessoal de mesa, por exemplo, pode ser considerado o dispositivo com o menor nível de abstração em um determinado jogo. Visto que o jogador que utiliza um computador pessoal deseja uma maior sensação de imersão no jogo, as informações precisam ser apresentadas ao jogador de maneira detalhada, o que exige muita troca de dados entre a aplicação-cliente e o servidor. Já um aparelho celular de limitados recursos computacionais oferece uma mínima sensação de imersão ao jogador. Sendo assim, são poucas as informações recebidas pela aplicação-cliente.

Cada elemento da área de interesse também deve conter uma informação a respeito da sua importância, para que o serviço saiba quais objetos podem ser descartados e quais devem ser enviados ao jogador. Para isto, foi incluído na classe *GameObject* (Figura 6.6) o atributo *priority*. Esta informação permite ao desenvolvedor do jogo criar categorias como, por exemplo, essencial, importante ou opcional, para cada ator, item ou NPC contido em uma área de interesse.

Diversos tipos de processamento podem ser realizados sobre as informações de conteúdo enviadas ao jogador. Neste trabalho, estes processamentos são divididos em duas categorias: *processamentos lógicos* e *processamentos de apresentação*. A primeira categoria engloba procedimentos que modificam o conteúdo da área de interesse, como, por exemplo, a quantidade de objetos ou a posição destes elementos no mundo virtual. Por sua vez, processamentos de apresentação preparam os dados para serem enviados à aplicação-cliente. Dependendo da forma com que estas informações são enviadas e também do dispositivo utilizado, pode ser necessá-

rio que estas informações recebidas pela aplicação-cliente sejam apresentadas ao jogador com a menor quantidade de processamento local possível. A Figura 6.18 apresenta o diagrama de classes para o conceito de estratégia de adaptação e suas relações com os processamentos lógicos e de apresentação.



**Figura 6.18** Diagrama de Classes – Estratégia de Adaptação.

Cada forma de processamento é representada por um módulo de adaptação de conteúdo. Desta maneira, os módulos são classificados em módulo lógico ou módulo de apresentação, que implementam, respectivamente, a interface *LogicModule* ou *PresentationModule*. A princípio, foram definidos os seguintes módulos lógicos:

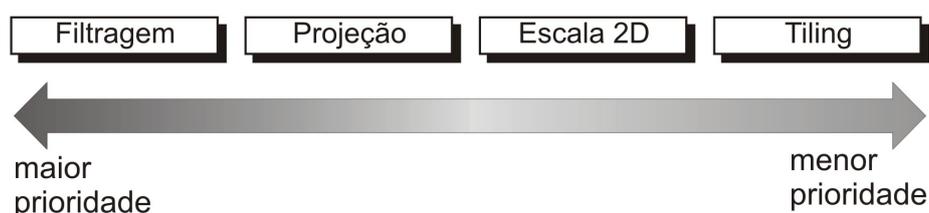
- *Módulo de filtragem de informações (FilteringModule)*: baseando-se no nível de abstração do dispositivo e na prioridade de cada elemento da área de interesse, este módulo é responsável por descartar os elementos que não precisem ser enviados ao jogador. Isto traz duas vantagens: a primeira é a redução do tráfego na rede, o que pode ser importante para jogadores utilizando algum tipo de conexão cuja cobrança é dada pela quantidade de bytes transmitidos, como GRPS. Outra vantagem é a menor carga de processamento e memória exigidos do dispositivo, já que são menos objetos a serem processados pela aplicação-cliente;
- *Módulo de escala bidimensional (ScalingModule)*: ajusta as posições dos objetos de uma área de interesse (bidimensional) baseando-se na resolução do dispositivo utilizado. Apesar de ser uma operação simples, o fato de ser realizada pelo serviço e não no dispositivo cliente diminui a carga de processamento neste último;
- *Módulo de tiling (TilingModule)*: adapta as posições dos objetos de uma área de interesse (bidimensional) para que possam ser apresentados como em jogos baseados em *tiles*

(*tile-based games*). Nestes jogos, um mapa é formado por blocos bidimensionais de igual tamanho, que o preenchem completamente e sem sobreposição. Logo, a posição de qualquer objeto apresentado neste mapa deve assumir uma quantidade limitada de valores, que podem ser mapeados por uma matriz com N linhas e M colunas;

- *Módulo de projeção tridimensional (ProjectionModule)*: realiza a projeção tridimensional das posições dos objetos para valores bidimensionais. Isto é muito útil em jogos 3D que precisem ser apresentados em dispositivos cujas características inviabilizem a utilização de gráficos tridimensionais (como, por exemplo, o tamanho da tela ou a capacidade de processamento). A posição dos objetos da área de interesse enviada ao jogador será definida, portanto, de maneira bidimensional.

No serviço, para cada estratégia de adaptação, podem ser utilizados quantos módulos lógicos forem necessários. Apesar de somente quatro terem sido concebidos, o serviço é extensível para que outros possam ser criados e acoplados à nossa abordagem. Desta forma, o resultado da transformação realizada por um módulo lógico serve de entrada para o módulo seguinte. Os módulos lógicos acima descritos possuem a propriedade de que, não importa a ordem em que eles processam um mesmo conjunto de dados de uma área de interesse, o resultado será sempre o mesmo. Contudo, o custo de processamento pode variar dependendo desta ordem. Por exemplo, caso o módulo de escala bidimensional trate o conteúdo de uma área de interesse antes do módulo de filtragem, será ajustada a posição de elementos que possivelmente serão descartados no módulo seguinte. Desta maneira, percebe-se que é mais vantajoso ordenar os dois módulos para que o de filtragem seja executado antes do de escala.

Em consequência, a cada módulo lógico é atribuída uma prioridade. Módulos com maior prioridade são executados primeiro em detrimento àqueles de menor prioridade. A seqüência mais comum para prioridades dos módulos lógicos descritos é apresentada na Figura 6.19. Nesta seqüência, prioriza-se a execução de módulos que possam facilitar o processamento dos módulos seguintes.



**Figura 6.19** Prioridades entre processamentos lógicos do Serviço de Adaptação de Conteúdo.

Por sua vez, em relação ao processamento de apresentação, foram definidos os seguintes módulos:

- *Módulo de apresentação textual*: baseado no idioma escolhido pelo jogador, este módulo cria um texto informando resumidamente o conteúdo da área de interesse. O idioma é armazenado como uma propriedade do dispositivo, obtido a partir do seu contexto. Desta forma, são informados quantos atores, itens e NPCs estão presentes, sendo também incluída a representação textual de cada um deles. É permitido ao desenvolvedor do jogo escolher como deve ser criada a representação textual de cada elemento na criação das entidades que herdam de *GameObject*;
- *Módulo de apresentação padrão*: neste módulo todas as informações resultantes da área de interesse são transmitidas de uma maneira padrão, de acordo com um protocolo de conhecimento do desenvolvedor do jogo.

Cada módulo de apresentação define inteiramente a forma com que o conteúdo da área de interesse é enviada ao jogador. Assim, faz sentido que somente um módulo de apresentação seja utilizado em cada estratégia de adaptação. Vale ressaltar que, se o desenvolvedor preferir, pode ser adicionado ao serviço outro módulo de apresentação para que os dados sejam transmitidos da forma desejada.

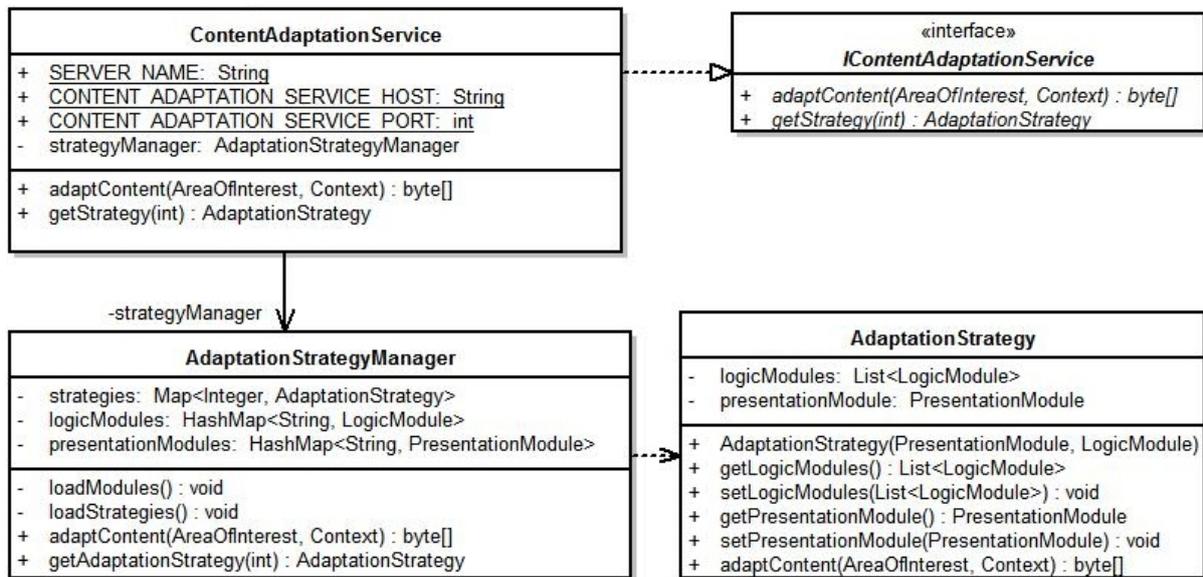
É importante ainda observar que o caráter genérico do serviço exige que as estratégias de adaptação utilizadas pelo jogo devem ser previamente definidas pela simulação do jogo. Deste modo, na inicialização do servidor do jogo, as estratégias são criadas e repassadas ao serviço.

Finalmente, para evitar uma degradação do desempenho do sistema na adaptação de conteúdo para jogadores com atualização freqüente do estado do jogo, foi utilizada a idéia de uma *cache* para armazenar localmente a estratégia de adaptação do jogador estacionário. Desta forma, o serviço oferece também uma forma de obter a estratégia de adaptação para um determinado dispositivo, de forma que a simulação de jogo possa obter a adaptação de conteúdo de um jogador sem a necessidade de chamar remotamente o serviço a cada ciclo de jogo.

#### 6.5.2.2 Arquitetura Interna

A Figura 6.20 apresenta a arquitetura proposta para o Serviço de Adaptação de Conteúdo como um diagrama de classes, utilizando a notação UML.

O Serviço de Adaptação de Conteúdo, representado pela interface *IContentAdaptationService*, fornece duas operações. A primeira, *adaptContent*, realiza a adaptação do conteúdo de



**Figura 6.20** Diagrama de Classes – Arquitetura Interna do Serviço de Adaptação de Conteúdo.

um jogador, tomando por base sua atual área de interesse e seu contexto atual. A segunda, *getStrategy*, permite que seja obtida qual a estratégia de adaptação (*AdaptationStrategy*), de acordo com o dispositivo usado pelo jogador. Cada uma destas operações é detalhada utilizando diagramas de seqüência.

O fluxo de mensagens da operação *adaptContent* ocorre de acordo com a Figura 6.21. Ela possui dois parâmetros: a área de interesse do jogador e o seu contexto. Logo que a operação é chamada a partir da simulação do jogo, o serviço delega a lógica da operação para o *AdaptationStrategyManager*, objeto que armazena todas as estratégias de adaptação previamente definidas. Este verifica qual a estratégia de adaptação que será utilizada, baseando-se no contexto do jogador, especificamente sobre o seu dispositivo. Para cada módulo lógico da estratégia de adaptação (se houver algum), a área de interesse sofre um processamento até ser serializada pelo módulo de apresentação da estratégia. O resultado é, então, retornado à simulação do jogo.

A operação *getStrategy* pode ser utilizada pela simulação de jogo para realizar a adaptação de conteúdo de jogadores estacionários, cuja taxa de atualização do estado do jogo pode ser muito freqüente. Desta forma, a simulação obtém a estratégia de adaptação do jogador e a armazena em uma *cache*, fazendo com que ela realize a adaptação localmente da mesma maneira que é feita no serviço. O fluxo de mensagens é mostrado na Figura 6.22.

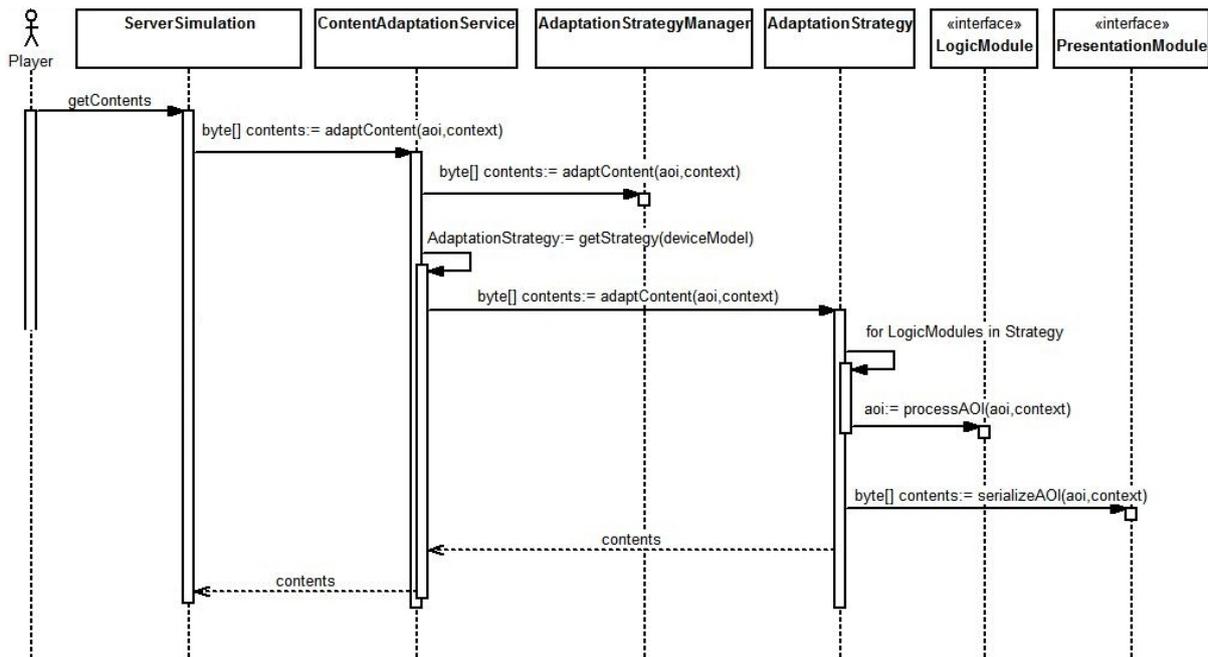
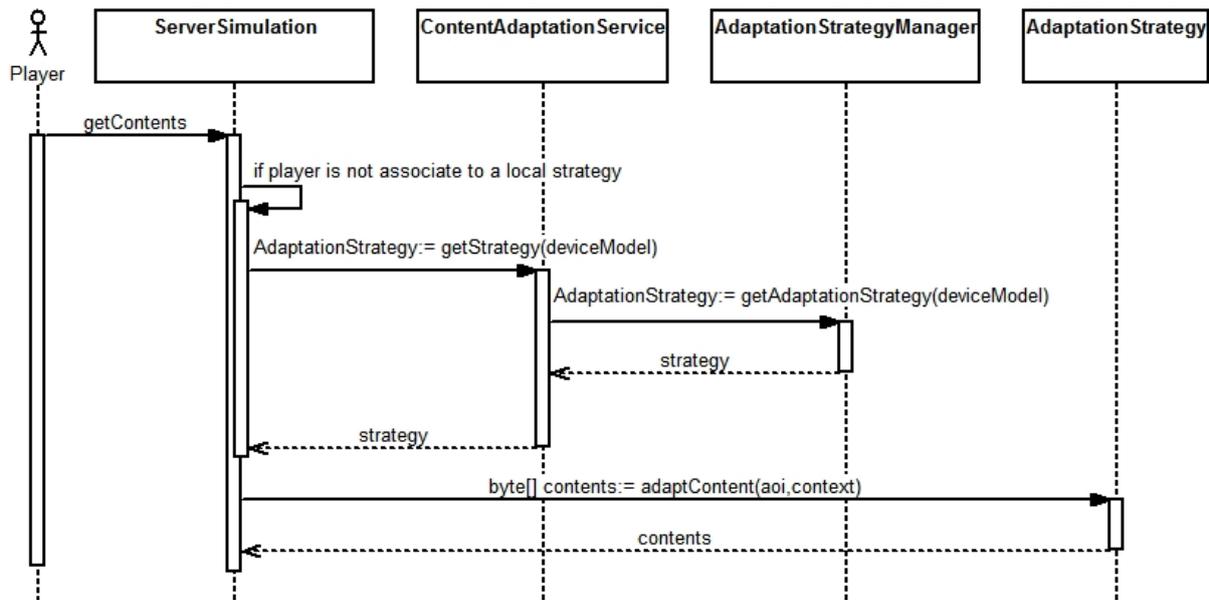


Figura 6.21 Diagrama de Sequência – Operação *adaptContent*.

### 6.5.3 Serviço de Adaptação de Interação

O uso de diferentes dispositivos indica que as entradas das ações dos jogadores móveis e estacionários são realizadas de formas diferentes. Porém, independente da configuração de execução do jogador, estas diferentes ações devem ser mapeadas para ações comuns dentro do mundo virtual do jogo. O modelo de aplicação PM2G permite que jogadores estacionários e móveis interajam diretamente no mundo virtual, realizando ações equivalentes.

O Serviço de Adaptação de Interação propõe-se a transformar as diferentes formas de entradas dos jogadores em diferentes contextos em ações padronizadas dentro de cada área de interesse do jogo. Em um contexto estacionário, a atuação do serviço pode ser dispensável, uma vez que o jogador estará efetivamente controlando as ações do seu personagem. No contexto móvel, por sua vez, a adaptação de interação é imprescindível, visto que a limitação da interface homem-máquina dos dispositivos móveis faz com que as típicas ações baseadas em movimentações ágeis do avatar sejam inadequadas neste contexto. Sendo assim, as ações de um jogador móvel devem ser realizadas sem a necessidade de uma conexão constante com o jogo ou de respostas imediatas aos seus comandos, podendo o jogador disparar uma ação a ser executada no mundo virtual enquanto ele realiza outras atividades fora do jogo, como deslocamento físico ou atendimento de chamadas no aparelho celular.



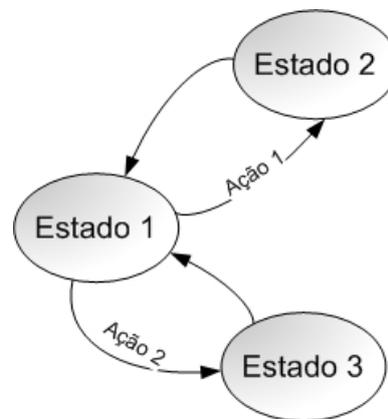
**Figura 6.22** Diagrama de Sequência – Operação *adaptStrategy*.

No Serviço de Adaptação de Interação, estes requisitos são acatados através da idéia proposta por Fox [Fox03]. As ações de jogadores móveis são realizadas no mundo virtual através de comandos de “alto nível”, que são expandidos em ações concretas. Na realidade, esta idéia segue exemplos já existentes em jogos comerciais. Por exemplo, o jogo Starcraft [BE05] permite que o jogador coloque tropas em estado de patrulha, onde em caso da presença de um adversário, as tropas atacam automaticamente. Este jogo permite também que tropas desloquem-se no mapa, através da indicação apenas de seu destino, sem que seja necessário um efetivo controle de seus avatares. O comportamento do personagem após o comando executado seria semelhante a um NPC, possivelmente controlado por um procedimento de inteligência artificial.

### 6.5.3.1 Modelagem

Para modelar este serviço, foi necessário definir como seria a interação do usuário com o jogo em um dispositivo móvel de interface homem-máquina limitada. Uma solução encontrada é fazer com que o jogo envie ao jogador o conjunto de ações de alto nível possíveis do seu personagem. O jogador, então, informa ao servidor do jogo a ação escolhida e este se responsabiliza, utilizando o Serviço de Adaptação de Interação, a mapear o comando em ações padrões do jogo.

Para realizar a obtenção das ações possíveis de um avatar em um dado instante, convém definir quais aspectos do jogo influenciam este procedimento. Em primeiro lugar, foi definido que o estado do avatar é um ponto de partida para descobrir quais ações ele pode realizar. Afinal, cada ação pode ser vista como uma espécie de transição de estados: ela indica qual o comportamento que o personagem deverá assumir a partir daquele momento. A Figura 6.23 exemplifica esta idéia. Nela estão associadas duas ações ao Estado 1 (Ação 1 e Ação 2), enquanto que a partir dos outros estados, não é possível realizar nenhuma ação. As transições que saem dos estados 2 e 3 não são, portanto, definidas por comandos do jogador.

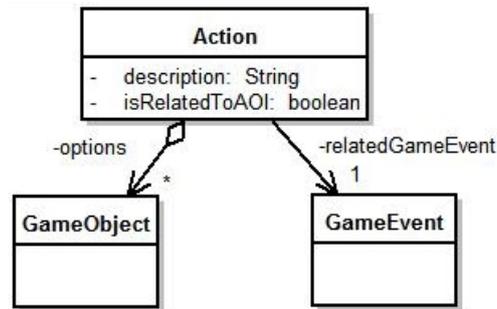


**Figura 6.23** Exemplo da relação entre estados e ações de um avatar.

Entretanto, é importante notar que não apenas o estado do jogador determina as suas ações. Dependendo do conteúdo da área de interesse, determinadas ações podem ser realizadas ou não. Por exemplo, um jogador não poderá realizar nenhuma ação de interação com itens virtuais caso não haja nenhum item virtual visível na sua área de interesse. Do mesmo modo, ele também não poderá interagir com outro personagem caso ele seja o único personagem em sua área de interesse naquele momento. Assim, foi definido que o conteúdo da área de interesse também influencia na obtenção das ações do jogador.

Com isto em mente, foi criada a entidade *Action*, que representa as ações de alto nível dos jogadores (Figura 6.24). Esta entidade possui uma descrição textual, usada na exibição das ações para os jogadores, além do atributo *isRelatedToAOI*, que informa se a ação está relacionada ao conteúdo da área de interesse do jogador. Esta informação é importante para permitir que algumas ações possam ocorrer sempre, independentemente do conteúdo da área de interesse do jogador. Um exemplo é uma ação para se desconectar do jogo, que a princípio pode ser realizada a qualquer momento.

Um objeto *Action* está sempre relacionado a um evento de jogo (*GameEvent*). Esta relação determina de maneira simples como o jogo irá interpretar a ação, pois um *GameEvent* é a forma



**Figura 6.24** Diagrama de Classes – Ação de um Jogador.

padrão com que a simulação de jogo trata os eventos recebidos dos jogadores.

Uma ação, como já foi dito, pode estar intrinsecamente relacionada a um elemento específico da área de interesse do jogador. Pode ser, por exemplo, um outro personagem com o qual se deseja interagir, ou um item qualquer. Deste modo, deve ser possível ao jogador, no momento em que seleciona uma ação deste tipo, especificar o objeto do jogo ao qual a ação se refere. Isto é possível com o uso de uma lógica do serviço que, ao oferecer uma ação a um jogador, liste opções relativas àquela ação. Por causa disto, ao objeto *Action* é atribuído um conjunto de objetos *GameObject*, cada um representando uma opção daquela ação que pode ser escolhida pelo jogador.

Para saber se um objeto da área de interesse será considerado uma opção de uma determinada ação, foi criado o atributo abstrato *isRelatedToAction*, que deve determinar para cada ação do jogo se o objeto se relaciona ou não com ela.

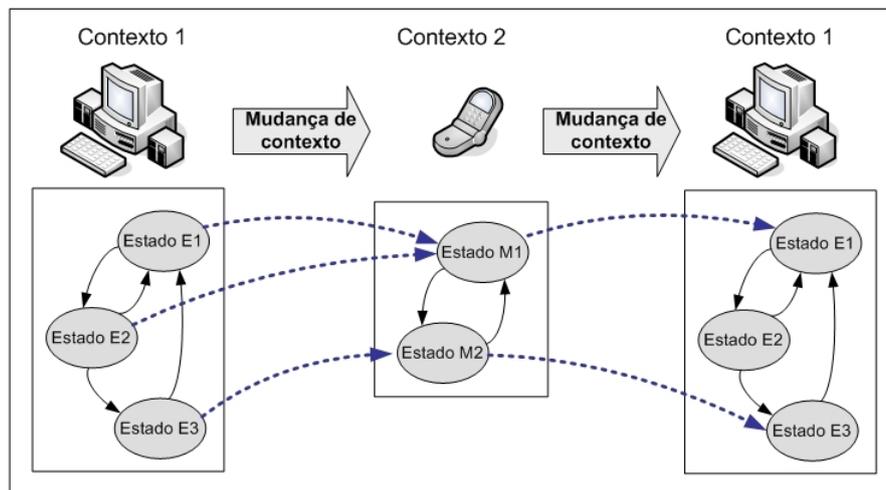
Vale ressaltar que o serviço, por ser genérico, não tem como identificar que ações podem ser realizadas para cada estado de um avatar. Na realidade, o jogo é que define quais são os seus estados e ações. Isto ocorre na inicialização do servidor do jogo, quando este mapeamento é definido e informado ao serviço.

Foi visto na Seção 6.5.2 que o Serviço de Adaptação de Conteúdo toma ciência do contexto de execução do jogador sempre que é chamado pela simulação do jogo. O Serviço de Adaptação de Interação, por sua vez, foi modelado para que não precise desta informação a todo momento.

Cada avatar possui um conjunto de estados que pode assumir, como já foi explicado. O que foi definido durante a modelagem deste serviço é que cada estado está diretamente relacionado a um nível de abstração do contexto de execução. O atributo *abstractionLevel* de um dispositivo (classe *Device*), além de ser utilizado pelo Serviço da Adaptação de Conteúdo para determinar a quantidade de informações enviadas ao jogador, passa a indicar o nível de abstração das suas ações.

Existem, na realidade, vários conjuntos de estados, para cada contexto que o jogador pode assumir no jogo. É desta forma que este serviço tem condições de saber quais ações o jogador poderá realizar para um dado contexto, baseando-se apenas no estado atual do seu avatar. Simplificadamente, caso o jogador esteja em um contexto móvel, seu avatar deverá assumir um entre os possíveis estados relacionados a este contexto. Se ele passar a jogar em um computador pessoal, o estado do seu avatar deverá refletir a mudança de contexto. Este processo também é realizado pelo Serviço de Adaptação de Interação. Ele deve saber, a partir do estado de um avatar e do novo contexto do jogador, qual o novo estado que será assumido.

A Figura 6.25 exemplifica a utilização de diferentes conjuntos de estados para cada contexto de execução, bem como associações entre estados de diferentes contextos, necessárias para atualizar o estado do avatar na mudança de contexto do jogador.



**Figura 6.25** Exemplo de mapeamento entre estados relacionados a diferentes contextos.

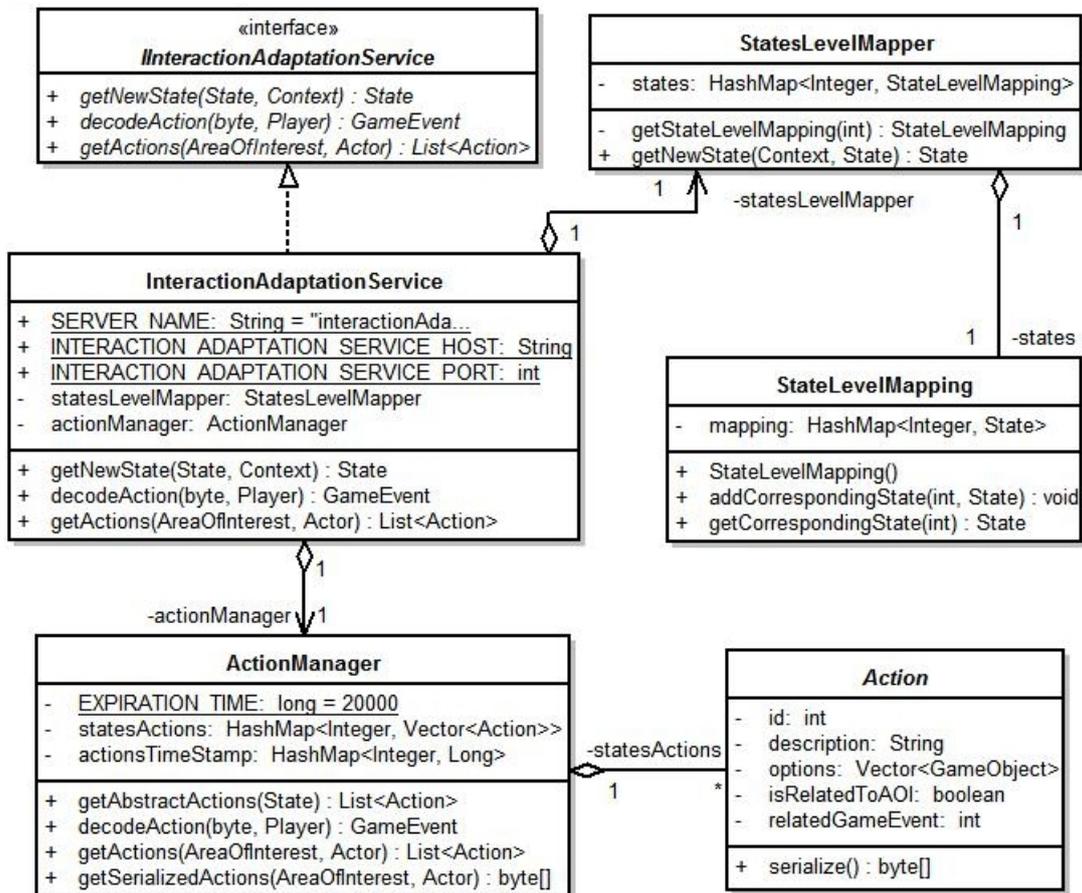
Devido ao seu caráter genérico, independente do jogo, o serviço deve ser informado pelo jogo como este procedimento deve ser realizado. O jogo deve, pois, associar cada estado a um certo número de outros estados, para cada contexto diferente do original.

Outro mecanismo oferecido pelo serviço é a utilização de um tempo de expiração para as ações enviadas ao jogador. Isto ameniza problemas de inconsistência que podem ocorrer quando um jogador móvel recebe um conjunto de ações e só executa a ação selecionada algum tempo depois.

Neste ínterim, o conteúdo da área de interesse pode ter mudado, de forma que tal ação não possa mais ser realizada. Assim, o serviço verifica se o prazo de realização da ação requisitada pelo jogador expirou ou não. Em caso positivo, o jogador deverá receber novamente um conjunto atualizado de ações para escolher qual ação tomar.

## 6.5.3.2 Arquitetura Interna

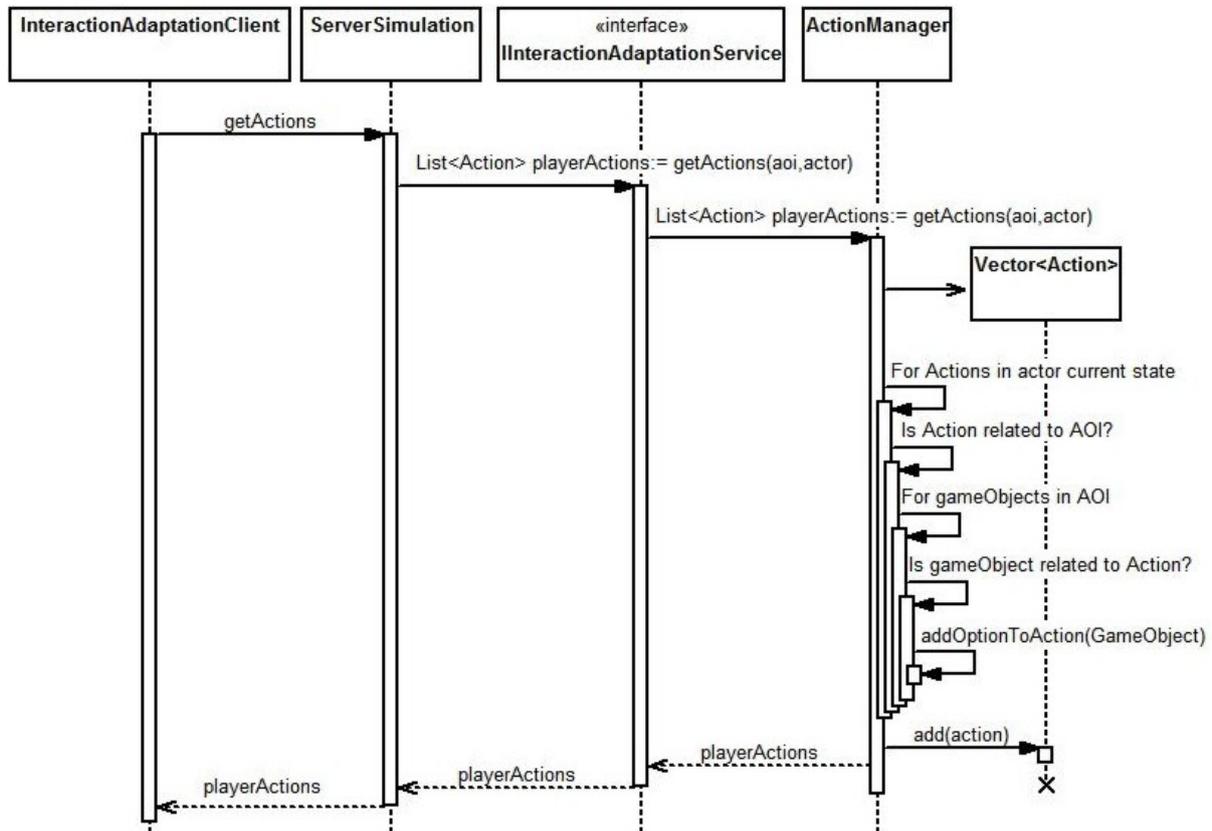
A Figura 6.26 apresenta a arquitetura proposta para o Serviço de Adaptação de Interação como um diagrama de classes em Unified Modelling Language (UML).



**Figura 6.26** Diagrama de Classes – Arquitetura Interna do Serviço de Adaptação de Interação

Com a operação *getActions* (Figura 6.27), o Serviço de Adaptação de Interação obtém as ações de um jogador. Esta operação é chamada pela simulação do jogo sempre que o jogador requisita as ações. Ela tem dois parâmetros: a área de interesse do jogador e o seu personagem. Assim que a operação é chamada, o serviço chama a operação de mesmo nome do *ActionManager*, que é a classe responsável por armazenar as ações para cada estado dos avatares e também realizar a lógica de obtenção dos estados a partir do conteúdo da área de interesse. Em seguida, é criado um conjunto vazio de ações ao qual serão adicionadas aquelas que satisfizerem determinadas restrições. O *ActionManager* obtém, então, as ações associadas ao estado do avatar para analisar quais delas devem ser enviadas ao jogador. Para cada uma destas ações, é verificado se ela tem relação com a área de interesse. Em caso negativo, a ação é imediatamente

adicionada ao conjunto resultante de ações, visto que nenhuma verificação adicional a respeito dela precisa ser feita. Este é o caso, como já mencionado, de uma operação de se desconectar do jogo, por exemplo.

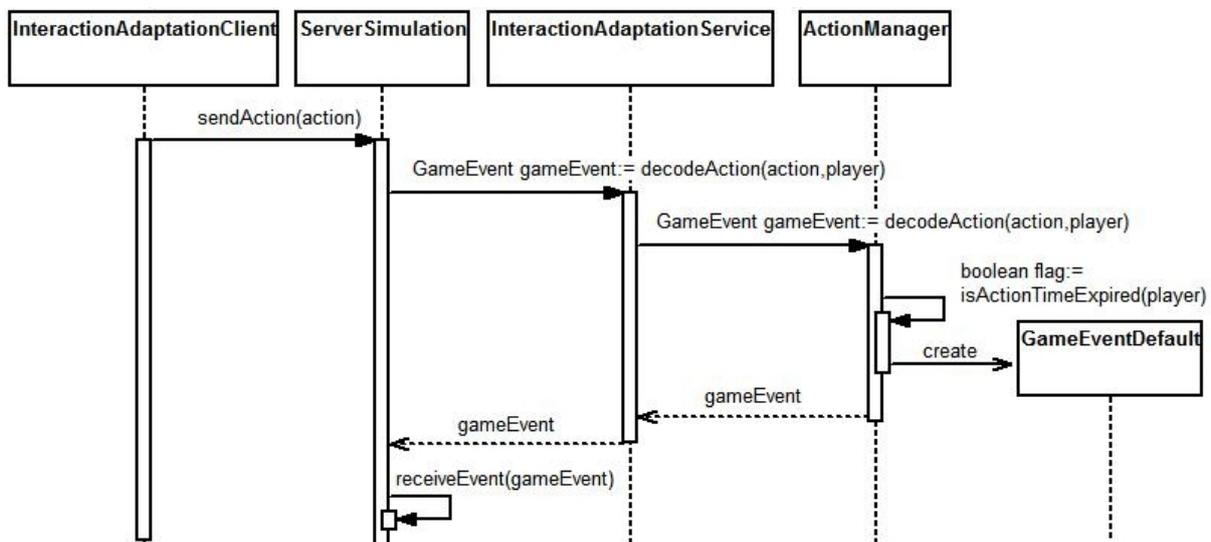


**Figura 6.27** Diagrama de Seqüência – operação *getActions*.

Contudo, caso a ação se relacione com a área de interesse, deve-se verificar se a área de interesse possui os elementos necessários para que a ação possa ser realizada. Isto é feito verificando-se a relação de cada objeto da área de interesse com aquela ação. Caso o objeto não tenha relação com a ação, é ignorado. Caso ele tenha relação com a ação, será incluído como uma opção daquela ação. Se a ação tiver ao menos uma opção, isto quer dizer que ela poderá ser realizada naquela área de interesse, sendo incluída no conjunto resultante de ações. Caso contrário, ela será descartada no momento e não será enviada ao jogador.

A operação *decodeAction* (Figura 6.28) do Serviço de Adaptação de Interação é responsável por mapear uma ação do jogador móvel em ações padrões do jogo. Esta operação resulta em um objeto *GameEvent*, que é o evento padrão de comunicação entre o servidor do jogo e clientes estacionários. Seus parâmetros são informações do jogador, obtidas pela simulação do

jogo quando a ação é enviada, e os dados serializados da ação. Tais dados indicam a ação escolhida e qual opção foi selecionada, caso tenha havido alguma opção para aquela ação. Assim que a operação é chamada pela simulação do jogo, o serviço chama a operação *decodeAction* de *ActionManager* para decodificar a ação e obter o evento de jogo relacionado. O *ActionManager*, por sua vez, apenas cria um novo *GameEventDefault*, implementação padrão da classe abstrata *GameEvent*, com os dados do jogador, da ação e da opção selecionada pelo jogador. Este evento é, então, retornado à simulação de jogo, que o processa normalmente como um evento de qualquer.



**Figura 6.28** Diagrama de Seqüência – operação *decodeAction*.

Por meio da operação *getNewState* (Figura 6.29) do Serviço de Adaptação de Interação, a simulação do jogo é capaz de obter o novo estado do avatar de um jogador quando ele muda de contexto de execução. Para tanto, precisa informar o estado atual do avatar e o novo contexto assumido pelo jogador. Logo que a operação é chamada pela simulação do jogo, é delegada a lógica da operação para um objeto *StatesLevelMapper*, que possui todos os mapeamentos entre cada estado e os estados de outro contexto que lhe são correspondentes. É obtido, então, o objeto *StateLevelMapping* relativo ao estado atual, que contém todos os estados correspondentes a este estado. Finalmente, sua operação *getCorrespondingState* permite obter o estado desejado passando como parâmetro o novo contexto do jogador.

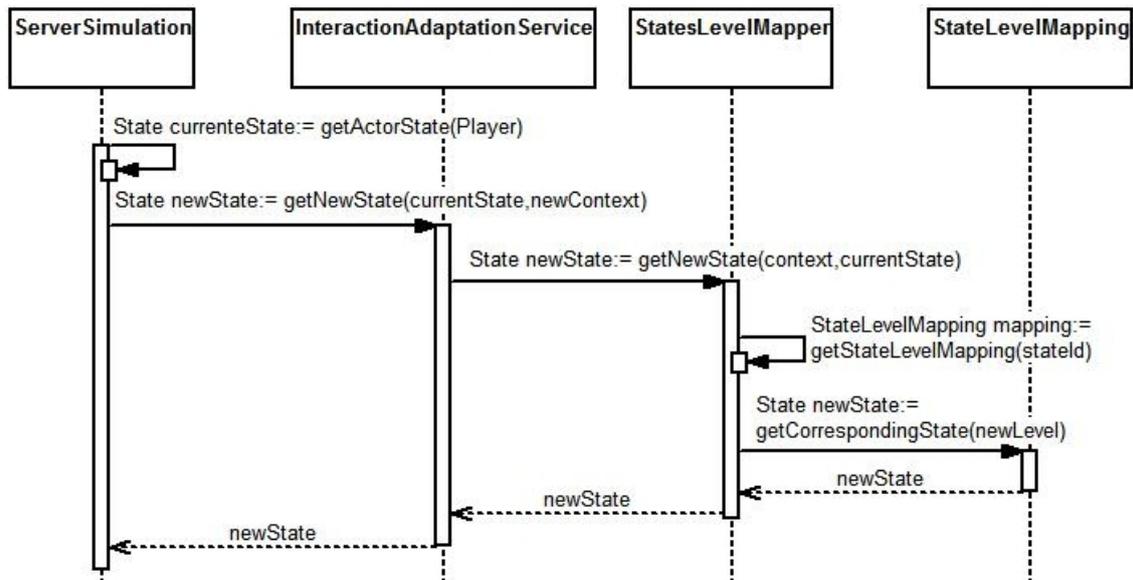


Figura 6.29 Diagrama de Seqüência – operação *getNewState*.

#### 6.5.4 Serviço de Notificação de Mensagens

O Serviço de Notificação de Mensagens tem a função de informar aos jogadores sobre acontecimentos de seu interesse, principalmente quando estes jogadores estiverem em um contexto móvel. Este serviço é o principal responsável por criar a idéia do jogador móvel estar sempre conectado ao jogo. A idéia é permitir que o jogador esteja sempre presente, sendo notificado através de diferentes meios de comunicação, como mensagens Short Message Service (SMS) ou e-mails.

No contexto estacionário, o jogador tem acesso direto ao mundo e seus eventos de interesse acontecem mediante sua presença, não fazendo sentido a emissão de notificações. O serviço de notificação de mensagens cria uma abstração para que jogadores móveis possam ser alertados sobre diferentes eventos relativos tanto em relação a eventos de sua área de interesse do mundo virtual, quanto a situações de seu mundo real, como desafios para partidas formadas por mini-mundos.

No mundo virtual, eventos representam acontecimentos relativos ao personagem do jogador, ao seu clã, ou qualquer outro elemento do jogo cuja alteração de estado seja de interesse do jogador. O comportamento assíncrono dos jogadores móveis indica que durante a execução de suas ações, seu personagem do jogador estará sujeito a influências de elementos de sua área de interesse, como ataque de outros jogadores. Este serviço surge com canal natural de aviso da execução e das possíveis interferências sofridas por estes comandos.

Em relação ao mundo real, este serviço também apresenta sua utilidade. O Serviço de Noti-

ificação de Eventos pode alertar a presença de objetos virtuais que estiverem situados próximos à localização física de um jogador, de acordo com seu deslocamento e estado. De forma semelhante, o serviço pode alertar o jogador a respeito de mini-mundos ou jogadores a serem desafiados em suas cercanias.

#### 6.5.4.1 Modelagem

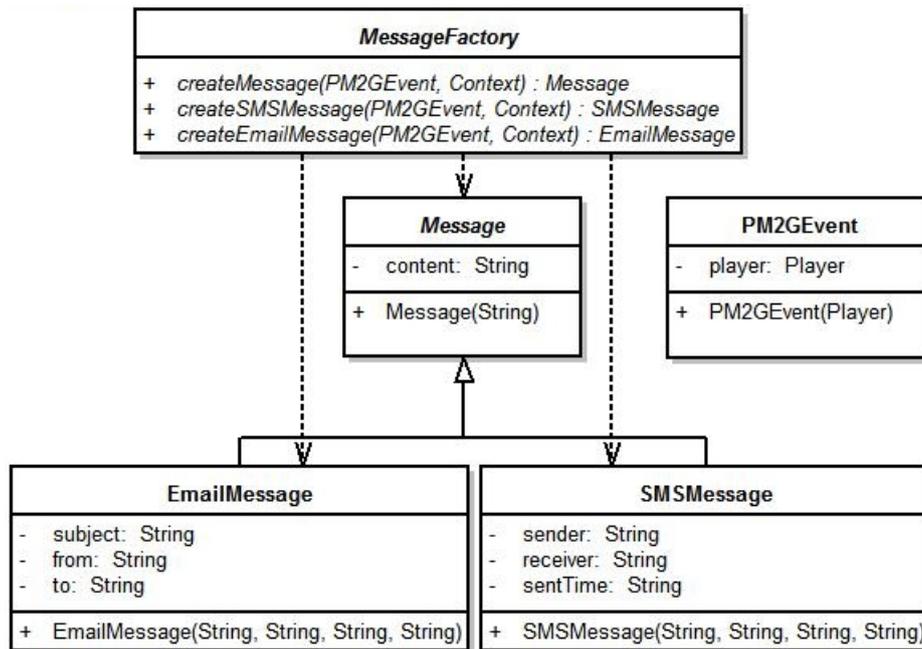
Em sua concepção, procurou-se criar a idéia de um serviço genérico, onde um evento pode ser disparado de qualquer outro componente da arquitetura que queira se utilizar do serviço para enviar mensagens para um jogador. Cada evento deve ser repassado ao serviço, que uma vez recebido, deve tomar as seguintes medidas:

1. identificar o jogador relacionado ao evento;
2. identificar o tipo do evento a ser informado ao jogador;
3. criar uma mensagem relativa ao evento, baseado no contexto atual do jogador, e;
4. enviar a mensagem ao jogador.

Nesta concepção, um evento, representado pela classe *PM2GEvent* da Figura 6.30, é sempre associado a um jogador. Esta classe representa qualquer evento, quer seja ele oriundo de eventos da simulação ou do mundo real do usuário. Porém, diferentes tipos de eventos podem inserir informações específicas. Por exemplo, se o desenvolvedor quer notificar um jogador que seu avatar está sendo atacado por outro jogador, é importante incluir a informação sobre este adversário. Sendo assim, cabe ao desenvolvedor, criar novos tipos de eventos como especializações de um evento convencional. Em outras palavras, um desenvolvedor deve estender a classe *PM2GEvent*, para criar novos eventos.

Uma vez que cada evento possui informações específicas, deve haver um mecanismo que trate cada evento individualmente. O tratamento de um evento pelo serviço refere-se à criação e envio de uma mensagem ao usuário, de acordo com o contexto deste jogador. Para viabilizar esta concepção, cada evento deve possuir um procedimento que forneça mensagens relativas ao evento, nas diferentes tecnologias de comunicação suportadas pelo jogo. De uma forma genérica, esta visão é representada pela classe abstrata *MessageFactory* da Figura 6.30, que a partir de um evento e do atual contexto do jogador, é capaz de criar mensagens em diferentes tecnologias de comunicação.

Toda mensagem (classe *Message*) possui um conteúdo (atributo *content*), que representa o texto a ser repassado ao jogador. Porém, diferentes tecnologias possuem atributos extras e



**Figura 6.30** Diagrama de Classes – Evento PM2G e Criadores de Mensagens.

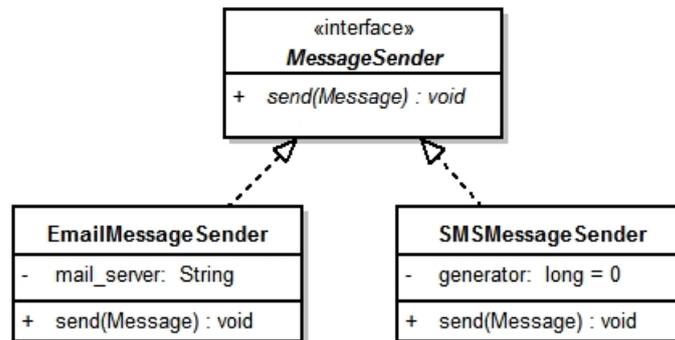
limitações específicas para a criação deste texto. Como exemplo, mensagens SMS possuem fortes limitações sobre o tamanho da mensagem a ser enviada. Além disso, cada evento utilizará dados particulares para criação da mensagem. Sendo assim, cada evento específico deverá especificar os passos necessários para a criação das mensagens em cada tecnologia. Com isto, cada classe que realize o tratamento de um evento, deve então estender a classe *MessageFactory*, fornecendo operações para a criação de mensagens nas diferentes tecnologias suportadas pelo jogo.

Uma vez criadas as mensagens, o serviço então envia a mensagem para o usuário, de acordo com o contexto do jogador. Diferentes alternativas de envio de mensagens possuem seus emissores, como email ou SMS. Estes recebem uma mensagem e a enviam ao jogador. Cada emissor segue a especificação da interface *MessageSender*, representada na Figura 6.31.

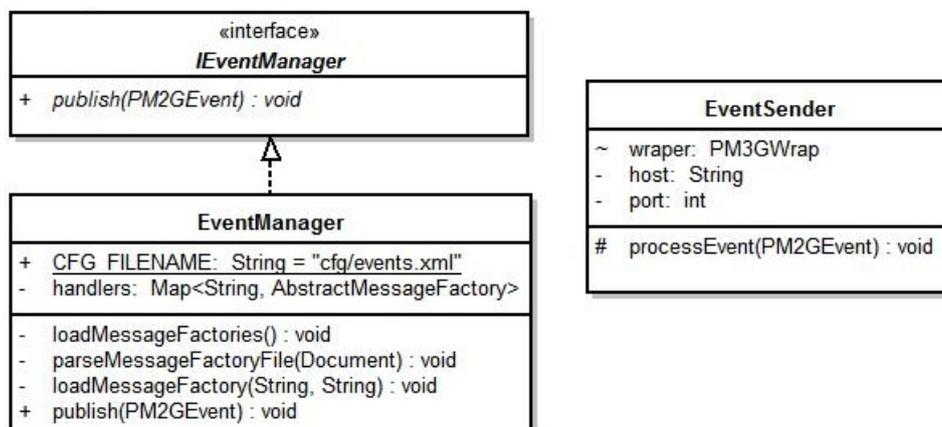
#### 6.5.4.2 Arquitetura Interna

A Figura 6.32 apresenta a arquitetura proposta para o Serviço de Notificação de Mensagens como um diagrama de classes em UML.

Componentes e elementos externos à arquitetura que queiram enviar eventos ao serviço, utilizam uma instância da classe *EventSender*. Esta classe possui um método *processEvent*, que recebe um evento, e o coloca em uma fila para ser tratado pelo serviço. A única operação



**Figura 6.31** Diagrama de Classes – Emissores de Mensagens.



**Figura 6.32** Diagrama de Classes – Arquitetura Interna do Serviço de Notificação de Mensagens

ofertada pelo serviço é representada pela interface *IEventManager*, através de seu método *publish*, que recebe um evento enviado por outro componente da arquitetura. O serviço possui um gerenciador (*EventManager*), que ao receber este evento, descobre qual é a classe de tratamento associada ao evento. Esta relação é determinada através de um arquivo de configuração, cuja estrutura é apresentada na Figura 6.33, onde estão definidos os mapeamentos entre eventos e seus respectivos procedimentos de tratamento.

A Figura 6.34 apresenta os passos seguidos pela execução do serviço. A simulação do jogo ou algum serviço interessado em enviar mensagens a um jogador, cria o evento e utiliza a classe *EventSender* para enviá-lo ao Gerenciador de Eventos (*EventManager*). Nesta classe é mantida uma estrutura que mapeia em memória as relações estabelecidas no arquivo de configuração descrito previamente. Um vez recebido um evento do jogo, o gerenciador recupera a classe que irá tratá-lo. Esta classe constrói a mensagem de acordo com o evento e o atual contexto do jogador. Uma vez criada a mensagem, esta é repassada ao Gerenciador de Mensagens (*MessageManager*). Este componente possui referências a todos os emissores de mensagens

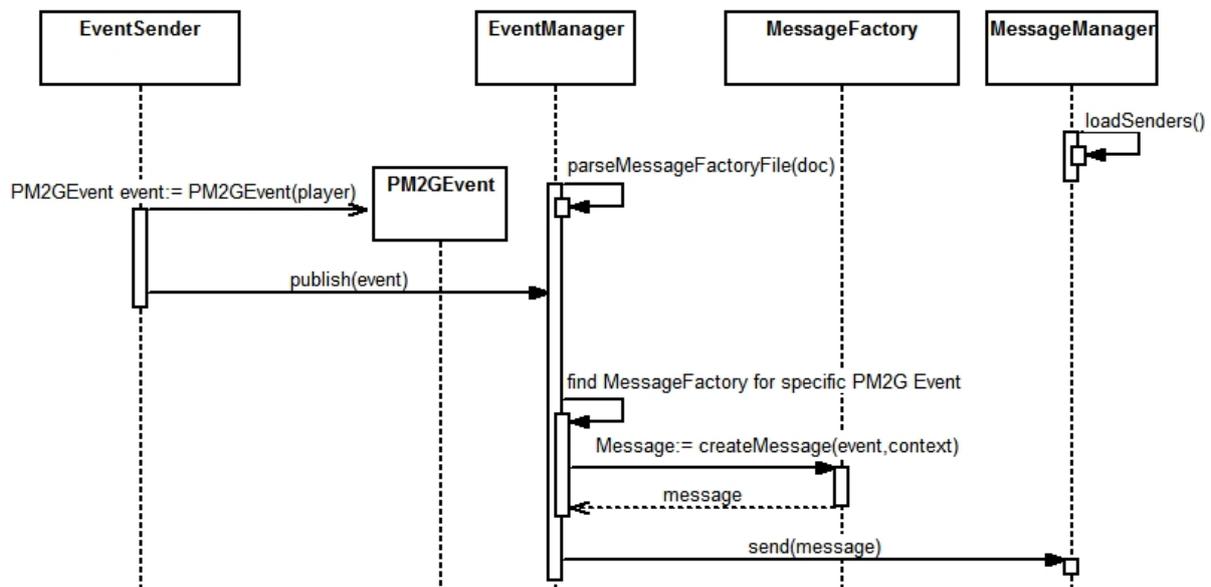
```

<?xml version="1.0" encoding="ISO-8859-1"?>
<event_manager>
  <event_conf>
    <event_class>Class1Name</event_class>
    <message_factory_class>Class1ObjectFactory</message_factory_class>
  </event_conf>
  <event_conf>
    <event_class>Class2Name</event_class>
    <message_factory_class>Class2ObjectFactory</message_factory_class>
  </event_conf>
  ...
  <event_conf>
    <event_class>ClassNName</event_class>
    <message_factory_class>ClassNObjectFactory</message_factory_class>
  </event_conf>
</event_manager>

```

**Figura 6.33** Configuração entre Eventos e *ObjectFactories*.

(*MessageSender*), que enviam mensagens de acordo com tecnologias específicas. A partir da mensagem recebida, o gerenciador utiliza então o emissor específico para enviar a mensagem ao jogador.



**Figura 6.34** Criação e envio de uma mensagem a partir de um evento.

### 6.5.5 Serviço de Integração de Mini-mundos

Por fim, o Serviço de Integração de Mini-Mundos diz respeito à integração do mundo virtual com cenários formados pelo conceito de *mini-mundo* do modelo de aplicação PM2G. Esta integração se dá pela transferência de informações entre o mundo virtual e as instâncias de mini-mundo existentes. Como ressaltado no modelo de aplicação, cada mini-mundo reflete um pequeno cenário de interação contextualizado com o mundo virtual do jogo PM2G. Desta forma, é função do Serviço de Integração de Mini-mundos fornecer operações que permitam a migração do perfil de um jogador a partir dos dados de seu personagem para o contexto local do mini-mundo.

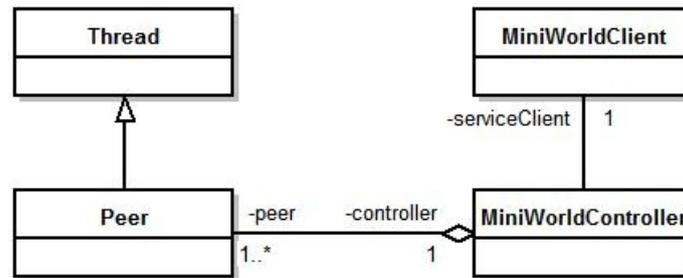
O perfil de um jogador refere-se a atributos e objetos que caracterizam o personagem de um usuário. O perfil de um jogador é modificado de acordo com sua participação no mini-mundo. Ao final de cada mini-mundo, os resultados de cada jogador são enviados ao serviço, onde são verificados e posteriormente integrados ao mundo virtual. Como exemplo, uma arma pode ser transferida do jogo PM2G para um mini-mundo, onde o jogador a utiliza em uma simulação de um duelo contra um adversário. Caso o jogador perca o duelo, sua arma pode ser incorporada pelo jogador vencedor, que passará a utilizá-la quando estiver utilizando outros dispositivos.

A integração de resultados apresenta-se como um potencial problema para questões relativas a trapaça, pois neste momento um jogador mal-intencionado pode injetar transformações ilegais ao estado do jogo, afetando potencialmente o mundo virtual e causando prejuízo aos demais jogadores. Embora o combate à trapaça não seja um problema diretamente atacado por este trabalho, medidas paliativas podem ser tomadas para atenuar o problema da integração, visto que este é uma questão inserida por nossa proposta.

Primeiramente, neste trabalho, mini-mundos seguem o modelo de “Jogos Soma Zero”. Estes jogos descrevem uma situação onde os ganhos e perdas de um jogador são equilibradas pelas perdas e ganhos dos outros jogadores. Este termo deriva da idéia que o total de ganhos de todos participantes diminuído do total de todas as perdas de todos participantes será igual a zero. Em outras palavras, não é possível que só haja vencedores, ou apenas perdedores. Deste modo, jogadores de mini-mundos não têm vantagens em criar jogos entre si, uma vez que algum dos participantes sairia prejudicado. Outra contra-medida para proteção à integração dos mini-mundos é obrigar a submissão automática dos resultados por mais de um jogador, para que os dados possam ser validados via replicação.

## 6.5.5.1 Modelagem Estática de um Mini-Mundo

Como explicado anteriormente, o conceito de mini-mundo representa jogos paralelos ao mundo virtual simulado pelo serviço PM2G. Estes jogos são concebidos como jogos multiusuário móveis tradicionais, mas que têm a necessidade de interação com o serviço para troca de informações e resultados com o jogo simulado no servidor PM2G.



**Figura 6.35** Diagrama de classes – Arquitetura de um Mini-Mundo PM2G.

Neste trabalho, um mini-mundo utiliza uma arquitetura cliente/servidor, onde um jogador assume o papel de servidor do jogo, enquanto outros se conectam à sua aplicação, assumindo o papel de clientes da aplicação. Para diferenciar o servidor de um mini-mundo do servidor PM2G, a partir deste ponto o servidor de um Mini-Mundo será denominado como Mestre (*Master*), enquanto jogadores conectados ao Mini-Mundo serão chamados de escravos (*slaves*).

A diferença entre um mini-mundo e um jogo multiusuário móvel convencional é que são utilizadas informações do personagem (avatar) do usuário no jogo PM2G. Estas informações são modificadas pela dinâmica do jogo entre os participantes do mini-mundo. Ao final, os resultados destas partidas são reenviadas ao serviço, que valida as informações, efetivando-as no servidor global do jogo PM2G. A Figura 6.35 apresenta uma visão simplificada da arquitetura de um Mini-Mundo. A classe *MiniWorldController* representa o controlador do jogo, independente se este assume o papel de Mestre ou escravo em um Mini-Mundo. A classe *Peer* trata de questões relativas à comunicação entre os jogadores, enquanto a classe *MiniWorldClient* lida com a interação do jogador com o serviço de integração de mini-mundos.

## 6.5.5.2 Modelagem Dinâmica – Interação Mini-Mundo e Serviço de Integração

A modelagem dinâmica da arquitetura dos mini-mundos será apresentada a partir da máquina de estados de seu controlador (*MiniWorldController*). Esta, por sua vez, será explicada na ordem temporal em que acontecem os eventos de um mini-mundo. A Figura 6.36 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 visualização mais detalhada de cada uma destas áreas ao longo desta seção.

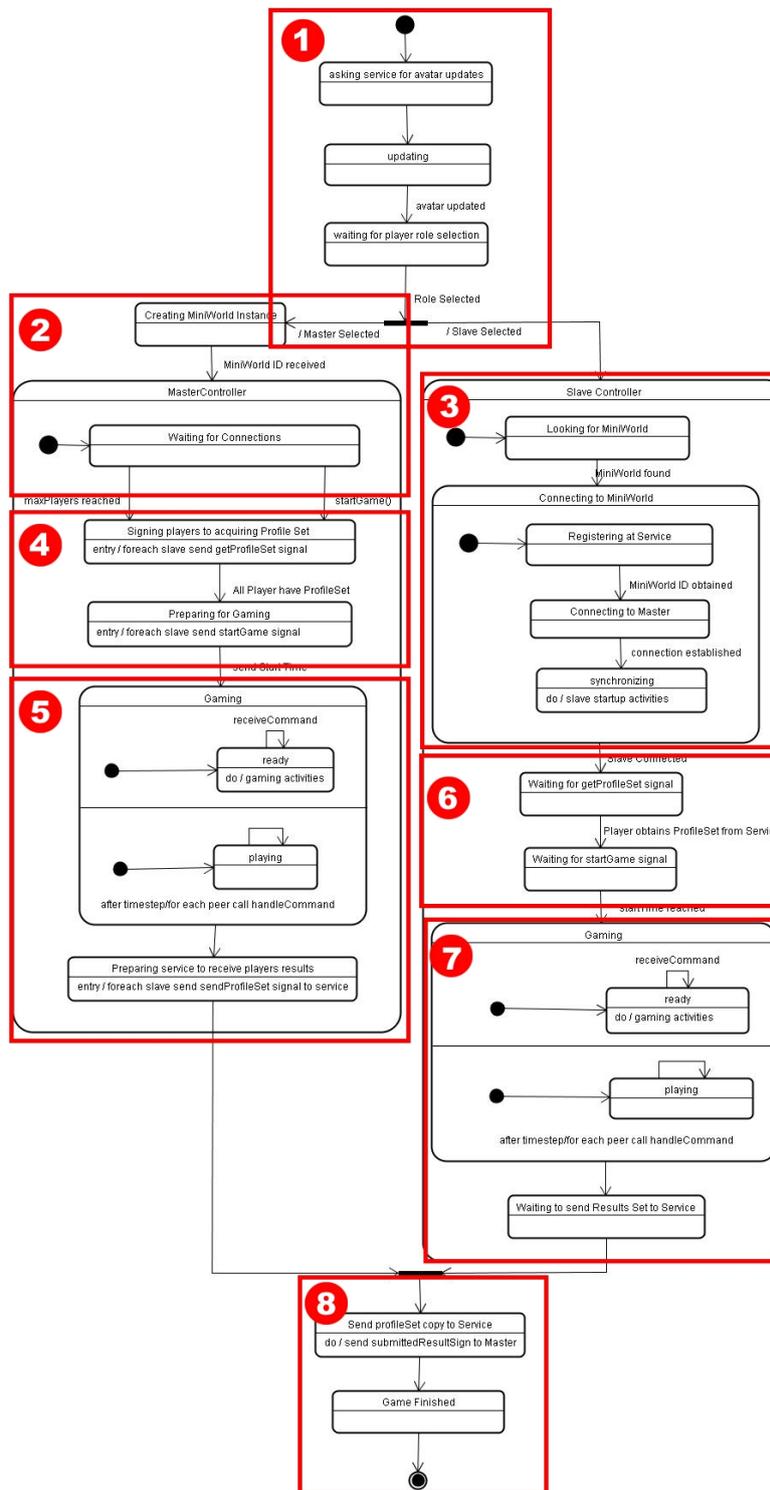
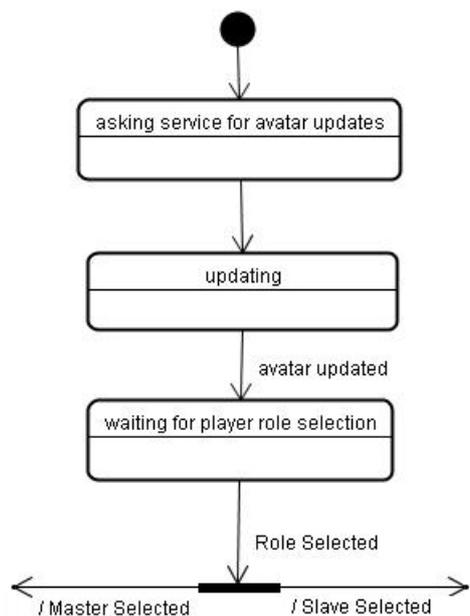


Figura 6.36 Máquina de estados do controlador de um Mini-Mundo.

## 6.5.5.3 Atualização de Avatares

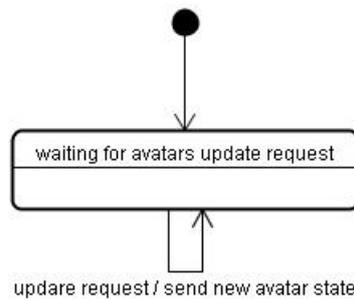
O ponto inicial da participação de um jogador em um mini-mundo é a sincronização das informações de seu personagem (avatar) com seu estado no jogo PM2G. Ressalta-se que é esperado que um jogador para participar de um mini-mundo possua uma conta válida, e possa se autenticar no jogo, de modo a obter seu identificador (*id*) de acesso. A Figura 6.37 apresenta os estados que modelam o comportamento de recuperação de informações do avatar de um jogador.



**Figura 6.37** Máquina de Estados do Controlador de Mini-Mundo – atualização de avatares

No estado *asking service for avatar updates* é realizada uma requisição ao serviço de integração de mini-mundos. O Serviço recupera as informações do perfil e as repassa ao jogador, que então atualiza o avatar no contexto do Mini-Mundo – estado *updating*. Depois de atualizar o estado do avatar, o controlador se encontra no estado *waiting for player role selection*, que representa a escolha entre conectar-se a um Mini-mundo previamente criado ou criar tal Mini-mundo. Maiores detalhes sobre os papéis de um jogador no mini-mundo serão apresentados adiante.

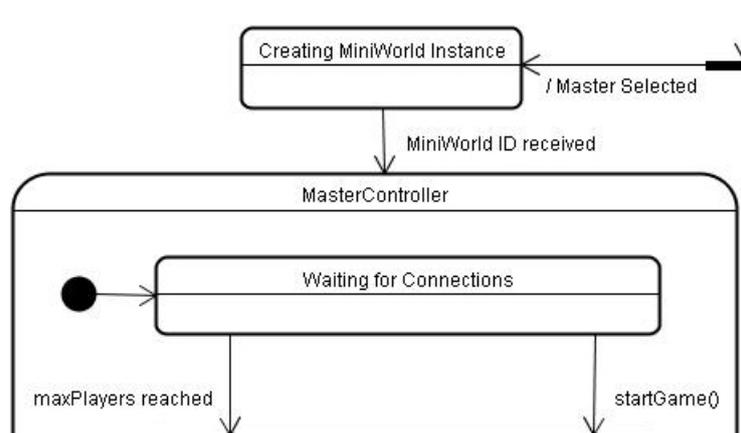
A atualização de avatares de clientes é apenas uma das requisições que o serviço de integração de Mini-Mundos deve disponibilizar. O estado para atualização de avatares é representado pelos estados do serviço apresentados pela Figura 6.38, uma das linhas de concorrência possíveis para o Serviço de Integração de Mini-Mundos.



**Figura 6.38** Máquina de Estados do Serviço de Integração de Mini-Mundos – atualização de avatares.

#### 6.5.5.4 Criação do Mini-Mundo e Tratamento de Conexões

A criação de um mini-mundo acontece a partir da escolha do jogador em assumir o papel de controlador mestre da rede. Neste caso, o controlador vai para o estado *creating Mini-World instance* – Figura 6.39. Neste estado, o jogo é criado no dispositivo no jogador, onde a dinâmica de troca de mensagens será realizada. Este é o caso onde o controlador mestre assumirá o papel de servidor do Mini-Mundo. Porém, também se faz necessário criar uma instância de uma referência ao Mini-Mundo no serviço de Integração de Mini-Mundos. Esta referência permitirá a integração dos resultados das instâncias ativas de Mini-Mundos com o jogo PM2G, assim como que outros jogadores possam ter conhecimento de Mini-Mundos em andamento.



**Figura 6.39** Máquina de Estados do Controlador de Mini-Mundo – Processamento de pedidos de criação, registro e atualização de estado.

O jogador que cria o Mini-Mundo assume o papel de “dono” do Mini-Mundo (*owner*). Neste papel, o controlador será responsável pelo controle do estado de um Mini-Mundo. Este

estado mudará de acordo com os vários estágios do ciclo de vida de um Mini-Mundo. Quando o mini-mundo é criado, seu estado no serviço é automaticamente estabelecido como *aberto*. Neste estado, passa-se a idéia que o jogo está em fase de espera pela conexão de outros jogadores. São repassados ainda ao serviço informações sobre o tipo do Mini-Mundo a ser criado. Estas informações incluem a quantidade máxima de jogadores suportados, o tipo de mini-mundo e a política de troca de mensagens (políticas serão explicadas no decorrer desta seção).

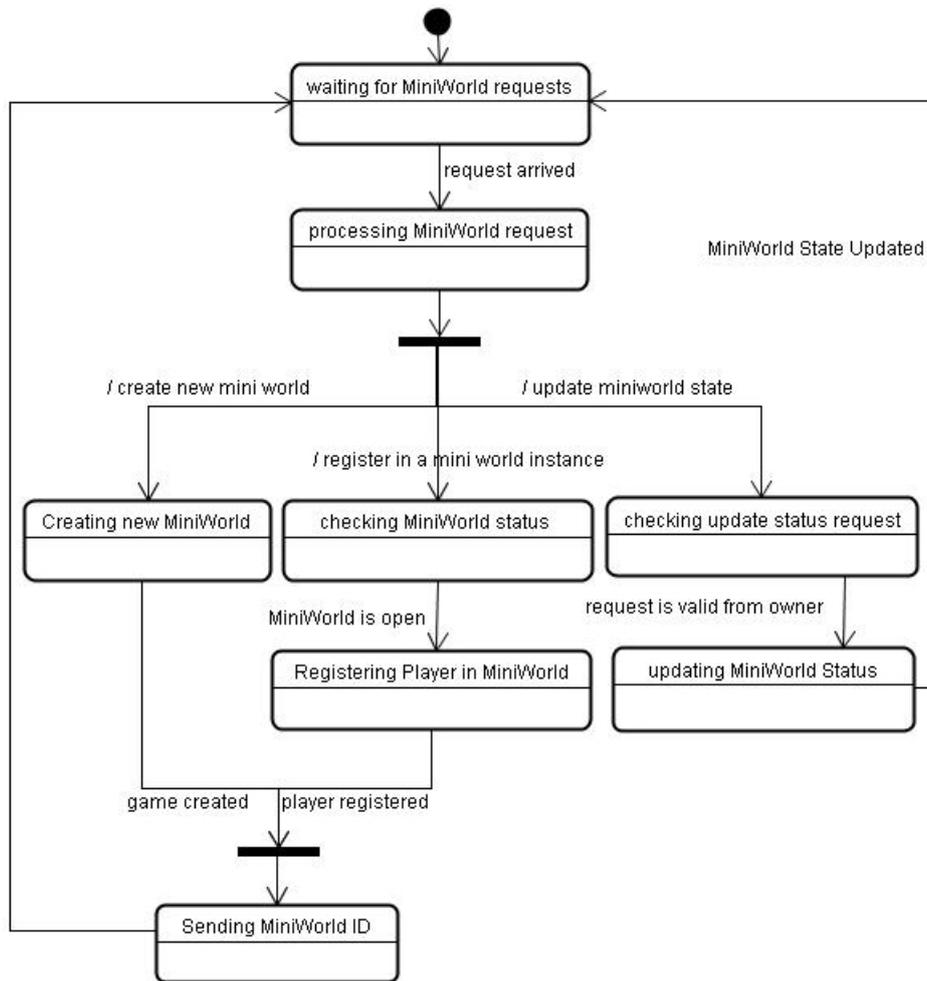
No lado do serviço, o pedido da criação de um Mini-Mundo é atendido pelo estado *processing Mini-World request*. O serviço cria a referência do Mini-Mundo estado (*creating new Mini-World*) e envia ao Mestre o identificador do Mini-Mundo (*Mini-World ID*) – estado *Sending Mini-World ID*. Com este identificador estabelecido, o controlador mestre passa a esperar conexões no estado *waiting for connections*, utilizando o identificador do Mini-Mundo para validar novas conexões. Toda vez que uma nova conexão é estabelecida, um objeto da classe *Peer* é criado e associado ao controlador mestre.

Um objeto *Peer* recebe as responsabilidades de sincronização e troca de mensagens pela rede com o dispositivo recém-conectado: o dispositivo escravo. A associação entre um *Peer* e um escravo é de um pra um. O mestre, em paralelo, continua esperando novas conexões. O jogo inicia por comando explícito do usuário que o criou ou quando o limite de conexões determinado pelo número de jogadores é atingido.

#### 6.5.5.5 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 o papel escolhido seja de escravo, o jogador busca no serviço qual o jogo que deseja participar, a partir da lista fornecida pelo serviço, representado pelo estado *looking for Mini-World*, apresentado na Figura 6.41.

Escolhido o jogo, segue-se o processo de conexão ao Mini-Mundo. Primeiramente, o controlador escravo passa ao estado *registering at service*, onde o mesmo busca registrar-se no serviço. O serviço verifica se o estado do Mini-Mundo ainda encontra-se aberto (*checking Mini-World status*). Em caso positivo, o serviço registra o jogador (*registering player in Mini-World*) e envia o id do mini-mundo ao controlador deste jogador (*sending Mini-World id*). Este identificador é utilizado pelo jogador para validar sua conexão com o mestre do jogo. A conexão controlador escravo e mestre é realizada de acordo com procedimentos particulares à tecnologia escolhida (bluetooth, por exemplo) – estado *connecting to Master*. A Figura 6.41 mostra também o estado *synchronizing*, no qual as atividades de inicialização necessárias aos *escravos* participantes da rede são realizadas. Essa inicialização acontece em paralelo, através



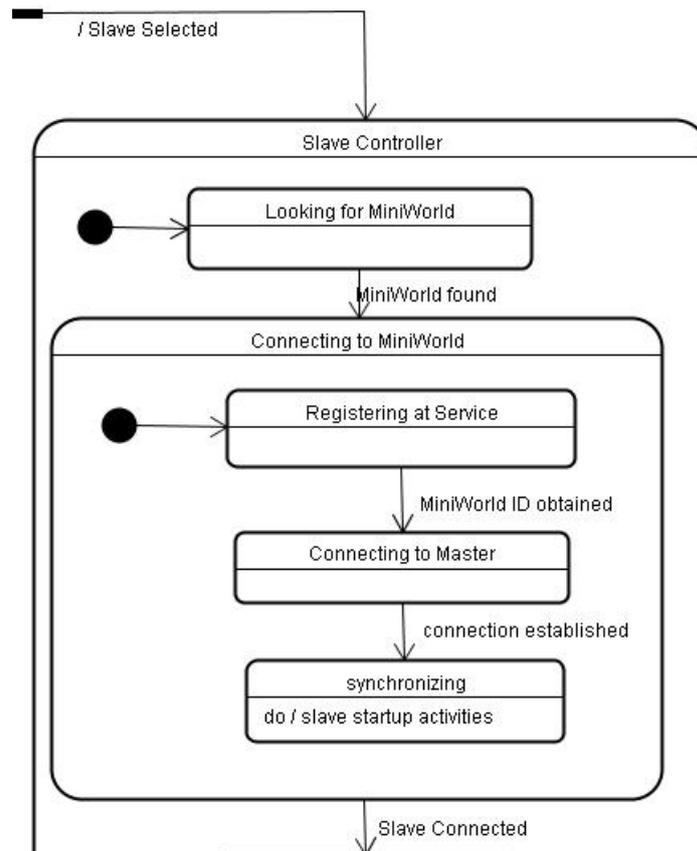
**Figura 6.40** Máquina de Estados do Serviço de Integração de Mini-Mundos – Processamento de pedidos de criação, registro e atualização de estado.

da troca de mensagens, com as atividades de inicialização do objeto *Peer* criado no mestre para esta conexão específica.

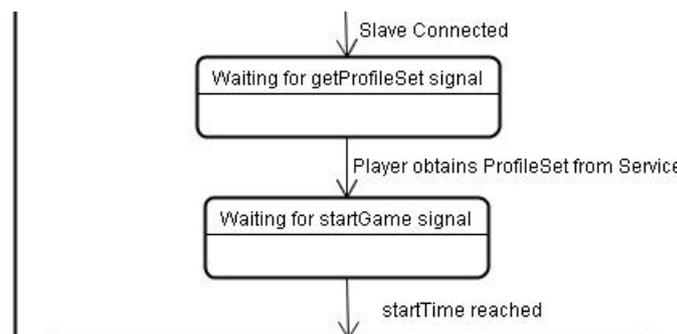
#### 6.5.5.6 Início e Decorrer do Jogo

Os escravos, depois de sincronizados, ficam em um estado de espera pelos procedimentos de início do jogo. Os procedimentos são apresentados através dos estados *Waiting for ProfileSet-Signal* e *Waiting for StartGame Signal* – Figura 6.42. Estes estados são estados de espera por sinalizações do mestre, explanados a seguir.

Como ressaltado anteriormente, o mestre pode, a qualquer momento, iniciar o jogo através de um comando explícito ou quando o limite de conexões é atingido. Em ambos os casos, o



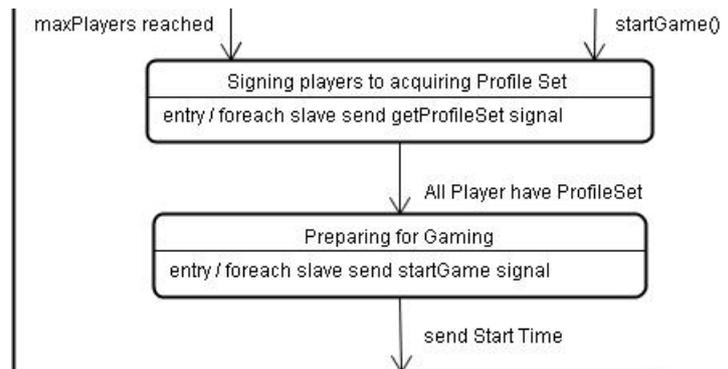
**Figura 6.41** Máquina de Estados do Controlador de Mini-Mundo – Localização, conexão e inicialização de escravos.



**Figura 6.42** Máquina de Estados do Controlador de Mini-Mundo – Espera pelo início do jogos por parte dos controladores escravos.

mestre envia um sinal para os escravos, indicando que cada jogador deve buscar no serviço o conjunto com as informações dos avatares de todos os jogadores participantes do Mini-Mundo (estado *Signing players to acquiring Profile Set*), apresentado na Figura 6.43.

O mestre é notificado assim que cada escravo recebe este conjunto. Após todos os jogadores terem obtido o conjunto de perfis, o mestre vai para o estado de *preparing for gaming*, onde envia a todos os escravos o sinal de início (*start*). Neste sinal, é repassada também a hora lógica no qual o jogo deve começar. O controlador-mestre também modifica o estado do Mini-Mundo para *em execução*, indicando que novos jogadores não podem mais entrar no mesmo.



**Figura 6.43** Máquina de Estados do Controlador de Mini-Mundo – Início do jogos pelo controlador mestre.

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

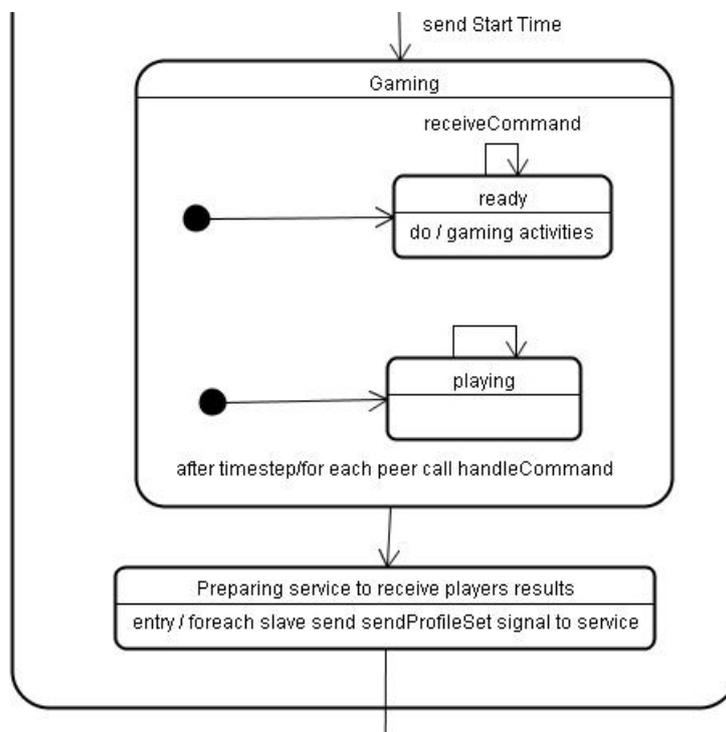
Iniciado o jogo, todos os participantes podem começar a enviar comandos, seguindo as restrições da política do jogo, escolhida previamente durante a criação do Mini-Mundo. Existem 3 políticas de jogo:

- 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 calculada, independentemente, 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;

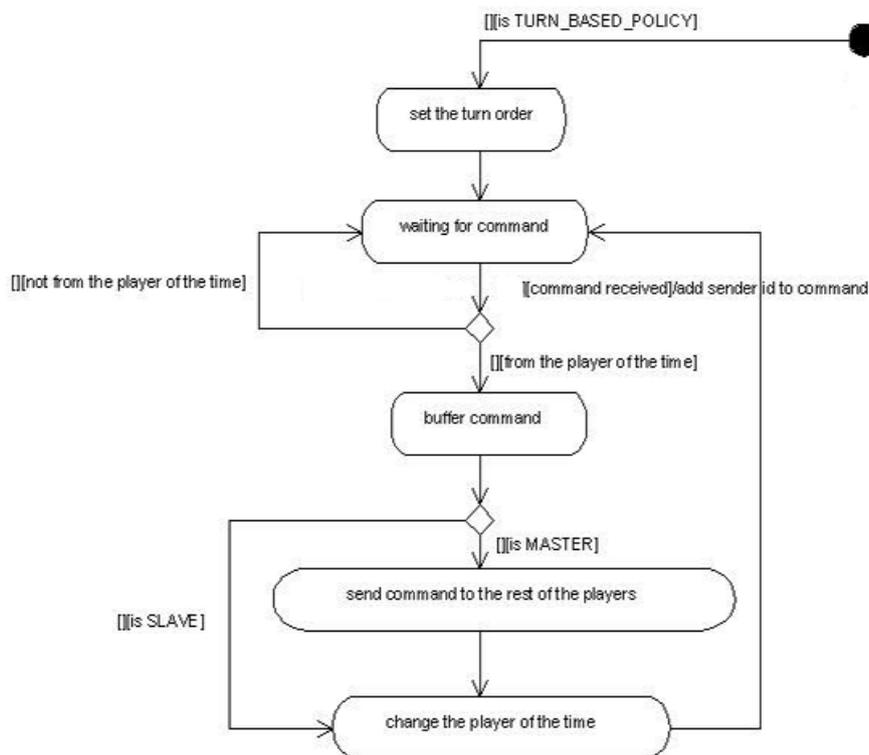
- Tempo-Real (*Real-time games*): Os comandos podem ser enviados a qualquer momento e serão executados na ordem que são recebidos pelo controlador.

Os estados, responsáveis pelo decorrer do jogo, do mestre e do escravo são respectivamente mostrados na Figura 6.44 e na Figura 6.48. 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 em um dado tempo estabelecido pelo jogo, 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 às regras de validação dos comandos, pelo estado *ready*, que deverão, também, ir 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 dispositivo local e comandos recebidos pela rede; e o envio de mensagens não gera evento, por isso não está especificado explicitamente.



**Figura 6.44** Máquina de Estados do Controlador de Mini-Mundo – Decorrer do jogo/Sinalização de término (Controlador Mestre).

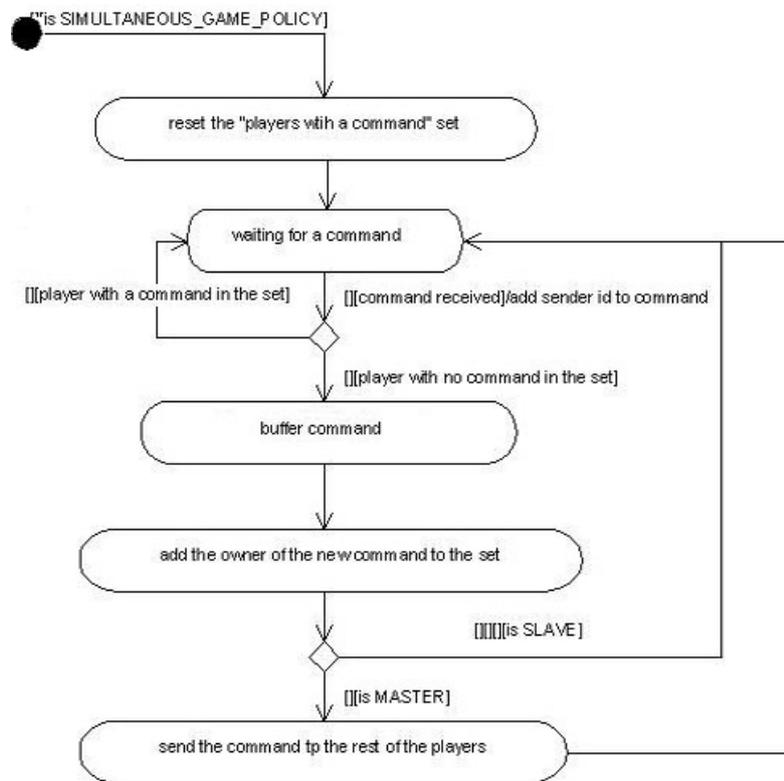
A Figura 6.45 mostra o fluxo de atividades de um jogo de turnos. Inicialmente, a seqüência em que os participantes vão jogar é definida. A partir desse ponto, o controlador fica esperando os comandos. Se um comando, do jogador da vez, chegar, ele é salvo e, caso o controlador tiver papel de mestre da rede, ele faz um *broadcast* do comando para finalmente trocar o jogador da vez. Se o controlador tiver papel de escravo, 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.



**Figura 6.45** Fluxo de atividades de jogos de turno.

A Figura 6.46 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. Inicialmente 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 seja o mestre, ele envia o comando a todos os outros participantes da rede e volta a esperar por comandos; caso seja escravo ele simplesmente volta a esperar por comandos. Quando todos os participantes enviarem seus comandos, eles serão executados e o fluxo reinicia.

A Figura 6.47 mostra o fluxo de atividades de um jogo de tempo-real. Esse fluxo é o mais



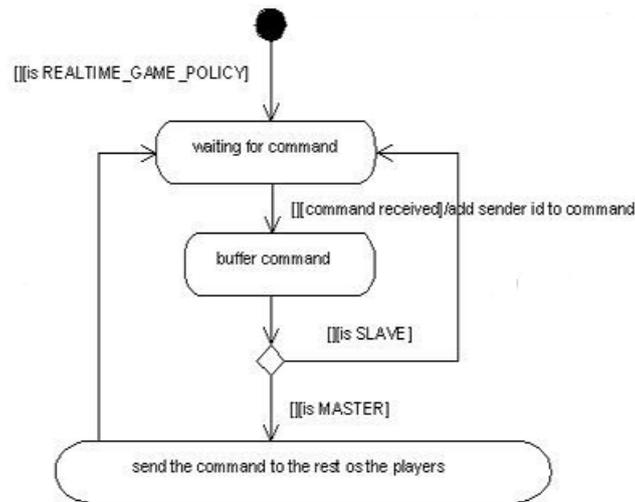
**Figura 6.46** Fluxos de atividades de jogos simultâneos.

simples. Qualquer comando recebido é salvo. Caso o controlador seja mestre, ele, além de salvar, envia um *broadcast* com o comando a todos os integrantes do Mini-mundo.

#### 6.5.5.7 Finalização de Resultados

Por fim, ao término da partida, os resultados do Mini-Mundo devem submetidos ao serviço. Essa submissão deve ser feita por os todos participantes do Mini-mundo para evitar trapaça. O término do jogo é controlado pelo Mestre, que modifica o estado do Mini-Mundo no serviço para que o mesmo possa receber os resultados dos jogadores – estado *em encerramento*. Uma vez modificado o estado no serviço, o mestre sinaliza para que os escravos enviem ao serviço, seu conjunto dos perfis, através do estado *Preparing Service to receive players results*, da Figura 6.44 e também envia seus resultados. Este conjunto reflete os estados dos avatares de todos os jogadores, resultante das interações no Mini-Mundo. A idéia é que todos os jogadores tenham uma cópia local do estado de todos os personagens, e que todos devem concordar sobre o estado final destes dados, quando do término do jogo.

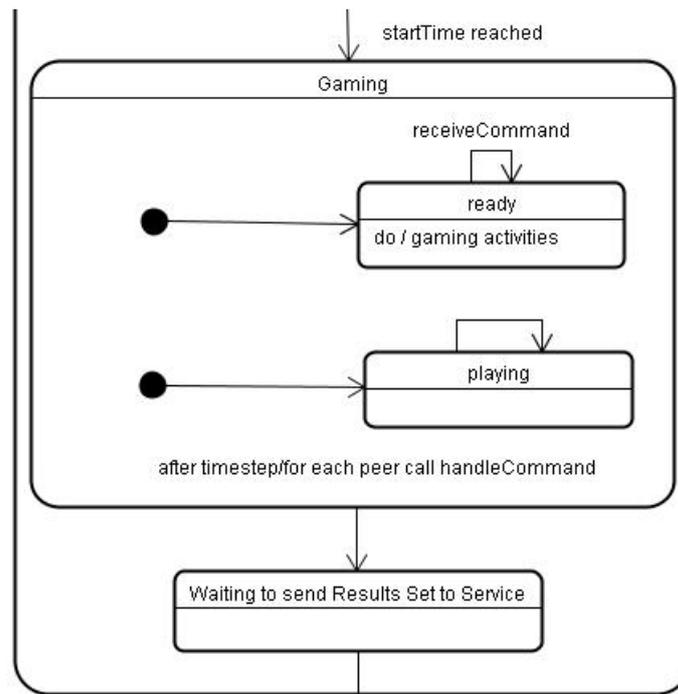
Os controladores escravos, após o término do jogo, ficam no estado *waiting to send results*



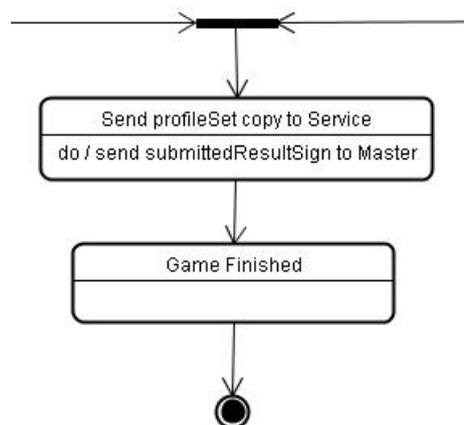
**Figura 6.47** Fluxos de atividades de jogos de tempo-real.

*to service* até receberem uma notificação do mestre para envio de seus resultados ao serviço, representado na Figura 6.48. Após receberem esta sinalização, cada controlador envia seus resultados, encerrando sua participação no mini-mundo, representados pelos estados *Send profileSet copy to Service*, na Figura 6.49.

Após o estado de um Mini-Mundo ser modificado para *em encerramento*, o serviço fica a espera da submissão dos resultados dos jogadores, representado pelo estado *waiting for mineworld results* (Figura 6.50). Para cada resultado submetido, o serviço verifica o conjunto de informações recebido com aqueles previamente enviados por outros jogadores. Após receber os resultados de todos os jogadores, inicia o processamento de validação destes dados. Este procedimento verifica se os dados de todos os jogadores são idênticos. Em caso positivo, o serviço aprova e atualiza o estado de todos os personagens de cada jogador no contexto do jogo PM2G. Caso os resultados não sejam consistentes, eles são descartados. Após esta validação, o fluxo de um Mini-mundo para o serviço é encerrado.



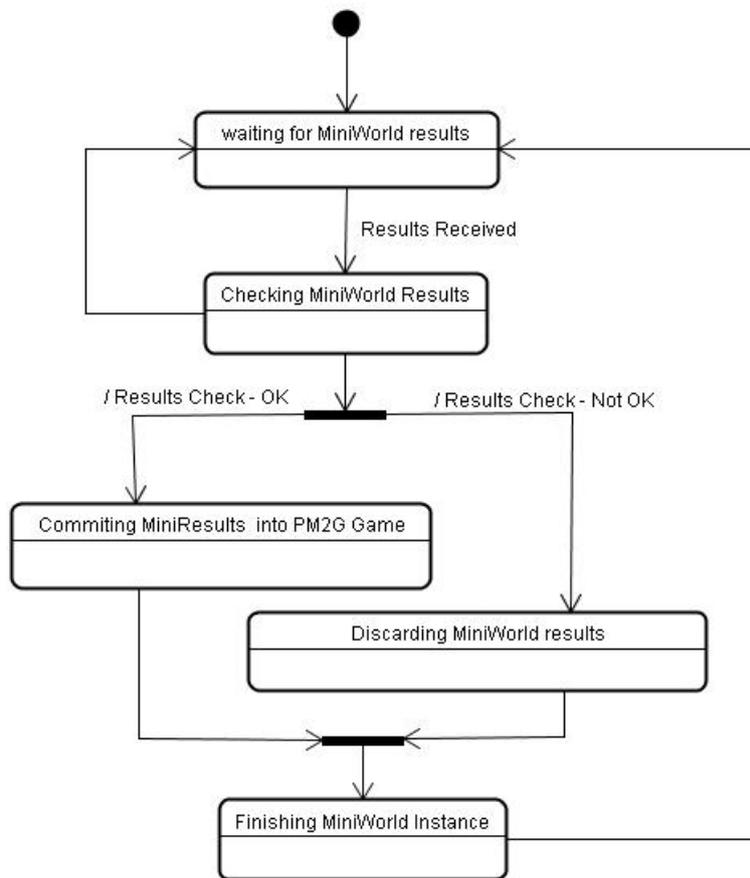
**Figura 6.48** Máquina de Estados do Controlador de Mini-Mundo – Decorrer do jogo/Sinalização de término (Controlador Escravo).



**Figura 6.49** Máquina de Estados do Controlador de Mini-Mundo – Submissão de Resultados.

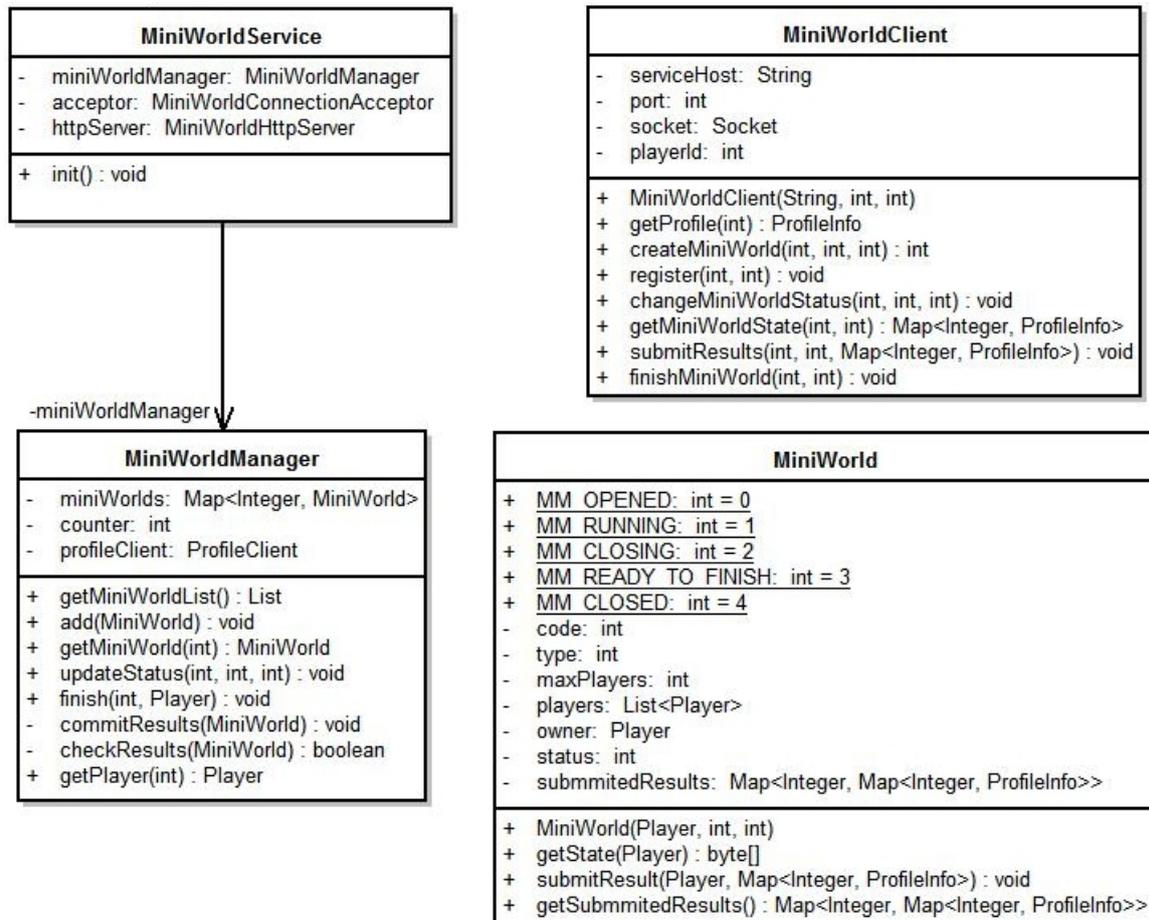
#### 6.5.5.8 Modelagem Estática do Serviço

A Figura 6.51 apresenta a arquitetura proposta para o Serviço de Integração de Mini-Mundos como um diagrama de classes, utilizando a notação UML. O Serviço utiliza o padrão de projeto *Acceptor* [Sch97] para tratar da comunicação entre clientes e o serviço. De forma semelhante



**Figura 6.50** Submissão de resultados pelo controlador de um Mini-Mundo.

ao Serviço de Gerenciamento de Contexto, são utilizadas duas classes, *MiniWorldConnectionAcceptor* e *MiniWorldHttpServer*, onde a segunda utiliza o protocolo HTTP para troca de mensagens entre dispositivos móveis que utilizam MIDP. O controle sobre as instâncias de Mini-Mundos é delegada ao Gerenciador do Mini-Mundo, representado pela classe *MiniWorldManager*, que apresenta as funcionalidades descritas para o controle do ciclo de vida de um Mini-Mundo. Cada referência a um mini-mundo utiliza a classe *MiniWorld* para guardar informações sobre jogadores, estado, resultados submetidos, dentre outras informações. A classe *MiniWorldClient* é utilizada por clientes para interação com o serviço.



**Figura 6.51** Diagrama de Classes – Arquitetura Interna do Serviço de Integração de Mini-Mundos

## 6.6 Considerações Finais

A integração de jogos multiusuários, mobilidade e heterogeneidade de dispositivos apresenta-se como uma tendência para jogos digitais futuros [Wal04]. Embora trabalhos apontem as vantagens desta integração [Fox03] [HIW04] [KW05] [Lin05], estes não apresentam um modelo adequado ao suporte de aplicações que busquem alcançar tal integração.

A arquitetura PM2G[TFR06b] busca preencher esta lacuna, através do conjunto de serviços especificados neste capítulo. Estes procuram guiar e facilitar a implementação destes jogos, seguindo a visão (cenários e modelos) descritos no Capítulo 5. O próximo capítulo apresenta experimentos realizados com a implementação parcial destes serviços, através de cenários de um jogo multiusuário.

# Experimentos

*Se os fatos não se encaixam na teoria, modifique os fatos*

— ALBERT EINSTEIN

A avaliação dos serviços propostos no capítulo anterior foi realizada via prototipação dos mesmos. Para avaliá-los, um projeto simples de um RPG digital foi elaborado, baseado em um jogo já existente. Este capítulo apresenta os passos realizados para a implementação deste jogo, incluindo a criação de seus elementos, bem como a configuração dos serviços PM2G, e os resultados obtidos a partir da execução deste protótipo.

## 7.1 Introdução

Ao longo da implementação dos serviços apresentados na arquitetura PM2G, foram definidas situações de jogos RPG que os validassem. Estes foram implementados considerando três diferentes plataformas: (i) computadores pessoais, (ii) dispositivos móveis com interface gráfica, como PDAs e telefones celulares mais avançados; e (iii) dispositivos móveis com interface textual, como telefones celulares mais simples. Este protótipo busca garantir a correção e eficiência das operações realizadas por cada serviço em diferentes contextos de execução.

Os eventos foram criados considerando diferentes níveis de imersão e abstração para cada dispositivo utilizado, baseados nos conceitos de percepção e interação do modelo de aplicação PM2G (Seção 5.3). Deste modo, a implementação para computadores pessoais exige uma participação mais intensa do jogador e permite que este tenha um maior controle do seu avatar, enquanto as implementações para dispositivos móveis visam uma interação de mais alto nível.

Estas situações típicas de um RPG digital incluem interações do avatar do jogador com outros jogadores e com itens no mundo do jogo. São elas:

- **Obter estado do jogo:** informa ao jogador o conteúdo de sua área de interesse, de diferentes maneiras, de acordo com o seu contexto;
- **Movimentação do avatar:** permite ao jogador mover seu avatar livremente pelo mundo virtual, dentro de sua área de interesse e também entre diferentes áreas de interesse;

- **Atacar adversário:** exemplo de interação entre dois jogadores, na qual seus dois avatares lutam entre si;
- **Coletar item virtual:** o avatar coleta um item virtual e o torna indisponível aos demais, modificando, assim, o conteúdo de sua área de interesse.

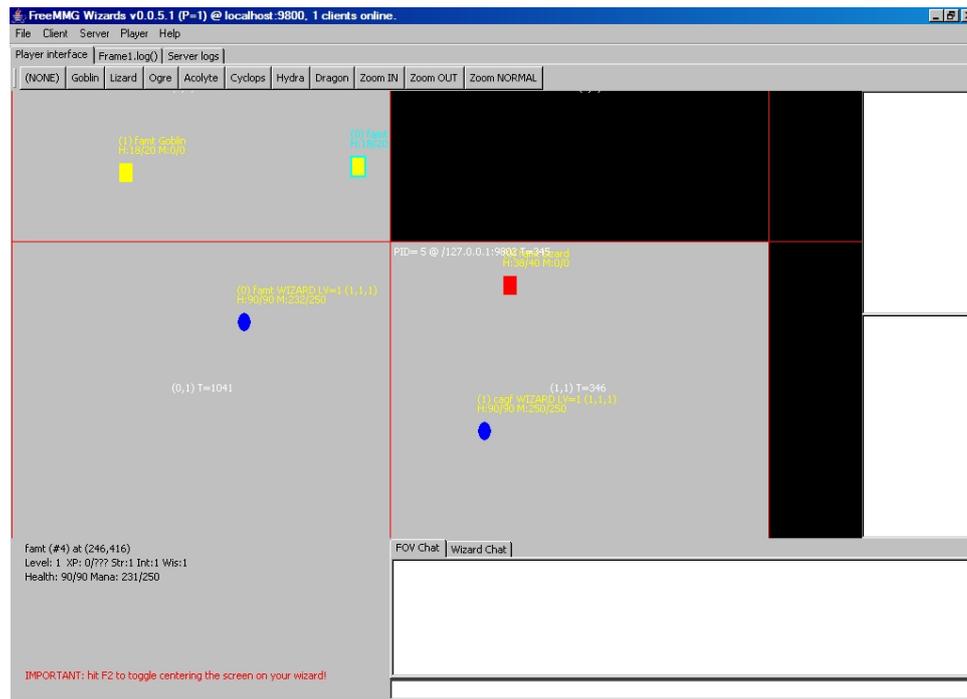
A partir destas situações foi elaborado um projeto simplificado de um jogo que contemplasse suas ocorrências para um jogador. As seções seguintes a esta introdução detalham os procedimentos realizados para a implementação deste protótipo. Primeiro, é apresentado o projeto de jogo utilizado. Em seguida, são apresentados os detalhes de criação de elementos do jogo e configuração dos serviços de *middleware* PM2G. Por fim, são apresentados dados sobre o reaproveitamento de código deste *middleware*.

## 7.2 Projeto do Jogo *Pervasive Wizards*

O jogo utilizado para validação dos serviços é uma versão de um jogo já existente, chamado *FreeWizards* [Cec05]. No jogo original, cada jogador controla um mago, que habita um mundo representado por um “tabuleiro” bidimensional, sem ornamentações, como diferentes tipos de terreno, obstáculos ou objetos. Magos controlam diferentes tipos de monstros, que são utilizados para atacar outros magos. A Figura 7.1 apresenta a *interface* do cliente de um jogador.

Os magos são representados por elipses, e os monstros por retângulos. Cada jogador possui um único mago. O jogador pode mover o seu mago livremente pelo tabuleiro, e também pode utilizar o mago para criar “monstros”. Tanto magos quanto monstros possuem dois atributos relevantes: pontos de vida e força de ataque. Quando dois objetos (magos ou monstros) colidem e são de posse de jogadores diferentes, ocorre um combate. O combate entre dois objetos A e B consiste em subtrair a força de ataque de A dos pontos de vida de B, e vice-versa. Após um combate, é possível que um dos objetos fique com uma quantidade de pontos de vida igual ou menor que zero. Neste caso, o objeto é removido do jogo, e o mago que controla o objeto “vitorioso” recebe uma bonificação que o torna mais forte (capacidade de criar monstros com maior frequência ou capacidade de criar monstros mais fortes). Em boa parte dos jogos de estado persistente, quanto mais o jogador participa do jogo, mais “forte” se torna o seu personagem. Caso o mago de um jogador seja destruído, o jogador pode revivê-lo, recolocando-o na mesma região original ou em outra região do mundo.

A versão utilizada para validar os conceitos propostos pelos serviços da arquitetura PM2G modifica o projeto original do jogo, de modo a adequá-lo aos jogos de representação de papéis (RPGs) e aos cenários definidos na Seção 7.1. Na versão modificada, magos não possuem mais



**Figura 7.1** Exemplo do estado da *interface* de um jogador, durante uma sessão do jogo FreeWizards [Cec05]

monstros, sendo o único avatar controlado pelo jogador. O mundo continua dividido em várias áreas retangulares que representam diferentes regiões do mundo virtual.

Além dos magos, cada região, opcionalmente, contém objetos que podem ser coletados pelos magos. Estes objetos são categorizados como armas, porções ou objetos de proteção. Ao coletar estes objetos, um mago se torna mais forte, aumentando seu poder de ataque ou sua capacidade de defesa. Os atributos de um mago foram modificados. Cada mago possui três atributos relevantes: pontos de vida, energia mística (comumente chamada em jogos de *mana*) e velocidade de deslocamento. Magos podem carregar consigo um item de cada categoria de objetos do jogo, ou seja, uma arma, uma porção e um elemento de proteção. A coleta de um objeto acontece quando um mago colide com o objeto. Caso o mago já possua um objeto da mesma categoria do objeto colidido, o objeto não é coletado.

Combates continuam acontecendo por meio de colisões entre magos, sendo que a posse de objetos modifica a quantidade de pontos de energia vital perdida ou retirada nos combates entre magos. Para efeito de ornamentação, as regiões do mundo podem conter também cachorros selvagens, que assumem o papel de um personagem automatizado (NPC), seguindo o modelo de aplicação PM2G. A Tabela 7.1 associa os demais elementos criados no protótipo com os termos definidos pelo modelo de aplicação PM2G.

**Tabela 7.1** Relação entre elementos do Jogo *Pervasive Wizards* e o modelo de aplicação PM2G

Elementos do jogo Pervasive Wizards	Elementos do modelo de aplicação PM2G
Armas, porções e elementos de proteção	Objetos
Magos	Avatares
Cachorros selvagens	Personagens automatizados (NPCs)
Tabuleiro bi-dimensional (Matriz NxM) de áreas retangulares de tamanho fixo	Mundo virtual
Região onde um mago esteja posicionado	Área de interesse

Apesar de simples, o projeto do jogo é suficiente para exigir a maior parte das funcionalidades implementadas pelos serviços da arquitetura proposta. O jogo foi rebatizado de *Pervasive Wizards*, onde os serviços da arquitetura e os clientes para computadores pessoais de mesa foram implementados utilizando a plataforma Java J2SE, versão 1.5. Os clientes móveis foram desenvolvidos usando a plataforma Java J2ME, versão 2.3.

## 7.3 Prototipação

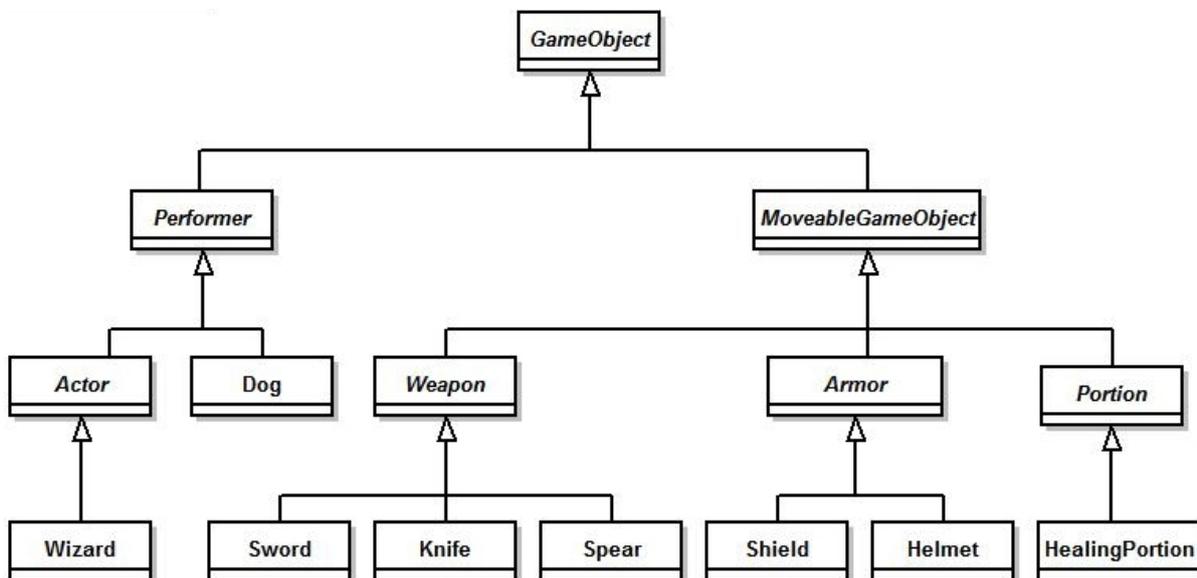
Para prototipação do jogo foram realizadas algumas etapas, também necessárias à criação de qualquer jogo a ser utilizado sobre o *middleware* PM2G. Estas tarefas consistem em criar as entidades do jogo a partir das entidades abstratas, mas especificamente, pelo *framework* de simulação do *middleware* PM2G, bem como configurar os serviços na inicialização do jogo, para serem utilizados ao longo da simulação. Além disto, foram implementados clientes para acesso ao jogo via computadores pessoais de mesa e para dispositivos móveis, textuais e gráficos. As subseções seguintes apresentam detalhes sobre a prototipação destas tarefas.

### 7.3.1 Definição das Entidades do Jogo

As classes concretas representam objetos, ações e estados de um avatar, além do próprio jogador do jogo *Pervasive Wizards*. Detalhes sobre estas classes são apresentadas a seguir.

- **Em relação aos usuários:** foi criada a entidade *DefaultPlayer*, que estende a classe *Player*, acrescentando informações para notificação de mensagens, como endereço eletrônico do jogador e identificador para recebimento de mensagens SMS;
- **Em relação aos objetos:** várias subclasses foram criadas, como mostra a Figura 7.2. Se-

gundo categorização de itens, a classe *MoveableGameObject* refere-se aos objetos que podem ser carregados pelos avatares. Estes podem ser armaduras (classe *Armor*), porções (classe *Portion*) ou armas (classe *Weapon*). Armaduras podem ser especializadas em elmos (classe *Helmet*) ou escudos (classe *Shield*). A classe *HealingPortion* representa uma porção que pode ser carregada pelo avatar e faz com que seus pontos de vida possam ser recuperados mais rapidamente. As classes *Knife*, *Sword* e *Spear* representam, respectivamente, facas, espadas e lanças. Além destes itens, há duas subclasses da classe *Performer*: *Dog* e *Wizard*, que é também subclasse de *Actor*. A classe *Dog* representa o NPC com o qual o jogador não interage, enquanto a classe *Wizard* representa o avatar do jogador;



**Figura 7.2** Diagrama de Classes – Subclasses criadas para entidades do Jogo *Pervasive Wizards*

- **Em relação aos estados de um jogador:** para jogadores utilizando computadores pessoais, foram definidos quatro estados: (i) *IdleState*, no qual o personagem não faz nada; (ii) *MovingState*, que indica movimentação do avatar; (iii) *FightingState*, no qual o avatar está lutando contra um adversário; e (iv) *RestingState*, estado em que o personagem recupera sua saúde com o passar do tempo e não pode ser atacado. Para os dois contextos móveis, três estados foram definidos: (i) *RestingMobileState*, no qual o avatar descansa de maneira semelhante ao *RestingState* e também não pode ser atacado; (ii) *FightingMobileState*, estado assumido pelo personagem após o jogador enviar o comando de atacar um adversário, e (iii) *CollectingMobileState*, estado assumido pelo avatar após o jogador

enviar o comando de coletar um item;

- **Em relação as ações de um jogador:** duas ações foram definidas para os jogadores móveis orientarem seus avatares: *GetItemAction* e *GoFightAction*. A primeira é realizada quando o jogador deseja coletar algum item na sua área de interesse, e a segunda ação corresponde ao comando do jogador para atacar algum adversário. A cada uma delas foi associado o código do evento de jogo correspondente.

### 7.3.2 Representação da Área de Interesse

Como apresentado na Tabela 7.1, a área de interesse de cada jogador é determinada pelos elementos presentes na região em que seu avatar esteja posicionado. No caso específico do Jogo *Pervasive Wizards*, a região do mundo é representada por uma área retangular, que contém os atores, personagens automatizados (NPCs) e objetos presentes naquela área do mundo virtual. Cada um destes elementos possuem propriedades específicas que refletem seu estado da região em dado instante no jogo. Por exemplo, é necessário saber informações sobre as posições de cada um destes elementos, ou mesmo, a ação que um avatar esteja realizando, para que estes possam ser representados nos clientes dos usuários.

Segundo suas possíveis representações, (Figura 6.8), a classe *SquareAOI* representa uma área de interesse de forma retangular. No jogo *Pervasive Wizards*, esta classe é especializada em um segmento. O mundo virtual é então composto por uma matriz de  $M \times N$  segmentos. A partir de cada segmento, o estado da simulação para um jogador é definido com as informações do segmento em que o mesmo esteja inserido em um dado instante. Para representar este estado, foi definida uma estrutura que engloba as informações relevantes dos elementos que compõem um segmento do mundo virtual. A Tabela 7.2 apresenta de forma resumida esta estrutura.

Esta estrutura representa as informações repassadas a jogadores em computadores pessoais, onde o conjunto maior de dados é transmitido. Para jogadores em dispositivos móveis, o serviço de adaptação de conteúdo realiza procedimentos que filtram ou transformam estas informações de acordo com as estratégias pré-definidas.

### 7.3.3 Implementação dos clientes de acesso ao Jogo

Para acesso dos jogadores nos diferentes contextos contemplados no jogo, foram implementadas três aplicações-clientes. Seguindo a arquitetura PM2G, estes clientes servem como portais de interação para o jogo, onde nenhum processamento local em relação ao estado do jogo é feito. O jogo repassa a informação da área de interesse do jogador, de acordo com o formato

**Tabela 7.2** Estrutura de Dados da Representação de uma área de interesse no Jogo *Pervasive Wizards*

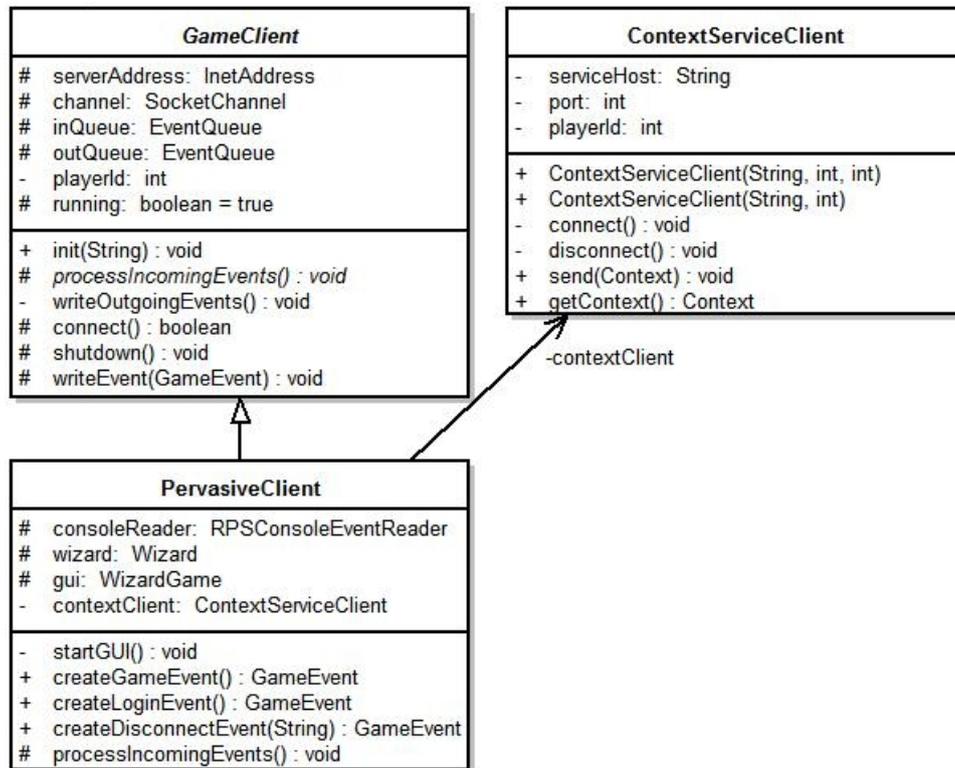
Tipo	Descrição
inteiro	Número de Atores presentes na região
Estrutura[inteiro]	Informação de cada ator presente na região
– inteiro	ID do ator
– byte	Tipo do ator
– inteiro	Coordenada X do ator no mundo global
– inteiro	Coordenada Y do ator no mundo global
– inteiro	Pontos de vida do ator
– inteiro	Pontos de mana do ator
– inteiro	Velocidade de deslocamento do ator
– Estrutura	Informações da ação atual do ator
– byte	Quantidade de objetos carregados pelo ator
– - Estrutura[byte]	Informações sobre cada objeto do ator
– inteiro	ID do objeto
– byte	Tipo do Objeto
inteiro	Número de NPCs presentes na região
Estrutura[inteiro]	Informação de cada NPC presente na região
– inteiro	ID do NPC
– byte	Tipo do NPC
– inteiro	Coordenada X do NPC no mundo global
– inteiro	Coordenada Y do NPC no mundo global
– inteiro	Velocidade de deslocamento do NPC
inteiro	Número de itens coletáveis presentes da região
Estrutura[inteiro]	Informação de cada item presente na região
– inteiro	ID do item
– byte	Tipo do Item
– inteiro	Coordenada X do item no mundo global
– inteiro	Coordenada Y do item no mundo global

definido pela estratégia de adaptação associada ao seu contexto. A aplicação-cliente apresenta ao jogador a percepção desta área de interesse, bem como outras informações relativas à interação do mesmo.

#### 7.3.3.1 Clientes para computadores pessoais de mesa

A classe *PervasiveClient* é utilizada para acesso ao jogo via computadores pessoais de mesa, representada pela Figura 7.3. Esta classe estende a classe *GameClient*, definindo ações para os eventos recebidos da simulação, ao mesmo tempo que utiliza a classe *ContextServiceClient*,

para envio de informações para o Serviço de Gerenciamento de Contexto. Sua execução é feita através do disparo via linha de comando, onde é passado como parâmetro, o IP do servidor do jogo.



**Figura 7.3** Diagrama de Classes – Aplicação-cliente para computadores pessoais.

No caso do cliente para computadores pessoais, operações adicionais foram implementadas de modo a permitir que o usuário possa criar novas contas de acesso. Embora estes mesmos procedimentos também fossem possíveis para dispositivos móveis, estes não foram implementados devido a dificuldade para entrada de dados alfanuméricos necessários para a criação das contas. Portanto, para a participação do jogador via dispositivo móvel, espera-se que o primeiro acesso do jogador seja feito via computador pessoal de mesa, onde sua primeira tarefa é o cadastro de sua conta. Os procedimentos são realizados de acordo com comandos fornecidos pelo jogador na aplicação-cliente. A Tabela 7.3 apresenta a lista dos possíveis comandos realizados por um jogador, a lista de parâmetros de cada comando e a descrição de sua funcionalidade.

A partir da conexão do jogador e inserção de seu avatar no jogo, este passa a receber continuamente as informações do segmento do mundo virtual onde o avatar tenha sido posto, através do comando *spawn*. Para renderizar a área de interesse do usuário, a aplicação-cliente para computadores pessoais utiliza também classes do *framework Golden T Engine*[GTS06], um

**Tabela 7.3** Lista de Comandos disponíveis para um jogador, via cliente para computadores pessoais.

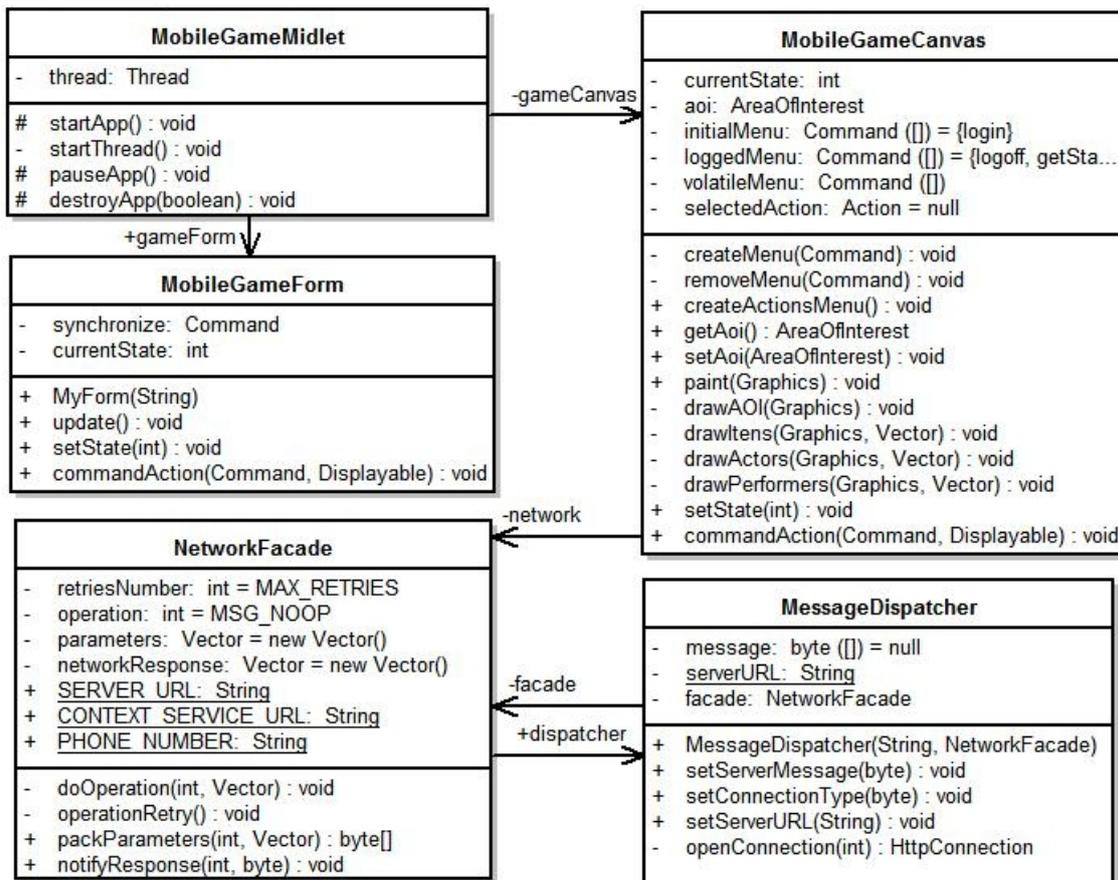
Comando	Parâmetros	Funcionalidade realizada
/sign	nickname, senha, e-mail, número telefonico para SMS	Caso não haja nenhum usuário com o nickname desejado, o servidor cria uma nova conta, e retorna com ID do jogador
/login	nickname, senha	Caso exista uma conta com o nickname passado, e cuja senha seja igual a passada pelo comando, o servidor retorna o ID do jogador, e estabelece o estado do jogador no servidor como logado
/spawn	posição X, posição Y	Caso o jogador esteja logado, o servidor insere o avatar do jogador na posição (X,Y) do mundo
/logout	-	Desconecta o jogador, retirando seu avatar o mundo

motor para o desenvolvimento de jogos em Java, que facilita o tratamento de questões como animações.

### 7.3.3.2 Clientes para dispositivos móveis

As aplicações-cliente para o contexto móvel foram projetadas para também receber as informações sobre o estado do jogo, bem como sobre as ações de alto nível que o jogador móvel poderá realizar, fornecidas pelo Serviço de Adaptação de Interação. O usuário pode se conectar no jogo usando a tupla (usuário/senha), da mesma forma que o cliente para computadores pessoais. No caso de uma conexão válida de um jogador móvel, o avatar do jogador é colocado na mesma posição em que este se encontrava na última sessão de acesso. A Figura 7.4 apresenta as principais classes desta aplicação.

Sendo desenvolvidas usando o perfil MIDP 2.0, a classe *MobileGameMidlet* representa a classe principal da aplicação. Esta possui referências para duas classes: (i) a classe *MobileGameCanvas* é utilizada por dispositivos móveis com interface gráfica, enquanto a (ii) classe *MobileGameForm* é utilizada para a apresentação da mensagem textual em dispositivos móveis mais simples. As classes *NetworkFacade* e *MessageDispatcher* tratam das questões relacionadas às conexões via HTTP com o servidor do jogo e com o serviço de adaptação de contexto.



**Figura 7.4** Diagrama de Classes – Aplicação-cliente para dispositivos móveis.

### 7.3.3.3 Envio de informações de contexto

Para os dispositivos contemplados no protótipo, foi simulada a mudança de informação de contexto, porém limitada a mudança do estado do jogador. As informações sobre o dispositivo utilizado são imutáveis para uma mesma aplicação-cliente, fazendo com o mesmo código do dispositivo seja repassado quando da mudança de estado pelo jogador. Neste protótipo, não foram utilizadas informações sobre a localização do jogador. A mudança de contexto para o cliente de computadores pessoais segue o mesmo padrão de disparo de comandos. O comando */context* faz com que a aplicação-cliente simule a mudança de contexto, através da API do *middleware*. No caso, das aplicações móveis, não foi utilizada nenhuma classe extra, e sim feita uma chamada HTTP ao serviço de contexto, onde são repassadas diretamente via fluxo de dados, as informações sobre o dispositivo e o novo estado.

### 7.3.4 Configuração dos Serviços

Embora concebidos para serem o mais genérico possível, os serviços definidos na arquitetura PM2G precisam de informações específicas do jogo *Pervasive Wizards* para seu correto funcionamento. Estas são carregadas pelos serviços no momento de sua inicialização (*bootstrap*), através de arquivos de configuração. Esta subseção apresenta a configuração destes serviços no jogo *Pervasive Wizards*.

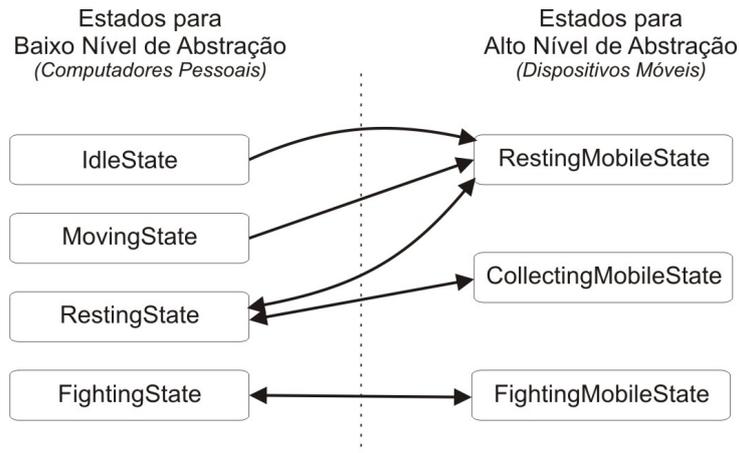
#### 7.3.4.1 Serviço de Adaptação de Conteúdo

Este serviço precisa ter armazenadas todas as estratégias de adaptação que serão utilizadas no jogo. Afinal, cada jogo pode definir um conjunto próprio de estratégias, baseado nos dispositivos para o qual é projetado. Para os cenários implementados, foram definidas três estratégias de adaptação. A primeira, para computadores pessoais, não possui módulos lógicos e o módulo de apresentação utilizado é o padrão (*DefaultPresentationMode*). Já a estratégia para dispositivos móveis com interface gráfica utiliza também o módulo de apresentação padrão, mas utiliza os módulos lógicos de escala bidimensional e filtragem de informações. Em outras palavras, a área de interesse é redimensionada para as dimensões do dispositivo-alvo, e informações irrelevantes sobre NPCs não são repassadas ao jogador. Finalmente, a estratégia de adaptação para telefones celulares de interface textual utiliza os mesmos módulos lógicos da estratégia anterior, porém com o módulo de apresentação textual.

#### 7.3.4.2 Serviço de Adaptação de Interação

Quanto a este serviço, é necessário ao jogo fornecer duas informações: uma associação entre estados correspondentes para diferentes níveis de abstração e um mapeamento entre os estados e as possíveis ações que o avatar pode realizar a partir deles. A primeira informação é necessária para que o serviço possa determinar qual o novo estado a ser assumido por um avatar cujo jogador mudou de contexto. No caso dos estados criados, foi definido que, quando o jogador muda para um contexto de alto nível de abstração, os estados *IdleState*, *MovingState*, e *RestingState* levam para o estado *RestingMobileState*, e *FightingState* leva para *FightingMobileState*. Se o jogador mudar para um contexto de baixo nível, foi definido que *CollectingMobileState*, *FightingMobileState* e *RestingMobileState* levam para *RestingState*. A Figura 7.5 ilustra as relações de transições entre os estados definidos na simulação, para diferentes contextos e níveis de abstração.

Ainda com relação ao Serviço de Adaptação de Interação, foi criada somente uma associa-

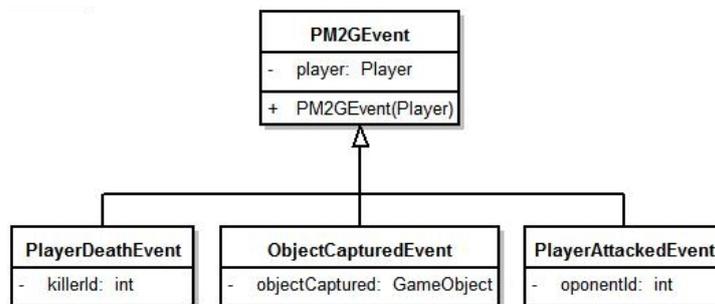


**Figura 7.5** Possíveis transições de estados em diferentes contextos de execução.

ção entre o *RestingMobileState* e as ações *GoFightAction* e *GetItemAction*. Isto quer dizer que o jogador móvel só poderá realizar ações caso seu avatar esteja neste estado.

#### 7.3.4.3 Serviço de Notificação de Mensagens

Para validação do serviço de notificação de mensagens, três eventos foram definidos como de interesse aos jogadores. São eles: (i) a coleta de um objeto pelo jogador, (ii) o fato de ter sido atacado por outro jogador e (iii) a derrota em um combate contra um adversário. Utilizando as classes abstratas do serviço, foram implementadas três novas classes, referentes a cada um dos eventos citados: *ObjectCapturedEvent*, *PlayerAttackedEvent* e *PlayerDeathEvent*. A Figura 7.6 apresenta o diagrama destas classes em UML. Cada um destes eventos insere informações específicas relacionadas com o mesmo, e que são utilizadas para a construção das mensagens de alertas aos jogadores. Por exemplo, o evento relativo à coleta de um objeto insere qual objeto foi coletado, enquanto os demais inserem a informação sobre qual jogador atacou ou derrotou o avatar do jogador que receberá a mensagem.



**Figura 7.6** Diagrama de Classes dos Eventos concebidos para o jogo *Pervasive Wizards*.

Seguindo o modelo definido pelo serviços, três classes, *ObjectCapturedMessageFactory*, *PlayerAttackedMessageFactory* e *PlayerDeathMessageFactory*, são responsáveis por criar as mensagens relativas a cada um desses eventos.

Na execução do serviço, utilizou-se uma estratégia simples de escolha do meio de entrega da mensagem ao jogador. Jogadores em computadores pessoais não recebem mensagens, visto que estes controlam seu avatar e presenciam os acontecimentos diretamente na tela de seu computador pessoal; Jogadores em dispositivos móveis com interface gráfica recebem uma notificação por email, enquanto dispositivos móveis com interface textual recebem uma mensagem SMS.

Os procedimentos de envio das mensagens por cada uma destas tecnologias é realizado por dois componentes: o FreeSMTP Server [Sof06] e o SMS Ozeki Server [OL06]. O primeiro é um servidor SMTP para plataforma Windows, que possibilita o envio de email diretamente por computador pessoal, sem a necessidade de comunicação com servidores SMTP de provedores de serviço. O segundo componente permite que aplicações escritas em diversas linguagens, inclusive Java, possam enviar e receber mensagens SMS através de um computador pessoal. O envio pode ser feito pela Internet usando o IP do *gateway* SMS de uma operadora ou usando um telefone *GSM* conectado ao computador via um cabo comum de transferência de dados. A integração entre as aplicações e o servidor Ozeki pode ser feito de várias formas, sendo que a utilizada pelo Serviço de Notificação de Mensagens foi através de arquivos texto que são escritos em um diretório específico, obedecendo um formato determinado pelo servidor Ozeki. Neste formato, cada arquivo possui três linhas, onde a primeira contém o telefone do emissor do SMS, a segunda a do destinatário e a terceira contém a mensagem de até 255 caracteres. O Servidor Ozeki fica em constante observação deste diretório, consumindo os arquivos e enviando as mensagens através do telefone celular. Para configuração do serviço, um arquivo de configuração foi definido, onde as informações sobre a integração de cada um destes componentes com serviço foi definida.

O último passo da configuração do serviço é a definição de arquivo de configuração, onde é feita a relação entre os eventos e suas classes de tratamento. Este arquivo é apresentado na Figura 7.8.

A notificação destas mensagens obedece a regra definida pelo estado de cada jogador, segundo a Tabela 6.2, que indica o estado de cada jogador em relação à notificação de mensagens.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<notification_service>
  <email>
    <mail_server>localhost</mail_server>
    <game_sender>famt@cin.ufpe.br</game_sender>
  </email>
  <sms>
    <phone_sender>818867XXXX</phone_sender>
    <sms_out_directory>c:/TMP/SMSOUT</sms_out_directory>
    <sms_sent_directory>c:/TMP/SMSSENT</sms_sent_directory>
    <sms_failed_directory>c:/TMP/SMSFAILED</sms_failed_directory>
  </sms>
</notification_service>

```

**Figura 7.7** Configuração do Serviço de Notificação de Eventos para o jogo *Pervasive Wizards*

#### 7.3.4.4 Serviço de Integração de Mini-Mundos

Em relação a este serviço, não foi implementada uma aplicação sobre uma tecnologia específica, como *bluetooth* ou *Wi-Fi*. Buscou-se mais testar as funcionalidades do serviço, seguindo as especificações detalhadas no capítulo anterior. Para isto, foi implementada uma aplicação que assume o papel de um cliente do serviço de integração de mini-mundos. Esta aplicação pode tanto assumir o papel de um Servidor do Mini-Mundo, como se conectar a um mini-mundo em execução.

A aplicação em questão não possui um projeto de jogo definido. As ações que simulam as interações entre clientes e o serviço, e entre clientes são realizadas por comandos textuais. A Tabela 7.4 lista os possíveis comandos, seus parâmetros, a descrição de sua funcionalidade, o retorno esperado de sua chamada e as possíveis exceções existentes para sua chamada.

No caso do jogo *PervasiveWizards*, o perfil (*Profile*) que é recuperado por cada jogador inclui as informações sobre seus pontos de vida e de mana. O serviço foi então configurado para fornecer estas informações a partir dos pedidos dos clientes, através do comando *request*.

Uma aplicação-cliente é responsável por criar a instância do mini-mundo (comando *create*), enquanto outras registram-se (comando *register*) como clientes do mesmo. Após a fase de inicialização, o jogador muda o estado do mini-mundo (comando *status*) para “em andamento”, fazendo com que todos os jogadores busquem no mini-mundo, o estado do jogo (comando *state*). O estado do Mini-Mundo é o conjunto dos perfis de todos os jogadores que participam do jogo. Cada jogador mantém uma cópia deste conjunto. O comando *set\_prop* é utilizado para realizar a mudança nestes atributos do perfil de um jogador em cada aplicação-cliente. Desta forma, busca simular a modificação destes valores, de acordo com a dinâmica do mini-mundo.

Para realizar os procedimentos de finalização do mini-mundo, o jogador que criou o mini-mundo modifica seu estado no serviço, para que o mesmo possa receber os resultados de cada

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<event_manager>
  <event_conf>
    <event_class>
      pm2g.proto.events.ObjectCapturedEvent
    </event_class>
    <message_factory_class>
      pm2g.proto.events.factory.ObjectCapturedMessageFactory
    </message_factory_class>
  </event_conf>
  <event_conf>
    <event_class>
      pm2g.proto.events.PlayerDeathEvent
    </event_class>
    <message_factory_class>
      pm2g.proto.events.factory.PlayerDeathMessageFactory
    </message_factory_class>
  </event_conf>
  <event_conf>
    <event_class>
      pm2g.proto.events.PlayerAttackedEvent
    </event_class>
    <message_factory_class>
      pm2g.proto.events.factory.PlayerAttackedMessageFactory
    </message_factory_class>
  </event_conf>
</event_manager>

```

**Figura 7.8** Configuração entre Eventos e *ObjectFactories* para o jogo *Pervasive Wizards*

jogador. Esta sinalização indica que todos as aplicações-clientes devem enviar ao serviço, seu estado local (comando *submit*). Após todas as aplicações-clientes terem enviado seu estado notificando o mestre, este jogador pede ao serviço que finalize o mini-mundo (comando *finish*). O serviço então compara todas as cópias recebidas. Caso apresentem o mesmo conjunto de atributos com valores idênticos, os resultados são integrados aos perfis de cada jogador. Caso contrário, os resultados são descartados, e o responsável pela criação do mini-mundo é notificado que o jogo foi corrompido (“trapaçado”).

## 7.4 Descrição dos experimentos

Nesta seção são apresentados os resultados da utilização do jogo *Pervasive Wizards*. Os resultados forma obtidos através do uso de emuladores para os dispositivos móveis, enquanto computadores pessoais foram utilizados para execução do servidor do jogo e dos cinco serviços da arquitetura PM2G.

**Tabela 7.4** Comandos aplicação de Mini-Mundos

Commando	Parâmetros	Descrição	Retorno	Exceções Possíveis
/set_player	int playerID	Estabelece o playerID do jogador na aplicação-cliente	-	-
/set_mw	int miniWorldID	Estabelece o ID do mini-mundo na aplicação-cliente	-	-
/request	int playerID	Solicita ao Serviço, o perfil do avatar do jogador com playerID	Profile	ProfileInvalid
/create	int playerID, int type, int numMax	Cria no Serviço, uma nova instância de um mini-mundo de um tipo específico, com um número máximo de jogadores. O jogador é estabelecido como owner do Mini-Mundo	int miniWorldID	ProfileInvalid
/register	int playerID, int miniWorldID	Registra o jogador em um mini-mundo	-	ProfileInvalid, PlayerAlreadyRegistered, MiniWorldFull, InvalidMiniWorld, MiniWorldAlreadyStarted
/set_prop	int playerID, String prop, int newValue	Muda no estado local do jogador, o atributo prop do jogador playerId para um novo valor	-	-
/state	int playerID, int miniWorldID, int newState	Recupera o estado do mini-mundo	Profile[]	ProfileInvalid, InvalidMiniWorld, PlayerNotRegistered
/status	int playerID, int miniWorldID	Modifica o estado do mini-mundo	-	NotOwner, InvalidMiniWorld
/submit	int playerID, int miniWorldID, Profile[] state	Submete ao serviço o estado final do jogo	-	ProfileInvalid, InvalidMiniWorld, InvalidParameter, MiniWorldNotClosed, PlayerNotRegistered
/finish	int playerID, int miniWorldID	Finaliza o mini-mundo	-	ProfileInvalid, InvalidMiniWorld, NotOwner, InvalidMiniWorldState, MiniWorldCheated

### 7.4.1 Obtenção do estado do jogo

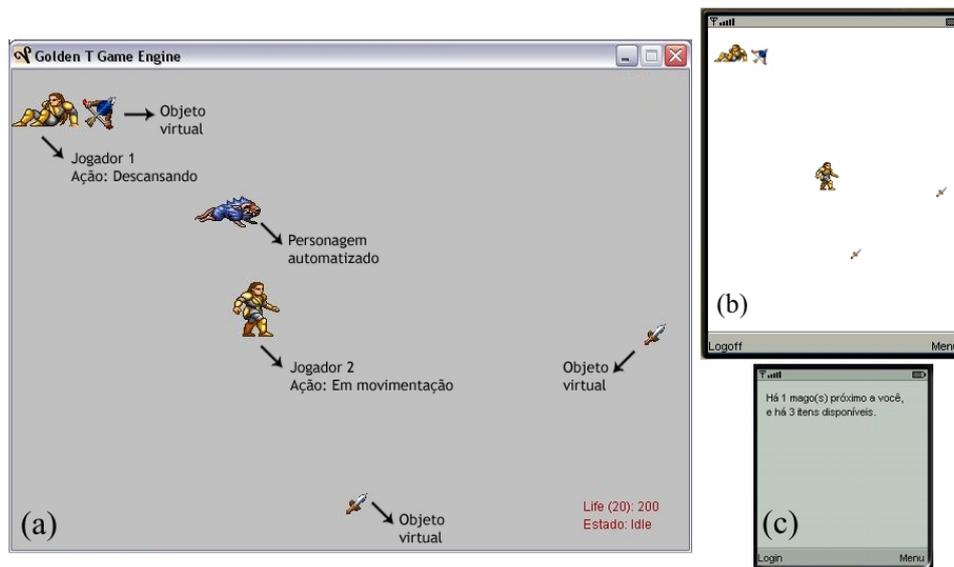
Antes de realizar qualquer interação com o seu avatar, o jogador deve estar ciente do conteúdo da sua área de interesse. Estas informações são repassadas em uma frequência, quantidade e nível de detalhamento que depende do contexto do jogador.

Jogadores em computadores pessoais são atualizados constantemente com estas informações a uma taxa de 100ms, mesmo quando não estão interagindo com o jogo no momento, sendo elas continuamente exibidas. Jogadores móveis, por sua vez, têm que requisitar o estado do jogo sempre que desejarem informações da sua área de interesse.

A quantidade de informações recebidas varia para cada contexto de execução dos jogadores. Os jogadores em computadores pessoais de mesa recebem todas as informações possíveis, sejam elas a respeito dos personagens dos outros jogadores, dos NPCs ou dos itens virtuais.

Jogadores móveis, por sua vez não recebem informações acerca dos NPCs. Além disso, para cada contexto móvel, o nível de detalhamento das informações é diferente: jogadores que utilizam dispositivos móveis com interface gráfica têm uma noção da posição de cada personagem na área de interesse, enquanto jogadores com aparelhos celulares de interface textual não têm este tipo de informação.

A Figura 7.9 mostra as visões de jogadores em cada um dos contextos para um mesmo instante do jogo. A primeira imagem (a) trata-se da tela da aplicação para computadores, em que são exibidos os avatares, itens virtuais e NPCs. A imagem (b) corresponde à tela da aplicação para dispositivos móveis com interface gráfica, e exibe apenas avatares e itens virtuais. A menor imagem (c) é uma tela para aparelhos celulares com interface textual, onde é apresentada uma mensagem informando quantos são os atores e itens presentes na área de interesse.



**Figura 7.9** Diferentes visões do estado de uma região do jogo *Pervasive Wizards*.

#### 7.4.2 Movimentação do avatar

Neste cenário, o jogador controla livremente a movimentação do seu avatar. Isto pode ser utilizado para um objetivo específico, como um ataque ou coleta de item virtual, mas pode também ser realizado apenas com o propósito de locomover o personagem.

A simples movimentação do avatar, sem fins específicos, foi definida como uma interação de baixo nível, pois exige do jogador um controle minucioso da posição atual do seu personagem. Desta forma, foi implementado apenas para o contexto de computadores pessoais. O

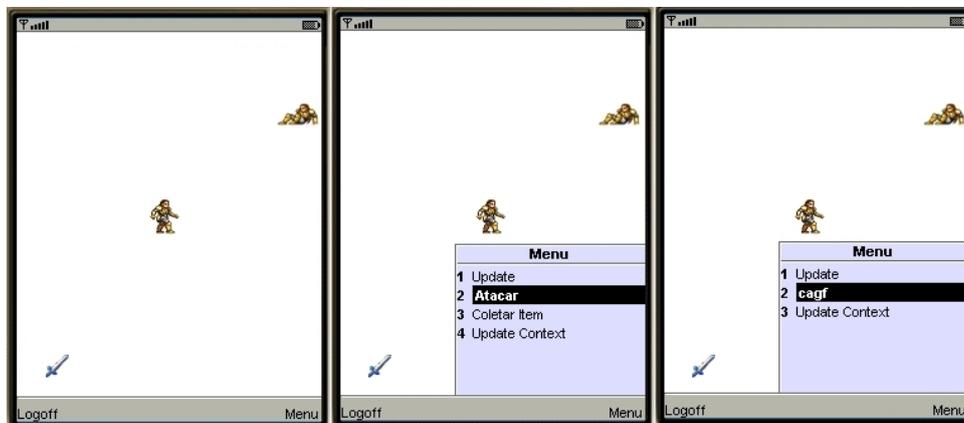
jogador interage utilizando seu dispositivo apontador (*mouse*), indicando a posição do mundo virtual para a qual deseja que seu personagem se dirija.

### 7.4.3 Ataque a um adversário

Para jogadores em computadores pessoais de mesa, o ataque é realizado quando o personagem se movimenta em direção a outro personagem e, então, colide com o mesmo. Neste momento, os dois personagens entram em estado de luta. Não existe, portanto, um comando explícito para atacar; em vez disso, o jogador tem que controlar seu personagem e guiá-lo até o adversário.

Por sua vez, o jogador móvel não tem um controle direto da movimentação do seu personagem. Para que seu avatar realize um ataque, ele realiza uma interação de alto nível com o jogo. Caso haja algum adversário na sua área de interesse, será possível a ele realizar o comando de ataque especificando o adversário. Seu avatar, então, se locomoverá de maneira automática até o adversário e os dois começarão a lutar entre si.

A Figura 7.10 exibe um comando explícito para a ação de ataque, disponível para o jogador no contexto móvel. São associados a esta ação os objetos da área de interesse que se relacionam com ela. No exemplo, o personagem do jogador *cagf*, possível alvo da ação, é exibido para que o jogador possa selecioná-lo.



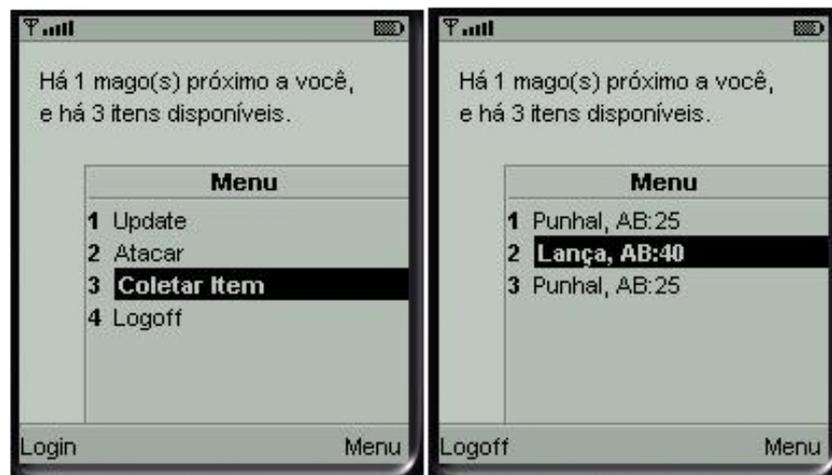
**Figura 7.10** Ação de ataque a um jogador no jogo *Pervasive Wizards*, utilizando um dispositivo móvel com interface gráfica

### 7.4.4 Coleta de item virtual

De forma semelhante ao ataque a um adversário, a coleta de item virtual é realizada pelo jogador estacionário quando o personagem se movimenta em direção a um item na sua área de

interesse e colide com ele. Também não existe um comando explícito para coletar um item virtual, sendo necessário ao jogador dirigir seu personagem até o item desejado.

Já o jogador móvel dispõe de um comando direto, caso haja itens virtuais visíveis na sua área de interesse. Quando a ação é enviada ao servidor, a simulação do jogo movimenta o personagem até o item virtual, até que haja uma colisão e o item seja coletado. A Figura 7.11 apresenta as opções oferecidas a um jogador a partir de sua área de interesse.



**Figura 7.11** Coleta de um item no jogo *Pervasive Wizards*, utilizando um dispositivo móvel com interface textual

#### 7.4.5 Simulação de Mini-Mundos

Foram simuladas instâncias de mini-mundos onde usuários trocam informações sobre seus perfis, e submetem o resultado desta interação ao serviço. Nestas experiências, simulou-se a tentativa de trapaça por parte de algum jogador para avaliar o comportamento do serviço ao efetivar os resultados do mini-mundo. A Figura 7.12 apresenta o relatório de execução do serviço durante a realização de duas instâncias de mini-mundos entre dois jogadores.

Na primeira instância, criada pelo jogador *famt*, o jogador *cagf* se conecta ao mini-mundo. Quando o jogo é posto de estado de execução, cada jogador recebe o estado a partir do serviço. Após suas interações, o jogador *famt* sinaliza ao serviço a finalização do Mini-Mundo. Os dois jogadores enviam seus respectivos estados ao serviço, que verifica a consistência dos mesmos, para integrá-los ao perfil de cada jogador. No caso da primeira instância, o servidor verificou que não houve problemas entre as cópias recebidas, e portanto, realizou a integração dos resultados.

Para a segunda instância criada, inverteu-se os papéis entre os jogadores: o jogador *cagf*

```

D:\pm2g\out>java -cp ..\xalan.jar;..\lib\xerces.jar;..\lib\ant.jar;..\lib\hsqldb.jar;..\lib\log4j-1.2.13.jar;..\lib\log4j-1.2.6.jar;..\lib\jaxen-core.jar;..\lib\jdom.jar;..\lib\xml-apis.jar;..\lib\ogniblen_0_2_3.jar;..\lib\junit-4.1.jar;..\lib\communication-protocol-1.1.jar;..\lib\moca-util-1.1.jar;..\lib\ecj-server-1.1.1.jar;..\lib\saxpath.jar;..\lib\ecj-client-1.1.1.jar;..\lib\jaxen-jdom.jar pm2g-services_mineworld.MineworldService
log4j:ERROR Could not find value for key log4j.appender.ECJFile
log4j:ERROR Could not instantiate appender named "ECJFile".
0 [main] INFO MiniWorldManager - MiniWorld Manager ready!
15 [main] INFO MiniWorldService - Acceptor running at Port:33001
15 [main] INFO MiniWorldService - HTTPAcceptor running at Port:33002
3388 [WorkerThread_0] INFO MiniWorldService -
Sessão Mini-Mundo 0, sem jogadores trapaceiros.
46276 [WorkerThread_4] INFO MiniWorldManager
49427 [WorkerThread_5] INFO MiniWorldService
53477 [WorkerThread_6] INFO MiniWorldService
62590 [WorkerThread_7] INFO MiniWorldManager
66718 [WorkerThread_8] INFO MiniWorldService
72419 [WorkerThread_9] INFO MiniWorldService
72419 [WorkerThread_9] INFO MiniWorldService
80784 [WorkerThread_10] INFO MiniWorldManager
80800 [WorkerThread_10] INFO MiniWorldService
80800 [WorkerThread_10] INFO MiniWorldManager
80800 [WorkerThread_10] INFO MiniWorldService
95287 [WorkerThread_11] INFO MiniWorldService
101549 [WorkerThread_12] INFO MiniWorldService
108013 [WorkerThread_13] INFO MiniWorldManager
Sessão Mini-Mundo 1, com jogadores trapaceiros.
150946 [WorkerThread_16] INFO MiniWorldService
154465 [WorkerThread_17] INFO MiniWorldService
204751 [WorkerThread_18] INFO MiniWorldManager
212963 [WorkerThread_19] INFO MiniWorldService
217233 [WorkerThread_20] INFO MiniWorldService
217233 [WorkerThread_20] INFO MiniWorldService
226131 [WorkerThread_21] INFO MiniWorldManager
ences for player famt's results
- Profile sent for player famt
- MiniWorld(0) created: owner->famt, maxPlayers ->2
- Player cagf has been registered in MiniWorld No. 0
- MiniWorld(0) status updated to 'Running'
- MiniWorld No. 0's State sent to Player famt
- MiniWorld No. 0's State sent to Player cagf
- MiniWorld(0) status updated to 'Closing'
- Results from Player famt has been received for MiniWorld No. 0
- Results from Player cagf has been received for MiniWorld No. 0
- MiniWorld(0) status updated to 'Ready to Finish'
- MiniWorld(0) results check was made successfully
- Profile updated for player No. famt
- Profile updated for player No. cagf
- MiniWorld(0) results committed
- MiniWorld(0) finished!
- Profile sent for player cagf
- Profile sent for player famt
- MiniWorld(1) created: owner->cagf, maxPlayers ->2
- Player famt has been registered in MiniWorld No. 1
- MiniWorld(1) status updated to 'Running'
- MiniWorld No. 1's State sent to Player cagf
- MiniWorld No. 1's State sent to Player famt
- MiniWorld(1) status updated to 'Closing'
- Results from Player cagf has been received for MiniWorld No. 1
- Results from Player famt has been received for MiniWorld No. 1
- MiniWorld(1) status updated to 'Ready to Finish'
- MiniWorld(1) results check failed due to inconsistent results for player famt's results

```

Figura 7.12 Amostra do acompanhamento da execução do serviço de integração de mini-mundos

criou o mini-mundo, enquanto o jogador *famt* conectou-se como um cliente. Os procedimentos seguem de forma similar à instância anterior, porém neste caso, buscou-se modificar os resultados de cada jogador, de forma a simular a tentativa de trapaça de um deles. Neste caso, o serviço ao receber os resultados, verificou que os mesmos não coincidem, fazendo com que a instância do Mini-Mundo fosse encerrada, sem que seus resultados fossem integrados ao jogo.

## 7.5 Avaliação do Middleware

Através do protótipo implementado, buscou-se avaliar o quanto o *middleware* PM2G facilita o desenvolvimento de jogos multiplataforma, seguindo os cenários propostos neste trabalho. Para isto, foram levantados dados sobre o quão genérico seria o *middleware* PM2G. Esta medida baseou-se na divisão das classes criadas no protótipo implementado em duas categorias. A primeira diz respeito às classes específicas do jogo *Pervasive Wizards*, ou seja, que foram criadas exclusivamente para uso neste jogo. A segunda categoria engloba as classes consideradas genéricas do *middleware*, e que podem ser reutilizadas por outros jogos PM2G.

Como resultado deste levantamento, a Tabela 7.5 apresenta os dados destas duas categorias,

de acordo com o número de classes relacionadas com os diferentes componentes da arquitetura PM2G. No total, foram contabilizadas 207 classes, sendo 151 consideradas genéricas, enquanto 51 específicas do jogo *Pervasive Wizards*. Das classes específicas, a maior parte concentra-se na definição dos elementos do jogo (Mago, NPCs e Objetos), representadas pelas 24 classes do framework de simulação. As 11 classes específicas do jogo relacionadas com o *framework* de comunicação englobam os clientes definidos para acesso do usuário ao jogo através dos diferentes dispositivos, bem como a definição de eventos específicos do jogo.

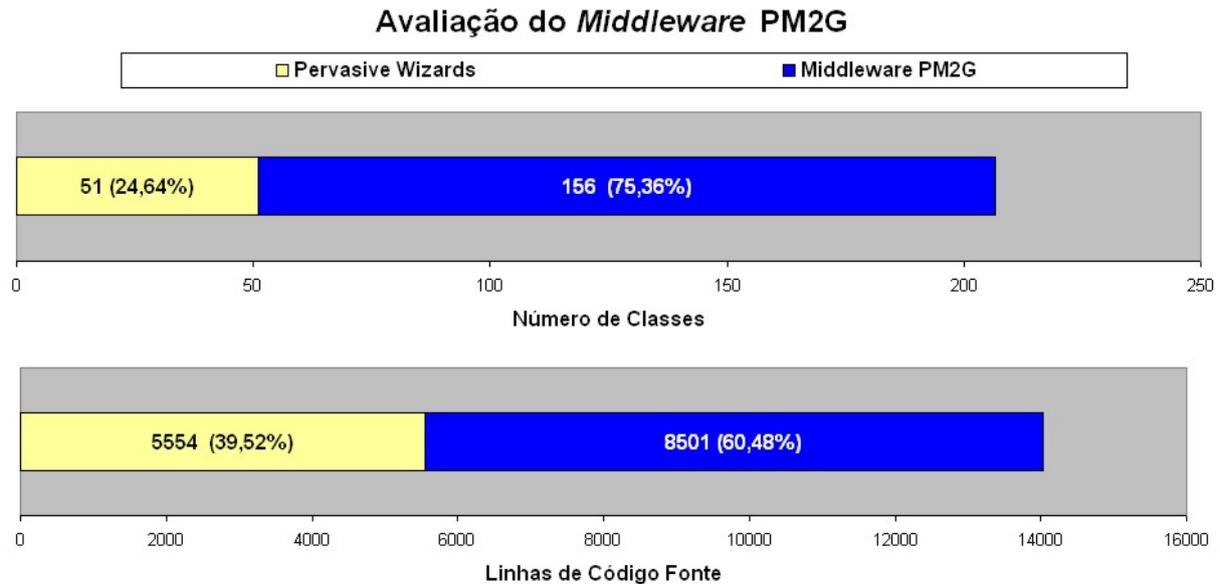
**Tabela 7.5** Avaliação do *Middleware*

	Classes Específicas		Classes Genéricas		Total de Classes
	Quantidade de Classes	%	Quantidade de Classes	%	
Utilitárias	1	1.96	0	0.00	1
<i>Framework</i> de Comunicação	11	21.57	44	28.21	55
<i>Framework</i> de Simulação	24	47.06	21	13.46	45
Gerenciamento de Contexto	0	0.00	21	13.46	21
Adaptação de Conteúdo	1	1.96	12	7.69	13
Adaptação de Interação	5	9.80	42	7.69	47
Notificação de Mensagens	6	11.76	48	11.54	54
Integração de Mini-Mundos	3	5.88	28	17.95	331
Total	51	100.00	156	100.00	207

Em relação aos serviços, destaca-se a total independência do jogo, do serviço de gerenciamento de contexto, visto que não houve necessidade de nenhuma interferência do desenvolvedor para uso do serviço. Em relação ao serviço de adaptação de conteúdo, faz-se necessário a implementação de uma única classe para definição das informações da área de interesse do jogador, e que serve de base para as transformações realizadas pelo serviço. Em relação ao serviço de adaptação de interação, foi necessário informar as ações de alto nível disponíveis para jogadores em dispositivos móveis. Estas informações são dependentes de cada projeto de jogo. No jogo *Pervasive Wizards*, as quatro ações definidas no projeto do jogo são implementadas através de 5 classes específicas do jogo, relacionadas com o serviço de adaptação de interação. Por sua vez, o serviço de notificação de mensagens precisa que para cada evento tratado, seja definida uma classe que representa o evento e outra que representa a classe (fábrica) que cria as mensagens para este evento. No caso do protótipo, foram definidos 3 eventos, conseqüentemente necessitando de 3 fábricas de mensagens, totalizando 6 classes. Por fim, o experimento realizado com o serviço de integração de mini-mundos, mesmo tendo a idéia

que cada mini-mundo será uma aplicação distinta, permitiu que fossem criadas 3 classes que auxiliam a implementação destas aplicações.

Através deste levantamento, verificou que mais de 75% do protótipo englobam classes genéricas, e que não necessitariam de qualquer interferência do desenvolvedor para a criação de novos jogos PM2G (vide Figura 7.13).



**Figura 7.13** Avaliação do *Middleware* PM2G: Quantidade de Classes e Linhas de Código

Neste mesmo levantamento, utilizou-se outra métrica para avaliar o esforço durante o experimento: o número de linhas de código fonte (excluindo linhas em branco e comentários) das classes das duas categorias. Observou-se que a relação obtida para a primeira categoria não é mantida. Isto indica que classes específicas para cada jogo têm uma granularidade mais fina que aquelas definidas pelas classes genéricas.

Mesmo com esta característica, a relação entre as classes que são efetivamente de responsabilidade de cada jogo indica que o *middleware* PM2G tende a atenuar o esforço e tempo de desenvolvimento de jogos com as características propostas neste trabalho.

## CAPÍTULO 8

# Conclusões

*Tudo aquilo que tem um início, tem um fim.*

— ANDY & LARRY WACHOWSKI (do filme “Matrix Revolutions”)

A integração entre de jogos multiusuário, mobilidade e heterogeneidade de dispositivos apresenta-se como uma tendência para jogos digitais futuros. Este trabalho apresentou uma visão para esta integração, através de cenários, modelos e uma arquitetura de suporte ao desenvolvimento de execução de jogos multiplataforma.

A arquitetura proposta segue o modelo tradicional de suporte a jogos multiusuário, oferecendo uma camada adicional de serviços de *middleware* que objetivam facilitar a programação destas aplicações, permitindo que uma única instância de jogo possa ser compartilhada entre jogadores em diferentes dispositivos. Nesta arquitetura, uma atenção especial é dada às questões relativas à adaptação da jogabilidade do usuário enquanto o mesmo migra entre dispositivos.

### 8.1 Contribuições

Primeiramente, neste trabalho foram estabelecidos cenários que buscam utilizar serviços e vantagens de aspectos da computação pervasiva para melhorar a experiência de jogadores.

A partir dos cenários, foram especificados modelos que (i) criam uma terminologia para elementos que fazem parte de um jogo com as características pretendidas, e (ii) ditam como se dá a participação de um usuário no jogo.

A principal contribuição é de natureza arquitetural. O conjunto dos serviços propostos no Capítulo 6 permite o desenvolvimento de jogos que seguem os modelos descritos previamente, sendo comprovados por experimentos descritos no Capítulo 7. Através destes serviços, pode-se:

- criar diferentes percepções de uma única instância de jogo para usuários que utilizem diferentes dispositivos, quer sejam estes móveis ou não, através do conceito de estratégias de adaptação;
- facilitar a execução de tarefas por jogadores que utilizem dispositivos móveis através

das ações geradas pelo serviço de adaptação de interação. Estas são essenciais para a integração de dispositivos que apresentam restrições de processamento e conectividade;

- permitir que o jogador esteja quase sempre acessível para receber informações sobre o andamento do jogo, através da notificação de mensagens;
- permitir que jogadores móveis possam interagir diretamente através de diferentes tecnologias de comunicação, e que os resultados destes jogos possam ser aproveitados no contexto do mundo virtual, contribuindo para aumento da dinâmica do jogo. Neste caso, o conceito de mini-mundo é também uma contribuição introduzida por este trabalho.

## 8.2 Trabalhos Futuros

Este trabalho poderá servir de base para novos estudos. Algumas sugestões são:

- Realizar mais experimentos utilizando informações de localização, integrando com outros trabalhos, como o MoCA [SER<sup>+</sup>04];
- Realizar estudos que melhorem a escalabilidade da arquitetura proposta, ao substituir a única instância do servidor do jogo por um *cluster* de servidores. Outra possibilidade é utilizar esquemas ponto a ponto entre jogadores[Cec05][HL04]. Neste segundo caso, computadores pessoais de certos jogadores poderiam assumir o papel de jogadores que utilizem dispositivos móveis, atuando como representantes (*proxies*) dos mesmos. Estes computadores assumiriam o papel de servidores para os jogadores móveis;
- Refinar o uso de contexto, fazendo com que informações técnicas mais específicas do dispositivo, como tempo de bateria, qualidade de conexão, dentre outros, tenham influência sobre o funcionamento dos serviços da arquitetura PM2G;
- Incorporar um mecanismo de modificação dinâmica dos serviços do *middleware*, permitindo a mudança sob demanda de estratégias de adaptação ou outras configurações;
- Expor os serviços a outros desenvolvedores para avaliação da facilidade de uso dos mesmos, e posterior refinamento;
- Incorporar soluções anti-trapaça, inclusive para aquelas introduzidas pelo modelo de aplicação PM2G;
- Expor os serviços a grupos de projetos de jogos, de modo a criar um experimento mais completo e aplicado à arquitetura proposta.

## Referências

- [Ale02] Thor Alexander. *Game Programming Gems*, volume 3, chapter A Flexible Simulation Architecture for Massively Multiplayer Games, pages 506 – 519. Charles River Media, 2002.
- [Ale03a] Thor Alexander. *Massively Multiplayer Game Development (Game Development)*, volume 1, chapter 2.1: Building a Massively Multiplayer Game Simulation Framework, Part 1: Structural Modeling, pages 103 – 114. Charles River Media, Inc., Rockland, MA, USA, 2003.
- [Ale03b] Thor Alexander. *Massively Multiplayer Game Development (Game Development)*, volume 1, chapter 2.1: Building a Massively Multiplayer Game Simulation Framework, Part 1: Behavioral Modeling, pages 115 – 122. Charles River Media, Inc., Rockland, MA, USA, 2003.
- [AMCRC04] Jalal Al-Muhtadi, Shiva Chetan, Anand Ranganathan, and Roy Campbell. Super spaces: A middleware for large-scale pervasive computing environments. In *PERCOMW '04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, page 198, Washington, DC, USA, 2004. IEEE Computer Society.
- [Aug04] Iara Augustin. *Abstrações para uma Linguagem de Programação Visando Aplicações Móveis em um Ambiente de Pervasive Computing*. PhD thesis, Universidade Federal do Rio Grande do Sul, Janeiro 2004.
- [AYBG02] Iara Augustin, Adenauer Correa Yamin, Jorge Luis Victoria Barbosa, and Cláudio Fernando Resin Geyer. ISAM, a software architecture for adaptive and distributed mobile applications. In *ISCC '02: Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02)*, page 333, Washington, DC, USA, 2002. IEEE Computer Society.
- [BBV03] David Brackeen, Bret Barker, and Laurence Vanhelsuwé. *Developing Games in Java*. New Riders Publishing, 2003.

- [BE05] Blizzard Entertainment. Starcraft, 2005. Disponível em <http://www.blizzard.com/starcraft/>. Acesso em 02 de Maio de 2007.
- [BE06] Blizzard Entertainment. The World of Warcraft Community Site, 2006. Blizzard, Inc. Disponível em <http://www.worldofwarcraft.com/>. Acesso em 02 de Maio de 2007.
- [Ber96] Philip A. Bernstein. Middleware: a model for distributed system services. *Commun. ACM*, 39(2):86–98, 1996.
- [BHLM02] S. Björk, J. Holopainen, P Ljungstrand, and R Mandryk. Special issue on ubiquitous games. *Personal and Ubiquitous Computing*, 6(5 – 6):358 – 361, December 2002.
- [BLME04] Marcel Busse, Bernd Lamparter, Martin Mauve, and Wolfgang Effelsberg. Lightweight qos-support for networked mobile gaming. In *SIGCOMM 2004 Workshops: Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04*, pages 85–92. ACM Press, 2004.
- [Blu04] Bluetooth.org. The Official Bluetooth Membership Site, 2004. Disponível em <http://www.bluetooth.org/spec/>. Acesso em 02 de Maio de 2007.
- [Bro98] Peter Brown. Triggering information by context. *Personal and Ubiquitous Computing*, 2(1), 1998.
- [But] IBM Butterfly.net. Butterfly.net: Powering next-generation gaming with on-demand computing. Disponível em <ftp://ftp.software.ibm.com/solutions/pdfs/g325-1938-00.pdf>. Acesso em 02 de Maio de 2007.
- [Cap03] Licia Capra. *Reflective Mobile Middleware for Context-Aware Applications*. PhD thesis, University College London, 2003.
- [CCRR02] James L. Crowley, Joëlle Coutaz, Gaeten Rey, and Patrick Reignier. Perceptual components for context aware computing. In *UbiComp '02: Proceedings of the 4th international conference on Ubiquitous Computing*, pages 117–134, London, UK, 2002. Springer-Verlag.
- [CDK05a] George Couloris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concept and Design*, chapter Mobile and Ubiquitous Computing, pages 657–719. Addison Wesley, 4th edition, June 2005.

- [CDK05b] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Addison Wesley/Pearson Education, 4th edition, June 2005.
- [Cec05] Fábio Reis Cecin. FreeMMG: uma arquitetura cliente-servidor e par-a-par de suporte a jogos maciçamente distribuídos. Master's thesis, Instituto de Informática. Universidade Federal do Rio Grande do Sul, Janeiro 2005.
- [Cen01] The Norwegian Computing Center. White paper – massive Multiplayer Game networking. Technical report, The Norwegian Computing Center, 2001.
- [CFG<sup>+</sup>03] Adrian David Cheok, Siew Wan Fong, Kok Hwee Goh, Xubo Yang, Wei Liu, and Farzam Farzbiz. Human pacman: a sensing-based mobile entertainment system with ubiquitous computing and tangible interaction. In *NETGAMES '03: Proceedings of the 2nd workshop on Network and system support for games*, pages 106–117. ACM Press, 2003.
- [CG04] V. Lo C. GauthierDickey, D. Zappala. A fully distributed architecture for massively multiplayer online games. In *SIGCOMM 2004 Workshops: Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04*. ACM Press, 2004.
- [Cha02] D. Chalmers. *Contextual Mediation to Support Ubiquitous Computing*. PhD thesis, Imperial College London, 2002.
- [CK00] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.
- [Day] Daydream. Bot Fighters. Disponível em <http://www.botfighters.com>. Acesso em 02 de Maio de 2007.
- [Dey01] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001.
- [DG99] C. Diot and L. Gautier. A distributed architecture for Multiplayer interactive applications on the internet, July/August 1999.
- [DRD<sup>+</sup>00] Alan Dix, Tom Rodden, Nigel Davies, Jonathan Trevor, Adrian Friday, and Kevin Palfreyman. Exploiting space and location as a design framework for

- interactive mobile systems. *ACM Trans. Comput.-Hum. Interact.*, 7(3):285–321, 2000.
- [EHL01] Maria R. Ebling, Guerny D. H. Hunt, and Hui Lui. Issues for context services for pervasive computing: Alcatel telecommunications review, January 2001.
- [Emm00] Wolfgang Emmerich. Software engineering and middleware: a roadmap. In *Proceedings of the conference on The future of Software engineering*, pages 117–129. ACM Press, 2000.
- [Eve06] Everquest. Everquest Live.com, 2006. Disponível em <http://everquest.com>. Acesso em 02 de Maio de 2007.
- [FBA<sup>+</sup>03] Martin Flintham, Steve Benford, Rob Anastasi, Terry Hemmings, Andy Crabtree, Chris Greenhalgh, Nick Tandavanitj, Matt Adams, and Ju Row-Farr. Where on-line meets on the streets: experiences with mobile mixed reality games. In *CHI '03: Proceedings of the conference on Human factors in computing systems*, pages 569–576. ACM Press, 2003.
- [For00] UPnP Forum. UPnP device architecture, June 2000. Disponível em <http://www.upnp.org/>. Acesso em 02 de Maio de 2007.
- [Fox03] David Fox. *Massively Multiplayer Game Development (Game Development)*, volume 1, chapter 5. Small Portals: Tapping into MMP Worlds via Wireless Devices, pages 244 – 254. Charles River Media, Inc., Rockland, MA, USA, 2003.
- [FRC03] Cláudio Fernando Resin Geyer Fábio Reis Cecin, Jorge Luis Victória Barbosa. Freemng: An hybrid peer-to-peer, client-server and distributed massively multiplayer game simulation mode. In *Proceedings of the 2th Brazilian Workshop on Games and Digital Entertainment*, November 2003.
- [FSR02] Frank Fitzek, Gerrit Schulte, and Martin Reisslein. System architecture for billing of multi-player games in a wireless environment using GSM/UMTS and WLAN services. In *NETGAMES '02: Proceedings of the 1st workshop on Network and system support for games*, pages 58–64. ACM Press, 2002.
- [GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

- [GL00] Brahim Ghribi and Luigi Logrippo. Understanding GPRS: the GSM packet radio service. *Computer Networks (Amsterdam, Netherlands: 1999)*, 34(5):763–779, 2000.
- [Gou03] Rudinei Goularte. *Personalização e adaptação de conteúdo baseadas em contexto para TV Interativa*. PhD thesis, Universidade de São Paulo – São Carlos, 2003.
- [GPVD99] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol: IETF request for comments, June 1999. Disponível em <http://www.ietf.org/rfc/rfc2608.txt?number=2608/>. Acesso em 02 de Maio de 2007.
- [GTS06] Golden T Studios. Golden T Studios – Golden T Game Engine (GTGE), 2006. Disponível em <http://www.goldenstudios.or.id/products/GTGE/>. Acesso em 02 de Maio de 2007.
- [GZL05] Chris GauthierDickey, Daniel Zappala, and Virginia Lo. Peer-to-peer support for Massively Multiplayer Games. In *Proceedings of the 24th Conference of the IEEE Communications Society (In Submission)*. IEEE Computer Society, March 2005.
- [Hel03] Ville Helin. Development of modern multiplayer games. Master’s thesis, Helsinki University of Technology, July 2003.
- [HIW04] JungHyun Han, Hoh Peter In, and Jong-Sik Woo. Towards Situation-Aware Cross-Platform Ubi-Game Development. In *APSEC*, pages 734–735, 2004.
- [HL04] Shun-Yun Hu and Guan-Ming Liao. Scalable Peer-to-Peer Networked Virtual Environment. In *Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04*, pages 129–133. ACM Press, 2004.
- [IBM99] IBM. Pervasive computing. *IBM System Journal*, 38(4), 1999.
- [IDG02] IDGA. IGDA online games white paper, 2002. IDGA, [http://www.igda.org/online/IGDAOnlineGamesWhite\\_paper\\_2002.pdf](http://www.igda.org/online/IGDAOnlineGamesWhite_paper_2002.pdf). Acesso em 02 de Maio de 2007.
- [IEE05] IEEE-802.11. IEEE 802.11, The Working Group Setting the Standards for Wireless LANs, maio 2005. <http://grouper.ieee.org/groups/802/11/>.

- [IU97] Hiroshi Ishii and Brygg Ullmer. Tangible bits: Towards seamless interfaces between people, bits and atoms. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 234–241, New York, NY, USA, 1997. ACM Press.
- [Joh05] Thienne Johnson. *Uma Arquitetura de Computação Pervasiva para Trabalho de Campo*. PhD thesis, Universidade Federal de Pernambuco, Março 2005.
- [JS04] Thienne Johnson and Djamel Sadok. Lightweight architecture to provide information dissemination throughout non-infrastructure fieldwork environments. In *Proceedings of 1st International Workshop on Mobility Aware Technologies and Applications*, 2004.
- [KLXH04] Björn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins. Peer-to-peer support for Massively Multiplayer Games. In *Proceedings of the 23rd Conference of the IEEE Communications Society*. IEEE Computer Society, July 2004.
- [Kri03] Jan Krikke. Samurai romanesque, j2me, and the battle for mobile cyberspace. *IEEE Computer Graphics and Applications*, 23(1):16–23, 2003.
- [KW05] Elina M.I. Koivisto and Christian Wenninger. Enhancing player experience in mmorpqs with mobile features. In *2nd International Digital Games Research Association Conference*, 2005.
- [LG01] Z. Lei and N. Georganas. Context-based media adaptation in pervasive computing. In *Proc. of Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2001.
- [Lin05] Craig A. Lindley. Game space design foundations for trans-reality Games. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 397–404, New York, NY, USA, 2005. ACM Press.
- [LoAC05] Laboratory of Advanced Collaboration. MoCA event-based communication interface homepage, 2005. Disponível em <http://www.lac.inf.puc-rio.br/moca/event-service/>. Acesso em 02 de Maio de 2007.
- [MC05] M1nd Cooperation. Alien Revolt, Maio 2005. Disponível em <http://www.alienrevolt.com/pt>. Acesso em 02 de Maio de 2007.

- [MD97] Pascoe J. Morse D.R., Ryan N. Enhanced reality fieldwork: the context-aware archaeologist assistant. *Computer Applications and Quantitative Methods in Archaeology*, 0, 1997.
- [MG05] Meantime Games. Meu Big Brother, January 2005. Disponível em <http://meubbb.globo.com>. Acesso em 02 de Maio de 2007.
- [MGS04] Microsoft Game Studios. Welcome to mythica, 2004. <http://mythica.com>. Acesso em 02 de Abril de 2004.
- [MI01] R.L. Mandryk and K.M. Inkpen. Supporting free play in ubiquitous computer Games. In *Proceedings UbiComp 2001 Workshop on Ubiquitous Gaming*, October 2001.
- [Mic99] Sun Microsystems. Jini technology architectural overview, January 1999. Disponível em <http://www.sun.com/software/jini/whitepapers/architecture.html>. Acesso em 03 de Maio de 2007.
- [Mic02] Microsoft. Microsoft DirectX: Home page, 2002. <http://www.microsoft.com/windows/directx/default.aspx>. Acesso em 02 de Maio de 2007.
- [Mic04] Microsoft. Microsoft Game studios | the official age of empires website, 2004. Disponível em <http://www.microsoft.com/games/empires/>. Acesso em 02 de Maio de 2007.
- [MSA<sup>+</sup>06] Andrea Menezes, Carlos Eduardo Silva, Daniel Arraes, Davi Pedrosa, Fernando Brayner, Isabel Silveira, Marcus Machado, Rafael Borges, Fernando Trinta, and Geber Ramalho. Uma plataforma para jogos móveis massivamente multiusuário. In *Annais do V SBGames, Simpósio Brasileiro de Jogos para Computador e Entretenimento Digital*. Universidade Federal de Pernambuco, Novembro 2006.
- [NCs05a] NCsoft. Lineage 2, The Chaotic Chronicle – the official web site, 2005. NCsoft, Inc. <http://www.lineage2.com>. Acesso em 02 de Maio de 2007.
- [NCs05b] NCsoft. Lineage, Dark Conquest – the official web site, 2005. NCsoft, Inc. <http://www.lineage.com>. Acesso em 02 de Maio de 2007.

- [NG] Newt Games. Mogi item hunt. Disponível em <http://www.mogimogi.com>. Acesso em 02 de Maio de 2007.
- [Nin05] Nintendo. Nintendo's official site for nintendo DS, 2005. Disponível em <http://www.nintendods.com/>. Acesso em 02 de Maio de 2007.
- [Nin06] Nintendo.com. Wii.Nintendo.com – In-Depth Regional Wii Coverage, 2006. Disponível em <http://wii.nintendo.com/>. Acesso em 02 de Maio de 2007.
- [Nok] Nokia. Nokia N-Gage | Pocket Kingdom. Disponível em <http://arena.n-gage.com/n-gage/web/en/pocketkingdom/index.jsp>. Acesso em 02 de Maio de 2007.
- [Nok03] Nokia. Tibiame Case Study, June 2003. Disponível em [http://ncsp.forum.nokia.com/downloads/nokia/documents/Tibia\\_v\\_1\\_0.pdf](http://ncsp.forum.nokia.com/downloads/nokia/documents/Tibia_v_1_0.pdf). Acesso em 02 de Maio de 2007.
- [Nok04] Nokia. Nokia N-Gage | home, 2004. Nokia N-Gage. <http://www.n-gage.com>. Acesso em 02 de Maio de 2007.
- [Obj05] ObjectWeb. ObjectWeb – HomePage, 2005. Disponível em <http://www.objectweb.org/>. Acesso em 02 de Maio de 2007.
- [OL06] Ozeki Ltda. Ozeki SMS Server for GSM, 2006. Disponível em <http://www.ozeki.hu>. Acesso em 02 de Maio de 2007.
- [OMAOa] Open Mobile Alliance OMA. Oma Technical Section – mobile games interoperability forum. Disponível em <http://www.openmobilealliance.org/tech/affiliates/mgif/mgifindex.html>. Acesso em 02 de Maio de 2007.
- [OMAOb] Open Mobile Alliance OMA. Open Mobile Alliance – home. Disponível em <http://www.openmobilealliance.org>. Acesso em 02 de Maio de 2007.
- [Ori02] Origin. ORIGIN Ultima On-line, 2002. Disponível em <http://www.uo.com>.
- [Pap03] Mike P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. *wise*, 00:3, 2003.
- [Pas98] Jason Pascoe. Adding generic contextual capabilities to wearable computers. In *International Symposium On Wearable Computers*, pages 92–99, 1998.

- [PDGSS05] R. Pellerin, F. Delpiano, E. Gressier-Soudan, and M. Simatic. Gasp : Un intergiciel pour les jeux en réseaux multijoueurs sur téléphones mobiles. *Deuxièmes Journées Francophones: Mobilité et Ubiquité 2005 – UBIMOB’05*, pages 61–64, 2005.
- [Pri00] M. Pritchard. How to hurt the hackers: The scoop on internet cheating and how you can combat it, July 2000. Gamasutra, <http://www.gamasutra.com/features/20000724/pritchard01.htm>. Acesso em 02 de Maio de 2007.
- [PTFR06] Davi Pedrosa, Fernando Trinta, Carlos Ferraz, and Geber Ramalho. Serviços de adaptação de jogabilidade para jogos multiplataforma multiusuário. In *Annais do V SBGames, Simpósio Brasileiro de Jogos para Computador e Entretenimento Digital*. Universidade Federal de Pernambuco, Novembro 2006.
- [PW02a] Lothar Pantel and Lars C. Wolf. On the impact of delay on real-time Multiplayer Games. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 23–29. ACM Press, 2002.
- [PW02b] Lothar Pantel and Lars C. Wolf. On the suitability of dead reckoning schemes for Games. In *Proceedings of the 1st workshop on Network and system support for games*, pages 79–84. ACM Press, 2002.
- [Rag05] Ragnarok. RAGNAROK online, 2005. Disponível em <http://iro.ragnarokonline.com/>. Acesso em 02 de Maio de 2007.
- [RHC<sup>+</sup>02] Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, Oct–Dec 2002.
- [Rom03] Manuel Roman. *An Application Framework for Active Space Applications*. PhD thesis, University of Illinois at Urbana-Champaign, Illinois, USA., 2003.
- [Sat01] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, pages 10–17, aug 2001.

- [Sch97] D. Schmidt. *Pattern Languages of Program Design*, chapter Acceptor and Connector: Design Patterns for Initializing Communication Services. Addison-Wesley, 1997.
- [Sch02] Douglas C. Schmidt. Middleware for real-time and embedded systems. *Commun. ACM*, 45(6):43–48, 2002.
- [SDA98] D. Salber, A. Dey, and G. Abowd. Ubiquitous computing: Defining an HCI research agenda for an emerging interaction paradigm: Tech. report git-gvu-98-01. feb. (1998, 1998).
- [SE06] Sony Entertainment. Playstation portable – PSP, 2006. Disponível em <http://www.us.playstation.com/psp.aspx>. Acesso em 02 de Maio de 2007.
- [SER<sup>+</sup>04] Vagner Sacramento, Markus Endler, Hana K. Rubinsztein, Luciana S. Lima, Kleider Goncalves, Fernando N. Nascimento, and Giulliano A. Bueno. Moca: A middleware for developing collaborative applications for mobile users. *IEEE Distributed Systems Online*, 5(10):2, 2004.
- [SH02] Kaukoranta Smed and Hakonen. A review on networking and multiplayer computer games. Technical Report 454, Turku Centre for Computer Science, 2002.
- [SKH01] Jouni Smed, Timo Kaukoranta, and Harri Hakonen. Aspects of networking in multiplayer computer games. In Loo Wai Sing, Wan Hak Man, and Wong Wai, editors, *Proceedings of International Conference on Application and Development of Computer Games in the 21st Century*, pages 74–81, Hong Kong SAR, China, November 2001.
- [Sla03] Stuart Slater. Rapid application development of games for undergraduate and postgraduate projects using directx. In *GAME-ON*, pages 253–, 2003.
- [SM03] Debashis Saha and Amitava Mukherjee. Pervasive computing: A paradigm for the 21st century. *Computer*, 36(3):25–31, 2003.
- [SM05] Sun Microsystems. Java 2 Platform, Micro Edition, 2005. Disponível em <http://java.sun.com/j2me/>. Acesso em 02 de Maio de 2007.
- [Sof06] Softstack.com. Free SMTP Server, 2006. Disponível em <http://www.softstack.com/freesmtp.html>. Acesso em 02 de Maio de 2007.

- [SSR<sup>+</sup>04] Anees Shaikh, Sambit Sahu, Marcel Rosu, Michael Shea, and Debanjan Saha. Implementation of a service platform for online Games. In *Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04*, pages 106–110. ACM Press, 2004.
- [SSS03] Debanjan Saha, Sambit Sahu, and Anees Shaikh. A service platform for on-line Games. In *Proceedings of the 2nd workshop on Network and system support for games*, pages 180–184. ACM Press, 2003.
- [ST94] Bill Schilit and M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32, 1994.
- [Ste03] Steam. Counter-strike.net – the official web site, 2003. Counter-Strike.net, Inc. Disponível em <http://www.counter-strike.net/>. Acesso em 02 de Maio de 2007.
- [TdAdAL<sup>+</sup>03] Dante Gama Torres, Gustavo Danzi de Andrade, André Roberto Gouveia do Amaral Leitão, Igor de Andrade Lima Gatis, Pedro Henrique de Macêdo, and Geber Lisboa Ramalho. Uma API para a criação de jogos multi-usuário de turno em telefones móveis. In *Proceedings of the 2th Brazilian Workshop on Games and Digital Entertainment*, November 2003.
- [Tea] YDream Team. Undercover. Disponível em <http://hk.playundercover.com>. Acesso em 02 de Maio de 2007.
- [Ter02] Terazona: Zona application framework whitepaper, 2002. Zona Inc. <http://www.zona.net/whitepaper/Zonawhitepaper.pdf>. Acesso em 02 de Maio de 2007.
- [TFR05] Fernando Trinta, Carlos Ferraz, and Geber Ramalho. Serviços de middleware para jogos massivos ubíquos. In *Annais do IV SBGames, Simpósio Brasileiro de Jogos para Computador e Entretenimento Digital*. Universidade de São Paulo – USP, Novembro 2005.
- [TFR06a] Fernando Trinta, Carlos Ferraz, and Geber Ramalho. Middleware services for pervasive multiplatform networked games. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*. ACM Press, October 2006.
- [TFR06b] Fernando Trinta, Carlos Ferraz, and Geber Ramalho. Uma proposta de cenários e serviços de suporte para jogos multiusuário multiplataforma pervasivos. In

*Annais do XXII Webmedia, Simpósio Brasileiro de Sistemas Multimídia e Web.* Universidade Federal do Rio Grande do Norte, Novembro 2006.

- [TPFR07] Fernando Trinta, Davi Pedrosa, Carlos Ferraz, and Geber Ramalho. Scenarios and middleware services for pervasive multiplatform networked games (working in progress paper). In *Percom 2007 : V IEEE International Conference on Pervasive Computing and Communications*. University of Texas, IEEE Computer Society, March 2007.
- [TV05] Adwait Tumbde and Shreepadma Venugopalan. A Voronoi Partitioning Approach to Support Massively Multiplayer Online Games. Maio 2005. Disponível em <http://www.cs.wisc.edu/vshree/Voronoi.pdf>.
- [Wal04] Gordon Walton. Nine things to look for in the next generation of MMOG, 2004. Disponível em [http://www.gdconf.com/archives/2004/walton\\_gordon.ppt](http://www.gdconf.com/archives/2004/walton_gordon.ppt). Acesso em 02 de Maio de 2007.
- [WGB99] M. Weiser, R. Gold, and J. S. Brown. The Origins of Ubiquitous Computing Research at PARC in the Late 1980s. *IBM Syst. J.*, 38(4):693–696, 1999.
- [Woo06] Bruce Sterling Woodcock. An analysis of MMOG subscription growth, 12.0, June 2006. Disponível em <http://www.mmogchart.com/>. Acesso em 02 de Maio de 2007.
- [Yam04] Adenauer Corrêa Yamin. *Arquitetura para um Ambiente de Grade Computacional Direcionado às Aplicações Distribuídas, Móveis e Conscientes do Contexto da Computação Pervasiva*. PhD thesis, Universidade Federal do Rio Grande do Sul, Junho 2004.
- [Yan03] Jeff Yan. Security design in online games. In *Proceedings of the 19th Annual Computer Security Applications Conference*, page 286. IEEE Computer Society, 2003.
- [YC02] J. Yan and H-J Choi. Security issues in online games. In *The Electronic Library: international journal for the application of technology in information environments*, volume 20, 2002.

