



UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Daniel Henriques Moreira

***SIMPATROL*: UM SIMULADOR DE
SISTEMAS MULTIAGENTES PARA O
PATRULHAMENTO**

Recife – PE
2008



UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Daniel Henriques Moreira

***SIMPATROL*: UM SIMULADOR DE SISTEMAS
MULTIAGENTES PARA O PATRULHAMENTO**

Dissertação apresentada como
requisito parcial para obtenção
do grau de Mestre em Ciência da
Computação.

Orientador: Prof. Dr. Geber
Lisboa Ramalho

Recife – PE
2008

Moreira, Daniel Henriques
SimPatrol: um simulador de sistemas
multiagentes para o patrulhamento / Daniel
Henriques Moreira. – Recife: O Autor, 2008.
xvi, 101 folhas: fig., tab., quadros

Dissertação (mestrado) – Universidade Federal
de Pernambuco. Cln. Ciência da Computação, 2008.

Inclui bibliografia e apêndice.

1. Inteligência artificial. I. Título.

006.3 CDD (22. ed.) MEI2009 – 111

Dissertação de Mestrado apresentada por **Daniel Henriques Moreira** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Simpatrol: Um Simulador de Smas para o Patrulhamento**", orientada pelo **Prof. Geber Lisboa Ramalho** e aprovada pela Banca Examinadora formada pelos professores:



Prof. Patricia Cabral de Azevedo Restelli Tedesco
Centro de Informática / UFPE

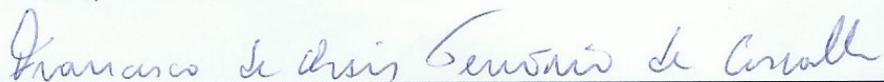


Prof. Augusto Cesar Pinto Loureiro da Costa
Departamento de Ciência da Computação / UFBA



Prof. Geber Lisboa Ramalho
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 4 de setembro de 2008.



Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO

Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

A meus pais, Tadeu e Márcia, por
todo o apoio de sempre.

AGRADECIMENTOS

Faço questão de agradecer:

Primeiramente a Deus, por sua assistência e patrulhamento impecáveis sobre mais esta fase da minha vida.

Aos meus familiares, pelo apoio e paciência incondicionais, especialmente aos meus pais e irmãos (que são o meu verdadeiro alicerce), e às minhas tias Sandra e Sônia (que sempre estiveram prontas a ajudar quando precisei).

Ao Centro de Informática (CIn) e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPQ), pela estrutura, suporte e financiamento, fundamentais para a concretização desta dissertação.

Aos professores Geber Ramalho (pela parceria e orientação), Patrícia Tedesco (pelas opiniões em artigos, pôsteres, etc.), Jacques Robin (pelas aulas de agentes inteligentes), e Liliane Salgado (pelo apoio constante e palavras motivadoras).

Ao amigo Luiz Josué, pela parceria na construção do simulador.

A todas as pessoas que ajudaram a amenizar a saudade da minha terra, em especial aos nortistas Antônio Jacob, Glauco Gonçalves, Luciana Costa, Rosana Cavalcante e Savana Carneiro (que fizeram de Recife um lugar mais próximo da Amazônia), e aos nordestinos Ariane Albuquerque, Diogo Carvalho, Gustavo Petry, João dos Prazeres, Juliana Miranda, Paulo Salgado e Weber Campos (por terem me acolhido tão bem nesta cidade que, sem sombra de dúvidas, aprendi a amar).

Aos colegas de trabalho (e hoje verdadeiros amigos) Bruno Barroso, Emanuel Estumano, Gustavo Carvalho, Hamilton Albuquerque, Leandro Pantoja e Raimundo Monteiro, pelo profissionalismo e cobertura nos momentos em que precisei estar ausente, e ao gerente Sandro Oliveira, pela sua compreensão.

Aos amigos de todo o sempre Igor Almeida e Frank Malcher (que mesmo à distância sempre se fizeram presentes).

E por fim, tenho agradecimentos em especial à Priscila Saboia, por todo o seu amor, sua amizade, parceria e companheirismo.

Muito obrigado a todos!

"Se fui capaz de ver mais longe,
foi apenas porque eu estava
apoiado sobre ombros de
gigantes."

Isaac Newton.

RESUMO

Apesar de sistemas multiagentes estarem se tornando cada vez mais comuns no desenvolvimento de *software*, há ainda que se chegar a um consenso sobre como modelar e comparar diferentes soluções de sistemas multiagentes para um mesmo problema. Em outras palavras, os pesquisadores de tal área ainda estão à procura de *benchmarks* que sirvam de referência para a comparação de abordagens distintas. Dada a importância do estabelecimento de *benchmarks* para sistemas multiagentes, uma questão que surge entre os pesquisadores da área de patrulhamento é a possibilidade e o potencial do problema do patrulhamento como um destes *benchmarks*. Apesar da quantidade de trabalho produzido até este ponto, tem-se encontrado dificuldades para comparar precisamente as soluções propostas para o patrulhamento. Estas dificuldades remetem principalmente à ausência de ferramentas (em especial um simulador unificado) que permitam aos estudiosos se concentrarem na solução do problema em si, e não na sua representação. Partindo deste cenário, este trabalho tem como objetivo discutir o estabelecimento do patrulhamento enquanto um *benchmark* para sistemas multiagentes. Adicionalmente, como um primeiro passo no sentido de preencher as lacunas que dificultam tal estabelecimento, introduz-se o *SimPatrol*, um novo simulador voltado para a tarefa de patrulhamento, fortemente inspirado pelos simuladores da *RoboCup* e TAC. Novos resultados são produzidos a partir de uma avaliação de parte das técnicas propostas pelas pesquisas anteriores frente às novas funcionalidades implementadas pelo simulador, em especial a inédita medida de ociosidade dos vértices, que leva em consideração o tempo de raciocínio gasto pelos patrulheiros.

PALAVRAS-CHAVES: sistemas multiagentes (SMA), patrulhamento multiagente, *benchmarks* para SMA, simuladores de SMA, *SimPatrol*.

ABSTRACT

Despite the fact that multi-agent systems are becoming increasingly common in software development, there is still need to reach a consensus on how to model and compare different multi-agent system solutions for the same problem. In other words, researchers of such area are still looking for benchmarks to serve as a reference for the comparison of different approaches. Given the importance of establishing benchmarks for multi-agent systems, an issue that arises to researchers of the patrolling field is the possibility and potential of the patrolling problem as one of these benchmarks. Despite the quantity of work produced up to this point, it is difficult to accurately compare the proposed solutions to the patrolling problem. These difficulties are mainly related to the lack of tools (in particular a unified simulator) to enable scholars to focus on resolving the problem itself, rather than represent it. Under this scenario, this paper aims to discuss the establishment of the patrolling problem as a benchmark for multi-agent systems. Additionally, as a first step towards filling gaps that hinder such establishment, it introduces *SimPatrol*, a new simulator dedicated to the task of patrolling, strongly inspired by the *RoboCup* and TAC simulators. New results are produced from the evaluation of some of the previously proposed patrolling techniques, when submitted to the new features implemented by the simulator, especially the unprecedented measurement of the idlenesses of the vertexes, taking into account the reasoning time spent by patrollers.

KEYWORDS: multi-agent systems (MAS), multi-agent patrolling, MAS benchmarks, MAS simulators, *SimPatrol*.

LISTA DE SIGLAS

BNF	<i>Backus-Naur form</i>
CC	<i>Cognitive coordinated (agents)</i>
CERN	Organização Européia de Pesquisa Nuclear
CIn	Centro de Informática da Universidade Federal de Pernambuco
CR	<i>Conscientious reactive (agents)</i>
DTD	<i>Document type definition</i>
GBLA	<i>Gray-box learner agents</i>
HPCC	<i>Heuristic pathfinder cognitive coordinated (agents)</i>
HPTB	<i>Heuristic pathfinder two-shots bidder (agents)</i>
HTTP	<i>Hypertext transfer protocol</i>
IA	Inteligência artificial
PP	Profundidade de percepção
RAM	<i>Random access memory</i>
SMA	Sistema multiagentes
SC	<i>Single cycle (patrolling strategy)</i>
TAC	<i>Trading agent competition</i>
TAC SCM	<i>Trading agent competition – supply chain management</i>
TCP/IP	<i>Transmission control protocol / internet protocol</i>
UDP/IP	<i>User datagram protocol / internet protocol</i>
UFPE	Universidade Federal de Pernambuco
XML	<i>Extensible markup language</i>

LISTA DE QUADROS

		Pág.
Quadro 2.1	Aspectos importantes sobre o <i>Soccerserver</i> .	13
Quadro 2.2	Algoritmo exercido pelo <i>Classic Server</i> durante um leilão.	16
Quadro 2.3	Aspectos importantes do <i>Classic Server</i> .	17
Quadro 3.1	Parâmetros do problema do patrulhamento relacionados ao seu território.	21
Quadro 3.2	Parâmetros do problema do patrulhamento relacionados aos patrulheiros.	22
Quadro 3.3	Parâmetros do problema do patrulhamento relacionados às sociedades de patrulheiros.	24
Quadro 3.4	Relações entre os domínios dos parâmetros do problema do patrulhamento com as categorias de ambientes de tarefa propostas por Russell e Norvig (2003).	26
Quadro 3.5	Resumo das contribuições dos trabalhos de Machado (2002), Almeida (2003), Santana (2004), Chevaleyre (2005) e Menezes (2006).	41
Quadro 3.6	Aspectos importantes do simulador de patrulhamento proposto por Machado (2002), Almeida (2003), Santana (2004) e Menezes (2005).	45
Quadro 4.1	Soluções para modelar o patrulhamento multiagente em respeito ao seu território.	50
Quadro 4.2	Cenários e soluções para modelar o patrulhamento multiagente em respeito às percepções dos agentes patrulheiros.	54
Quadro 4.3	Soluções para modelar o patrulhamento multiagente em respeito às percepções de seus agentes.	54

Quadro 4.4	Soluções para modelar o patrulhamento multiagente em respeito à comunicação entre os agentes patrulheiros.	58
Quadro 4.5	Soluções para modelar o patrulhamento multiagente em respeito às sociedades de patrulheiros.	60
Quadro 4.6	Cenários e soluções para modelar o patrulhamento multiagente em respeito à vitalidade dos agentes patrulheiros.	62
Quadro 4.7	Possíveis percepções e ações para um patrulheiro, com suas respectivas limitações.	63
Quadro 4.8	Requisitos não-funcionais e suas respectivas soluções.	65
Quadro 4.9	Exemplo de percepção sobre o grafo a ser patrulhado.	67
Quadro 4.10	Atores presentes no diagrama de casos de uso principal do <i>SimPatrol</i> .	69
Quadro 4.11	Detalhamento do conteúdo do pacote <i>model</i> .	70
Quadro 4.12	Detalhamento do conteúdo do pacote <i>control</i> .	71
Quadro 4.13	Detalhamento do conteúdo do pacote <i>view</i> .	73
Quadro 4.14	Detalhamento do conteúdo do pacote <i>util</i> .	74
Quadro 4.15	Aspectos importantes sobre o <i>SimPatrol</i> .	75

LISTA DE TABELAS

		Pág.
Tabela 4.1	Ociosidades médias coletadas em decorrência do patrulhamento exercido por cinco agentes do tipo HPCC, sobre os mapas do conjunto proposto por Santana (2004).	80
Tabela 4.2	Ociosidades médias, medidas em segundos, coletadas em decorrência do patrulhamento exercido por cinco agentes dos tipos CR, CC, HPCC e CS, sobre os mapas do conjunto proposto por Santana (2004).	85
Tabela 4.3	Ociosidades máximas, medidas em segundos, coletadas em decorrência do patrulhamento exercido por cinco agentes dos tipos CR, CC, HPCC e CS, sobre os mapas do conjunto proposto por Santana (2004).	86

LISTA DE FIGURAS

		Pág.
Figura 2.1	Interface gráfica com o usuário de uma instância do <i>Soccermonitor</i> .	12
Figura 3.1	Exemplo de esqueletização (Machado 2002).	27
Figura 3.2	Mapas <i>A</i> (a) e <i>B</i> (b) propostos por Machado (2002).	33
Figura 3.3	Um exemplo de grafo para patrulhamento.	35
Figura 3.4	Modelos de territórios propostos por Santana (2004). Em (a) tem-se o mapa <i>A</i> , (b) o mapa <i>B</i> , (c) o mapa <i>Circle</i> , (d) o mapa <i>Corridor</i> , (e) o mapa <i>Islands</i> , e (f) o mapa <i>Grid</i> .	37
Figura 3.5	Agentes patrulhando segundo a estratégia do ciclo único.	38
Figura 4.1.	Modelo de dados para a simulação de território.	51
Figura 4.2	Modelo de dados para a simulação de tempo.	52
Figura 4.3	Simulação de tempo em ciclos.	53
Figura 4.4	Modelo de dados para a simulação de percepções.	54
Figura 4.5	Percepção do grafo (a), e percepção dos outros agentes (b), ambas com profundidade igual a um (1). O agente <i>perceptor</i> está no centro do território.	55
Figura 4.6	Modelo de dados para o controle de percepções permitidas e suas respectivas limitações.	55
Figura 4.7	Modelo de dados para a simulação de ações.	56
Figura 4.8	Modelo de dados para o controle de ações permitidas e suas respectivas limitações.	57
Figura 4.9	Modelo de dados para a simulação de comunicações.	59
Figura 4.10	Modelo de dados para a simulação de sociedades.	60
Figura 4.11	Modelo de dados para a simulação da vitalidade dos agentes.	62

Figura 4.12	Modelo de dados para a coleta de métricas.	64
Figura 4.13	Classes de objetos <i>XMLáveis</i> .	67
Figura 4.14	Exemplo de grafo percebido por um agente.	67
Figura 4.15	Diagrama de casos e uso principal do <i>SimPatrol</i> .	68
Figura 4.16	Diagrama de pacotes do <i>SimPatrol</i> .	69
Figura 4.17	Modelo de dados das medidas de configuração de uma simulação.	72
Figura 4.18	Interface gráfica de usuário do <i>SimPatrol</i> .	73
Figura 5.1	Configuração de <i>hardware</i> utilizada nos experimentos.	79
Figura 5.2	Médias das ociosidades médias de cada mapa do conjunto proposto por Santana (2004), patrulhados por cinco agentes do tipo HPCC. Adaptado de Santana (2004).	80
Figura 5.3	Convergências das ociosidades médias em experimentos com as <i>configurações 01</i> (de cinco agentes do tipo HPCC), para cada mapa do conjunto proposto por Santana (2004).	81
Figura 5.4	Médias das ociosidades médias de cada mapa do conjunto proposto por Santana (2004), patrulhados por cinco agentes do tipo HPCC.	82
Figura 5.5	Ociosidades médias, medidas em segundos, coletadas em decorrência do patrulhamento exercido por cinco agentes dos tipos CR, CC, HPCC e SC, sobre os mapas do conjunto proposto por Santana (2004).	86
Figura 5.6	Ociosidades máximas, medidas em segundos, coletadas em decorrência do patrulhamento exercido por cinco agentes dos tipos CR, CC, HPCC e SC, sobre os mapas do conjunto proposto por Santana (2004).	87
Figura 5.7	Ociosidades médias, medidas em ciclos, coletadas em decorrência do patrulhamento exercido por cinco agentes dos tipos CR, CC, HPCC e SC, sobre os mapas do conjunto proposto por Santana (2004).	87

LISTA DE EQUAÇÕES

		Pág.
Equação 3.1	Ociosidade instantânea média do grafo.	29
Equação 3.2	Ociosidade instantânea máxima do grafo.	29
Equação 3.3	Ociosidade média do grafo.	29
Equação 3.4	Ociosidade máxima do grafo.	29
Equação 3.5	Normalização de um valor coletado para uma métrica.	30

SUMÁRIO

Capítulo 1. Introdução.....	01
Capítulo 2. <i>Benchmarks</i> para Sistemas Multiagentes.....	04
2.1. <i>Benchmarks</i>	04
2.1.1. <i>Definição</i>	05
2.1.2. <i>Bons Benchmarks</i>	07
2.2. Estado-da-arte de <i>Benchmarks</i> para Sistemas Multiagentes.....	08
2.2.1. <i>RoboCup</i>	09
<i>Soccerserver</i>	10
2.2.1. <i>Trading Agent Competition</i>	13
<i>TAC Classic e o Classic Server</i>	14
2.3. Considerações Finais.....	18
Capítulo 3. Patrulhamento.....	20
3.1. O Problema do Patrulhamento.....	20
3.1.1. <i>Parâmetros do problema do patrulhamento</i>	21
3.1.1.1. <i>Parâmetros Relacionados ao Território</i>	21
3.1.1.2. <i>Parâmetros Relacionados aos Patrulheiros</i>	22
3.1.1.3. <i>Parâmetros Relacionados às Sociedades de Patrulheiros</i>	24
3.2. Patrulhamento Multiagente.....	25
3.2.1. <i>Métricas do Patrulhamento Multiagente</i>	28
3.3. Estado-da-arte do Patrulhamento Multiagente.....	31
3.3.1. <i>Agentes Pioneiros</i>	31
3.3.2. <i>Agentes Heurísticos</i>	33
3.3.3. <i>Agentes Aprendizes por Reforço</i>	35
3.3.4. <i>Agentes Baseados no Caixeiro Viajante</i>	37
3.3.5. <i>Agentes Negociantes</i>	38
3.3.6. <i>Críticas aos Trabalhos Anteriores</i>	40
3.4. Pode o Patrulhamento ser um <i>Benchmark</i> para Sistemas Multiagentes?.....	43
3.5. Considerações Finais.....	46

Capítulo 4. <i>SimPatrol</i>: Requisitos e Arquitetura.....	48
4.1. Requisitos Atendidos pelo <i>SimPatrol</i>.....	48
4.1.1. Requisitos Funcionais.....	49
4.1.1.1. <i>Requisitos de Simulação de Território.....</i>	50
4.1.1.2. <i>Requisitos de Simulação de Tempo.....</i>	51
4.1.1.3. <i>Requisitos de Simulação de Percepções.....</i>	53
4.1.1.4. <i>Requisitos de Simulação de Ações.....</i>	56
4.1.1.5. <i>Requisitos de Simulação de Comunicações.....</i>	58
4.1.1.6. <i>Requisitos de Simulação de Sociedades.....</i>	60
4.1.1.7. <i>Requisitos de Simulação da Vitalidade dos Agentes.....</i>	61
4.1.1.8. <i>Requisitos de Coleta de Métricas.....</i>	63
4.1.2. Requisitos Não-funcionais.....	64
4.2. Arquitetura.....	68
4.2.1. <i>Pacote model.....</i>	70
4.2.2. <i>Pacote control.....</i>	71
4.2.3. <i>Pacote view.....</i>	73
4.2.4. <i>Pacote util.....</i>	74
4.3. Considerações Finais.....	74
Capítulo 5. Experimentos e Resultados.....	77
5.1. <i>Hardware</i> Utilizado nos Experimentos.....	77
5.2. Reprodução de Resultados do Simulador Anterior.....	79
5.2.1. <i>Discussão sobre a Reprodução de Resultados do Simulador Anterior.....</i>	82
5.3. Resultados Considerando o Tempo de Raciocínio dos Agentes.....	84
5.3.1. <i>Discussão sobre os Novos Resultados.....</i>	87
5.4. Considerações Finais.....	89
Capítulo 6. Conclusões e Trabalhos Futuros.....	90
6.1. Contribuições para o Patrulhamento.....	90
6.2. Trabalhos Futuros.....	91
Apêndice A. DTD de objetos XMLáveis no <i>SimPatrol</i>.....	94
Referências Bibliográficas.....	98

Capítulo 1

Introdução

Como apontado por Hanks, Pollack e Cohen (1993), a Inteligência Artificial (IA) passou a se concentrar menos em tecnologias de componentes isolados, e mais em sistemas completos e integrados. Neste cenário, sistemas multiagentes (SMA) oferecem uma boa alternativa para a concepção de tais sistemas integrados. Isto porque, segundo Jennings e Wooldridge (1998), modelar estes sistemas através do conceito de *agentes* autônomos, percebendo e agindo sobre um *ambiente*, permite um alto nível de abstração e independência entre as suas partes integrantes, graças à decomposição dos problemas a serem resolvidos e à atribuição dos subproblemas resultantes aos agentes.

Ademais, conforme Russell e Norvig (2003), a adoção do paradigma de agentes não só tem possibilitado, mas também tem sistematizado a combinação de técnicas pertencentes a diversas subáreas da IA anteriormente isoladas (IA simbólica, IA conexionista, IA evolucionária, etc.), no sentido de produzir sistemas inteligentes.

Assim, por conta desta simplificação e sistematização do projeto e implementação de sistemas complexos (em geral concorrentes e distribuídos), SMA têm se tornado cada vez mais comuns no desenvolvimento de *software* (Hübner, Bordini e Vieira, 2004). Todavia, apesar desta popularidade crescente, há ainda que se chegar a um consenso sobre como modelar e comparar diferentes soluções de SMA para um mesmo problema (Hanks, Pollack e Cohen, 1993). Em outras palavras, os pesquisadores da área de SMA ainda estão à procura de *benchmarks* que sirvam de referência para a comparação de abordagens distintas.

Em termos gerais, *benchmarks* são problemas precisamente descritos, passíveis de resolução por diversas técnicas, e representantes de uma classe de problemas. Hanks, Pollack e Cohen (1993) evidenciaram a importância de se estabelecer *benchmarks* para SMA, assim como Stone (2002), e Drogoul, Landau e Muñoz (2007) criticaram os *benchmarks* para SMA pertencentes às competições de computação mais populares da atualidade: *RoboCup* e TAC.

Segundo Stone (2002), e Drogoul, Landau e Muñoz (2007), ainda que *RoboCup* e TAC tenham atraído nos últimos anos um número considerável de competidores, e tenham motivado pesquisas importantes dentro da área de SMA, elas possuem desvantagens que estão relacionadas principalmente ao seu aspecto de disputa. Esta característica de competição acaba por fomentar problemas como a obsessão por vencer, barreiras à entrada de novos competidores, eventuais vantagens financeiras de alguns participantes sobre os demais, etc. No caso específico da *RoboCup*, questiona-se ainda a representatividade de parte dos problemas que seus competidores têm resolvido (Drogoul, Landau e Muñoz, 2007).

De posse desta discussão sobre o estabelecimento e a qualidade de *benchmarks* para SMA, uma questão que surge entre os pesquisadores da área de *patrulhamento* é a possibilidade e o potencial do problema do patrulhamento como um destes *benchmarks*.

Informalmente, patrulhamento é o ato de se deslocar no espaço com o intuito de visitar regularmente regiões de interesse. O objetivo é minimizar o tempo entre duas visitas a um mesmo lugar.

Em respeito à sua utilidade, o patrulhamento pode ser aplicado em domínios de problemas onde supervisão, inspeção e controle são necessários, a exemplo de tarefas de vigilância, cobertura de espaço e navegação (Machado, 2002). Por tal relevância, diversas arquiteturas de agentes e mecanismos deliberativos já foram aplicados ao problema do patrulhamento. Técnicas com base em comportamentos reativos (Machado, 2002), agentes heurísticos (Almeida, 2003), aprendizagem por reforço (Santana, 2004), teoria dos grafos (Chevaleyre, 2005) e mecanismos de negociação (Menezes, 2006) foram desenvolvidas.

Todavia, apesar da quantidade de trabalho produzido até este ponto, pesquisadores têm encontrado dificuldades para comparar precisamente as soluções propostas para o patrulhamento (Moreira et al., 2007). Estas dificuldades remetem principalmente à ausência de ferramentas (em especial um simulador unificado) que permitam aos estudiosos se concentrarem na solução do problema em si, e não na sua representação.

Partindo deste cenário, este trabalho tem como objetivo discutir o estabelecimento do patrulhamento como um *benchmark* para SMA. Adicionalmente, como um primeiro passo no sentido de preencher as lacunas que dificultam tal estabelecimento, introduz-se o *SimPatrol*, um novo simulador de SMA voltado para a

tarefa de patrulhamento, fortemente inspirado pelos simuladores da *RoboCup* e TAC.

De modo a se alcançar tal objetivo, há uma série de metas específicas:

- 1) Discutir o estabelecimento de *benchmarks* para SMA e resumir o seu estado-da-arte;
- 2) Apresentar o problema do patrulhamento (destacando-se a identificação de seus parâmetros mais relevantes) e resumir o seu estado-da-arte;
- 3) Avaliar a qualidade do problema do patrulhamento enquanto um *benchmark* para SMA;
- 4) Introduzir o *SimPatrol* enquanto um novo simulador de SMA voltado para o patrulhamento, destacando-se as novas funcionalidades implementadas, quando comparado às versões do simulador de patrulhamento anteriores (Machado, 2002; Almeida, 2003; Santana, 2004; Menezes, 2006);
- 5) Descrever experimentos e apresentar resultados de patrulhamento obtidos com o *SimPatrol*.

De posse destes objetivos, esta dissertação procura alcançá-los da seguinte maneira: o capítulo 2 discute o estabelecimento de *benchmarks* para SMA, evidenciando as características que tornam um *benchmark* um “bom *benchmark*” para a IA. Na seqüência, o capítulo 3 define o problema de patrulhamento, apresentando em seguida um resumo das contribuições de Machado (2002), Almeida (2003), Santana (2004), Chevaleyre (2005) e Menezes (2006), que constituem o atual estado-da-arte desta área. Neste mesmo capítulo, discutem-se a possibilidade e o potencial do problema do patrulhamento como um *benchmark* para SMA.

O capítulo 4, por sua vez, introduz o *SimPatrol*, evidenciando os requisitos atendidos e revelando a arquitetura adotada, além de destacar as novas funcionalidades implementadas. Em seguida, o capítulo 5 busca a validação do simulador, efetuando comparações com as versões do simulador de patrulhamento utilizadas nos trabalhos de Machado (2002), Almeida (2003), Santana (2004) e Menezes (2006), sobretudo quanto à reprodução dos resultados obtidos. Além disso, novos resultados são produzidos a partir de uma avaliação das técnicas propostas pelas pesquisas anteriores frente às novas funcionalidades implementadas. Finalmente, o capítulo 6 realiza algumas considerações finais, no sentido de se destacar as contribuições desta dissertação, bem como os trabalhos futuros.

Capítulo 2

Benchmarks para Sistemas Multiagentes

De modo a se embasar a proposta do problema do patrulhamento como um *benchmark* para SMA, faz-se necessário, primeiramente, abordar o movimento de estabelecimento de *benchmarks* para a área de SMA (apontado por Hanks, Pollack e Cohen, 1993).

Portanto, este capítulo tem como metas definir o que são *benchmarks* para SMA, apontar as características que tornam qualquer *benchmark* um *bom benchmark*, e resumir o estado-da-arte desta área, através de uma análise das competições *RoboCup* e TAC – apontadas por Stone (2002), e Drogoul, Landau e Muñoz (2007) como reunidoras dos *benchmarks* mais populares da atualidade.

Para se alcançar tais objetivos, o texto a seguir está organizado da seguinte maneira: a seção 2.1 define o que são *benchmarks* (em especial do ponto de vista da IA e dos SMA) e quais características os tornam bons *benchmarks*. Na seção seguinte (2.2), o estado-da-arte de *benchmarks* para SMA é apresentado através de um resumo das competições *RoboCup* e TAC, mais uma análise dos seus respectivos simuladores. Na seqüência, a seção 2.3 encerra o assunto, com algumas considerações finais.

2.1. *Benchmarks*

Como apontado por Drogoul, Landau e Muñoz (2007), a Ciência da Computação herdou do processo de industrialização a necessidade de se avaliar a performance de sistemas lógicos distintos (algoritmos, arquiteturas, etc.) aplicados a um mesmo problema p , de modo a se identificar quais apresentam os melhores resultados, e especificamente em que situações. Informalmente, tal problema p pode ser encarado como um *benchmark*.

Como dito por Hanks, Pollack e Cohen (1993), *benchmarks* são sobretudo problemas passíveis de uma descrição precisa e rigorosa que simplificam uma realidade mais complexa e sofisticada. O primeiro objetivo desta simplificação é

possibilitar comparações justas entre soluções distintas, dado que ela facilita a submissão de tais soluções às mesmas situações controladas.

Em respeito à IA, tendo-se em vista o fato de que ela é a *ciência do artificial* (Pereira, 2001), os *benchmarks* desempenham um papel importante que excede a questão da comparação de sistemas concorrentes. Isto se dá porque, além de uma veia teórico-analítica, a IA possui uma faceta empírica importante, na qual a experimentação controlada é fundamental. Isto torna os *benchmarks*, para tal ciência, peças chaves do seu processo científico.

Drogoul, Landau e Muñoz (2007) também evidenciam esta questão ao afirmarem que, antes da aplicação em problemas reais, é comum – dentro da IA – testar-se os métodos, algoritmos ou técnicas desenvolvidas em instâncias dos problemas simplificadas e rigorosamente definidas (i.e. *benchmarks*), de modos que seja possível vários pesquisadores resolvê-las com seus próprios sistemas.

Assim, o processo experimental consiste-se de o pesquisador variando características do ambiente simulado, ou até mesmo da tarefa de *benchmark*, com o intuito de medir os efeitos de tais mudanças sobre o desempenho do sistema que ele está propondo. Com estas informações, ele deve ser capaz de identificar fenômenos isolados irrelevantes e resultados genéricos importantes, explicando adequadamente o porquê de sua solução se comportar da maneira observada.

Na seqüência, definições mais formais para *benchmarks* são apresentadas, sobretudo do ponto de vista da Ciência da Computação, da IA e dos SMA.

2.1.1. Definição

Para a Ciência da Computação, "um *benchmark* é uma aplicação ou problema dedicado à avaliação das performances de sistemas" (Drogoul, Landau e Muñoz, 2007). No caso específico da IA, *benchmark* é todo problema:

- *Suficientemente genérico*, a ponto de poder ser resolvido por várias técnicas diferentes;
- *Suficientemente específico*, a ponto de permitir que tais técnicas sejam comparadas entre si;

- *Suficientemente representativo* de uma classe de problemas reais interessantes.

Além disso, deve-se garantir que os resultados da aplicação de uma determinada técnica a um *benchmark* possam ser medidos de maneira determinística e permitam se estabelecer uma ordem total entre os custos das aplicações de várias técnicas.

Exemplos de *benchmarks* para a IA já populares no meio científico são o *problema da mochila* (Martello e Toth, 1990), o *problema das n-rainhas* (Russell e Norvig, 2003), o *problema do caixeiro viajante* (Cook, 2008), entre outros. Tratados por Drogoul, Landau e Muñoz (2007) como *benchmarks clássicos da IA*, estes problemas destacam-se por serem suficientemente genéricos – característica comprovada pela quantidade de técnicas já desenvolvidas e aplicadas nas suas resoluções – suficientemente específicos, comprovado pela presença na literatura de estudos comparativos destas técnicas (Martello e Toth, 1990; Russell e Norvig, 2003; Cook, 2008) e suficientemente representativos de uma classe de problemas interessantes: os *problemas NP-completos* (Garey e Johnson, 1979). Complementarmente, permitem o cálculo do custo de um método de resolução (em termos de números de movimentos, ou complexidade algorítmica, ou quantidade de memória necessária) de maneira independente da tecnologia utilizada, para que então se ordene as várias técnicas segundo seus desempenhos.

Contudo, conforme Hanks, Pollack e Cohen (1993), à medida que a IA tem se focado menos em componentes tecnológicos e mais em sistemas completos e integrados (em especial SMA), estes *benchmarks* tradicionais têm revelado suas limitações. Drogoul, Landau e Muñoz (2007) afirmam que estas limitações residem no fato de estes *benchmarks* clássicos preocuparem-se em avaliar a performance de agentes lógicos funcionando individualmente, característica que não faz sentido para SMA.

Nestes termos, que características deve apresentar, afinal, um *benchmark* para SMA? Da mesma forma que os demais *benchmarks* para a IA, ele deve ser suficientemente genérico e suficientemente específico. A particularidade de *benchmarks* para SMA reside justamente na questão da representatividade de uma *classe de problemas*. SMA diferem de outros sistemas porque a complexidade de seus problemas vai além da *teoria da NP-completude* (Garey e Johnson, 1979). Como citado por Stone (2002), esta complexidade abrange outros elementos como

colaboração, coordenação dos agentes, aquisição de conhecimento, raciocínio e planejamento em tempo real, combinação de sensores, etc., o que confere à sua representatividade uma participação (e contribuições) em cada um destes conceitos.

Dois *benchmarks* propostos para avaliar o desempenho se SMA e já bastante estudados na literatura são o *problema da presa-predador* (Stephens e Merx, 1990; Levy e Rosenschein, 1992), e o *problema da colheita* (Steels, 1989; Goss et al., 1990). Entretanto, a quantidade de parâmetros que se pode variar em cada um destes *benchmarks* é muito grande. Focando-se no caso do problema da presa-predador – por exemplo – podem-se variar, entre outras coisas, a velocidade das presas e predadores, o raio de percepção dos agentes, a abrangência do seu mecanismo de comunicação, etc. Como verificado por Drogoul, Landau e Muñoz (2007), o resultado desta situação é não ser incomum duas ou mais técnicas aplicadas a um mesmo *benchmark* não resolverem de fato o mesmo problema, por terem relaxado (ou enrijecido) em suas soluções parâmetros diferentes. Esta questão acaba por prejudicar a qualidade destes *benchmarks*.

E em se falando da qualidade de *benchmarks*, o que caracteriza um *bom benchmark*?

2.1.2. Bons Benchmarks

Segundo Drogoul, Landau e Muñoz (2007) um bom *benchmark* não só deve facilitar a implementação de novas técnicas, como também deve permitir aos pesquisadores concentrarem-se mais na solução do que na representação do problema (não lhes cabendo, inclusive, controlar a maneira como os parâmetros de entrada do mesmo estão sendo relaxados durante uma eventual simulação). Neste sentido, Hanks, Pollack e Cohen (1993) introduzem o conceito de *testbeds*.

Testbeds são ambientes de *software* completos que, além de oferecer uma interface de configuração para os parâmetros de um *benchmark*, asseguram que diferentes técnicas sejam testadas em iguais condições e avaliadas da mesma forma. Em outras palavras, eles permitem o controle da forma como a realidade está sendo simplificada através do *benchmark* a que dão suporte. Hanks, Pollack e Cohen (1993) citam vários *testbeds* que dão suporte ao problema da presa-predador

e ao problema da colheita. Todavia, frente à variedade destes *testbeds*, que não contam com simuladores unificados que reúnam todas as estratégias já desenvolvidas e as compare de uma maneira justa (submetendo-as, por exemplo, a um mesmo gerador de números aleatórios, ou à mesma contagem de tempo, ou aos mesmos valores para os parâmetros do problema), estes *benchmarks* têm perdido espaço para torneios unificados como a *RoboCup* e a TAC.

Na seqüência, o estado-da-arte de *benchmarks* para SMA é representado justamente por estas competições.

2.2. Estado-da-arte de *Benchmarks* para Sistemas Multiagentes

Segundo Stone (2002), tanto *RoboCup* quanto TAC têm atraído nos últimos anos um número considerável de competidores e têm motivado pesquisas importantes dentro da área de SMA. Isto se dá porque reúnem *benchmarks* populares (como é o caso da *RoboCup*, cujo tema central é desenvolver estratégias para vencer uma partida de futebol, um dos esportes mais populares do mundo), e instigantes (por concentrarem, em uma época do ano, verdadeiros torneios para confrontar técnicas distintas).

No caso específico deste trabalho, o interesse nestas competições reside em seus respectivos simuladores. Simuladores estes que, ao longo dos seus anos de existência, têm assegurado avaliações confiáveis de várias técnicas submetidas às mesmas condições, com liberdade para codificação de soluções nas linguagens de programação mais populares do mercado e meio científico (Tiobe Company, 2008).

Em suma, totalmente à revelia dos pontos negativos apontados por Drogoul, Landau e Muñoz (2007) e por Stone (2002) para estes torneios (a exemplo da obsessão dos competidores por ganhar, ou de barreiras para entrar, ou das vantagens que um competidor com mais recursos financeiros possa apresentar em relação aos demais), o estado-da-arte para *benchmarks* em SMA será representado nesta dissertação por estas duas competições. Não só porque elas são as mais populares da área em questão, mas também porque seus simuladores podem apontar requisitos importantes a serem atendidos pelo *SimPatrol*.

2.2.1. *RoboCup*

A *RoboCup* é uma iniciativa conjunta internacional que tem o objetivo de promover a IA, a robótica e campos relacionados, por meio de *benchmarks* que, além de permitir a integração e comparação de diversas tecnologias, sejam populares e desafiadores o suficiente para estimular, a cada ano, a participação de vários pesquisadores de todas as partes do mundo.

Tais *benchmarks* são concretizados em *ligas de futebol*, organizadas em *simulação* (onde *agentes virtuais* são submetidos a um *software* simulador que controla as suas percepções e assegura o resultado de suas ações), e em torneios distintos de *robôs* separados por tipo e tamanho dos seus *hardwares*. Há ainda *ligas de resgate* (em uma tentativa de aumentar a contribuição social do projeto), sendo uma de *simulação* – mais uma vez com agentes virtuais submetidos a um *software* simulador – e uma de *robôs*.

Todavia, como atestado pelo próprio *site* do evento na *internet* (Robocup, 2008), o tópico principal de pesquisa do projeto gira em torno do *problema de se desenvolver agentes capazes de vencer uma partida de futebol*. Chen et al. (2002) reforçam esta questão ao afirmarem que "o corpo principal da *RoboCup* se consiste de várias ligas de futebol". Tanto isto é verdade, que o mote do projeto é, no ano de dois mil e cinquenta (2050), apresentar um time de robôs humanóides completamente autônomos que vençam o time humano campeão mundial de futebol.

Sobre este desafio, Chen et al. (2002) asseguram que, mesmo não sendo eventualmente alcançado, pesquisas importantes nos campos de *agentes autônomos*, *colaboração multiagente*, *desenvolvimento de estratégias*, *raciocínio em tempo real* e *combinação de sensores* terão sido realizadas. Qualquer pessoa interessada em conhecer estes avanços pode obter as bibliotecas dos trabalhos participantes de edições anteriores no *site* do projeto (Robocup, 2008).

Em respeito às várias ligas existentes, merece destaque nesta dissertação a de *simulação de futebol*, onde agentes de *software* movendo-se independentemente jogam uma partida de futebol em um campo virtual criado em um computador. O interesse nesta liga reside justamente no *testbed* por ela consolidado, que conta

com um sistema de simulação física de futebol implementado em um simulador de nome *Soccerserver* (Chen et al., 2002).

Soccerserver

O *Soccerserver* é um *software* que permite a agentes autônomos organizados em dois times disputarem uma partida de futebol. Esta partida, por sua vez, é executada em um formato *cliente-servidor*. O *servidor* – que é o próprio *Soccerserver* – provê um campo virtual, simula a dinâmica de movimentação da bola e dos jogadores, e assegura o andamento do jogo de acordo com as regras do futebol. Ademais, ele se comunica com os *clientes* via portas *UDP/IP* (Kurose e Rossi, 2004). Dado que até onze (11) jogadores podem se juntar a cada um dos dois times, resultando em um total de vinte e dois (22) agentes autônomos compartilhando o mesmo campo, o *ambiente de tarefas* (Russell e Norvig, 2003) simulado pelo *Soccerserver* acaba por se tratar de um ambiente multiagente bastante dinâmico.

Sobre os clientes, a idéia é que cada um deles seja um processo em separado que se conecta ao servidor através de uma porta específica, e controla as ações de um jogador. Em termos práticos, por meio desta porta *UDP/IP* designada, o cliente envia comandos de controle para o seu jogador, e recebe do *Soccerserver* informações sensoriais. Em outras palavras, o programa cliente funciona como o cérebro do jogador: ele recebe sensações visuais e de tato do servidor, e envia comandos de controle, constituídos por requisições de execução de ações específicas (e.g. *chutar a bola*, *virar-se*, *correr*, etc.). O servidor então recebe estas mensagens, atende às requisições e atualiza o ambiente. Adicionalmente, ele provê a todos os jogadores informações sensoriais (e.g. dados visuais sobre a posição da bola, traves e outros jogadores).

Sobre o *Soccerserver*, é importante salientar que ele se trata de um sistema de tempo real funcionando em intervalos de tempo discretos (*ciclos*). Cada ciclo tem uma mesma duração específica, e as ações que precisam ser executadas em um determinado ciclo devem chegar ao servidor durante o intervalo de tempo correto. Do contrário, um cliente eventualmente atrasado perderá a oportunidade de atuar decisivamente para seu time. Por este motivo, os mecanismos de raciocínio

implementados nos clientes devem apresentar performances aceitáveis, sob pena de custarem a vitória da partida.

Internamente, o *Soccerserver* controla o andamento do jogo baseando-se em consultas e atualizações de vários *modelos*. Há *modelos de ações*, *percepções*, *comunicações*, *colisões*, *stamina dos agentes*, etc. O modelo de ações, por exemplo, é constituído de um conjunto pré-definido de ações – como *catch* (apreender a bola), *kick* (chutar), *turn* (virar-se) e *run* (correr) – além de ser *ruidoso*, o que significa que nem sempre a requisição de ação feita por um jogador, após ser atendida pelo servidor, apresentará os resultados esperados – i.e. o ambiente é não-determinístico.

Semelhantemente, o modelo de percepções conta com um repertório de percepções, a exemplo do *tato* e da *visão* (com uma *taxa de atualização* e *ângulo de percepção* bem definidos, implicando que o ambiente não é completamente observável). Da mesma forma que o modelo de ações, este modelo é *ruidoso*, significando que nem sempre os jogadores têm uma percepção correta do campo e dos demais jogadores (mais uma vez reforçando o fato do ambiente ser não-determinístico).

Sobre o modelo de comunicações, este se dá baseado em protocolos do tipo *fale-e-ouça* (*say-and-hear*) e com um alcance bem determinado, a partir do agente que fala. Há ainda o modelo de colisões, que ajuda o servidor a detectar as colisões entre os objetos, e atualizar de uma forma fisicamente plausível os demais modelos internos. Por fim, o modelo de *stamina* prevê, para cada agente, um comportamento de perdas energéticas referentes a cada ação, assim como também um fenômeno de recuperação energética a cada ciclo da partida.

Uma questão importante sobre os modelos internos reside no fato de eles não estarem voltados à visualização gráfica da partida de futebol de forma alguma. Em suma, não cabe ao *Soccerserver* preocupar-se em exibir o jogo, plotar imagens dos jogadores ou realizar cálculos geométricos de como desenhar o campo. Para esta tarefa existe um segundo aplicativo – o *Soccermonitor* – que também se conecta ao *Soccerserver* via porta UDP/IP e dele recebe informações suficientes para uma representação gráfica da partida em tempo de simulação (ver figura 2.1), além de *replays* posteriores ao seu término. Ademais, várias instâncias do *Soccermonitor* podem se conectar ao *Soccerserver*, possibilitando diversas visualizações simultâneas de um mesmo jogo.

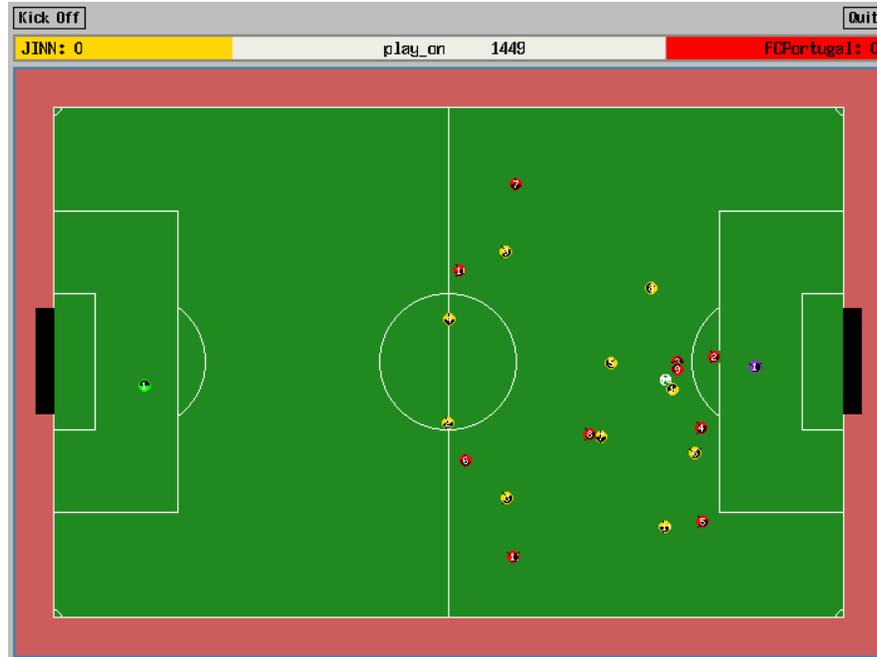


Figura 2.1 – Interface gráfica com o usuário de uma instância do *Soccermonitor*.

De posse deste entendimento superficial sobre os modelos internos do *Soccerserver*, pode-se perceber a maior vantagem decorrente das abstrações por eles asseguradas: pesquisadores que estejam desenvolvendo a inteligência dos jogadores não precisam se preocupar com problemas de *movimentação robótica*, *reconhecimento de objetos*, *tecnologias de comunicação* ou mesmo aspectos de *hardware*. Desta forma, eles podem focar em conceitos de mais alto-nível, como *raciocínio automático*, *cooperação entre os agentes*, *coordenação* e *aprendizagem*.

Adicionalmente, dado que uma partida é executada segundo o paradigma cliente-servidor, não há restrições ao modo como os times são desenvolvidos. O único requisito existente diz respeito ao fato de os jogadores serem capazes de estabelecer conexões UDP/IP com o servidor. Nestes termos, os programas podem ser escritos em qualquer linguagem, desde que enviem às portas UDP/IP corretas as requisições no formato esperado. Chen et al. (2002) apresentam a forma normal de Backus (BNF, *Backus-Naur Form*) para as mensagens trocadas entre o *Soccerserver* e seus clientes.

O quadro 2.1 resume alguns aspectos importantes sobre o *Soccerserver*.

Aspecto	Valor
Problema a ser resolvido pelos agentes	Ganhar uma partida de futebol
Arquitetura do <i>software</i>	Cliente-servidor (o <i>Soccerserver</i> é o servidor, e os agentes são os clientes)
Visualização gráfica da simulação	Cliente UDP/IP <i>Soccermonitor</i> , que interpreta mensagens enviadas pelo servidor e reproduz a simulação
Interface de comunicação do ambiente com os agentes	Portas UDP/IP
Linguagem de programação dos agentes	Qualquer uma, desde que haja suporte à troca de mensagens via portas UDP/IP
Quantidade de sociedades de agentes	Duas (2), já que há dois times (i.e. ambiente multi-social)
Quantidade de agentes	Ambiente multiagente cooperativo (entre agentes de um mesmo time) e competitivo (entre agentes de times diferentes)
Sincronismo entre percepções e ações	Ambiente assíncrono (i.e. percepções são regularmente fornecidas aos agentes de maneira independente ao atendimento de suas requisições por ações)
Confiabilidade das percepções	Percepções ruidosas
Confiabilidade das ações	Ações ruidosas
Observabilidade do ambiente	Ambiente parcialmente observável (graças às percepções limitadas e ruidosas)
Tipo de comunicação entre os agentes	Baseada em protocolos <i>fale-e-ouça</i> e com limitações de alcance
Confiabilidade da comunicação entre agentes	Não-confiável (i.e. ruidosa)

Quadro 2.1 – Aspectos importantes sobre o *Soccerserver*.

2.2.2. Trading Agent Competition

A *competição de agentes negociantes* (TAC, do inglês *trading agent competition*) é um fórum internacional que tem o objetivo de prover *benchmarks* de negociação, estimular pesquisadores a desenvolverem diversas técnicas que dotem agentes virtuais da capacidade de negociação, e promover competições oficiais anuais entre estes agentes. Segundo Stone (2002), a TAC foi motivada em vários aspectos pela *RoboCup* (ver seção anterior), já que – apesar de se basear em problemas completamente diferentes de *jogar futebol* – ela também é uma competição multiagente.

Atualmente, a TAC conta com dois *benchmarks* representados por duas competições principais (Tac, 2008). A primeira e mais antiga delas, chamada de *TAC Classic*, trata-se de um problema inserido no domínio complexo e de crescimento rápido dos mercados eletrônicos (*e-markets*), e diz respeito a uma situação em que vários agentes de turismo precisam negociar recursos entre si e montar pacotes de viagens. A mais recente, chamada *TAC SCM* – do inglês *supply chain management* – descreve um cenário de fabricação de computadores pessoais que inicia com a obtenção de componentes, passa pela montagem das máquinas e termina com a venda dos produtos aos clientes.

Ainda que sejam cenários diferentes, ambas as competições confrontam agentes virtuais por meio de um *software* simulador apropriado (há um para cada competição), que lhes fornece informações sobre os recursos negociáveis e deles recebe atitudes de negociação, sempre assegurando a concorrência leal dentro do mercado virtual, através de medidas que evitam ou punem eventuais trapaças. Ademais, mesmo se tratando de problemas diferentes, com simuladores distintos, os princípios tecnológicos empregados nos dois *softwares* são basicamente os mesmos: mesma plataforma de desenvolvimento – no caso, *Java* (Deitel e Deitel, 2003) – mesma arquitetura cliente-servidor, etc. Por esta razão, somada a restrições de agenda e espaço, esta dissertação aborda apenas um destes problemas e seu respectivo *testbed*: a *TAC Classic* e seu simulador chamado *Classic Server*.

TAC Classic e o Classic Server

A competição *TAC Classic* envolve oito (8) agentes autônomos que, emulando agentes de turismo, atendem – cada um – oito (8) clientes que pretendem viajar de *TACtown* a *Tampa*, curtir alguns *eventos de entretenimento*, para então voltar a *TACtown*, em um período equivalente a cinco (5) dias. Cada cliente em particular possui suas próprias preferências por dias de chegada e partida de *Tampa* (dentro do intervalo de cinco dias considerado), quartos de hotel (há dois hotéis disponíveis, sendo um mais luxuoso – o *Tampa Towers* – e outro mais modesto – o *Shoreline Shanties*), e eventos de entretenimento (há três disponíveis: entradas para um *ringue de lutas*, um *parque de diversões* e um *museu*).

Nestes termos, o objetivo dos agentes é montar um pacote de viagens para cada cliente seu, constituído de (1) *passagens de avião*, (2) *reservas de hotel* e (3) *entradas para eventos de entretenimento*, tentando satisfazer ao máximo as preferências de cada um. A complexidade desta situação aumenta devido ao fato de os recursos constituintes dos pacotes serem limitados e oferecidos em iguais condições aos oito agentes: estes devem ser capazes de realizar lances para comprar e vender tais recursos em *leilões* separados por tipo de recurso. Cada tipo de recurso, por sua vez, é cotado em um mercado separado com suas próprias regras. O mercado de passagens de aviões, por exemplo, conta com seu próprio mecanismo de perturbação de preços iniciais (que muda a cada intervalo definido de tempo), aumentando a dinamicidade do ambiente.

Desta forma, em um determinado momento da competição, podem estar ocorrendo vários leilões simultâneos (sobre produtos diferentes), assegurando-se que em uma partida sejam realizados até vinte e oito (28) leilões, sendo oito (8) para negociar passagens aéreas, oito (8) para negociar reservas de hotéis, e doze (12) para negociar *tickets* de entretenimento. Tais leilões devem ser fechados dentro de um intervalo de nove minutos (*9 min*), podendo cada um deles ser aleatoriamente fechado a cada minuto destes nove (i.e. os agentes não sabem quais dos leilões irão fechar no minuto corrente).

Com o intuito de se garantir que os agentes sejam todos expostos a situações equivalentemente complexas, eles são submetidos a várias rodadas de nove minutos, de modo a se variar as preferências de seus clientes em cada uma delas. Ao final da competição, ganha aquele agente que melhor satisfaz seus oito clientes, gastando a menor quantidade de dinheiro possível (dado que cada recurso tem um valor inicial associado).

Neste contexto, o papel do *Classic Server* é garantir que os clientes de cada agente tenham suas preferências por recursos geradas não só aleatoriamente, mas também de uma maneira que impeça que determinados agentes recebam clientes mais fáceis de serem satisfeitos do que os demais. Além disso, é de responsabilidade do *software* manter os mercados, promover os leilões, enviar os valores cotados para os recursos e assegurar a correta flutuação de preços.

Funcionando segundo o paradigma cliente-servidor – tal qual o *Soccerserver* da *RoboCup* – o *Classic Server* é um *servidor HTTP* (Kurose e Rossi, 2004) que atende aos vários agentes de turismo que a ele se conectam e enviam informações

em formato *XML* (Deitel et al., 2001), devidamente encapsuladas em requisições HTTP. As respostas são então enviadas pelo servidor também em formato XML, sendo garantido que, para cada requisição enviada por um cliente, uma resposta – nem que seja de erro – é retornada ao mesmo.

Dado que um servidor pode simular simultaneamente vários jogos, contendo, cada jogo, até vinte e oito leilões concomitantes, cabe a ele controlar quais agentes conectados participam de quais jogos. Além disso, toda vez que um agente envia uma *string* em formato XML que expressa um lance relativo a algum leilão, este deve identificar a qual leilão ele se refere através de um *auctionID* (i.e. um valor único de identificação do leilão) publicado pelo servidor, que deve constar na *string* submetida.

Assim, pode-se afirmar que cada jogo controlado pelo *Classic Server* é um ambiente multiagente bastante dinâmico. Cada cliente conectado, que exerce a função de um agente de turismo, é um processo independente que deve participar dos vários leilões realizados na partida. Como vários agentes podem enviar requisições com lances para um mesmo leilão concorrentemente, cabe ao servidor enfileirá-los segundo a ordem de chegada.

Na prática, um lance é uma *string* que representa a intenção de um agente em comprar ou vender um recurso, e em que quantidade de exemplares. Referências sobre o formato dos lances podem ser obtidas em Tac Classic (2008). Para realizar os leilões, o servidor exerce o algoritmo constante no quadro 2.2.

1. A cotação inicial dos recursos a serem leiloados é enviada para todos os agentes;
2. Enquanto não é hora de encerrar o leilão (decidido aleatoriamente):
 - 2.1. Um lance é retirado da fila de lances do leilão;
 - 2.2. Este lance é avaliado segundo as regras do leilão e do mercado dos recursos em questão;
 - 2.3. Se o lance é válido, a cotação de preços dos recursos é atualizada;
 - 2.4. O agente que enviou o lance recebe informações sobre a validade do mesmo, bem como a cotação de preços dos recursos;
3. Terminado o leilão, o servidor define quais agentes ficarão com quais recursos e registra os seus gastos;
4. As aquisições dos recursos por parte dos agentes são enviadas para todos.

Quadro 2.2 – Algoritmo exercido pelo *Classic Server* durante um leilão.

Como se pode observar, para cada ação que requisita, um agente de turismo sempre tem um retorno com a percepção da cotação de preços do mercado, mais informações sobre a validade de seu lance (sendo incluídas, inclusive, justificativas

para uma eventual rejeição do mesmo). Conforme Stone (2002), esta característica, ao contrário da *RoboCup*, garante aos agentes que eles só receberão suas percepções após terem suas requisições de ações atendidas pelo servidor; por este motivo, fica facilitado lançar mão do paradigma tradicional da IA de disparar ações em função das percepções para implementar a inteligência dos mesmos. Contudo que não extrapole o tempo de ocorrência do leilão, um agente pode gastar o tempo que quiser para deliberar sobre o seu próximo lance, sob pena de realizar menos lances que os demais agentes – na hipótese de estes serem mais rápidos; nesta situação, o agente mais lento ficaria em desvantagem na disputa pelos recursos, por se pronunciar menos vezes.

Aspecto	Valor
Problema a ser resolvido pelos agentes	Negociar recursos em leilões para montar pacotes de viagem baratos
Arquitetura do <i>software</i>	Cliente-servidor (o <i>Classic Server</i> é o servidor, e os agentes são os clientes)
Visualização gráfica da simulação	Não-aplicável
Interface de comunicação do ambiente com os agentes	Protocolo HTTP
Linguagem de programação dos agentes	Qualquer uma, desde que haja suporte ao estabelecimento de conexões HTTP
Quantidade de sociedades de agentes	Uma (1), a <i>sociedade de agentes negociantes</i> (i.e. ambiente mono-social)
Quantidade de agentes	Ambiente multiagente competitivo
Sincronismo entre percepções e ações	Ambiente síncrono (i.e. percepções são fornecidas a um agente sempre que ele age, após atendimento de uma requisição sua por ações)
Confiabilidade das percepções	Percepções exatas
Confiabilidade das ações	Ações exatas
Observabilidade do ambiente	Ambiente parcialmente observável (graças à flutuação de preços dos mercados e aleatoriedade do horário de término de um leilão)
Tipo de comunicação entre os agentes	Sem comunicação entre os agentes
Confiabilidade da comunicação entre agentes	Não-aplicável

Quadro 2.3 – Aspectos importantes do *Classic Server*.

Adicionalmente, dado que os leilões são realizados segundo o paradigma cliente-servidor, não há restrições ao modo como os agentes de turismo são desenvolvidos. O único requisito existente diz respeito ao fato de eles serem capazes de realizar requisições HTTP junto ao servidor, com mensagens no formato XML esperado.

O quadro 2.3 (na página anterior) resume alguns aspectos importantes sobre o *Classic Server*.

2.3. Considerações Finais

Este capítulo trouxe uma discussão sobre o movimento de estabelecimento de *benchmarks* para a área de SMA, tendo definido o que são *benchmarks* para SMA, e apontado as características que tornam qualquer *benchmark* um *bom benchmark*.

Além disso, o estado-da-arte desta área foi apresentado através de um resumo das competições *RoboCup* e TAC. Isto porque ambas reúnem os *benchmarks* mais populares da atualidade.

De fato, tais *benchmarks* são suficientemente genéricos, dadas a diversidade e a quantidade de soluções propostas pelos vários competidores (Robocup, 2008; TAC, 2008). Também são suficientemente específicos, uma vez que as técnicas propostas são confrontadas em torneios. Em termos gerais, espera-se que um vencedor tenha apresentado uma solução melhor que um perdedor.

Sobre a questão da representatividade, a TAC pode ser considerada suficientemente representativa, dado que seus agentes negociantes lidam com problemas cujas soluções contribuem para as áreas de raciocínio, planejamento e aprendizagem. Entretanto, no caso da *RoboCup*, a representatividade é de alguma forma questionável.

Como apontado por Drogoul, Landau e Muñoz (2007), uma técnica, por exemplo, que tenha solucionado o problema de extração de dados relevantes a partir da combinação de sensores de um jogador, certamente contribui para a área de combinação de sensores, sendo – portanto – representativa. Todavia, uma solução para o problema de se evitar impedimentos durante um ataque ao gol adversário pode não ser tão representativa.

Apesar desta questão sobre a representatividade, a *RoboCup*, juntamente com a TAC, contam com *testbeds* estáveis e consolidados, que garantem uma boa qualidade para os seus *benchmarks*. Como apontado neste capítulo, tais *testbeds* são o *Soccerserver* (da liga de simulação da *RoboCup*), e o *Classic Server* (da competição *TAC Classic*, da TAC).

De posse desta discussão sobre *benchmarks*, um questionamento que surge entre os pesquisadores da área de *patrulhamento* é a possibilidade e o potencial do problema do *patrulhamento* como um destes *benchmarks*. Afinal, pode o *patrulhamento* ser um bom *benchmark* para SMA? Antes de se tratar deste tópico, no entanto, faz-se necessário apresentar o problema do *patrulhamento*.

Estes são os assuntos do próximo capítulo.

Capítulo 3

Patrulhamento

Este capítulo aborda o estabelecimento do problema do patrulhamento como um *benchmark* para SMA.

Para tanto, a seção 3.1 define o problema do patrulhamento e aponta seus parâmetros, que são utilizados na modelagem computacional de situações reais de patrulhamento. Na seção seguinte (3.2), o referido problema ganha um novo enunciado dentro do paradigma de SMA, sendo então explicadas as métricas utilizadas por Machado (2002), Almeida (2003), Santana (2004), Chevaleyre (2005) e Menezes (2006) para avaliar as estratégias propostas em seus estudos. Em seguida, um resumo destes trabalhos e algumas críticas são apresentados na seção 3.3, de modo a se elucidar o estado-da-arte deste ramo de pesquisa. De posse deste conhecimento, a seção 3.4 avalia a possibilidade e a qualidade do patrulhamento como um *benchmark* para SMA, enquanto a seção 3.5 traz algumas considerações finais sobre o assunto.

3.1. O Problema do Patrulhamento

Patrulhamento é o ato de se mover no espaço com o intuito de visitar regularmente um conjunto de regiões de interesse. Os motivos para as visitas de tais regiões estão relacionados à detecção da ocorrência de eventos (aqui chamados *eventos patrulhados*). Trazendo para situações do mundo real, um evento patrulhado pode ser a presença de um intruso em uma das salas de um museu, ou a execução de código malicioso em uma das estações de uma rede de computadores, entre várias outras possibilidades.

Ainda que considerar os eventos patrulhados venha a enriquecer a pesquisa sobre patrulhamento, no contexto desta dissertação, para efeito de simplificação, o problema do patrulhamento está restrito à otimização das visitas, sendo ignorada a natureza dos eventos patrulhados. Em outras palavras, o problema do patrulhamento é aqui determinado como a minimização do intervalo de tempo

compreendido entre duas visitas a uma mesma região, considerando todas as regiões de interesse do espaço.

3.1.1. Parâmetros do Problema do Patrulhamento

Almeida (2003) apresentou uma tipologia para o problema do patrulhamento, dando uma direção para a identificação de alguns de seus parâmetros, bem como dos respectivos domínios. Em linhas gerais, estes parâmetros dizem respeito ao *território* a ser patrulhado, ou aos *patrulheiros* que efetuarão o patrulhamento, ou ainda às *sociedades de patrulheiros* envolvidos na tarefa.

Na seqüência, cada um destes três elementos (*território*, *patrulheiros* e *sociedades de patrulheiros*) é tomado como referência para organizar a apresentação dos parâmetros do problema do patrulhamento.

3.1.1.1. Parâmetros Relacionados ao Território

O quadro 3.1 resume os parâmetros do problema do patrulhamento relacionados ao seu *território* (i.e. o espaço cujas regiões de interesse devem ser visitadas).

Parâmetro	Domínio
Continuidade do território	Discreta, Contínua
Dinamicidade do território	Estática, Dinâmica
Homogeneidade do território	Homogênea (todas as regiões de interesse têm a mesma prioridade), Heterogênea (regiões de interesse têm prioridades distintas)

Quadro 3.1 – Parâmetros do problema do patrulhamento relacionados ao seu território.

Como se pode observar, relativamente ao território, um parâmetro importante é a sua *continuidade*. Por um lado há territórios que são claramente *discretos* (e.g. uma rede de computadores, cujas estações devem ser supervisionadas), sendo caracterizados por possuir um conjunto de regiões de interesse bem definidas. Por outro lado, há territórios *contínuos* (e.g. um campo minado cujas minas devem ser

detectadas) que são caracterizados por, sem um mapeamento adequado, possuir regiões de interesse que não estão claramente separadas umas das outras.

Mantendo-se em mente os territórios, outro parâmetro relevante é a sua *dinamicidade*. Enquanto há territórios marcados por não terem seus conjuntos de regiões de interesse modificados com o passar do tempo (e.g. um museu cujas salas devem ser vigiadas), há aqueles cujos conjuntos são bastante dinâmicos (e.g. uma rede de computadores onde novas estações a serem supervisionadas podem ser adicionadas em qualquer hora). Tais territórios são chamados, respectivamente, *estáticos* e *dinâmicos*.

Ainda sobre os territórios, um terceiro parâmetro importante é a *prioridade* das regiões de interesse. Há problemas de patrulhamento cujas prioridades para visitar cada uma das regiões de interesse são todas iguais; tais problemas são ditos possuir *territórios homogêneos*. Em contrapartida, há problemas com certas regiões de interesse que têm maior prioridade sobre as demais (e.g. um banco onde a sala do cofre principal deve ser visitada mais vezes do que as outras); tais problemas são ditos possuir *territórios heterogêneos*.

3.1.1.2. Parâmetros Relacionados aos Patrulheiros

Uma vez que o problema de patrulhamento está relacionado às visitas das regiões de interesse, fica implícito o fato de existirem *patrulheiros* executando tal tarefa. Almeida (2003) percebeu que analisar alguns dos aspectos relacionados aos patrulheiros levaria à identificação de mais parâmetros. O quadro 3.2 resume os parâmetros do problema do patrulhamento relacionados aos patrulheiros.

Parâmetro	Domínio
Profundidade de percepção (<i>PP</i>) dos patrulheiros	$PP \in [0, \infty[$
Confiabilidade das percepções	Percepções exatas, Percepções ruidosas
Confiabilidade das ações	Ações exatas, Ações ruidosas
Tipos de comunicação	Inexistente, Baseada em mensagens, Baseada em marcas, Baseada em <i>blackboard</i>
Confiabilidade da comunicação	Confiável, Não-confiável
Vitalidade dos patrulheiros	Com gasto energético, Sem gasto energético

Quadro 3.2 – Parâmetros do problema do patrulhamento relacionados aos patrulheiros.

Como se pode observar, do ponto de vista dos patrulheiros, um parâmetro importante é a *profundidade de percepção*. Matematicamente, seu domínio é $[0, \infty)$; zero (0) significa que os patrulheiros não podem perceber o território de maneira alguma; qualquer outro valor n positivo significa que os patrulheiros podem perceber o território n unidades (ciclos, metros, etc.) à frente; se n é suficientemente grande (tendendo a infinito), então os patrulheiros são capazes de perceber o território por inteiro em um único instante.

Um outro parâmetro de relevância é a *confiabilidade das percepções* e a *confiabilidade das ações* dos patrulheiros. Em geral, o esperado para as percepções (visão, por exemplo) é que elas estejam corretas em todas as suas ocorrências (i.e. elas são *exatas*). No entanto, não é incomum na robótica haver percepções capturadas com ruídos, que incutem erros nas informações obtidas. Nestes casos, as percepções são ditas *ruídas*. De modo semelhante, o esperado para as ações é que elas tenham os seus efeitos garantidos em cem por cento (100%) das situações (i.e. elas são *exatas*). Contudo, mais uma vez do ponto de vista da robótica, há vezes em que os atuadores falham, havendo ruídos nas ações. Nestes casos, tais ações são ditas *ruídas*.

Outro parâmetro interessante é o *tipo das comunicações* entre os patrulheiros. Enquanto em algumas situações não há a necessidade de se comunicar, em outras os patrulheiros precisam difundir mensagens livremente. Há ainda situações nas quais os patrulheiros devem se comunicar através do depósito de *marcas* no território. Em outros casos, patrulheiros trocam informações por meio de uma estrutura de dados compartilhada (chamada *blackboard*) que é passível de leitura e escrita por todos.

Ainda sobre a comunicação, outro parâmetro relevante é a sua *confiabilidade*. De modo simplificado, um processo de comunicação pode ser *confiável*, se a comunicação entre os patrulheiros ocorre asseguradamente com sucesso em cem por cento (100%) das situações (i.e. um patrulheiro consegue fazer os demais perceberem sua mensagem em todas as suas tentativas). Do contrário, tal processo de comunicação é dito *não-confiável*.

Por fim, um último parâmetro obtido do ponto de vista dos patrulheiros é a sua *vitalidade*. Enquanto em algumas instâncias do problema do patrulhamento não há a necessidade de se associar custos energéticos às ações e percepções dos

patrulheiros, em outras (especialmente em aplicações de robótica) os patrulheiros são suscetíveis à perda de *stamina*, necessitando recarregá-la eventualmente.

3.1.1.3. Parâmetros Relacionados às Sociedades de Patrulheiros

Uma vez que a tarefa de patrulhamento pode ser executada por um grupo de patrulheiros, a noção de *sociedades de patrulheiros* torna-se útil para diversificar as aplicações do problema do patrulhamento. O quadro 3.3 resume os parâmetros do problema do patrulhamento relacionados às sociedades de patrulheiros.

Parâmetro	Domínio
Quantidade de sociedades	Mono-social, Multi-social
Dinamicidade das sociedades	Fechada, Aberta
Homogeneidade das sociedades	Homogênea, Heterogênea

Quadro 3.3 – Parâmetros do problema do patrulhamento relacionados às sociedades de patrulheiros.

Enquanto em algumas situações não há a necessidade de se ter mais de uma sociedade, em outras ter duas ou mais sociedades diferentes pode facilitar o desenvolvimento de uma solução (e.g. uma sociedade de aliados e uma sociedade de inimigos, em um jogo de computador). Esta observação leva à identificação de mais um parâmetro: a *quantidade de sociedades*. Uma instância qualquer do problema do patrulhamento pode ser *mono-social* (se ela possui apenas uma sociedade de patrulheiros), ou *multi-social* (do contrário).

Mantendo-se em vista o aspecto social, um parâmetro adicional relevante é a *dinamicidade das sociedades*. Sucintamente, existem *sociedades fechadas*, cuja quantidade de patrulheiros não muda em tempo de patrulhamento, e existem *sociedades abertas*, cujos patrulheiros podem nascer ou morrer em tempo de patrulhamento.

Finalmente, há também um terceiro parâmetro chamado *homogeneidade da sociedade*. Enquanto algumas sociedades são chamadas *homogêneas* (i.e. todos os patrulheiros são do mesmo tipo e desempenham o mesmo papel), outras são rotuladas *heterogêneas*, uma vez que seus membros não apresentam o mesmo

comportamento (e.g. uma sociedade que tem um coordenador central decidindo quais regiões de interesse os demais patrulheiros devem visitar).

3.2. Patrulhamento Multiagente

Machado (2002) percebeu que, devido à tarefa de patrulhamento ser executada por um grupo de patrulheiros, o problema pode ser resolvido adequadamente por uma abordagem multiagente.

De acordo com Russell e Norvig (2003), a adoção da perspectiva de *agentes* ajudou a reorganizar os subcampos anteriormente isolados da IA (e.g. IA simbólica, IA conexionista, IA evolucionária, IA estatística, etc.), facilitando a combinação de seus resultados no desenvolvimento de sistemas inteligentes. Ainda se baseando nos dois autores, "um *agente* é qualquer coisa que pode ser vista como percebendo seu *ambiente* através de sensores e atuando no mesmo por meio de atuadores". De modo simplificado, depois de perceber o ambiente e antes de agir sobre ele, um agente deve decidir suas próximas ações, levando a um processo deliberativo no qual várias técnicas de IA podem ser livremente aplicadas.

Trazendo a abordagem de agentes inteligentes para o problema do patrulhamento, os patrulheiros são equivalentes aos agentes, enquanto o território a ser patrulhado é o ambiente. Russell e Norvig (2003) também introduziram o conceito de *ambiente de tarefas*, enumerando algumas das suas propriedades ao longo de *dimensões de categorias*. Parafraseando-os, "ambientes de tarefas são os problemas para os quais os agentes racionais são a solução". Portanto, o problema do patrulhamento em si é o ambiente de tarefas, e alguns dos seus parâmetros identificados anteriormente podem ser usados para categorizá-lo ao longo de algumas das dimensões propostas pelos referidos pesquisadores.

O quadro 3.4 relaciona os domínios de alguns dos parâmetros do problema do patrulhamento com algumas das categorias de ambientes de tarefa propostas por Russell e Norvig (2003).

Considerando a primeira dimensão listada, relacionada à questão *contínuo-versus-discreto*, as relações são diretas: um *território contínuo* caracteriza um *ambiente de tarefas contínuo*, assim como um *território discreto* caracteriza um

ambiente de tarefas discreto. Machado (2002) destacou a dificuldade de se patrulhar territórios contínuos. Uma vez que, em tais casos, os limites entre as regiões de interesse não são claramente definidos, um agente acaba tendo uma quantidade praticamente infinita de possibilidades de movimentação, bem como um número muito grande de regiões para visitar (muitas delas interceptando as mesmas porções do espaço).

Dimensão (Russell e Norvig 2003)	Classificação dos ambientes de tarefa (Russell e Norvig 2003)	Domínio do parâmetro	Parâmetro
Contínuo <i>versus</i> Discreto	Ambiente de tarefas contínuo	Território contínuo	Continuidade do território
	Ambiente de tarefas discreto	Território discreto	
Determinístico <i>versus</i> Estocástico	Ambiente de tarefas determinístico	Território estático e percepções e ações exatas	Dinamicidade do território e Confiabilidade das percepções e ações dos patrulheiros
	Ambiente de tarefas estocástico	Território dinâmico ou percepções ou ações ruidosas	
Completamente Observável <i>versus</i> Parcialmente Observável	Ambiente de tarefas parcialmente observável	$PP \in [0, \infty[$ (i.e. não grande o suficiente) ou percepções ruidosas	Profundidade de percepção (PP) dos patrulheiros e Confiabilidade das suas percepções
	Ambiente de tarefas totalmente observável	PP tendendo a ∞ (i.e. grande o suficiente) e percepções exatas	
Agente Único <i>versus</i> Multiagente	Ambiente de tarefas multiagente		

Quadro 3.4 – Relações entre os domínios dos parâmetros do problema do patrulhamento com as categorias de ambientes de tarefa propostas por Russell e Norvig (2003).

Inspirando-se na abordagem da robótica, Machado (2002) propôs o uso de técnicas de *esqueletização* (do inglês *skeletonization*) para simplificar os territórios a serem patrulhados (ver figura 3.1). Sucintamente, um método de esqueletização mapeia o território a ser patrulhado para um grafo, cujos vértices são as regiões de interesse a serem regularmente visitadas, e cujas arestas expressam as maneiras possíveis de se deslocar entre tais regiões. Machado (2002) enumerou ainda as técnicas de esqueletização mais utilizadas.

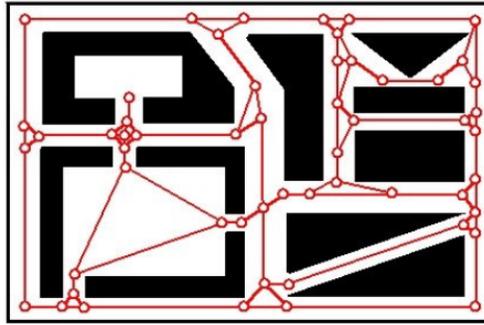


Figura 3.1 – Exemplo de esqueletização (Machado 2002).

Como percebido por Almeida (2003), além da simplificação da representação dos territórios a serem patrulhados, a adoção da abordagem de grafos também resulta em uma abstração interessante para o problema do patrulhamento: com ela, as soluções eventualmente propostas podem ser facilmente aplicadas a ambos os casos de territórios discretos e contínuos, sem a necessidade de se adicionar ou remover características. Isto significa que, se um pesquisador desenvolve uma estratégia de patrulhamento, esta pode ser utilizada para resolver um problema de patrulhamento de redes de computadores, ou qualquer outro problema que, do contrário, envolva um território contínuo. Por este motivo, todas as soluções propostas para o problema do patrulhamento definem uma estratégia para os agentes se moverem sobre um grafo (Machado 2002, Almeida 2003, Santana 2004, Chevaleyre 2005 e Menezes 2006).

Fazendo referência à segunda dimensão listada no quadro 3.4, relacionada ao aspecto *determinístico-versus-estocástico*, fica claro que um *território estático* e *percepções* e *ações* dos patrulheiros sempre *exatas* necessariamente compreendem um *ambiente de tarefas determinístico*. Isto ocorre porque, na situação em questão, pode-se garantir que o próximo estado do ambiente depende única e exclusivamente das ações executadas pelos agentes. Por outro lado, no caso de *territórios dinâmicos*, ou *percepções* ou *ações ruidosas* por parte dos patrulheiros, o próximo estado do ambiente pode ser inesperado, por conta de uma alteração não prevista do território, ou mesmo percepções ou ações incorretas, ainda que os agentes não tenham agido de forma alguma. Por tal razão, estas últimas hipóteses caracterizam *ambientes de tarefa estocásticos*.

Na seqüência, a próxima dimensão relatada lida com a *observabilidade* do ambiente. Nas situações em que os agentes não podem perceber o território por

completo em um único instante (i.e. eles possuem um limite na sua profundidade de percepção), ou as percepções são *ruidosas*, o ambiente é classificado como *parcialmente observável*. Em outras situações, nas quais os agentes podem perceber todo o ambiente em único instante (i.e. sua profundidade de percepção é grande o suficiente), e as percepções são garantidamente *exatas*, o ambiente é dito ser *totalmente observável*.

Finalmente, a última dimensão relacionada diz respeito à quantidade de agentes no ambiente de tarefas. Considerando que a grande maioria das instâncias de problemas do patrulhamento envolve vários patrulheiros visitando as regiões de interesse, pode-se assumir que elas são todas *sistemas multiagentes*; esta afirmação leva a uma redefinição do problema do patrulhamento, chamado *patrulhamento multiagente*:

Patrulhamento multiagente é um sistema multiagente cujo ambiente é representado por um grafo e cujos agentes atuam como *tokens* itinerantes que se movem sobre este grafo, visitando os vértices através das arestas. Deste modo, o objetivo é minimizar o intervalo de tempo compreendido entre duas visitas para cada vértice.

Definido o problema do patrulhamento multiagente, um aspecto importante a ser analisado são as *métricas* empregadas na avaliação das estratégias propostas para resolvê-lo. Desta forma, a próxima seção introduz as métricas mais relevantes já utilizadas na literatura para medir a performance das soluções de patrulhamento.

3.2.1. Métricas do Patrulhamento Multiagente

Levando-se em consideração que o objetivo do patrulhamento multiagente é minimizar o intervalo de tempo entre duas visitas consecutivas para cada vértice, uma métrica importante é justamente este intervalo de tempo, chamado *ociosidade*. Em linhas gerais, a *ociosidade instantânea* de um vértice $k - i_k(t)$ – (ou simplesmente *ociosidade de um vértice k*) mede o intervalo de tempo compreendido entre a última visita até o momento atual t , para um vértice específico k . Tal métrica é a mais fundamental, dado que é utilizada para se obter todas as demais.

Diretamente baseadas nesta métrica, há a *ociosidade instantânea média* – $i(t)$ – i.e. a média das ociosidades instantâneas para todos os vértices no momento t (ver equação 3.1), e há também a *ociosidade instantânea máxima* para o grafo – $\max i(t)$ – que é justamente a maior ociosidade instantânea encontrada no momento t , mais uma vez considerando-se todos os vértices do grafo (ver equação 3.2).

$$i(t) = \frac{\sum_{k=0}^{ord} i_k(t)}{ord}, \quad \text{ord é a ordem do grafo}^1$$

Equação 3.1 – Ociosidade instantânea média do grafo.

$$\max i(t) = i_m(t) \mid (\forall m, n \in V, \quad m \neq n \quad \wedge \quad i_m(t) \geq i_n(t)),$$

V é o conjunto de vértices do grafo

Equação 3.2 – Ociosidade instantânea máxima do grafo.

Adicionalmente, coletar as ociosidades instantâneas médias a uma taxa específica permite o cálculo da *ociosidade média do grafo* – $I(t)$ – que é a média de todos os valores coletados até o presente momento t (ver equação 3.3). Similarmente, pode-se medir a *ociosidade máxima do grafo* – $\max I(t)$ – que é a maior ociosidade já encontrada no grafo, até o presente momento t (ver equação 3.4).

$$I(t) = \frac{1}{t} \int_0^t i(t) dt$$

Equação 3.3 – Ociosidade média do grafo.

$$\max I(t) = \max i(m) \mid (\forall m, n \leq t, \quad m \neq n \quad \wedge \quad \max i(m) \geq \max i(n))$$

Equação 3.4 – Ociosidade máxima do grafo.

¹ A cardinalidade do conjunto de vértices de um grafo chama-se sua *ordem* (Gould, 1988)

Dadas as métricas apresentadas, um passo natural é saber como elas são utilizadas para comparar duas ou mais estratégias distintas de patrulhamento. Novamente lembrando-se que elas são todas baseadas nos intervalos de tempo compreendidos entre duas visitas para cada vértice (i.e. as ociosidades dos vértices), e que o objetivo do patrulhamento multiagente é justamente minimizar tais valores, uma estratégia de patrulhamento é considerada melhor que outra se produz valores inferiores para tais métricas.

Machado (2002) percebeu que para uma avaliação completa de uma estratégia de patrulhamento, a quantidade de agentes deve ser de alguma forma considerada. Isto porque, ainda que seja esperado que a adição de mais patrulheiros sempre resulte em melhoras de performance para uma estratégia (ao menos em termos absolutos), não se pode garantir que este ganho seja justificável, devido a gastos computacionais adicionais (e até mesmo gastos financeiros, no caso de uma aplicação de robótica). Pensando nestes termos, Machado (2002) propôs um método de normalização de métricas, de modo a refletir o *ganho* decorrente da adição de mais agentes (ver equação 3.5).

$$metrica_normalizada = métrica_absoluta \times \frac{número_de_agentes}{número_de_vértices}$$

Equação 3.5 – Normalização de um valor coletado para uma métrica.

Sucintamente, a equação proposta funciona da seguinte maneira: imaginando que cinco (5) agentes estão patrulhando um grafo com cem (100) vértices, suponha-se que a ociosidade média obtida por uma estratégia *s* depois de um tempo *t* de simulação seja igual a cinco ($5 = 100 \times 5 / 100$). Considerando agora outra situação, suponha-se que, para a mesma estratégia *s*, o uso de dez (10) agentes resulte em uma ociosidade média de setenta segundos (70 sec), após o mesmo intervalo de tempo *t* de simulação. Neste caso, o valor normalizado é igual a sete ($7 = 70 \times 10 / 100$), revelando que, apesar do número de agentes ter sido duplicado (de 5 para 10), o ganho com tal modificação foi pequeno (de 5 para 7).

Moreira et al. (2007) perceberam que, devido às métricas coletadas serem todas baseadas no conceito de ociosidade, não importa como os agentes decidem a ordem e o momento exato de visitar os vértices; para impactar sobre a coleta das métricas, tudo o que deve ser feito é visitar os vértices o mais rapidamente possível.

Esta é uma característica importante que, juntamente com a representação em grafo para o território a ser patrulhado, assegura um nível de abstração muito interessante para o problema do patrulhamento.

Explicadas as métricas mais relevantes do patrulhamento multiagente, a próxima seção resume os trabalhos já desenvolvidos nesta área, evidenciando suas contribuições.

3.3. Estado-da-arte do Patrulhamento Multiagente

Percebendo que a tarefa de patrulhamento envolve situações que requerem comportamento coordenado e tomada de decisão entre os patrulheiros, Machado (2002) iniciou o estudo do assunto dentro do paradigma de SMA. Desde então, uma série de trabalhos nesta linha têm sido realizados. Enquanto algumas pesquisas testaram empiricamente várias técnicas distintas – como *comportamento reativo* e *cognitivo* (Machado 2002, e Almeida 2003), *aprendizagem de máquina* (Santana 2004) e *mecanismos de negociação* (Menezes 2006) – Chevaleyre (2005) desenvolveu uma análise teórica acerca do assunto. Na seqüência, um resumo das contribuições de cada uma destes autores é apresentado em ordem cronológica.

3.3.1. Agentes Pioneiros

O trabalho de Machado (2002) foi pioneiro na área de patrulhamento, apresentando uma discussão original acerca da tarefa de patrulhamento multiagente, bem como uma avaliação empírica de possíveis soluções. Em termos práticos, diversas arquiteturas *reativas* e *cognitivas*² de SMA foram propostas, além de critérios *de avaliação*, *cenários de experimentação* e uma primeira versão de um *simulador de patrulhamento*.

² Segundo Russell e Norvig (2003), arquiteturas reativas e cognitivas de agentes se diferenciam pela presença ou não de um *objetivo* explícito. No caso do patrulhamento, nas arquiteturas reativas, os patrulheiros tomam decisões locais, escolhendo o próximo vértice a visitar dentro da vizinhança do vértice em que se encontram. Já nas arquiteturas cognitivas, os patrulheiros elegem um vértice como objetivo, podendo este estar em qualquer lugar do grafo: daí, planejam e percorrem uma rota até ele.

Tendo em vista as arquiteturas de SMA, Machado (2002) fundamentou o processo de decisão dos seus agentes unicamente nas ociosidades dos vértices do grafo a ser patrulhado. Enquanto em algumas destas arquiteturas, a ociosidade considerada é *individual* (i.e. cada agente tem a sua própria percepção das ociosidades, baseadas apenas nas suas próprias visitas aos vértices), em outras a ociosidade é *compartilhada* (i.e. todos os agentes têm a mesma percepção das ociosidades, baseadas nas visitas de todos indistintamente).

Das estratégias propostas, merecem destaque por seu desempenho a arquitetura de *agentes reativos conscientes* (agentes CR, do inglês *conscientious reactive agents*), e a arquitetura de *agentes coordenados cognitivos* (agentes CC, do inglês *cognitive coordinated agents*).

Os agentes CR são caracterizados por escolher como próximo passo o vértice de sua vizinhança que apresenta maior ociosidade individual. Tais agentes são ditos *reativos* uma vez que não possuem um objetivo a longo prazo para cumprir; simplesmente percebem o meio e decidem a sua próxima ação. Adicionalmente, são ditos *conscientes* porque lembram os vértices que já visitaram anteriormente, dado que guardam para si informações das ociosidades individuais.

Os agentes CC, por sua vez, são caracterizados por contar com um *coordenador central* (daí a expressão *coordenados*), que avalia as ociosidades compartilhadas do grafo e recomenda a cada patrulheiro, através do envio de mensagens, o vértice que possui a maior ociosidade e ainda não é objetivo de alguém. Quanto aos patrulheiros, uma vez definido os seus objetivos, estes devem então planejar o seu deslocamento até eles, utilizando para isso o algoritmo de Dijkstra (daí a expressão *cognitivos*).

Em respeito aos critérios de avaliação, pela primeira vez foram apresentados os conceitos de *ociosidade média* e *pior ociosidade* (ver seção 3.2.1). Com relação aos cenários de experimentação propostos, foi no trabalho de Machado (2002) que os mapas *A* e *B* – apresentados na figura 3.2(a) e 3.2(b) – foram definidos, vindo a compor, posteriormente, o conjunto de mapas apontados por Santana (2004) como configurações ideais para se comparar diferentes técnicas de patrulhamento.

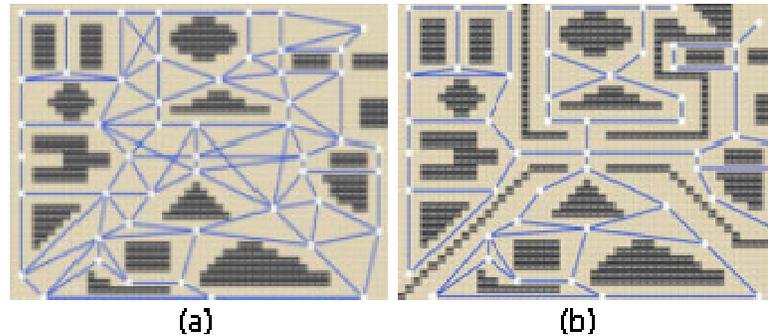


Figura 3.2 – Mapas A (a) e B (b) propostos por Machado (2002).

Finalmente, tendo em vista o simulador proposto, este lidava apenas com grafos cujas arestas eram unitárias e contava a passagem do tempo em *ciclos*, coincidentes com o ciclo *perceber-raciocionar-agir* dos patrulheiros.

3.3.2. Agentes Heurísticos

Almeida (2003) percebeu a importância de se utilizar grafos com pesos para representar os territórios a serem patrulhados. Como apontado pelo autor, arestas valoradas permitem expressar a realidade com maior propriedade, já que os pesos a elas associados podem refletir, dentre várias outras possibilidades, o comprimento das ruas de uma cidade, ou os tempos gastos para se percorrer os enlaces de uma rede de computadores.

Deste modo, frente à limitação de lidar apenas com grafos cujas arestas têm tamanho unitário, Almeida (2003) estendeu as arquiteturas anteriormente propostas por Machado (2002), de modo a fazê-las funcionar em grafos com arestas valoradas. Para tanto, dois tipos principais de alterações foram feitos:

- 1) O processo de decisão de um agente passou a se fundamentar não só nas ociosidades (individuais ou compartilhadas) dos vértices, mas também na distância compreendida entre o seu vértice atual e os vértices candidatos a objetivo;
- 2) Para os agentes cognitivos, o planejamento de rotas até os vértices-objetivo passou a utilizar um mecanismo de *pathfinding* mais sofisticado, que lança mão sobre a ociosidade dos vértices intermediários no cálculo do menor caminho.

Tendo em vista o processo de decisão de um patrulheiro, os vértices candidatos a objetivo passaram a ser avaliados não só de maneira diretamente proporcional às suas ociosidades, mas também inversamente proporcional às distâncias compreendidas entre eles e o vértice corrente do patrulheiro. Com isto, um *bom objetivo* para um agente passou a ser não só um vértice de *ociosidade alta* (quando comparado aos demais), mas também de *boa proximidade* (em relação aos demais).

Em respeito ao planejamento mais sofisticado de rotas – no caso das arquiteturas cognitivas – o procedimento avaliativo citado pôde então ser empregado na seleção do próximo vértice a ser expandido, dentro do algoritmo de Dijkstra. A figura 3.3 traz uma situação que ajuda a ilustrar os resultados obtidos com o emprego deste novo mecanismo.

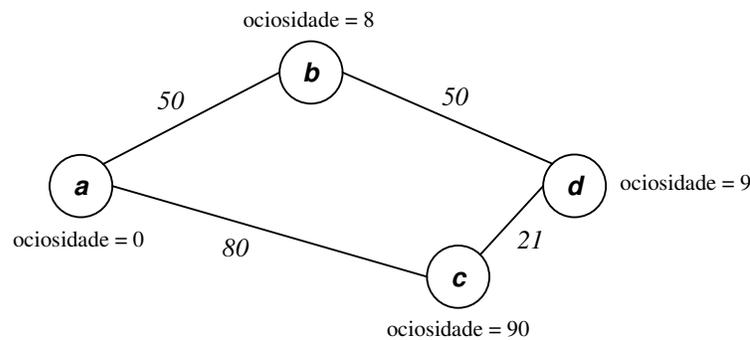


Figura 3.3 – Um exemplo de grafo para patrulhamento.

Com a utilização do algoritmo de Dijkstra convencional, baseado apenas no tamanho das arestas, um agente querendo se deslocar do vértice *a* para o vértice *d* utilizaria o caminho *a-b-d* (que é menor), de tamanho cem (100). Entretanto, existe um outro caminho *a-c-d* um pouco maior (exatamente uma unidade maior que o caminho *a-b-d*, neste caso), que passa por um vértice de maior ociosidade – no caso o vértice *c*, com ociosidade igual a noventa (90), cerca de dez vezes maior que os demais. Tomar este outro caminho certamente impactaria numa redução dos valores calculados pelas métricas descritas na seção 3.2.1, aspecto que aponta uma melhor estratégia de patrulhamento.

Nestes termos, a idéia por detrás do mecanismo de *pathfinding* refinado é que os patrulheiros dêem preferência aos caminhos que contêm os vértices de maior

ociosidade, ainda se mantendo o compromisso com a minimização do comprimento total das rotas.

Dentre as estratégias propostas por Almeida (2003), merece destaque por seu desempenho a arquitetura de *agentes coordenados cognitivos roteadores heurísticos* (agentes HPCC, do inglês *heuristic pathfinder cognitive coordinated agents*), uma espécie de "evolução" dos agentes CC. Grosso modo, os agentes HPCC são caracterizados por contar com um coordenador central (daí a expressão *coordenados*), que avalia as ociosidades compartilhadas do grafo e recomenda a cada patrulheiro, através do envio de mensagens, o vértice melhor avaliado e que ainda não é objetivo de alguém. Quanto aos patrulheiros, uma vez definido os seus objetivos (daí a expressão *cognitivos*), estes devem então planejar o seu deslocamento até ele, utilizando para isso o algoritmo de *pathfinding* refinado (daí a expressão *roteadores heurísticos*).

Além de contribuir com a melhoria na representação do ambiente a ser patrulhado, Almeida (2003) estendeu o simulador outrora construído por Machado (2002) para funcionar com grafos cujas arestas são valoradas.

3.3.3. Agentes Aprendizes por Reforço

Em seus estudos sobre o patrulhamento, Santana (2004) observou que nos trabalhos de Machado (2002) e Almeida (2003), para todas as arquiteturas propostas, sempre havia uma instância do problema (uma topologia em particular do grafo a ser patrulhado, por exemplo) na qual uma boa estratégia (em termos gerais) apresentava um desempenho ruim.

Frente a esta situação, o pesquisador motivou-se a desenvolver agentes adaptativos, que pudessem de alguma maneira estabelecer por si mesmos uma boa estratégia de patrulhamento, ajustando-se autonomamente à instância do problema de patrulhamento a que estivessem sendo submetidos. Para tanto, o autor entendeu que utilizar técnicas de *aprendizagem de máquina* representava um caminho interessante para se alcançar tal objetivo.

Dentre as técnicas de aprendizagem de máquina existentes, Santana (2004) recorreu à *aprendizagem por reforço* para propor suas arquiteturas de agentes. Em

linhas gerais, na aprendizagem por reforço um agente toma decisões baseado em uma representação apropriada do ambiente (*estado*), e recebe um *feedback* sobre o resultado de suas ações (*recompensa*). Nestes termos, ele aprende o que fazer (i.e. como mapear estados em ações) ao maximizar um sinal numérico que recebe como recompensa por suas ações.

Dado que a aplicação da aprendizagem por reforço no problema do patrulhamento não é direta, Santana (2004) precisou primeiramente remodelá-lo com base em conceitos do *processo de decisão de Markov* e do *processo de decisão semi-markoviano*. Uma vez remodelado, o problema tornou-se então passível de solução pelo algoritmo de aprendizagem *q-learning*, apropriadamente empregado em todas as arquiteturas propostas pelo autor.

Dentre as estratégias de patrulhamento desenvolvidas por Santana (2004), merece destaque por seu desempenho a arquitetura de *agentes aprendizes de caixa-cinza* (agentes GBLA, do inglês *gray-box learner agents*), que avaliam a utilidade de suas ações de maneira *egoísta* (i.e. sem considerar a utilidade de fato para todos), calculando a recompensa de suas ações com base na ociosidade instantânea – $i(t)$, ver seção 3.2.1. Sucintamente, cada agente GBLA age de acordo unicamente com o seu próprio processo de decisão markoviano (ou semi-markoviano), funcionando para os demais patrulheiros como uma *caixa-preta*, amenizada pelo fato de que eles trocam mensagens sobre os vértices que pretendem visitar e incluem esta informação na sua representação de estados do ambiente (daí a expressão *caixa-cinza*).

Além de propor arquiteturas de agentes adaptativos, o trabalho em questão foi o primeiro a apresentar cenários de simulação que fossem de fato representativos das diferentes topologias que um agente pode vir a enfrentar em situações reais. A figura 3.4 mostra os mapas utilizados nos experimentos. Como se pode observar, os mapas (a) e (b), aproveitados do trabalho de Machado (2002) representam topologias genéricas, muito e pouco conectadas, respectivamente. O mapa (c) (*Circular*) representa uma topologia de círculo, enquanto o mapa (d) (*Corridor*) representa um corredor. O mapa (e) (*Islands*), por sua vez, representa uma topologia de ilhas, onde há uma grande variação nos tamanhos das arestas. Finalmente, o mapa (f) (*Grid*) representa uma topologia de grade.

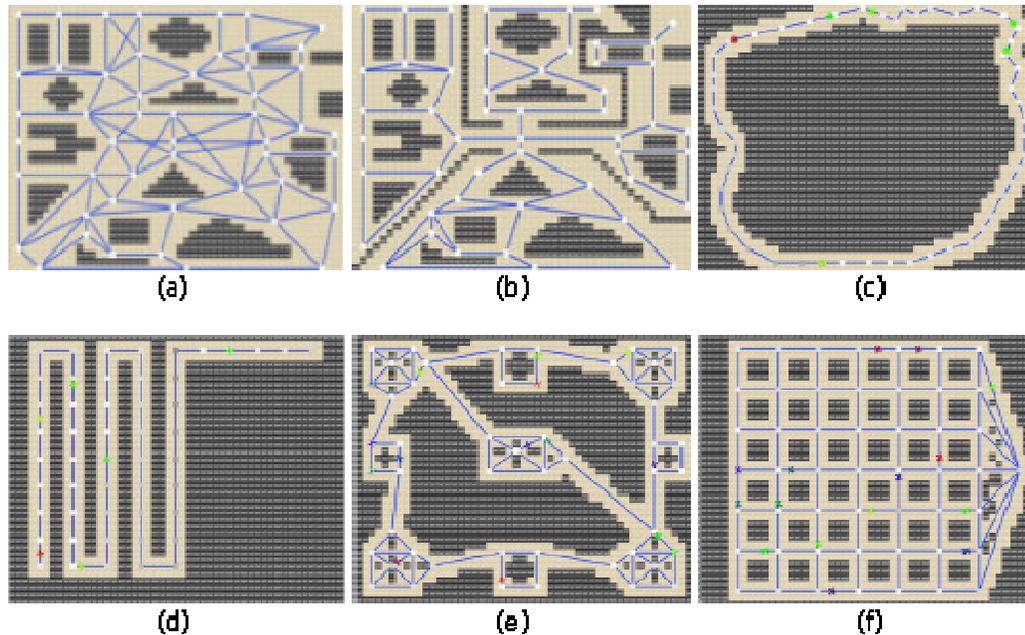


Figura 3.4 – Modelos de territórios propostos por Santana (2004). Em (a) tem-se o mapa A, (b) o mapa B, (c) o mapa *Circular*, (d) o mapa *Corridor*, (e) o mapa *Islands*, e (f) o mapa *Grid*.

3.3.4. Agentes Baseados no Caixeiro Viajante

De modo a complementar os trabalhos desenvolvidos até o momento, que vinham sendo feitos baseados sobretudo no empirismo, Chevaleyre (2005) realizou as primeiras análises inteiramente teóricas acerca do problema do patrulhamento, tratando-o como uma instância de um *problema de otimização combinatória*. Isto porque, partindo de sua própria definição, o objetivo dos patrulheiros é *minimizar* o tempo compreendido entre duas visitas a um mesmo vértice, para todos os vértices do grafo.

Assim, dentre várias verificações, o pesquisador chegou à conclusão de que, para um único agente, a melhor estratégia de patrulhamento é a de percorrer os vértices do grafo através da solução do *caixeiro viajante*, dado que ela *otimiza* (i.e. minimiza) os valores da métrica de ociosidade máxima – $\max l(t)$, ver seção 3.2.1.

Em respeito às instâncias multiagentes do problema, Chevaleyre (2005) propôs duas categorias principais de estratégias, onde em uma os agentes devem sempre percorrer a solução do caixeiro viajante de alguma forma, e na outra os

agentes devem particionar de alguma maneira o grafo entre si, para que então calculem, frente aos subgrafos resultantes, soluções individuais do caixeiro viajante.

Das soluções propostas pelo autor, merece destaque pelo seu desempenho a chamada *estratégia de ciclo único* (SC, do inglês *single cycle*), onde os patrulheiros são colocados a percorrer a solução do caixeiro viajante, todos no mesmo sentido, mantendo entre seus predecessores e sucessores a mesma distância. A figura 3.5 mostra esta situação para dois patrulheiros.

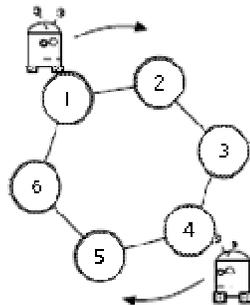


Figura 3.5 – Agentes patrulhando segundo a estratégia do ciclo único.

3.3.5. Agentes Negociantes

Seguindo os passos de Machado (2002), Almeida (2003) e Santana (2004), Menezes (2006) propôs novas arquiteturas de agentes dotados de mecanismos de *negociação* para decidir como patrulhar um determinado território.

Como apontado pela autora, do ponto de vista dos SMA, a negociação é um processo que permite a grupos de agentes comunicarem-se entre si, para estabelecer acordos mutuamente aceitáveis, tendo em vista a solução de um mesmo problema. Em outras palavras, os agentes negociantes precisam chegar a um consenso na distribuição de tarefas ou recursos, seja porque não possuem individualmente capacidade para resolver o problema em questão, seja porque não detêm em seu poder recursos suficientes para fazê-lo.

Em termos práticos, segundo Menezes (2006), a negociação multiagente envolve componentes básicos, a saber: os *objetos negociáveis* (que são os recursos a serem trocados entre os agentes), o *protocolo de negociação* (que define como os agentes devem interagir para oferecer e solicitar instâncias dos objetos a serem

negociados), e as *estratégias dos agentes* (que definem, para cada agente, o quanto eles estão dispostos a realizar concessões de seus recursos, em troca de recursos que estão em posse de outros agentes, sempre visando a maximização do seu contentamento).

Focando-se no problema do patrulhamento, os objetos negociáveis são os vértices que os patrulheiros devem visitar. Nestes termos, tais patrulheiros devem negociar entre si quais vértices cada um é responsável por visitar regularmente. A principal vantagem desta solução é a simplificação do problema do patrulhamento, dado que instâncias menores suas – i.e. instâncias com uma menor quantidade de vértices a serem visitados – são geradas e equilibradamente distribuídas entre os agentes.

Há, no entanto, a exigência de que os *mecanismos de negociação* (i.e. o protocolo de negociação e as estratégias dos agentes) sejam simples, sob pena de introduzirem uma nova etapa na solução do problema do patrulhamento que, apesar de simplificado, pode acabar por continuar tão ou mais complexo quanto antes, caso tais mecanismos sejam muito complicados.

Partindo deste ponto de vista, Menezes (2006) propôs a utilização de *leilões* para implementar os mecanismos de negociação, dado que eles constituem cenários de interação entre os agentes bastante simples para o estabelecimento de acordos.

Em um cenário de patrulhamento típico, cada agente recebe um conjunto aleatório de vértices do grafo a ser patrulhado, e avalia a qualidade deste conjunto com base na proximidade dos seus elementos. No caso de um agente detectar que um de seus vértices não pode ser visitado por ele em um tempo aceitável (devido a estar muito distante dos demais), ele inicia então um leilão, exercendo o papel de *leiloeiro*. Os demais patrulheiros, no papel de *arrematantes*, checam em seus conjuntos se há um vértice cuja troca com o vértice leilado é vantajosa. No caso de este vértice existir, o arrematante efetua um *lance* oferecendo-o. Assim, o leiloeiro escolhe o melhor lance (i.e. o vértice oferecido mais próximo aos demais do seu conjunto), e estabelece o acordo (no caso a troca) com o seu arrematante.

Neste processo, deve-se destacar que o vértice leilado tem valores distintos para cada agente arrematante, e os lances são secretos (um arrematante não sabe quais vértices os demais patrulheiros estão oferecendo). Em respeito às diversas arquiteturas de agentes propostas, estas se diferem principalmente quanto ao tipo de leilão promovido pelos agentes (há leilões de uma, duas ou mais rodadas), ao

modo como os agentes decidem o vértice a ser leilado (com base unicamente na distância entre os vértices, ou considerando também a ociosidade dos mesmos), e ao modo como os agentes se deslocam para patrulhar os vértices sob sua responsabilidade.

Das estratégias propostas, merece destaque por seu desempenho a *arquitetura de agentes roteadores heurísticos arrematantes em dois lances* (agentes HPTB, do inglês *heuristic pathfinder two-shots bidder agents*). Como sugere o nome, tais patrulheiros podem efetuar dois lances distintos em cada leilão que participam, na tentativa de arrematar o vértice oferecido (daí a expressão *two-shots bidder*), e avaliam a proximidade dos vértices e planejam a maneira de visitá-los com base no roteamento heurístico empregado nos agentes HPCC (ver seção 3.3.2).

Por fim, o trabalho de Menezes (2006) levanta uma questão interessante em relação à filosofia inerente aos SMA: dado que os agentes devem ser completamente autônomos, mais do que forçar um outro indivíduo a realizar um serviço, um agente deve buscar convencer os demais a cooperarem com ele. Para tanto, eles precisam *negociar*, e a pesquisadora certamente propiciou este mecanismo para o caso do problema do patrulhamento.

3.3.6. Críticas aos Trabalhos Anteriores

O quadro 3.5 resume as principais contribuições dos trabalhos anteriores em relação ao problema do patrulhamento. As arquiteturas citadas correspondem justamente àquelas que apresentaram os melhores desempenhos, dentro de cada pesquisa (i.e. produziram os menores valores para as métricas do problema – ver seção 3.2.1).

Ainda que estas contribuições tenham sido muito importantes, estes trabalhos não estão livres de críticas. Moreira et al. (2007) apresentaram algumas delas. Uma fonte de sugestões de melhorias, por exemplo, diz respeito aos parâmetros do problema do patrulhamento que ainda não foram levados em consideração pelos pesquisadores. Aspectos como grafos cujos nodos têm prioridades distintas para serem visitados (i.e. *territórios heterogêneos*), ou cujos nodos e arestas podem ser tornar indisponíveis em tempo de patrulhamento (i.e. *territórios dinâmicos*, ver seção 3.1.1.1) não foram considerados nas primeiras estratégias propostas. De modo

semelhante, *sociedade abertas* de agentes e a possibilidade de se associar *custos energéticos* às ações e percepções dos patrulheiros ainda não foram exploradas (ver seções 3.1.1.3 e 3.1.1.2, respectivamente).

Autor (ano)	Contribuições	Arquiteturas de destaque
Machado (2002)	<ul style="list-style-type: none"> Arquiteturas reativas e cognitivas de SMA; Processo de decisão dos agentes baseado unicamente nas ociosidades dos vértices do grafo a ser patrulhado; Métricas de ociosidade média e pior ociosidade; Mapas <i>A</i> e <i>B</i> para patrulhamento; Primeira versão do simulador de patrulhamento. 	<p><i>Cognitive coordinated agents</i> (CC) e <i>conscientious reactive agents</i> (CR)</p>
Almeida (2003)	<ul style="list-style-type: none"> Aprimoramento das arquiteturas propostas por Machado (2002) para fazê-las funcionar em grafos com arestas valoradas; Processo de decisão dos agentes baseado não só nas ociosidades dos vértices, mas também na distância compreendida entre o vértice atual e os vértices candidatos a objetivo; Extensão do simulador de Machado (2002) para lidar com grafos cujas arestas são valoradas. 	<p><i>Heuristic pathfinder cognitive coordinated agents</i> (HPCC)</p>
Santana (2004)	<ul style="list-style-type: none"> Arquiteturas de agentes capazes de aprender por reforço a patrulhar um determinado território; Cenários de simulação representativos das diferentes topologias que um agente pode vir a enfrentar em situações reais: mapas <i>A</i> e <i>B</i> de Machado (2002), mais outros quatro (<i>Circle</i>, <i>Corridor</i>, <i>Islands</i> e <i>Grid</i>). 	<p><i>Gray-box learner agents</i> (GBLA)</p>
Chevaleyre (2005)	<ul style="list-style-type: none"> Primeiras análises inteiramente teóricas acerca do problema do patrulhamento, tratando-o como uma instância de um problema de otimização combinatória; Conclusão de que, para um único agente, a melhor estratégia de patrulhamento é a de percorrer os vértices do grafo através da solução do caixeiro viajante. 	<p><i>Single Cycled agents</i> (SC)</p>
Menezes (2006)	<ul style="list-style-type: none"> Arquiteturas de agentes negociantes, que estabelecem acordos entre si para decidir quais vértices do grafo a ser patrulhado ficam a encargo de serem visitados regularmente por quais patrulheiros. Tais acordos são estabelecidos por meio de leilões. 	<p><i>Heuristic pathfinder two-shots bidder agents</i> (HPTB)</p>

Quadro 3.5 – Resumo das contribuições dos trabalhos de Machado (2002), Almeida (2003), Santana (2004), Chevaleyre (2005) e Menezes (2006).

Esta situação de variações do problema a serem exploradas ainda se agrava pelo fato de as várias versões existentes do simulador de patrulhamento –

decorrentes dos trabalhos anteriores – terem sido concebidas focando-se em necessidades locais e isoladas destas pesquisas. Desta maneira, elas não estão de forma alguma preparadas para, por exemplo, guardar as prioridades de visitação dos vértices do grafo a ser patrulhado, ou executar mecanismos de atualização de uma eventual sociedade aberta de patrulheiros. Um desenvolvedor capacitado poderia obviamente estender estas versões – e.g. Almeida (2003) estendeu o simulador proposto por Machado (2002) para que este lidasse com grafos cujas arestas são valoradas – mas no atual estágio em que se encontram as várias versões, sem qualquer controle de quais são as existentes, qual a precedência entre elas e quais aspectos cada uma já traz implementados, tal tarefa fica muito prejudicada.

No entanto, uma das características mais importantes que ainda não foram analisadas está relacionada à contagem do tempo. Levando-se em consideração as avaliações feitas sobre o desempenho das estratégias de patrulhamento anteriormente propostas, todas foram experimentadas em versões de um simulador que conta o tempo em ciclos de *percepção-raciocínio-ação* dos agentes. Portanto, quando um pesquisador submetia sua estratégia a cento e vinte e três (123) ciclos de simulação – por exemplo – ele estava na verdade permitindo que seus agentes percebessem, raciocinassem e agissem exatas cento e vinte e três (123) vezes. Em tais situações, enquanto os agentes estavam deliberando sobre a sua próxima ação, era assegurado que o ambiente não se modificava de forma alguma. Similarmente, as métricas de ociosidade eram calculadas com base nestes valores, o que significa que, no pior caso, a ociosidade máxima possível para um grafo era cento e vinte e três (123) – indicando que pelo menos um nodo nunca fora visitado.

Dada a diversidade de técnicas aplicadas e mecanismos de raciocínio, percebeu-se que tal método de simulação de tempo ignora o tempo real gasto pelos agentes para tomar suas decisões; deste modo, se uma estratégia escolhe de fato as melhores ações em cada turno, mas sob pena de gastar uma quantidade de tempo muito grande, ela não é penalizada de forma alguma, ainda que, em termos reais, os seus patrulheiros demorem muito pensando entre uma visita e outra.

Longe de ofuscar as contribuições exercidas pelos trabalhos anteriores, as críticas aqui apresentadas fazem parte do processo natural de investigação científica acerca do problema do patrulhamento, além de indicarem tópicos para pesquisas futuras.

3.4. Pode o Patrulhamento ser um *Benchmark* para Sistemas Multiagentes?

Apesar da quantidade de soluções já propostas (Machado, 2002; Almeida, 2003; Santana, 2004; Chevaleyre, 2005; Menezes, 2006), o patrulhamento multiagente ainda não foi analisado como um *benchmark* para SMA. Na hipótese de isto ser feito, sua popularidade aumentaria (mais pesquisadores da área de SMA se interessariam pelo tópico), e como uma consequência natural, mais técnicas e ferramentas relacionadas seriam desenvolvidas.

Mas afinal, considerando o conceito de *benchmarks* para SMA apresentado no capítulo 2, pode o problema do patrulhamento multiagente ser estabelecido como um destes *benchmarks*? Para se obter esta resposta, devem-se satisfazer os seguintes questionamentos:

- O patrulhamento multiagente é suficientemente genérico, a ponto de poder ser resolvido por várias técnicas diferentes?
- O patrulhamento multiagente é suficientemente específico, a ponto de permitir que tais técnicas sejam comparadas entre si?
- O patrulhamento multiagente é suficientemente representativo?

A resposta para a primeira pergunta é comprovadamente positiva graças ao número de técnicas já desenvolvidas para resolver o problema do patrulhamento. Como mostrado na seção 3.3, uma série de trabalhos têm sido realizados desde 2002. Enquanto alguns pesquisadores testaram empiricamente várias técnicas – a exemplo de comportamento reativo (Machado, 2002), aprendizagem de máquina (Santana, 2004) e mecanismos de negociação (Menezes, 2006) – Chevaleyre (2005) apresentou uma análise teórica do assunto. Ademais, mais esforços têm sido empregados pelo grupo de pesquisas em IA da Universidade Federal de Pernambuco (UFPE) no sentido de desenvolver novos mecanismos de patrulhamento.

Levando o segundo item em consideração, a resposta é novamente positiva. O patrulhamento multiagente é específico o suficiente a ponto de permitir que as diversas técnicas existentes sejam comparadas entre si; do contrário, Santana (2004), Chevaleyre (2005) e Menezes (2006) não teriam comparado suas estratégias aos métodos de patrulhamento anteriores a seus trabalhos. Devido às

métricas bem definidas (ver seção 3.2.1), e ao conjunto de seis (6) mapas propostos por Santana (2004) – ver seção 3.3.3 – os resultados obtidos das várias técnicas podem ser medidos deterministicamente, permitindo o estabelecimento de uma ordem total entre eles.

Por fim, tendo-se em mente a terceira questão, o patrulhamento multiagente pode ser considerado suficientemente representativo, por duas razões. Primeiramente, do ponto de vista da IA, ele lida com situações cujas soluções contribuem para os campos de colaboração e coordenação de agentes (Machado, 2002), planejamento (Almeida, 2003), aprendizagem de máquina (Santana, 2004), comunicação entre agentes (Menezes, 2006), etc. Segundamente, o patrulhamento é representativo, dada a sua vasta possibilidade de aplicações. Como apontado por Machado (2002), ele pode ser útil na detecção de falhas em *intranets*, ou no deslocamento de personagens em jogos de estratégia, ou ainda na detecção de páginas novas ou recém-modificadas na *internet*; enfim, qualquer contexto que exija vigilância, inspeção ou controle distribuídos.

Respondidos os três questionamentos, um detalhe relevante diz respeito a quão bom o patrulhamento multiagente é, enquanto um *benchmark*. Lembrando os preceitos apresentados por Drogoul, Landau e Muñoz (2007) de que um bom *benchmark* é aquele que facilita a representação, a compreensão e o desenvolvimento de novos métodos, permitindo o enfoque na solução do problema em si, em detrimento da sua representação, o patrulhamento multiagente o faz das seguintes formas:

- Os territórios a serem patrulhados são simplificados através de grafos. Desta forma, quando um agente patrulheiro visita os vértices de um grafo, ele pode representar um vigilante que faz a ronda noturna visitando as salas de um museu, ou talvez um aplicativo de inspeção do estado dos enlaces de uma *intranet*, ou ainda um aventureiro caçando fantasmas nos quartos de uma mansão assombrada.
- As métricas coletadas são todas baseadas no intervalo de tempo compreendido entre duas visitas consecutivas a um mesmo vértice – i.e. na *ociosidade* (ver seção 3.2.1). Portanto, não importa como os agentes decidem em qual ordem e em qual o momento os nodos devem ser visitados; para impactar sobre as métricas coletadas, tudo o que eles precisam fazer é visitá-los rapidamente.

Nestes termos, o que está faltando para viabilizar a larga utilização e divulgação do patrulhamento como um *benchmark* para SMA? Certamente, um primeiro passo é criar ferramentas, em particular um simulador unificado, que facilitem a implementação e a comparação das soluções propostas por qualquer pesquisador.

O quadro 3.6 resume os principais aspectos do simulador proposto por Machado (2002), Almeida (2003), Santana (2004), e Menezes (2006).

Aspecto	Valor
Problema a ser resolvido pelos agentes	Visitar os vértices de um grafo, percorrendo-o através de suas arestas
Arquitetura do <i>software</i>	<i>Stand-alone</i> (tanto os agentes quanto as classes de controle do simulador são compilados como um único aplicativo, que roda localmente e em auto-suficiência)
Visualização gráfica da simulação	Há um módulo de visualização dentro do aplicativo <i>stand-alone</i> , que está acoplado às classes internas de negócio da simulação
Interface de comunicação do ambiente com os agentes	Chamadas de métodos nativas da linguagem C++
Linguagem de programação dos agentes	Obrigatoriamente em C/C++
Quantidade de sociedades de agentes	Uma (1), a <i>sociedade de patrulheiros</i> (i.e. ambiente mono-social)
Quantidade de agentes	Ambiente multiagente cooperativo
Sincronismo entre percepções e ações	Ambiente síncrono – o simulador funciona em etapas: (i) todos os agentes percebem o ambiente, (ii) todos os agentes deliberam e agem, (iii) o ambiente é atualizado e o ciclo recomeça
Observabilidade do ambiente	A observabilidade do ambiente fica a critério dos agentes, que definem em código quais métodos de consulta ao ambiente devem chamar, e com quais limitações (alcance, etc.)
Confiabilidade das percepções	Percepções exatas
Confiabilidade das ações	Ações exatas
Tipo de comunicação entre os agentes	Inexistente, baseada em mensagens, baseada em marcas, baseada em <i>blackboard</i> (a critério dos agentes)
Confiabilidade da comunicação entre agentes	Comunicação confiável

Quadro 3.6 – Aspectos importantes do simulador de patrulhamento proposto por Machado (2002), Almeida (2003), Santana (2004) e Menezes (2005).

Sobre este simulador – e suas várias versões – ele ainda não constitui um recurso que permita de fato aos pesquisadores implementar suas técnicas facilmente e compará-las às anteriores. Alguém que decida utilizá-lo deverá estudar cuidadosamente seus mecanismos internos de funcionamento, devido ao alto acoplamento dos módulos de *software*. Ademais, caso esta pessoa decida adicionar novos comportamentos aos agentes outrora programados, ela precisará primeiramente conhecer os modelos internos de grafos, agentes, ações e percepções utilizados, além de dominar como os eventuais *objetos* de controle os manipulam. Em termos práticos, ela terá que *estender* um conjunto de *classes* específicas, sendo obrigada a codificar suas rotinas na linguagem C/C++.

Frente às limitações apresentadas para o simulador anterior, e de posse do conhecimento acerca dos simuladores da *RoboCup (Soccerserver)* e da TAC (mais especificamente da *TAC Classic, o Classic Server*), identifica-se uma oportunidade valiosa para reunir o que há de melhor nestes *softwares* e construir um simulador que consolide de fato um *testbed* para o problema do patrulhamento.

3.5. Considerações Finais

Este capítulo trouxe uma definição para o problema do patrulhamento, tendo apontado os seus parâmetros, bem como as métricas que foram utilizadas nos trabalhos anteriores para avaliar as soluções propostas.

Além disso, o estado-da-arte desta área foi apresentado através de um resumo das pesquisas de Machado (2002), Almeida (2003), Santana (2004), Chevaleyre (2005) e Menezes (2006).

Adicionalmente, tais estudos foram criticados quanto aos parâmetros que ainda não foram levados em consideração na proposta de novas estratégias de patrulhamento (a exemplo de territórios dinâmicos e heterogêneos, sociedades abertas de patrulheiros, e agentes com perdas energéticas associadas às suas ações e percepções). Houve ainda críticas quanto à não consideração do tempo de raciocínio gasto pelos agentes, para calcular as ociosidades dos vértices, e quanto à ausência de um simulador de patrulhamento unificado.

Por fim, este capítulo respondeu positivamente ao questionamento sobre a possibilidade e a qualidade do patrulhamento como um *benchmark* para SMA, destacando a importância de se estabelecer um *testbed* para o problema.

De fato, desenvolver um *testbed* associado ao patrulhamento – nos moldes do *Soccerserver* da *RoboCup* (ver seção 2.2.1), e do *Classic Server*, da TAC (ver seção 2.2.2) – confunde-se com a construção de um novo simulador unificado para o patrulhamento. Por isto, este trabalho propõe o *SimPatrol*, um simulador de SMA que oferece um *testbed* ao problema do patrulhamento, e contempla todos os requisitos outrora atendidos pelas versões do simulador de patrulhamento anteriores, além de trazer novas funcionalidades.

Em suma, tal programa, mais do que partir de interesses particulares de pesquisas por algum aspecto em especial do problema do patrulhamento, tem sua concepção baseada na definição do problema do patrulhamento multiagente em si, bem como seus parâmetros e respectivos domínios, métricas e o atual estado-da-arte (todos discutidos neste capítulo).

No próximo capítulo, os requisitos e a arquitetura deste *software* são apresentadas.

Capítulo 4

***SimPatrol*: Requisitos e Arquitetura**

Motivando-se pelo movimento de estabelecimento de *benchmarks* para a área de SMA (analisado no capítulo 2), e pela possibilidade do estabelecimento do patrulhamento como um destes *benchmarks*, este trabalho propõe um *testbed* que dê suporte à tarefa de patrulhamento: o *SimPatrol*.

Em linhas gerais, o desenvolvimento do *SimPatrol* representa a oportunidade de se minimizar, para os trabalhos futuros, a necessidade do pesquisador se preocupar com a representação do problema e, conseqüentemente, voltar-se mais ao patrulhamento em si.

De posse dos conhecimentos apresentados acerca do simulador da *RoboCup* (*Soccerserver*, ver seção 2.2.1), do simulador da TAC (mais especificamente da *TAC Classic*, o *Classic Server*, ver seção 2.2.2), e do simulador utilizado nos trabalhos de Machado (2002), Almeida (2003), Santana (2004) e Menezes (2006) – ver seção 3.3.6 – requisitos não-funcionais relevantes podem ser identificados para o *testbed*. Além disso, a definição do problema do patrulhamento e de seus parâmetros (apresentados no capítulo 3) indicam os requisitos funcionais a que este *software* deve atender.

Desta forma, este capítulo tem justamente como meta introduzir o referido *software* como um novo simulador de SMA voltado para o patrulhamento. Mais especificamente, pretende-se apontar os requisitos atendidos pelo simulador, com o intuito de definir claramente o que ele faz (seção 4.1), e apresentar a arquitetura do programa, com o intuito de esclarecer como ele funciona (seção 4.2). Na seção seguinte (4.3), considerações finais sobre o desenvolvimento do *testbed* são apresentadas.

4.1. Requisitos Atendidos pelo *SimPatrol*

Como apontado por Pressman (2001), os *requisitos* de um *software* definem os serviços que ele deve fornecer e dispõem sobre as restrições à operação do mesmo.

Em outras palavras, os requisitos estão relacionados ao *quê* um sistema faz, sem preocupações com o *como*.

Nestes termos, o objetivo desta seção é exatamente definir quais as capacidades do *SimPatrol* – enquanto um simulador de SMA voltado para a tarefa do patrulhamento – e especificar o porquê destas capacidades. Para tanto, decidiu-se dividi-las em dois conjuntos distintos: um relacionado às funções e serviços exercidos pelo programa (referentes aos seus *requisitos funcionais*), e um relacionado às circunstâncias sobre as quais ele deve operar (i.e. seus *requisitos não-funcionais*).

Na seqüência, cada um destes conjuntos é melhor apresentado.

4.1.1. Requisitos Funcionais

De maneira geral, os requisitos funcionais do *SimPatrol* são retirados dos parâmetros do problema do patrulhamento e seus respectivos domínios, bem como da definição do patrulhamento multiagente, suas métricas e o seu atual estado-da-arte (todos discutidos no capítulo 3).

De modo a organizar a apresentação destes requisitos, eles estão agrupados em oito (8) conjuntos, a saber:

- Requisitos de simulação de território;
- Requisitos de simulação de tempo;
- Requisitos de simulação de percepções;
- Requisitos de simulação de ações;
- Requisitos de simulação de comunicações;
- Requisitos de simulação de sociedades;
- Requisitos de simulação da vitalidade dos agentes;
- Requisitos de coleta de métricas.

Na seqüência, cada um destes grupos é discutido.

4.1.1.1. Requisitos de Simulação de Território

Nesta seção são definidas as capacidades que o *SimPatrol* possui para representar o território cujo patrulhamento pretende-se simular. O quadro 4.1 parte dos parâmetros do problema do patrulhamento relacionados ao seu território (ver seção 3.1.1.1) e apresenta as soluções implementadas no simulador.

Parâmetro	Domínio	Soluções
Continuidade do território	Discreta	<ul style="list-style-type: none"> Modelo interno de grafos.
	Contínua	<ul style="list-style-type: none"> Não necessário.
Dinamicidade do território	Estática	<ul style="list-style-type: none"> Grafos com <i>vértices</i> e <i>arestas permanentes</i> em tempo de simulação.
	Dinâmica	<ul style="list-style-type: none"> Grafos com <i>arestas dinâmicas</i>, que podem ficar <i>disponíveis</i> ou <i>indisponíveis</i> em tempo de simulação; Grafos com <i>vértices dinâmicos</i> que podem ficar <i>disponíveis</i> ou <i>indisponíveis</i>¹ em tempo de simulação.
Homogeneidade do território	Homogênea	<ul style="list-style-type: none"> Grafos cujos vértices são todos iguais quanto às suas prioridades de visitaç�o.
	Heterogênea	<ul style="list-style-type: none"> Grafos cujos vértices t�m um valor de prioridade de visitaç�o associado, expresso por um n�mero natural. Zero (0) � a prioridade m�xima.

Quadro 4.1 – Soluções para modelar o patrulhamento multiagente em respeito ao seu território.

Como se pode observar, o *SimPatrol* n o simula territ rios cont nuos, caracter stica que   condizente com a pr pria defini o do patrulhamento multiagente (ver se o 3.2): os territ rios s o mapeados para grafos, o que significa que eles sempre s o discretos, e os agentes funcionam como *tokens* itinerantes que percorrem estes grafos. Em rela o   dinamicidade dos v rtices e das arestas (para territ rios din micos), o planejado   que eventos de *se tornar dispon vel* e *se tornar indispon vel* sejam regidos por distribui es de probabilidade em fun o do tempo de simula o.

Tendo em vista o valor das prioridades de visita o (para territ rios heterog neos), dado que a probabilidade mais alta tem valor zero (0), tomando dois valores quaisquer relativos a dois v rtices distintos, o de maior prioridade ser  aquele cujo valor   mais pr ximo do zero. Adicionalmente, com o intuito de facilitar a diferencia o entre as regi es de interesse, representadas pelos v rtices, estes s o

¹ Sempre que um v rtice se torna indispon vel, suas arestas tamb m o ficam.

rotulados. De forma semelhante, em acordo com o trabalho de Almeida (2003) e pesquisas posteriores, as arestas são *valoradas*, de modo a aumentar a capacidade de representação da realidade (ver seção 3.3.2).

Por fim, partindo de um cenário em que patrulheiros vigiam uma cidade cujas ruas são eventualmente de sentido único, identifica-se a necessidade do *SimPatrol* contemplar *multigrafos orientados* (Gould, 1988) em seu modelo interno.

A figura 4.1 expressa o modelo de dados que dá suporte às soluções projetadas para a simulação de território. Trata-se de um *diagrama de classes* (Eriksson et al., 2004).

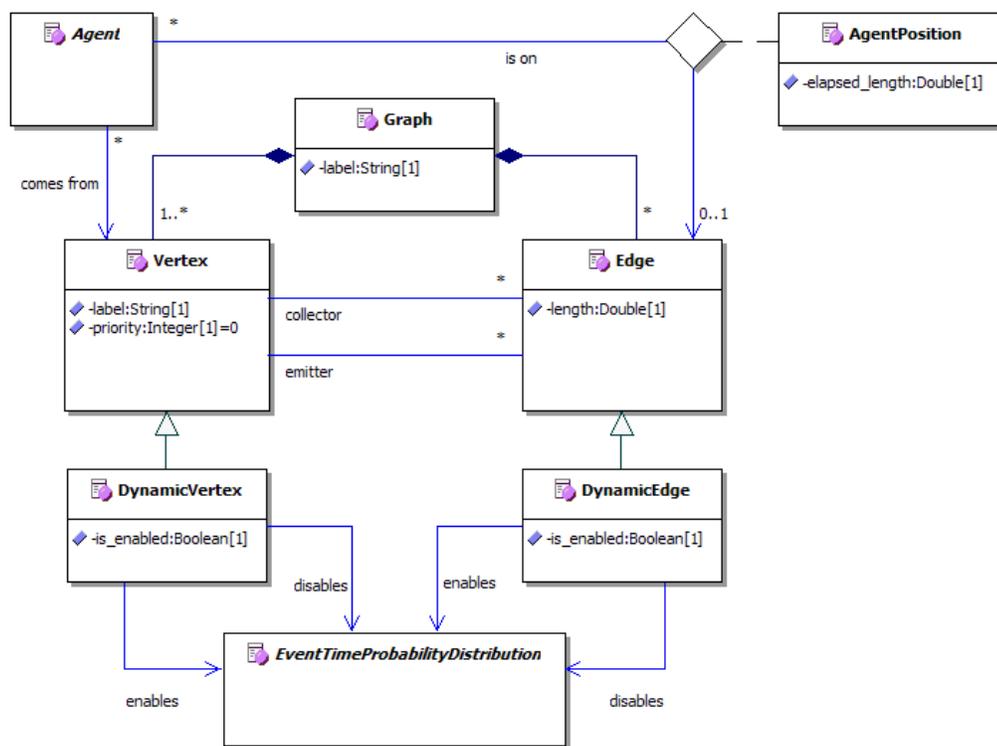


Figura 4.1 – Modelo de dados para a simulação de território.

4.1.1.2. Requisitos de Simulação de Tempo

Esta seção destina-se a apresentar as capacidades do *SimPatrol* relacionadas à contagem da passagem de tempo.

Como apontado por Hanks, Pollack e Cohen (1993), um modelo de tempo bem definido é imprescindível para qualquer sistema que almeja constituir um bom

testbed para um *benchmark*. Com o *SimPatrol*, em especial, a contagem de tempo é um aspecto determinante, dado que as métricas do problema por ele tratado baseiam-se no conceito de *ociosidade* (i.e. o tempo compreendido entre duas visitas consecutivas a um mesmo vértice – ver seção 3.2.1).

Levando-se em consideração as avaliações feitas sobre o desempenho das estratégias de patrulhamento anteriormente propostas (Machado, 2002; Almeida, 2003; Santana, 2004; Chevaleyre, 2005; Menezes, 2006), todas foram coletadas em versões de um simulador que conta o tempo em ciclos de *percepção-raciocínio-ação* dos agentes. A fim de que o *SimPatrol* consiga reproduzir tais resultados, o *software* possibilita a mesma contagem de tempo.

Por outro lado, como apontado na seção 3.3.6, dado que este modelo de tempo constitui uma das maiores críticas às pesquisas anteriores, o *SimPatrol* oferece a alternativa de medir a ociosidade dos vértices e o tempo de simulação em valores reais (i.e. medidos em segundos e frações de segundos). Isto possibilita que o tempo gasto pelos agentes para raciocinar sobre suas próximas ações impacte no cálculo das ociosidades dos vértices.

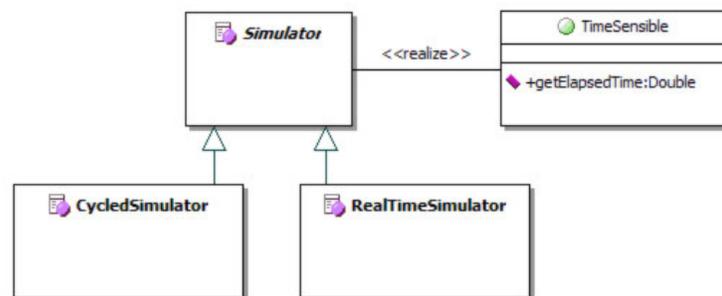


Figura 4.2 – Modelo de dados para a simulação de tempo.

O modelo de dados expresso na figura 4.2 apresenta as duas *especializações* do simulador (*classe Simulator*) que representa o *SimPatrol*. Uma delas – a *CycledSimulator* – conta o tempo segundo o ciclo de *percepção-raciocínio-ação* dos agentes. A outra, por sua vez, mede a passagem do tempo em valores reais (a *RealTimeSimulator*). Ademais, um simulador sempre *realiza* a *interface TimeSensible*, indicando que ele é sensível à passagem do tempo. A idéia por detrás desta solução é que cada uma das duas especializações do simulador implemente o *método getElapsedTime* à sua maneira, de modo a retornar corretamente o tempo de simulação já decorrido.

A figura 4.3 especifica a contagem de tempo para o simulador que sincroniza as percepções e as ações dos agentes (*Cycled Simulator*), tal qual os simuladores anteriores de patrulhamento. Trata-se de um *diagrama de atividades* (Eriksson et al., 2004).

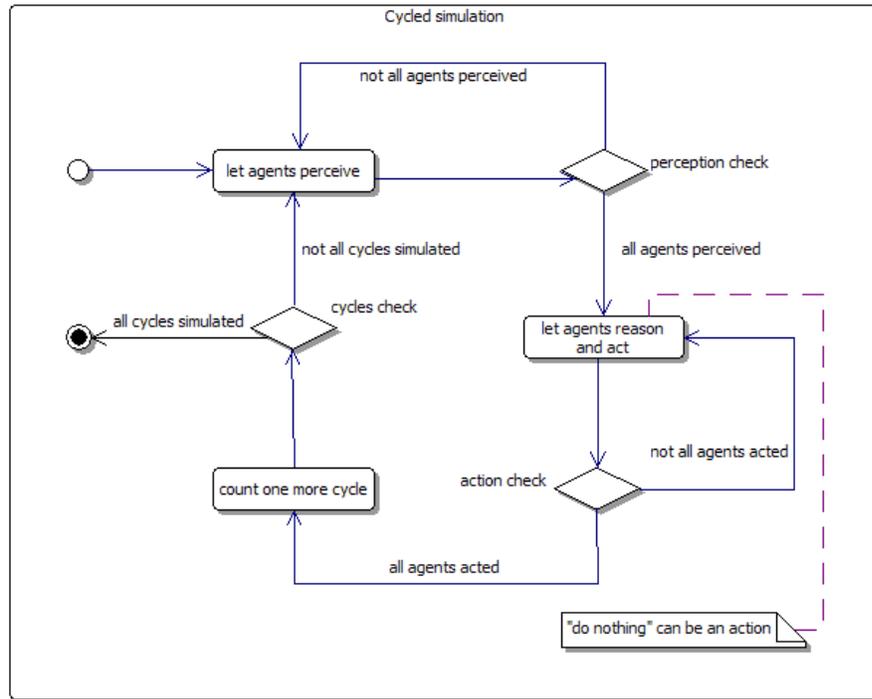


Figura 4.3 – Simulação de tempo em ciclos.

Já para o simulador que conta o tempo em valores reais (*RealTime Simulator*), os agentes percebem o ambiente segundo uma taxa específica – medida em *percepções por segundo* – e podem atuar sobre ele a qualquer momento; em respeito ao tempo de simulação, este tem uma duração específica em segundos (ou frações de segundos) e, portanto, a simulação tem uma hora certa para acabar.

4.1.1.3. Requisitos de Simulação de Percepções

Esta seção ocupa-se em definir as capacidades que o *SimPatrol* possui em respeito às percepções dos agentes. O quadro 4.2 contém dois (2) cenários motivadores e as respectivas soluções dadas para dotar o programa da habilidade de simulá-los.

Cenário	Soluções
Em uma aplicação de robótica, um robô está sendo construído para patrulhar um território inicialmente desconhecido. De que forma ele pode realizar esta atividade, dado que não possui nem ao menos um mapa do referido território?	<ul style="list-style-type: none"> • Agentes com capacidade de percepção do grafo que representa o território; • Agentes com capacidade de percepção de si mesmos, para determinar, entre outras coisas, suas posições no grafo.
Em uma aplicação de robótica, um robô está sendo construído para patrulhar um museu. De que forma ele pode realizar esta atividade, dado que não sabe quando ou onde um eventual intruso invadirá o território do referido museu?	<ul style="list-style-type: none"> • Agentes com capacidade de percepção de outros agentes no território.

Quadro 4.2 – Cenários e soluções para modelar o patrulhamento multiagente em respeito às percepções dos agentes patrulheiros.

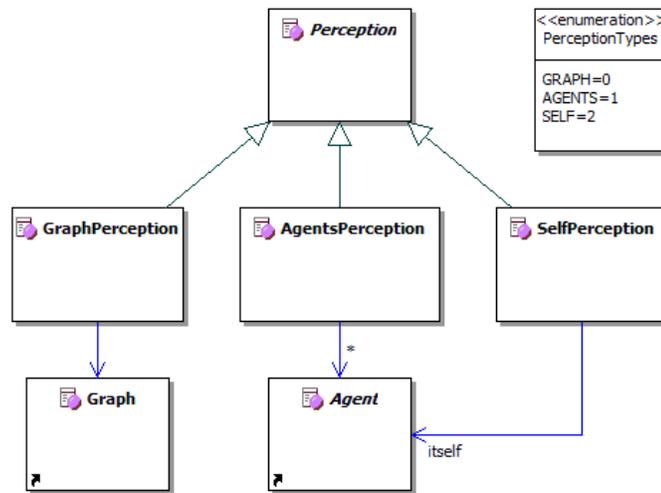


Figura 4.4 – Modelo de dados para a simulação de percepções.

A figura 4.4 expressa o modelo de dados que dá suporte a estas soluções. Na seqüência, o quadro 4.3 toma os parâmetros do problema do patrulhamento relacionados às percepções dos agentes (ver seção 3.1.1.2), e aponta mais soluções implementadas no simulador.

Parâmetro	Domínio	Soluções
Profundidade de percepção (PP) dos patrulheiros	$PP \in [0, \infty[$	<ul style="list-style-type: none"> • Simulador controla a profundidade de percepção dos patrulheiros baseado em configurações dos mesmos.
Confiabilidade das percepções	Percepções exatas	<ul style="list-style-type: none"> • Simulador não introduz erros nas percepções dos agentes.
	Percepções inexatas	<ul style="list-style-type: none"> • Não implementado.

Quadro 4.3 – Soluções para modelar o patrulhamento multiagente em respeito às percepções de seus agentes.

Como se pode observar, o *SimPatrol* pode controlar as percepções dos agentes baseando-se em valores de profundidade configuráveis. A idéia é que, para um agente capaz de perceber o território com profundidade um (1), por exemplo, o simulador só lhe proporcione porções do grafo a partir do vértice em que ele se encontra mais um nível adiante – ver figura 4.5 (a). De modo semelhante, este mecanismo pode ser aplicado ao se perceber os outros agentes – figura 4.5 (b).

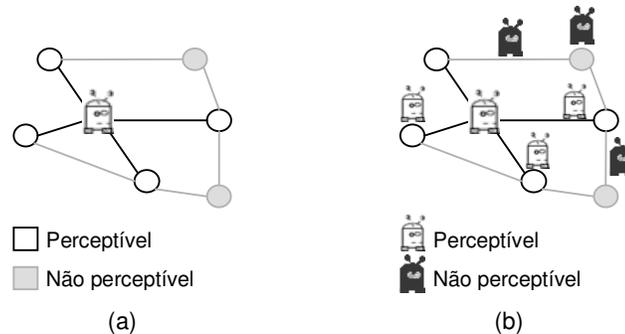


Figura 4.5 – Percepção do grafo (a), e percepção dos outros agentes (b), ambas com profundidade igual a um (1). O agente *perceptor* está no centro do território.

Além de controlar o modo como as percepções são fornecidas aos agentes, o *SimPatrol* ainda decide – também baseando-se em configurações – quais percepções cada um pode obter. Com isto, fica garantido que um patrulheiro nunca receba dados de percepções para as quais ele não esteja autorizado. Em outras palavras, ao contrário do que acontece com os simuladores anteriores da patrulha, fica a encargo do *SimPatrol* controlar o nível de observabilidade do ambiente, ao invés do pesquisador, que antes precisava definir no código de seus agentes o que consultar do ambiente e com quais limitações. A figura 4.6 traz o modelo de dados relacionado a estes controles.

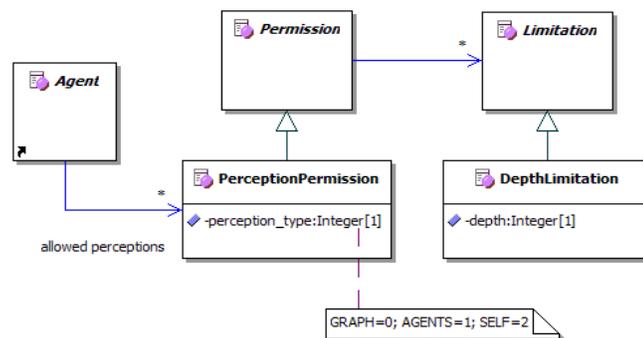


Figura 4.6 – Modelo de dados para o controle de percepções permitidas e suas respectivas limitações.

Partindo do diagrama de classes anterior, a idéia contida é que cada agente da simulação possua um conjunto de percepções permitidas (*allowed perceptions*) explicitamente definido. Tais percepções permitidas tratam-se na verdade da especialização de *permissões* (representadas pela classe *Permission*) que o *SimPatrol* deve verificar, e cada uma delas pode estar associada a um conjunto de limitações (representadas pela classe *Limitation*) que regem o modo como o simulador deve atendê-las. Nestes termos, a profundidade de uma percepção (correspondente a instâncias da classe *DepthLimitation*) nada mais é que uma especialização destas limitações.

Voltando ao quadro 4.3, pode-se observar que o *SimPatrol*, ao contrário do *Soccerserver* da *RoboCup* (ver seção 2.2.1), não introduz ruídos nas percepções dos agentes (i.e. as percepções são completamente confiáveis). Por restrições de tempo e espaço, nenhum modelo de perturbação das mesmas foi elaborado.

4.1.1.4. Requisitos de Simulação de Ações

Esta seção apresenta as capacidades que o *SimPatrol* possui em respeito às ações dos agentes.

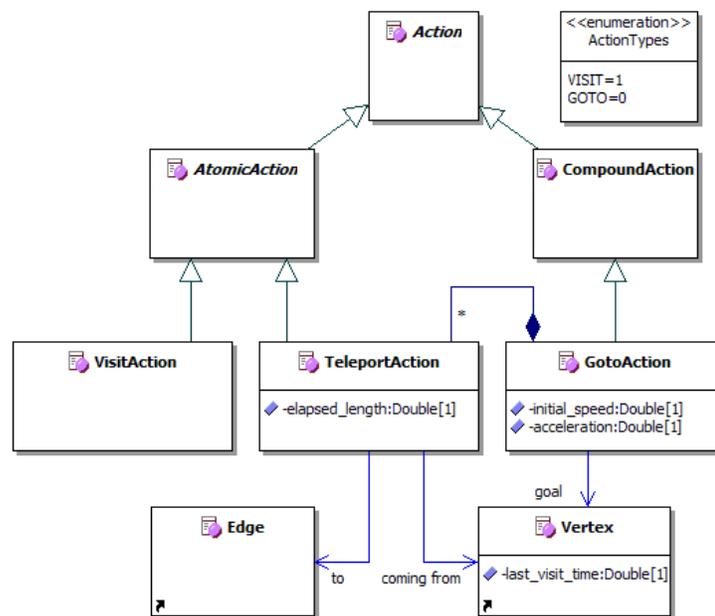


Figura 4.7 – Modelo de dados para a simulação de ações.

Tomando a definição do patrulhamento multiagente apresentada na seção 3.2, duas ações básicas são identificadas para os agentes. Lembrando que eles são *tokens* itinerantes que *se movem* sobre o grafo a ser patrulhando, *visitando* os vértices através das arestas, tais ações são *mover-se* e *visitar*. A figura 4.7 (na página anterior) expressa o modelo de dados que dá suporte a estes conceitos.

Como se pode observar, o *SimPatrol* divide as ações dos agentes em dois conjuntos distintos. Um conjunto diz respeito às *ações atômicas* (classe *AtomicAction*) que são instantâneas e têm seus efeitos aplicados sobre o ambiente com um único passo. É o caso da ação *visitar* (classe *VisitAction*), que registra no vértice corrente a hora da visita (em ciclos ou em segundos, dependendo da contagem de tempo – ver seção 4.1.1.2). De modo a suportar esta ação, a classe vértice (*Vertex*) ganha o *atributo last_visit_time*.

O outro conjunto está relacionado às ações compostas (classe *CompoundAction*), que – ao contrário das atômicas – têm seus efeitos aplicados sobre o ambiente com vários passos. É o caso da ação de se mover (classe *GotoAction*), que deve refletir o tempo gasto pelos patrulheiros para percorrer as arestas, segundo os seus comprimentos (*length*). Em termos gerais, o *SimPatrol* transforma a ação de se mover em uma seqüência de ações atômicas de teletransporte (classe *TeleportAction*), segundo a velocidade inicial (medida em unidades de *length* por ciclo ou por segundo) e a aceleração (medida em unidades de *length* por ciclo ao quadrado, ou por segundo ao quadrado) do movimento (atributos *initial_speed* e *acceleration* da classe *GotoAction*).

De forma semelhante à simulação de percepções, o *SimPatrol* limita o modo como os agentes executam suas ações e define quais ações um agente pode executar. Para tanto, há um modelo de permissões de ações e suas respectivas limitações, expresso na figura 4.8.

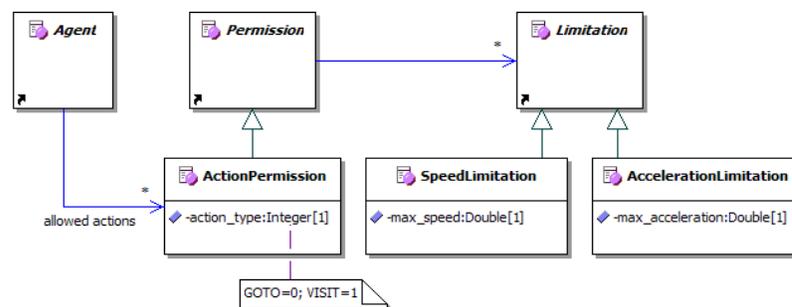


Figura 4.8 – Modelo de dados para o controle de ações permitidas e suas respectivas limitações.

Focando-se nas limitações de velocidade (classe *SpeedLimitation*) e aceleração (classe *AccelerationLimitation*), estas definem, respectivamente, a velocidade máxima e a aceleração máxima permitidas para o patrulheiro, ao se mover.

Por fim, o modelo contido no *SimPatrol* não prevê ruídos nos resultados esperados em decorrência da execução das ações (i.e. as ações são completamente confiáveis). Por restrições de tempo e espaço, nenhum modelo de perturbação das mesmas foi elaborado.

4.1.1.5. Requisitos de Simulação de Comunicações

Nesta seção são definidas as capacidades que o *SimPatrol* possui em respeito aos processos de comunicação dos agentes patrulheiros. O quadro 4.4 parte dos parâmetros do problema do patrulhamento relacionados à comunicação entre os agentes (ver seção 3.1.1.2) e apresenta as soluções implementadas no simulador.

Parâmetro	Domínio	Soluções
Tipos de comunicação	Mensagens	<ul style="list-style-type: none"> • Ação de propagação de mensagens pelo grafo, com uma profundidade específica de alcance; • Percepção de mensagens propagadas pelo grafo, desde que o perceptor esteja dentro do alcance das mesmas.
	Marcas	<ul style="list-style-type: none"> • Ação de deposição de marcas no vértice ou na aresta corrente; • Percepção de marcas depositadas nos vértices ou nas arestas da porção corrente do grafo percebida.
	<i>Blackboard</i>	<ul style="list-style-type: none"> • Adição da ociosidade compartilhada na percepção dos vértices.
	Inexistente	<ul style="list-style-type: none"> • Proibição de percepções e ações comunicativas.
Confiabilidade da comunicação	Confiável	<ul style="list-style-type: none"> • Percepções e ações comunicativas exatas.
	Não-confiável	<ul style="list-style-type: none"> • Não implementado.

Quadro 4.4 – Soluções para modelar o patrulhamento multiagente em respeito à comunicação entre os agentes patrulheiros.

Como se pode observar, o *SimPatrol* simula tal processo de comunicação através de pares de percepções e ações comunicativas. Com exceção da comunicação baseada em *blackboard*, que atualmente está resolvida através da

adição da *ociosidade compartilhada*² na percepção sobre os vértices, os demais tipos de comunicação revelam a necessidade de classes adicionais nos modelos de percepções e ações do simulador. Este aspecto pode ser observado na figura 4.9.

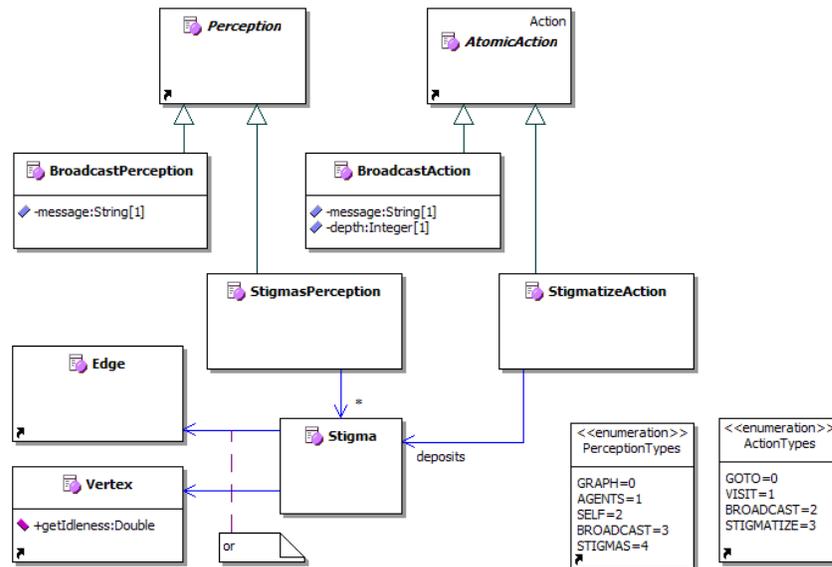


Figura 4.9 – Modelo de dados para a simulação de comunicações.

Tomando a situação de *comunicação inexistente*, o *SimPatrol* garante que os agentes não se comunicarão, ao lançar mão do mecanismo anteriormente descrito de permissões para percepções e ações (ver figuras 4.6 e 4.8). Em suma, um agente só é capaz de perceber mensagens e difundi-las, por exemplo, se tiver permissões para isso. Focando-se na ação de propagar mensagens (classe *BroadcastAction*), há ainda a possibilidade de impor limitações de profundidade à mesma (classe *DepthLimitation*, constante na figura 4.6).

Finalmente, sobre a confiabilidade das comunicações, o *SimPatrol* não prevê ruídos nas percepções, nem ruídos nos resultados esperados em decorrência da execução de ações comunicativas (i.e. as comunicações são completamente confiáveis). Por restrições de tempo e espaço, nenhum modelo de perturbação das mesmas foi elaborado.

² A *ociosidade compartilhada* diz respeito à ociosidade de um vértice que é calculada com base nas visitas de todos os agentes indistintamente. Para obtê-la, a classe *Vertex* oferece o método *getDlness* (ver figura 3.9).

4.1.1.6. Requisitos de Simulação de Sociedades

Esta seção destina-se a apresentar as capacidades que o *SimPatrol* possui em respeito ao modo como os agentes patrulheiros podem estar organizados em uma simulação. Desta forma, o quadro 4.5 parte dos parâmetros do problema do patrulhamento relacionados às *sociedades de patrulheiros* (ver seção 3.1.1.3) e apresenta as soluções que tornam possível reproduzir seus domínios no simulador.

Parâmetro	Domínio	Soluções
Quantidade de sociedades	Mono-social	• Modelo interno de <i>sociedade de agentes</i> .
	Multi-social	• Possibilidade da coexistência de várias sociedades de agentes em um mesmo ambiente a ser patrulhado.
Dinamicidade das sociedades	Fechada	• Sociedades fechadas cujos agentes são <i>perpétuos</i> (i.e. não podem morrer nem ser criados em tempo de simulação).
	Aberta	• Sociedades abertas cujos agentes são <i>temporários</i> (i.e. podem morrer e ser criados em tempo de simulação).
Homogeneidade das sociedades	Homogênea	• Agentes de uma mesma sociedade com as mesmas permissões para percepções e ações.
	Heterogênea	• Agentes de uma mesma sociedade diferentes entre si quanto às suas permissões para percepções e ações.

Quadro 4.5 – Soluções para modelar o patrulhamento multiagente em respeito às sociedades de patrulheiros.

A figura 4.10 expressa o modelo de dados que dá suporte às soluções projetadas para a simulação de sociedades.

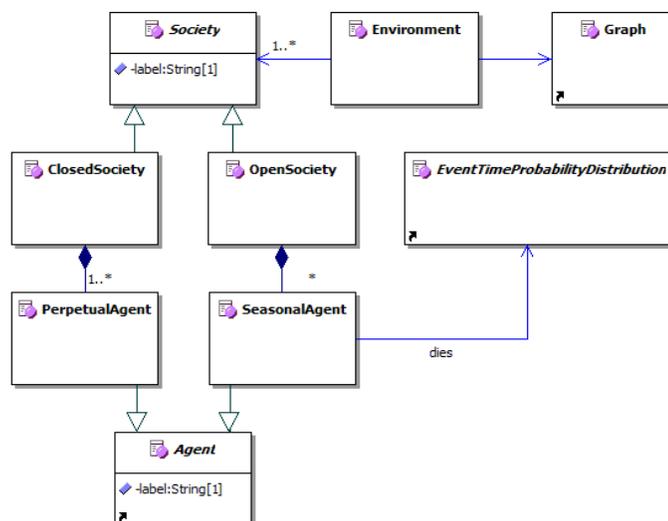


Figura 4.10 – Modelo de dados para a simulação de sociedades.

Como se pode observar, um *ambiente* de simulação (representado pela classe *Environment*) – além de *agregar* o grafo a ser patrulhado (classe *Graph*) – agrega uma ou mais sociedades de agentes (classe *Society*). Estas, por sua vez, podem ser abertas (*OpenSociety*) ou fechadas (*ClosedSociety*), reunindo agentes perpétuos (*PerpetualAgent*) e temporários (*SeasonalAgent*), respectivamente.

Focando-se nos agentes temporários, estes podem *morrer* segundo uma distribuição de probabilidade em função do tempo de simulação, e podem ser criados a qualquer momento, bastando, para isso, submeter ao simulador um *comando* de adição de agentes na sociedade adequada. Este mecanismo é melhor explicado na seção 4.2.2.

Por fim, o *SimPatrol* permite que os agentes de uma mesma sociedade sejam diferentes entre si ao atribuir, para cada um deles, um conjunto individual de permissões para percepções e ações (ver figuras 4.6 e 4.8). Desta forma, dentro de uma mesma sociedade, parte dos agentes pode ter permissão, por exemplo, para enviar mensagens, enquanto outra parte pode ter permissões exclusivas para se mover sobre o grafo.

4.1.1.7. Requisitos de Simulação da Vitalidade dos Agentes

Nesta seção são apresentadas as capacidades que o *SimPatrol* possui em respeito ao modo como os agentes podem perder energia ao perceber o ambiente e executar suas ações, e – de maneira semelhante – recuperar esta energia.

Como identificado no quadro 3.2, um dos parâmetros do problema do patrulhamento é justamente a vitalidade dos patrulheiros. Tomando por base o domínio deste parâmetro, mais os cenários descritos no quadro 4.6, as soluções utilizadas para dotar o simulador da habilidade de simulá-los são apresentadas neste mesmo quadro.

Cenário	Soluções
---------	----------

Em uma simulação de patrulhamento, um pesquisador pretende apenas avaliar o desempenho de sua estratégia, sem, no entanto, preocupar-se com as perdas e recuperações energéticas de seus agentes.

- Permissões para percepções e para ações sem limitações energéticas associadas.

Em uma experiência de robótica, um grupo de robôs equipados com baterias internas foi designado para patrulhar um labirinto. A cada percepção realizada, e a cada ação executada, uma certa quantidade de energia das baterias é gasta, exigindo que sejam distribuídos, ao longo do labirinto, pontos de recarga energética, para que os robôs mantenham sua autonomia. Nestes termos, de que forma o patrulhamento deve ser realizado, frente à questão do gasto de energia e a existência de pontos de recarga?

- Agentes dotados de nível de energia;
- Auto-percepção por parte dos agentes quanto ao seu nível de energia;
- Ação de recarga energética;
- Permissões para percepções e para ações associadas a limitações de gasto energético específicas;
- Impedimento de execução de ações e omissão de percepções por conta de níveis energéticos insuficientes;
- Grafos com vértices marcados como pontos de recarga energética.

Quadro 4.6 – Cenários e soluções para modelar o patrulhamento multiagente em respeito à vitalidade dos agentes patrulheiros.

A figura 4.11 expressa o modelo de dados que dá suporte às soluções projetadas para a simulação da vitalidade dos agentes. Como se pode observar, os agentes podem obter o seu nível interno de energia ao perceberem a si mesmos (classe *SelfPerception*, ver figura 4.4), consultando seus atributos de *stamina*.

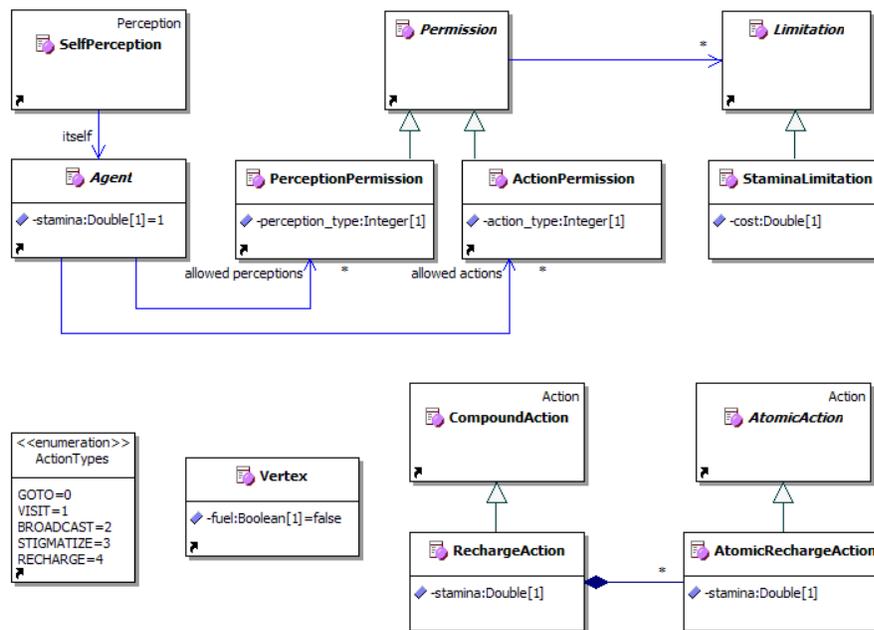


Figura 4.11 – Modelo de dados para a simulação da vitalidade dos agentes.

Sobre a ação de recarga energética, existe a classe *RechargeAction* que, tratando-se de uma ação composta, é resolvida pelo simulador em vários passos (i.e. várias ações atômicas de aumento do valor do atributo *stamina*, por meio da classe *AtomicRechargeAction*) que devem refletir o tempo gasto por um patrulheiro (em ciclos ou em segundos, dependendo do tipo de contagem de tempo) para se abastecer, sempre que estiver em um vértice cujo atributo *fuel* é verdadeiro. Opcionalmente, uma limitação de velocidade (classe *SpeedLimitation*, ver figura 4.8) – medida em unidades de *stamina* por ciclo, ou por segundo – pode ser associada à permissão para um agente se abastecer, de modo a controlar a recarga de sua *stamina*.

Por fim, para que o *SimPatrol* diminua o valor de *stamina* de um agente sempre que ele percebe o ambiente ou age, a cada permissão individual para uma determinada percepção ou ação pode ser associada uma limitação de *stamina* (classe *StaminaLimitation*), que expressa o custo (atributo *cost*) da ocorrência deste elemento.

O quadro 4.7 reúne todas as percepções e as ações possíveis para um patrulheiro, correlacionando-as com suas respectivas possíveis limitações.

Percepções	Limitações	Ações	Limitações
<i>GraphPerception</i>	<i>DepthLimitation</i> , <i>StaminaLimitation</i>	<i>GotoAction</i>	<i>SpeedLimitation</i> , <i>AccelerationLimitation</i> , <i>StaminaLimitation</i>
<i>AgentsPerception</i>	<i>DepthLimitation</i> , <i>StaminaLimitation</i>	<i>VisitAction</i>	<i>StaminaLimitation</i>
<i>SelfPerception</i>	<i>StaminaLimitation</i>	<i>BroadcastAction</i>	<i>DepthLimitation</i> , <i>StaminaLimitation</i>
<i>BroadcastPerception</i>	<i>StaminaLimitation</i>	<i>StigmatizeAction</i>	<i>StaminaLimitation</i>
<i>StigmasPerception</i>	<i>StaminaLimitation</i>	<i>RechargeAction</i>	<i>SpeedLimitation</i> , <i>StaminaLimitation</i>

Quadro 4.7 – Possíveis percepções e ações para um patrulheiro, com suas respectivas limitações.

4.1.1.8. Requisitos de Coleta de Métricas

Esta seção encerra a apresentação dos requisitos funcionais do *SimPatrol*, focando-se nas métricas do problema do patrulhamento a que o simulador dá suporte. Como

apontado na seção 3.2.1, são quatro (4) as métricas principais, todas calculadas com base na ociosidade dos vértices; esta, por sua vez, pode ser obtida através do método *getIdleness*, da classe *Vertex* (ver figura 4.9).

A figura 4.12 expressa o modelo de dados que dota o simulador da capacidade de coletar tais métricas.

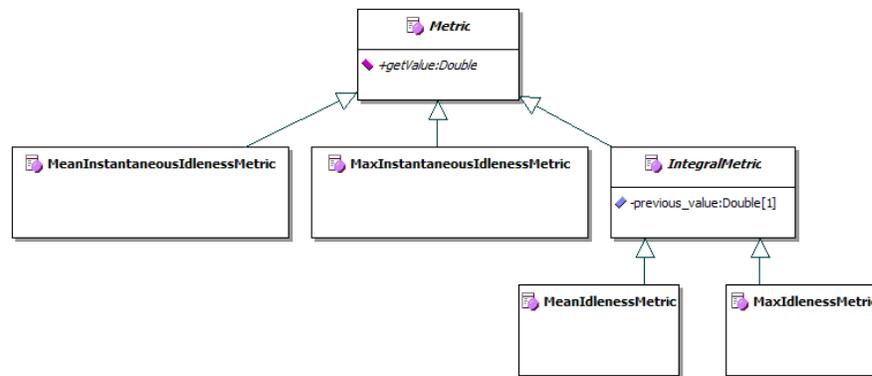


Figura 4.12 – Modelo de dados para a coleta de métricas.

Como se pode observar, cada métrica deve implementar o método *abstrato* *getValue*, que expressa o seu valor no momento corrente. Destacam-se, aqui, as *métricas integrais* (representadas pela classe *IntegralMetric*) que, não-episódicas, são coletadas a uma taxa específica e o cálculo do seu valor corrente depende dos valores anteriormente coletados (atributo *previous_value*).

Na seqüência, os requisitos não-funcionais do *SimPatrol* são discutidos.

4.1.2. Requisitos Não-funcionais

Como citado anteriormente, os requisitos não-funcionais – no escopo desta dissertação – referem-se às restrições sob as quais o *SimPatrol* deve operar. Tais requisitos têm como justificativa principal garantir uma boa qualidade para o simulador enquanto candidato a *testbed* para o problema do patrulhamento. Desta forma, grande parte destes requisitos é retirada da experiência obtida com os simuladores anteriores do patrulhamento, mais os *softwares Soccerserver (RoboCup)* e *Classic server (TAC)*, todos discutidos no capítulo 2.

O quadro 4.8 apresenta um lista de requisitos não-funcionais interessantes para o *SimPatrol* e resume como eles estão solucionados no programa.

Requisito não-funcional	Soluções
Baixo acoplamento dos módulos do <i>software</i> , dando-se atenção especial para a clara distinção entre o que é o simulador, o que são os agentes, e o que são as ferramentas de coleta de métricas e reprodução da simulação.	<ul style="list-style-type: none"> • Arquitetura cliente-servidor, onde o simulador comporta-se como o servidor, e os agentes e as ferramentas de coleta de métricas e reprodução da simulação comportam-se como os clientes.
Codificação dos agentes em qualquer linguagem ou tecnologia.	<ul style="list-style-type: none"> • Os agentes se comportam como clientes do <i>SimPatrol</i>, conectando-se a ele através de portas <i>UDP/IP</i> ou <i>TCP/IP</i> (Kurose e Rossi, 2004); • Por meio destas portas, os agentes recebem percepções do simulador, e a ele enviam ações que desejam executar; • Protocolos de comunicação baseados em mensagens escritas em XML.
Codificação das ferramentas de coleta das métricas do patrulhamento em qualquer linguagem ou tecnologia.	<ul style="list-style-type: none"> • Tais ferramentas se comportam como clientes do <i>SimPatrol</i>, conectando-se a ele através de portas <i>UDP/IP</i> e dele recebendo os valores das métricas; • Protocolos de comunicação baseados em mensagens escritas em XML.
Codificação das ferramentas de reprodução da simulação em qualquer linguagem ou tecnologia.	<ul style="list-style-type: none"> • Tais ferramentas se comportam como clientes do <i>SimPatrol</i>, conectando-se a ele através de portas <i>UDP/IP</i>, e dele recebendo dados para exibição da simulação; • Protocolos de comunicação baseados em mensagens escritas em XML.

Quadro 4.8 – Requisitos não-funcionais e suas respectivas soluções.

Focando-se na questão de o *SimPatrol* utilizar uma arquitetura cliente-servidor, a idéia básica é que o mesmo se comporte como um servidor ao qual se conectam três (3) tipos de clientes. O primeiro tipo se refere aos agentes patrulheiros, que recebem – do simulador – percepções do ambiente, e a ele enviam requisições por ações. O segundo tipo diz respeito às ferramentas de coleta das métricas do problema do patrulhamento, que recebem do servidor, em tempo de simulação, valores de uma das quatro métricas apresentadas na seção 3.2.1, a uma taxa de exemplares por segundo. Por fim, o terceiro tipo de cliente corresponde às ferramentas que recebem, do *SimPatrol*, dados em quantidade e qualidade suficientes para uma reprodução da simulação (de maneira semelhante ao *SocceMonitor* da *RoboCup*, ver seção 2.2.1).

Voltando-se a atenção para os clientes que correspondem aos agentes patrulheiros, o mecanismo obtido com esta solução é semelhante ao adotado no *Soccerserver* da *RoboCup* e no *Classic server* da TAC (ver seção 2.2): cada cliente deste tipo funciona como o cérebro de um único patrulheiro, recebendo do servidor as percepções adequadas, e requisitando ao mesmo a execução de ações, conforme a estratégia de patrulhamento implementada.

Em termos práticos, cada patrulheiro se conecta ao *SimPatrol* através de uma porta TCP/IP específica, caso a simulação realize a contagem de tempo em ciclos de *percepção-raciocínio-ação* dos agentes, ou por meio de uma porta UDP/IP específica, caso a simulação realize a contagem de tempo em valores reais. A preferência no uso de um protocolo em detrimento do outro, de acordo com a simulação de tempo, reside justamente nas diferenças entre os dois quanto à confiabilidade e a velocidade de transmissão dos dados.

Sucintamente, por assegurar a entrega correta das informações, sob pena de gastar mais tempo com a checagem de dados e estabelecimento de conexão (Kurose e Rossi, 2004), o protocolo TCP/IP é indicado para a simulação em ciclos, dado que esta não possui compromisso com a passagem de tempo em valores reais. Além disso, a utilização deste protocolo anula a possibilidade de um ciclo ser perdido em decorrência de transmissões falhas de percepções ou requisições por ações.

Por outro lado, tendo-se em mente as simulações em tempo real, o protocolo UDP/IP mostra-se mais indicado, graças a sua rapidez decorrente do controle simplificado dos dados transmitidos (Kurose e Rossi, 2004). Neste caso, levando em consideração que as percepções são enviadas aos agentes segundo uma taxa de exemplares por segundo, uma eventual transmissão incorreta de dados é compensada pelas transmissões corretas posteriores.

Por fim, sobre o formato das mensagens trocadas entre o servidor e os seus clientes, o mecanismo utilizado para organizar as informações é baseado em marcações (i.e. XML, tal qual o *Classic server* da TAC – ver seção 2.2.2). Sejam percepções enviadas pelo simulador aos agentes, sejam ações submetidas pelos agentes ao servidor, sejam as métricas enviadas às ferramentas de coleta das mesmas, tudo o que entre e sai do *SimPatrol* está em formato XML. A figura 4.13 especifica alguns dos principais elementos do simulador que realizam a interface *XMLable*, de modo a possuírem versões em XML.

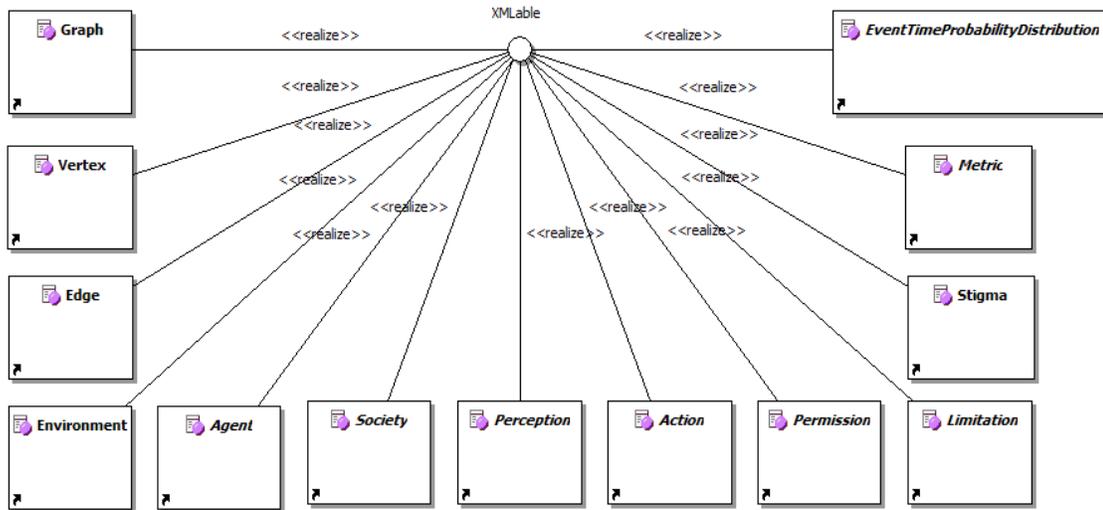


Figura 4.13 – Classes de objetos XMLáveis.

Para fins ilustrativos, o quadro 4.9 exemplifica a percepção de um agente sobre o grafo em que ele se encontra, em formato XML. O grafo correspondente à percepção encontra-se na figura 4.14.

```

<perception type="0">
  <graph label="graph_perceived">
    <vertex id="a" label="a" idleness="0" priority="0" fuel="false"/>
    <vertex id="b" label="b" idleness="8" priority="0" fuel="false"/>
    <vertex id="c" label="c" idleness="17" priority="0" fuel="false"/>

    <edge id="ab" emitter_id="a" collector_id="b" oriented="false" length="6"/>
    <edge id="bc" emitter_id="b" collector_id="c" oriented="false" length="4"/>
    <edge id="ca" emitter_id="c" collector_id="a" oriented="false" length="8"/>
  </graph>
</perception>
    
```

Quadro 4.9 – Exemplo de percepção sobre o grafo a ser patrulado.

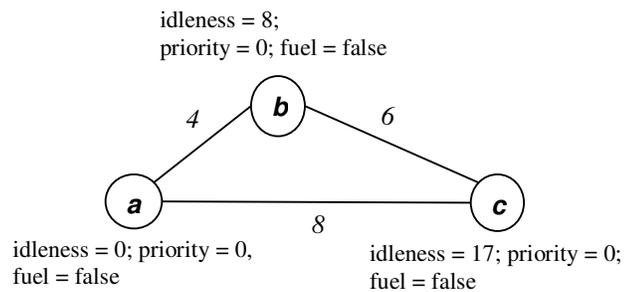


Figura 4.14 – Exemplo de grafo percebido por um agente.

O apêndice A contém as *DTD* (Deitel et al., 2001) que regulamentam o modo como os objetos presentes na figura 4.13 devem ser lidos e escritos em XML. Na seqüência, a arquitetura do *SimPatrol* é devidamente apresentada, com o intuito de esclarecer como ele funciona.

4.2. Arquitetura

O *diagrama de casos de uso* expresso na figura 4.15 sintetiza os requisitos atendidos pelo *SimPatrol*. O quadro 4.10 descreve os *atores* presentes neste diagrama.

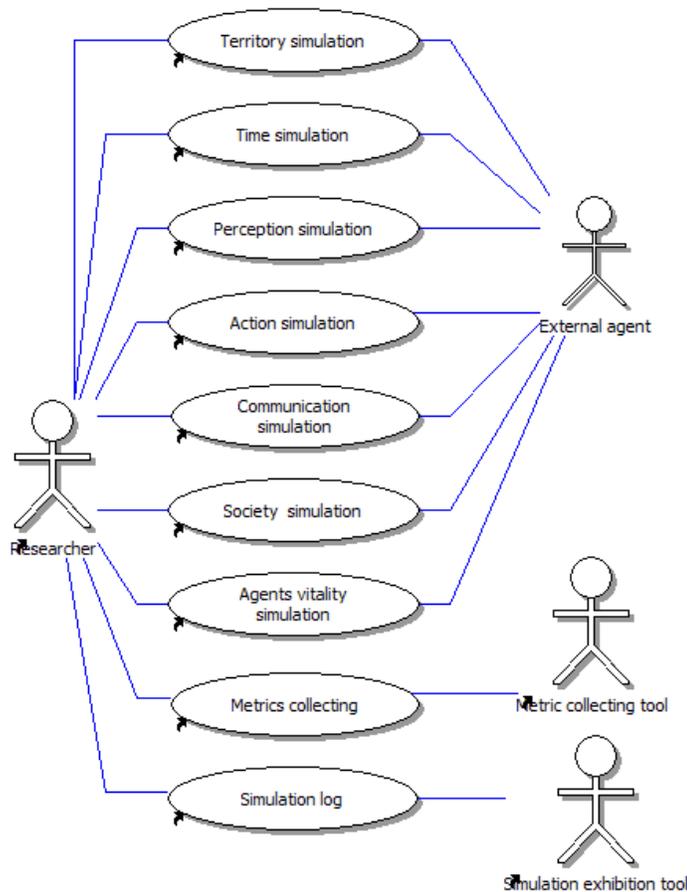


Figura 4.15 – Diagrama de casos e uso principal do *SimPatrol*.

Ator	Descrição
<i>Researcher</i>	Pesquisador interessado em simular uma estratégia de patrulhamento, e que interage com o simulador por meio de sua interface gráfica.
<i>External agent</i>	Cliente do <i>SimPatrol</i> responsável por controlar um dos patrulheiros durante uma simulação. Interage com o simulador por meio de uma porta TCP/IP ou UDP/IP, recebendo percepções e enviando ações ao programa.
<i>Metric collecting tool</i>	Cliente do <i>SimPatrol</i> responsável por coletar os valores de uma das métricas do problema do patrulhamento. Interage com o simulador por meio de uma porta UDP/IP, recebendo tais valores.
<i>Simulation exhibition tool</i>	Cliente do <i>SimPatrol</i> responsável por coletar os eventos decorrentes da execução de uma simulação. Interage com o simulador por meio de uma porta UDP/IP, recebendo tais eventos.

Quadro 4.10 – Atores presentes no diagrama de casos de uso principal do *SimPatrol*.

De posse destes requisitos, a arquitetura utilizada na construção do *SimPatrol* pode então ser apresentada, de modo a elucidar seu funcionamento. A figura 4.16 exibe uma visão estrutural do simulador. Trata-se de um *diagrama de pacotes* (Eriksson et al., 2004). Na seqüência, cada um destes *pacotes* é devidamente detalhado.

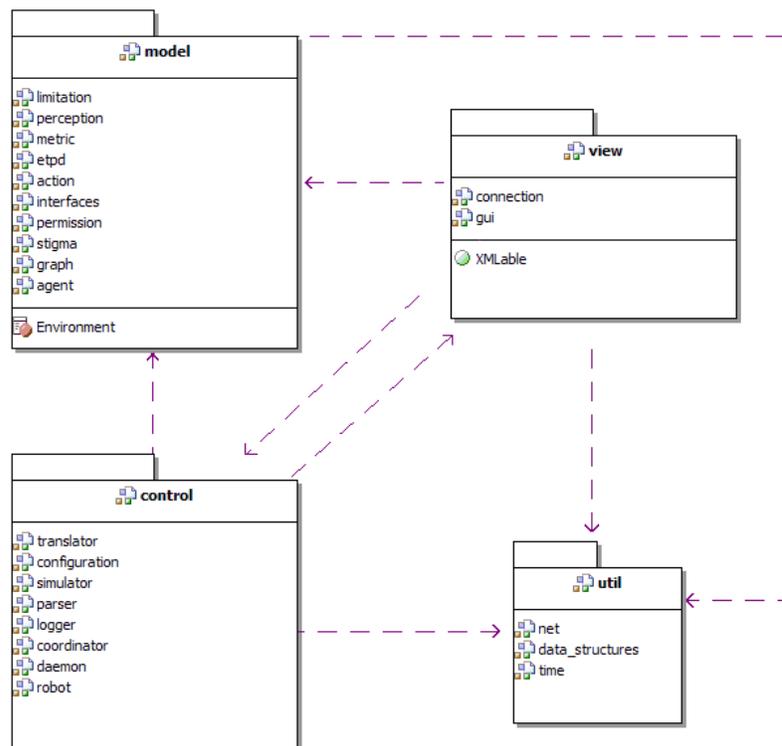


Figura 4.16 – Diagrama de pacotes do *SimPatrol*.

4.2.1. Pacote *model*

O quadro 4.11 lista o conteúdo do pacote de nome *model*. O objetivo deste pacote é agrupar as *classes de negócio* do simulador, i.e. as classes cujas instâncias constituem os elementos manipulados durante toda a simulação, a exemplo dos agentes e suas sociedades, suas percepções e ações, o grafo patrulhado, etc.

Item	Conteúdo
Pacote <i>model.action</i>	Classes que implementam as possíveis ações para os agentes em uma simulação.
Pacote <i>model.agent</i>	Classes que implementam os agentes ³ e as sociedades de agentes em uma simulação.
Pacote <i>model.etpd</i>	Classes que implementam as distribuições de probabilidade ⁴ da ocorrência de um evento, baseadas no tempo de simulação.
Pacote <i>model.graph</i>	Classes que implementam o grafo a ser patrulhado em uma simulação.
Pacote <i>model.interfaces</i>	Interfaces realizadas pelas classes do pacote <i>model</i> .
Pacote <i>model.limitation</i>	Classes que implementam as limitações impostas às permissões para percepções e para ações dos agentes, em uma simulação.
Pacote <i>model.metric</i>	Classes que implementam as métricas do problema do patrulhamento multiagente.
Pacote <i>model.perception</i>	Classes que implementam as possíveis percepções para os agentes de uma simulação.
Pacote <i>model.permission</i>	Classes que implementam as permissões para percepções e para ações específicas dos agentes, em uma simulação.
Pacote <i>model.stigma</i>	Classes que implementam os tipos de marcas (<i>stigmas</i>) passíveis de depósito sobre o grafo, por parte dos agentes, em uma simulação.
Classe <i>Environment</i>	Classe que implementa um ambiente de simulação, composto pelo grafo a ser patrulhado, mais as sociedades de agentes (ver figura 4.10).

Quadro 4.11 – Detalhamento do conteúdo do pacote *model*.

³ Deve-se observar que estes agentes não desempenham qualquer estratégia de patrulhamento. Eles funcionam apenas como peças de um tabuleiro, registrando o estado do patrulheiro que representam.

⁴ O gerador de números aleatórios e os distribuidores de números aleatórios foram todos aproveitados da biblioteca de nome *Colt* (<<http://public.web.cern.ch/Public/Welcome.html>>), produzida pela Organização Européia de Pesquisa Nuclear (CERN).

4.2.2. Pacote control

O quadro 4.12 detalha o conteúdo do pacote de nome *control*. O objetivo deste pacote é agrupar as classes cujas instâncias manipulam os objetos das classes de negócio, controlando o fluxo das simulações e assegurando a coerência do ambiente (correta posição dos agentes, resultado esperado para execuções de ações, percepções adequadas, coleta de métricas do patrulhamento, etc.)

Item	Conteúdo
Pacote <i>control.configuration</i>	Classes que implementam as <i>medidas</i> (i.e. <i>atitudes</i>) de configuração de uma simulação.
Pacote <i>control.coordinator</i>	Classe de nome <i>Coordinator</i> , cuja instância é responsável por sincronizar as percepções e as ações dos agentes, em uma simulação com contagem de tempo em ciclos (ver figura 4.3).
Pacote <i>control.daemon</i>	Classes que implementam os <i>daemons</i> do <i>SimPatrol</i> , i.e. os objetos autônomos que servem os clientes conectados ao simulador, sejam eles os agentes que controlam os patrulheiros da simulação, ou as ferramentas de coleta de métricas do patrulhamento, ou as ferramentas de exibição da simulação.
Pacote <i>control.event</i>	Classes que implementam os eventos decorrentes da execução de uma simulação e que servem de base para a exibição da mesma. Há eventos gerados pela movimentação de patrulheiros, visitas de vértices, indisponibilidade de vértices ou arestas, etc.
Pacote <i>control.parser</i>	Classe de nome <i>CompoundActionParser</i> , cuja instância é responsável por transformar as <i>ações compostas</i> (ver figura 4.7) enviadas pelos agentes em seqüências de <i>ações atômicas</i> , cada uma a ser executada em um momento específico da simulação.
Pacote <i>control.robot</i>	Classes cujas instâncias (chamadas de <i>robôs</i>) asseguram autonomamente a coerência do ambiente de simulação, controlando a dinamicidade dos vértices, das arestas (ver figura 4.1) e das sociedades abertas (ver figura 4.10), conforme as distribuições de probabilidade para tais eventos, e o tempo de simulação.
Pacote <i>control.simulator</i>	Classes que implementam os tipos de simuladores (simulador com contagem de tempo em ciclos, e simulador com contagem de tempo em valores reais) que constituem o <i>SimPatrol</i> (ver figura 4.2).
Pacote <i>control.translator</i>	Classes que implementam os mecanismos de recuperação de objetos a partir de mensagens recebidas em formato XML.

Quadro 4.12– Detalhamento do conteúdo do pacote *control*.

Focando-se no subpacote *control.configuration*, o destaque para o mesmo reside em seu conteúdo, que define as possíveis *medidas de configuração* de uma

determinada simulação. Em outras palavras, é neste contêiner onde estão implementadas as classes que, quando têm suas instâncias submetidas ao simulador, causam impactos sobre a execução de uma simulação (aqui simplesmente chamadas de *configurações*). A figura 4.17 contém o modelo de dados que dá suporte a este mecanismo.

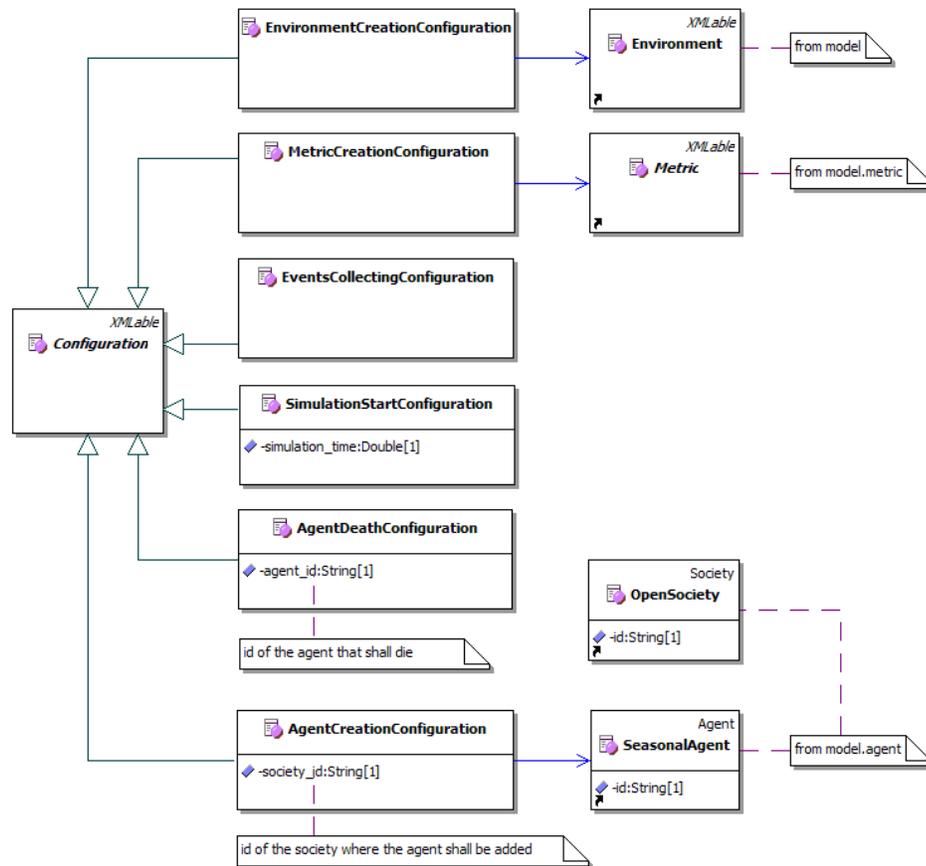


Figura 4.17 – Modelo de dados das medidas de configuração de uma simulação.

Como se pode observar, há configurações para submeter o ambiente de simulação (representada pela classe *EnvironmentCreationConfiguration*), inserir e retirar agentes de sociedades abertas (representadas, respectivamente, pelas classes *AgentCreationConfiguration* e *AgentDeathConfiguration*), determinar quais métricas do patrulhamento devem ser coletadas (classe *MetricCreationConfiguration*), possibilitar a coleta de eventos decorrentes da execução da simulação – que servem de base para a exibição da mesma – (classe *EventCollectingConfiguration*), e ordenar o início de uma simulação (classe *SimulationStartConfiguration*).

Por conta deste aspecto, o *SimPatrol* disponibiliza uma porta TCP/IP adicional voltada especificamente para receber tais configurações (coerentemente escritas em formato XML).

4.2.3. Pacote view

O quadro 4.13 detalha o conteúdo do pacote de nome *view*. O objetivo deste pacote é agrupar as classes cujas instâncias efetuam o *interfaceamento externo* do simulador com seus clientes, sejam eles os agentes que se conectam ao servidor, ou as ferramentas de coleta das métricas do patrulhamento, ou os aplicativos de exibição da simulação, ou mesmo os pesquisadores que o utilizam como um *testbed* para o problema do patrulhamento.

Item	Conteúdo
Pacote <i>view.connection</i>	Classes que implementam as conexões oferecidas pelo <i>SimPatrol</i> para trocar dados com os clientes que funcionam como os agentes, ou as ferramentas de coleta de métricas, ou as ferramentas de exibição da simulação.
Pacote <i>view.gui</i>	Classes que implementam a interface gráfica de usuário do simulador.
Interface <i>XMLable</i>	Interface que quando realizada por uma classe, dá a ela a capacidade de retornar versões suas em XML (ver figura 4.13).

Quadro 4.13 – Detalhamento do conteúdo do pacote *view*.

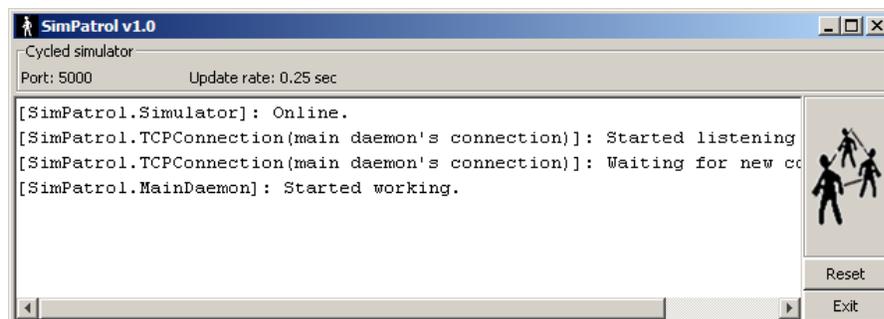


Figura 4.18 – Interface gráfica de usuário do *SimPatrol*.

A figura 4.18 mostra a interface gráfica do *SimPatrol*. Como se pode observar, existe um campo principal de exibição de informações textuais sobre a simulação

realizada, e dois botões principais, sendo um para reiniciar o simulador (botão *Reset*) – capaz de terminar uma eventual simulação em andamento – e um para sair do programa (botão *Exit*), capaz de fechar conexões e liberar todos os recursos ocupados pelo *software*.

Há ainda a possibilidade de se configurar a porta TCP/IP disponibilizada pelo programa para obter o envio de *configurações* (ver seção anterior). Na figura em questão, a porta utilizada é a 5000.

4.2.4. Pacote *util*

O quadro 4.14 detalha o conteúdo do pacote de nome *util*. O objetivo deste pacote é agrupar as classes utilitárias do simulador, a exemplo de estruturas de dados (filas e lista dinâmicas, *heaps* mínimos, etc.), portas TCP/IP e UDP/IP, e mecanismos de contagem de tempo (relógios, cronômetros, etc.).

Item	Conteúdo
Pacote <i>util.data_structures</i>	Classes que implementam as estruturas de dados utilizadas por todo o código do simulador.
Pacote <i>util.net</i>	Classes que implementam as portas TCP/IP e UDP/IP utilizadas nas conexões oferecidas pelo simulador.
Pacote <i>util.time</i>	Classes que implementam os mecanismos de contagem de tempo para o simulador (relógios, cronômetros, etc.).

Quadro 4.14 – Detalhamento do conteúdo do pacote *util*.

4.3. Considerações Finais

Este capítulo apresentou os requisitos e a arquitetura do *SimPatrol*.

Qualquer pessoa interessada em obter o simulador e seu código fonte pode fazê-lo através do endereço <<http://simpatrol.googlecode.com/svn/SimPatrol/code>>. De modo semelhante, um pesquisador interessado em sua modelagem e documentação pode obtê-los nos respectivos caminhos: <<http://simpatrol.googlecode.com/svn/SimPatrol/models>> e <<http://simpatrol.googlecode.com/svn/SimPatrol/docs>>. Há ainda a possibilidade de se

obter uma versão da documentação do *software* no formato *Javadoc* (Sun, 2008) em <<http://simpatrol.googlecode.com/svn/SimPatrol/javadoc>>.

Sobre os clientes que se conectam ao *SimPatrol*, há implementações de agentes que seguem as principais estratégias de patrulhamento descritas nos trabalhos de Machado (2002), Almeida (2003) e Chevaleyre (2005). Há ainda um cliente para coleta de métricas que guarda os valores em arquivos, e um cliente que, de forma semelhante, guarda os eventos decorrentes de uma eventual simulação em arquivos. Todos estes clientes podem ser obtidos no endereço <<http://simpatrol.googlecode.com/svn/SimPatrolClients>>, e existem exemplares desenvolvidos em *C++* e *Java*.

Aspecto	Valor
Problema a ser resolvido pelos agentes	Visitar os vértices de um grafo, percorrendo-o através de suas arestas
Arquitetura do <i>software</i>	Cliente-servidor (o <i>SimPatrol</i> é o servidor, e os agentes e as ferramentas de coleta de métricas e exibição da simulação são os clientes)
Visualização gráfica da simulação	Cliente UDP/IP que recebe do servidor eventos em formato XML decorrentes da execução da simulação
Interface de comunicação do ambiente com os agentes	Portas UDP/IP ou TCP/IP
Linguagem de programação dos agentes	Qualquer uma, desde que haja suporte à troca de mensagens em formato XML via portas UDP/IP ou TCP/IP
Quantidade de sociedades de agentes	Ambiente multi-social, com quantidade de sociedades configurável
Quantidade de agentes	Ambiente multiagente cooperativo
Sincronismo entre percepções e ações	Ambiente síncrono – para simulações com contagem de tempo baseada no ciclo <i>perceber-raciocinar-agir</i> dos agentes – ou ambiente assíncrono – para simulações com contagem de tempo baseada em valores reais (em segundos ou frações de segundos)
Confiabilidade das percepções	Percepções exatas
Confiabilidade das ações	Ações exatas
Observabilidade do ambiente	Ambiente parcialmente observável, com profundidade de percepção configurável e controlada pelo simulador
Tipo de comunicação entre os agentes	Configurável como inexistente, baseada em mensagens, baseada em marcas, ou baseada em <i>blackboard</i> , e controlada pelo simulador
Confiabilidade da comunicação entre agentes	Comunicação confiável

Quadro 4.15 – Aspectos importantes sobre o *SimPatrol*.

O quadro 4.15 (na página anterior) resume alguns aspectos importantes sobre o *SimPatrol*.

Por fim, existe um protótipo de ferramenta para configurar o ambiente de uma simulação. Com este aplicativo, há a possibilidade de se importar os grafos utilizados nas simulações dos trabalhos de Machado (2002), Almeida (2003), Santana (2004) e Menezes (2006), e configurar as sociedades de agentes, sejam elas abertas ou fechadas. Como saída, este programa gera um arquivo em formato XML que contém o ambiente a ser simulado. Ele está disponível em <<http://simpatrol.googlecode.com/svn/SimPatrolEnvironmentEditor>>.

Na seqüência, o capítulo 5 descreve alguns testes realizados com o simulador, apresentando os resultados obtidos.

Capítulo 5

Experimentos e Resultados

De posse dos requisitos planejados para o *SimPatrol*, e da arquitetura utilizada na construção do *software*, este capítulo tem como metas validá-lo (efetuando algumas comparações com as versões do simulador de patrulhamento anteriores, em termos de resultados produzidos), e trazer novas contribuições ao campo de pesquisa do patrulhamento (sobretudo quanto à reavaliação do desempenho de parte das arquiteturas de agentes anteriormente propostas, aqui em valores reais de ociosidade – i.e. medidas em segundos).

Para se alcançar tais objetivos, os recursos de *hardware* (máquinas, processadores, quantidades de memória RAM, configurações da rede de computadores, etc.) utilizados nos experimentos são apresentados na seção 5.1. Logo em seguida (seção 5.2), os experimentos realizados na tentativa de reproduzir os resultados de parte dos trabalhos anteriores são descritos, bem como os resultados efetivamente obtidos com o novo simulador, em decorrência da realização de tais experimentos, são apresentados.

Na seqüência (seção 5.3), descrevem-se os experimentos conduzidos na tentativa de reavaliar o desempenho de parte das arquiteturas anteriormente propostas, desta vez com a contagem da ociosidade dos vértices em segundos, e se apresentam os resultados obtidos com o novo simulador, em decorrência da realização destes experimentos.

Por fim, a seção 5.4 encerra o capítulo, trazendo algumas considerações finais sobre os experimentos realizados.

5.1. *Hardware* Utilizado nos Experimentos

Esta seção tem como objetivo descrever os recursos de *hardware* utilizados nos experimentos. Como explicado na seção 4.1.2, a arquitetura do *SimPatrol* é cliente-servidor: o simulador funciona como o servidor, ao qual clientes devem se conectar para controlar os agentes patrulheiros, ou receber os valores coletados para as

métricas do problema do patrulhamento, ou receber os eventos gerados em decorrência da simulação (i.e. movimentação dos agentes, visita de vértices, etc.).

Partindo deste aspecto, foi escolhido um computador pessoal comum para rodar exclusivamente o simulador e oferecer portas de conexão TCP/IP ou UDP/IP às demais máquinas do cenário. Aqui simplesmente chamado de *servidor*, este computador apresentava as seguintes configurações: processador *AMD Sempron* com velocidade de um gigahertz e oito décimos (1,8 GHz), e memória RAM de quinhentos e doze megabytes (512 MB), dos quais cento e vinte e oito megabytes (128 MB) estavam destinados ao simulador.

Para receber os valores coletados pelo servidor para as métricas do problema do patrulhamento, foi destinado um computador pessoal que rodava clientes capazes de guardar tais valores em arquivos. Aqui chamado de *cliente de métricas*, este computador possuía as seguintes configurações: processador *Intel Celeron* com velocidade de um gigahertz (1,0 GHz), e memória RAM de cento e vinte e oito megabytes (128 MB), inteiramente disponibilizada aos clientes em questão (havia um cliente para cada tipo de métrica, descritas na seção 3.2.1).

Em relação aos clientes de controle dos agentes patrulheiros, a configuração ideal seria a de se utilizar um computador para cada cliente, todos com as mesmas configurações de *hardware* (de modo a se garantir que um agente não viesse a apresentar vantagens de processamento sobre os demais). Todavia, devido a restrições nos recursos disponíveis, a configuração utilizada de fato foi rodar todos os clientes em uma mesma máquina, como processos concorrentes independentes e com a mesma prioridade. Sobre a máquina utilizada para tal (aqui chamada de *cliente de agentes*), lançou-se mão de um computador pessoal comum, com processador *Intel Pentium III* de oitocentos megahertz (800 MHz) e seiscentos e quarenta megabytes (640 MB) de memória RAM, igualmente disponibilizada para os agentes.

A figura 5.1 expressa a configuração descrita anteriormente. Como se pode observar, as três máquinas utilizadas faziam parte de uma mesma rede *ethernet* (Kurose e Rossi, 2004), ligadas fisicamente por um *switch* com capacidade de transmitir dados a uma velocidade de cem megabits por segundo (100 Mbps). De modo semelhante, cada computador contava com uma interface de rede *ethernet*, com capacidade de transmitir dados em modo *full duplex* (i.e. capaz de receber e enviar dados simultaneamente) a cem megabits por segundo (100Mbps).



Figura 5.1 – Configuração de *hardware* utilizada nos experimentos.

Por fim, todas as três máquinas rodavam o mesmo sistema operacional, a saber o *MS Windows XP Home Edition*.

5.2. Reprodução de Resultados do Simulador Anterior

Esta seção tem como objetivo avaliar a capacidade do *SimPatrol* em reproduzir os resultados apresentados pelos trabalhos anteriores do patrulhamento (Machado, 2002; Almeida, 2003; Santana, 2004; Chevaleyre, 2005; e Menezes, 2006).

Em suma, frente à grande quantidade de arquiteturas de agentes existentes, e à diversidade de cenários passíveis de reprodução no novo simulador, optou-se por escolher uma arquitetura em particular, e submetê-la ao conjunto de seis mapas proposto por Santana (2004) – ver seção 3.3.3 – coletando-se assim, com o novo programa, os valores para as métricas de ociosidade descritas na seção 3.2.1.

Desta forma, das várias arquiteturas disponíveis, decidiu-se por lançar mão dos agentes HPCC de Almeida (2003), e tentar reproduzir os resultados descritos no trabalho de Santana (2004) relacionados à aplicação desta arquitetura ao já citado conjunto de mapas. As razões para a escolha de tal arquitetura dizem respeito, principalmente, ao nível de complexidade dos seus agentes que – por sua simplicidade – permitiu um rápido desenvolvimento dos mesmos, bem como reduziu a possibilidade de haver grandes disparidades entre a implementação utilizada por Santana (2004), e a implementação utilizada neste trabalho.

Em linhas gerais, para a arquitetura de agentes HPCC, dentre vários experimentos conduzidos, Santana (2004) tomou populações de cinco agentes e gerou aleatoriamente, para cada mapa, um total de seis (6) configurações distintas

entre si quanto ao posicionamento inicial dos patrulheiros, resultando em um total de trinta e seis (36) cenários diferentes de cinco (5) agentes do tipo HPCC.

A figura 5.2, adaptada de Santana (2004), traz – para cada mapa – a média das ociosidades médias coletadas em cada uma das seis configurações iniciais geradas pelo pesquisador. Cada uma destas configurações foi submetida a simulações com um total de quinze mil (15000) ciclos de *percepção-raciocínio-ação* dos agentes, desprezando-se os três mil (3000) primeiros ciclos, referentes à chamada *fase transitória* (conforme descrito por Machado, 2002).

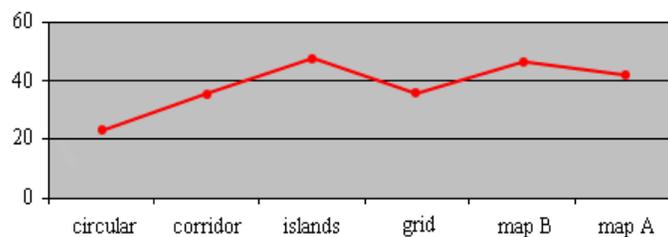


Figura 5.2 – Médias das ociosidades médias de cada mapa do conjunto proposto por Santana (2004), patrulhados por cinco agentes do tipo HPCC. Adaptado de Santana (2004).

Nestes termos, tendo em vista a reprodução destes resultados no *SimPatrol*, tomou-se cada um dos seis mapas e gerou-se aleatoriamente, para cada um deles, seis (6) configurações diferentes de cinco (5) agentes do tipo HPCC, resultando em um total de trinta e seis (36) cenários diferentes quanto ao mapa do território a ser patrulhado, e quanto à posição inicial dos cinco patrulheiros.

Mapa	circular	corredor	islands	grid	B	A
Configuração 01	44,9283	57,3095	73,2725	52,6673	66,8545	57,3867
Configuração 02	46,8724	57,6240	73,4945	54,7323	67,2092	58,8762
Configuração 03	46,3105	57,0189	73,0674	54,1355	66,5269	59,2638
Configuração 04	47,9327	58,7660	74,3005	55,8584	68,4966	58,5182
Configuração 05	46,9467	57,7041	73,5510	54,8112	67,2994	60,6707
Configuração 06	47,6663	58,4791	74,0980	55,5755	68,1731	59,3624
Média	46,7762	57,8169	73,6307	54,6300	67,4266	59,0130
Desvio padrão	1,0770	0,6758	0,4770	1,1439	0,7619	1,0809

Tabela 5.1 – Ociosidades médias coletadas em decorrência do patrulhamento exercido por cinco agentes do tipo HPCC, sobre os mapas do conjunto proposto por Santana (2004).

A tabela 5.1 (na página anterior) apresenta as ociosidades médias coletadas em cada um dos cenários, e as respectivas médias destes valores, com seus desvios padrão, organizados por mapa do território a ser patrulhado.

Semelhantemente aos experimentos de Santana (2004), cada cenário foi submetido a simulações de quinze mil (15000) ciclos, desprezando-se os três mil (3000) primeiros ciclos no cálculo das ociosidades médias. Tomando a informação de que, com a configuração de *hardware* apresentada na seção anterior, cerca de dois mil e quinhentos (2500) ciclos eram executados por hora, para cada um dos trinta e seis (36) cenários foram gastas cinco horas (5 h) de execução, perfazendo um total de cento e oitenta horas (180 h) de processamento total para a realização dos experimentos.

A figura 5.3 expressa as convergências da ociosidade média para as *configurações 01* de cada mapa, dentro dos quinze mil (15000) ciclos de execução de cada experimento. O último valor, no extremo de cada gráfico, corresponde exatamente aos valores da primeira linha da tabela 5.1.

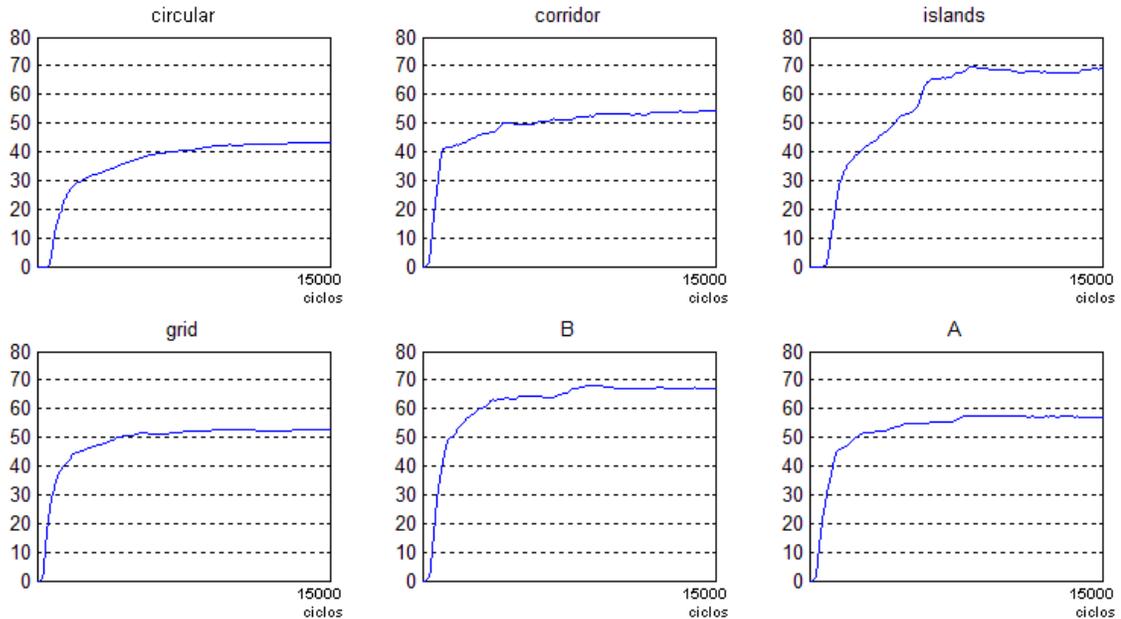


Figura 5.3 – Convergências das ociosidades médias em experimentos com as *configurações 01* (de cinco agentes do tipo HPCC), para cada mapa do conjunto proposto por Santana (2004).

A figura 5.4 reúne graficamente as médias das ociosidades médias, expressas na penúltima linha da tabela 5.1. Como se pode observar, com o

SimPatrol houve um acréscimo de cerca duas dezenas de ciclos para os valores de cada métrica coletada, quando comparados aos resultados apresentados por Santana (2004) – ver figura 5.2.

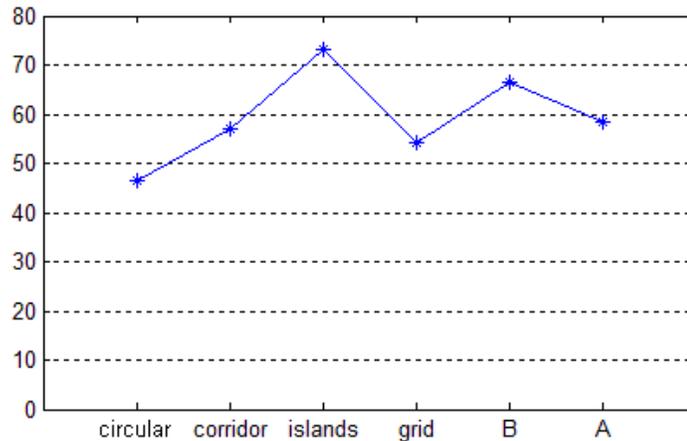


Figura 5.4 – Médias das ociosidades médias de cada mapa do conjunto proposto por Santana (2004), patrulhados por cinco agentes do tipo HPCC.

5.2.1. Discussão sobre a Reprodução de Resultados do Simulador Anterior

Tomando-se os valores relacionados à tentativa de reprodução dos resultados fornecidos pelas versões do simulador de patrulhamento anteriores, tem-se que destacar, primeiramente, a questão da convergência das ociosidades médias advindas dos experimentos assistidos pelo *SimPatrol* (ver figura 5.3). O fato de os agentes HPCC terem estabelecido uma estratégia de visitação dos vértices que, suficientemente estável, promoveu a convergência das ociosidades médias em cada mapa patrulhado, indica que há coerência por parte do novo simulador quanto ao fornecimento de percepções e atendimento de requisições de ações enviadas pelos agentes. Caso estas tarefas fossem feitas de modo incorreto, tais convergências seriam bastante prejudicadas, ou até mesmo impossibilitadas de ocorrer.

Voltando-se aos valores das médias das ociosidades médias obtidos (ver tabela 5.1 e figura 5.4), a sua relevância fica por conta do acréscimo de cerca de duas dezenas de ciclos, quando comparados aos valores apresentados por Santana

(2004). Razões para este fato residem sobretudo no modo como cada simulador (*SimPatrol* e versões do simulador de patrulhamento anteriores) conta os ciclos de *percepção-raciocínio-ação* dos agentes: por algum motivo, o *SimPatrol* conta mais ciclos que os outros programas no cálculo da ociosidade dos vértices.

Dado que não existe uma documentação unificada acerca destes outros *softwares*, nem se sabe ao certo qual das versões existentes foi utilizada por Santana (2004) para produzir os resultados aqui discutidos, fica difícil apontar uma causa concreta. Todavia, há suspeitas de que, em decorrência de o *SimPatrol* tratar as *visitas* aos vértices como ações, toda vez que este simulador atende a requisição por uma visita, ele encerra o ciclo do agente requisitante (não o impedindo, no entanto, de executar mais ações). Nas situações em que todos os demais agentes já agiram, o agente requisitante poderá não ter a oportunidade de iniciar imediatamente (i.e. no ciclo corrente) seu deslocamento para o próximo vértice, incorrendo em um atraso que adiciona uma unidade na ociosidade compartilhada do vértice objetivado.

Ainda assim, apesar das discrepâncias nos valores em questão, deve-se reiterar que, quando confrontadas as ociosidades obtidas com o *SimPatrol* entre si (e separadas por mapas), o comportamento da “função” resultante permanece o mesmo (basta comparar as figuras 5.2 e 5.4). Tendo sido o acréscimo praticamente o mesmo para todos os valores (cerca de duas dezenas de ciclos), conservou-se o comportamento dos cinco agentes HPCC perante cada um dos mapas, no seguinte sentido: eles apresentam um melhor desempenho ao patrulhar o mapa *circular*, têm praticamente a mesma desenvoltura ao patrulhar os mapas *corridor*, *grid* e *A* (que é inferior ao desempenho referente ao mapa *circular*), e produzem os valores mais altos de ociosidade para o mapa *islands*, seguido pelo mapa *B*.

O fato de este comportamento ter se conservado reforça o indício de que o *SimPatrol* simula corretamente a tarefa de patrulhamento em ciclos.

5.3. Resultados Considerando o Tempo de Raciocínio dos Agentes

Como apontado anteriormente (ver seção 3.3.6), uma das principais fontes de críticas aos trabalhos anteriores sobre o patrulhamento diz respeito ao método de contagem de tempo adotado, que utiliza os ciclos de *percepção-raciocínio-ação* dos agentes como unidade de medida. O problema deste mecanismo está no fato de ele ignorar o tempo real gasto pelos patrulheiros para tomar suas decisões: se uma estratégia escolhe de fato as melhores ações em cada turno, mas sob pena de gastar uma quantidade de tempo muito grande, ela não é penalizada de forma alguma, ainda que, em termos reais, os seus patrulheiros tenham demorado muito pensando entre uma visita e outra.

Assim, decidiu-se por planejar experimentos cujos resultados demonstrassem o real desempenho de parte das arquiteturas de agentes mais importantes. Em termos mais práticos, inspirando-se nos trabalhos de Santana (2004), Almeida et al. (2004) e Chevaleyre (2004) – que publicaram previamente comparações entre as principais arquiteturas até então propostas – o objetivo desta seção é selecionar algumas destas arquiteturas e as reavaliar no *SimPatrol*, medindo-se a ociosidade dos vértices em valores reais de contagem de tempo.

Para tanto, optou-se pela submissão, ao conjunto de seis mapas proposto por Santana (2004), das seguintes arquiteturas de agentes: agentes CR, CC, HPCC e SC (ver quadro 3.5).

Em termos gerais, os agentes CR foram escolhidos por constituírem a arquitetura reativa de melhor desempenho (Machado, 2002). Como tais patrulheiros gastam pouco tempo decidindo suas próximas ações (dado que são reativos), há um interesse em se observar o quanto eles são vantajosos quando comparados às arquiteturas mais sofisticadas de agentes.

Sobre as arquiteturas CC e HPCC, optou-se pela sua utilização devido ao curto tempo necessário para o desenvolvimento de seus agentes, e devido à complexidade de raciocínio dos seus agentes coordenadores (Almeida, 2003), cujo tempo gasto para determinar os objetivos dos patrulheiros nunca fora medido.

Em respeito aos agentes SC, estes foram escolhidos por constituírem a arquitetura que apresentara os melhores desempenhos nos testes comparativos realizados por Santana (2004), Almeida et al. (2004), e Chevaleyre (2004). Por fim,

devido a restrições de agenda e espaço, não foram coletados resultados para as demais arquiteturas.

Tomando as publicações de Santana (2004), Almeida et al. (2004) e Chevaleyre (2004), citadas anteriormente, lançar mão de cinco (5) agentes de cada arquitetura produz os resultados que melhor esclarecem o quanto uma é melhor que outra. Por este motivo, esta é a quantidade de patrulheiros utilizada nos experimentos. Desta forma, para cada um dos seis mapas, gerou-se aleatoriamente uma configuração de posicionamentos iniciais para os cinco agentes, que foi reproduzida em cada uma das quatro arquiteturas selecionadas. Sobre o tempo de simulação escolhido, optou-se por execuções de quatro horas (4h). O motivo para este valor advém dos quinze mil (15000) ciclos de simulação adotados por Santana (2004), Almeida et al. (2004) e Chevaleyre (2004), e na velocidade de deslocamento aqui escolhida para os agentes: cada um é posto a caminhar com velocidade igual a uma unidade de aresta por segundo.

Pensando-se comparativamente, no paradigma de ciclos, um agente leva – por exemplo – sete (7) ciclos para percorrer uma aresta com comprimento igual a sete (7). Em tempo real, com a velocidade citada para os experimentos, ele leva sete segundos (7 sec). Como neste sentido há equivalência entre ciclos e segundos, optou-se por realizar simulações de aproximadamente quinze mil segundos (15000 sec), que equivalem a quatro horas e dez minutos (4 h e 10 min), arredondadas para quatro horas. De modo semelhante, dado que, nas simulações em ciclos, as coletas dos valores das métricas descartavam os três mil (3000) primeiros ciclos, para o caso de tempo real, ignoraram-se os três mil primeiros segundos (3000 sec) de cada experimento.

	agentes CR	agentes CC	agentes HPCC	agentes SC
circular	29,1513	87,2319	52,3386	25,4400
corridor	52,4014	154,6092	91,3718	46,7775
islands	64,7430	171,9352	98,9715	43,5060
grid	115,0236	129,1508	89,7264	53,7276
B	94,8183	210,2792	100,0973	85,4906
A	98,7927	148,1586	101,4306	47,7132

Tabela 5.2 – Ociosidades médias, medidas em segundos, coletadas em decorrência do patrulhamento exercido por cinco agentes dos tipos CR, CC, HPCC e SC, sobre os mapas do conjunto proposto por Santana (2004).

A tabela 5.2 (na página anterior) reúne as ociosidades médias obtidas para cada uma das quatro arquiteturas selecionadas, frente a cada um dos seis mapas propostos por Santana (2004). A figura 5.5 toma estes valores e os organiza graficamente.

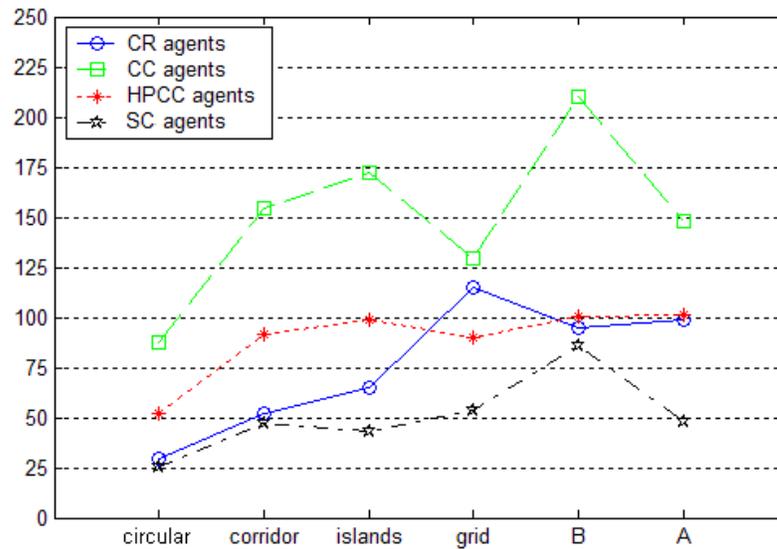


Figura 5.5 – Ociosidades médias, medidas em segundos, coletadas em decorrência do patrulhamento exercido por cinco agentes dos tipos CR, CC, HPCC e SC, sobre os mapas do conjunto proposto por Santana (2004).

Na seqüência, a tabela 5.3 apresenta os valores de ociosidade máxima coletados em cada experimento, para então serem dispostos graficamente na figura 5.6.

	agentes CR	agentes CC	agentes HPCC	agentes SC
circular	100,1526	267,2354	239,5031	73,1533
corredor	310,3995	512,0427	597,5876	119,8858
islands	285,4392	489,3159	601,4800	161,5751
grid	115,0236	432,7347	388,3317	112,0829
B	695,4324	749,9709	801,6993	402,5064
A	435,0007	439,2716	519,5568	144,1079

Tabela 5.3 – Ociosidades máximas, medidas em segundos, coletadas em decorrência do patrulhamento exercido por cinco agentes dos tipos CR, CC, HPCC e SC, sobre os mapas do conjunto proposto por Santana (2004).

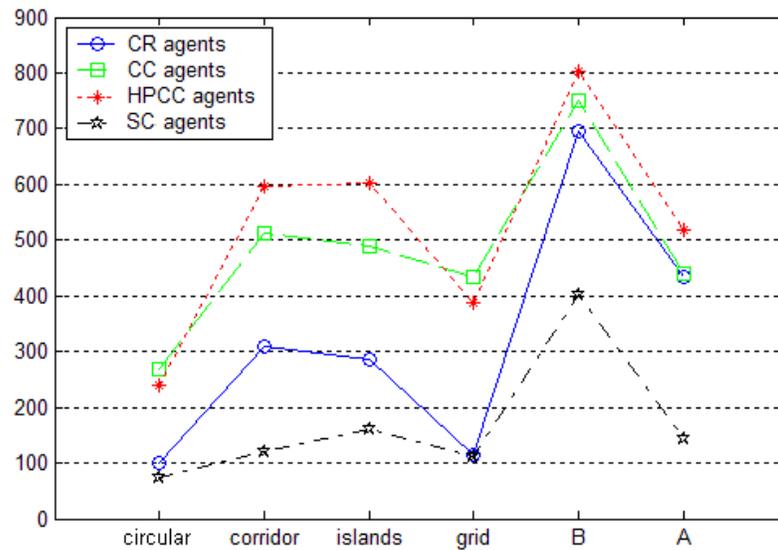


Figura 5.6 – Ociosidades máximas, medidas em segundos, coletadas em decorrência do patrulhamento exercido por cinco agentes dos tipos CR, CC, HPCC e SC, sobre os mapas do conjunto proposto por Santana (2004).

5.3.1. Discussão sobre os Novos Resultados

Os trabalhos de Santana (2004), Almeida et al. (2004) e Chevaleyre (2004) embasam o consenso atual da ordem de desempenho das arquiteturas de agentes patrulheiros consideradas mais importantes. A figura 5.7, adaptada de Santana (2004), expressa as ociosidades médias, medidas em ciclos de *percepção-raciocínio-ação* dos agentes, para as quatro arquiteturas avaliadas em tempo real nesta dissertação, utilizando-se cinco (5) patrulheiros de cada uma.

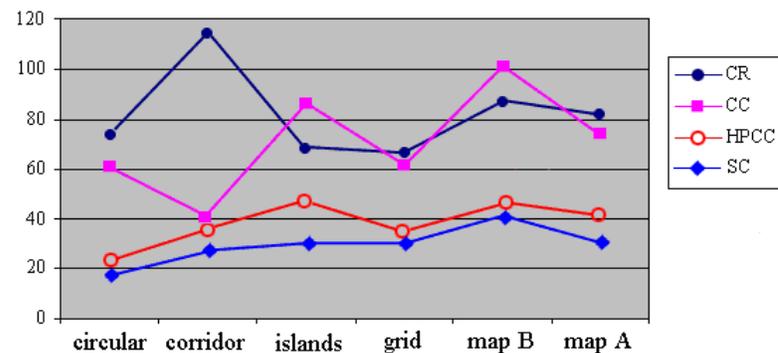


Figura 5.7 – Ociosidades médias, medidas em ciclos, coletadas em decorrência do patrulhamento exercido por cinco agentes dos tipos CR, CC, HPCC e SC, sobre os mapas do conjunto proposto por Santana (2004).

Como se pode observar, a arquitetura CR, por exemplo, apresenta um dos piores desempenhos, produzindo valores de ociosidade distantes dos obtidos com a arquitetura HPCC. Contudo, a medição inédita das ociosidades dos vértices em segundos produz uma reviravolta neste consenso: como se pode verificar na figura 5.5, a arquitetura de agentes CR produz, para alguns dos mapas, valores de ociosidade média inferiores aos da arquitetura HPCC.

Explicações para estes resultados residem justamente na consideração do tempo gasto pelos agentes para deliberar sobre suas próximas ações. Os agentes HPCC, além de contarem com um coordenador central que decide, para cada agente, qual o próximo vértice a ser visitado (constituindo um gargalo em potencial para o sistema), necessitam – uma vez definido o seu objetivo – planejar a melhor maneira de se deslocar até ele. Portanto, o tempo total de decisão destes patrulheiros compreende o tempo gasto para se comunicar com o coordenador, mais o tempo gasto para o coordenador calcular o vértice-objetivo (sendo, neste processo, requisitado concorrentemente por todos os patrulheiros), mais o tempo gasto por este coordenador para responder de volta ao agente, mais o tempo gasto pelo agente para planejar o caminho a ser traçado.

De contrapartida, os agentes CR simplesmente percebem a vizinhança e, em uma fração de segundos, escolhem como próximo vértice a ser visitado aquele de maior ociosidade. A rapidez destes agentes em seus processos deliberativos é tão determinante, que influencia inclusive nas ociosidades máximas decorrentes de seus patrulhamentos. Como se pode verificar na figura 5.6, os agentes CR superam os agentes HPCC (e CC) em todos os mapas, do ponto de vista das ociosidades máximas.

Em relação aos agentes SC, estes se mantiveram no patamar de melhor estratégia, considerando-se tanto as ociosidades médias, quanto as ociosidades máximas. A explicação para este fato tem a ver com os seus mecanismos internos de funcionamento: os patrulheiros calculam uma única vez a solução do problema do caixeiro viajante para o mapa em questão, e a partir dela traçam um planejamento, ao qual seguem por toda a simulação reativamente. Ademais, o tempo gasto para calcular o traçado inicialmente recai justamente nos ciclos da fase transitória, sendo ignorados no cálculo das ociosidades.

Curiosamente, para os mapas cujos vértices possuem em média os menores graus (*circular e corredor*), os agentes CR apresentaram performance similar aos

agentes SC, considerando-se as ociosidades médias coletadas (ver figura 5.5). Razões para este comportamento estão relacionadas à quantidade pequena de vértices e ociosidades a serem checadas pelos patrulheiros reativos em cada vértice visitado, devido às vizinhanças reduzidas.

5.4. Considerações Finais

Este capítulo descreveu os experimentos de patrulhamento realizados com o *SimPatrol*, e apresentou discussões sobre os resultados obtidos com tais experimentos.

Em suma, uma parte destes experimentos foi conduzida no sentido de reproduzir os resultados coletados com as versões do simulador de patrulhamento anteriores, em respeito ao patrulhamento efetuado por agentes do tipo HPCC, para cada um dos seis mapas propostos por Santana (2004). Conforme as discussões apresentadas, tais reproduções mostraram-se satisfatórias, comprovando o bom funcionamento do simulador.

A outra parte dos experimentos foi conduzida no sentido de avaliar o tempo de raciocínio gasto pelos agentes CR, CC, HPCC e SC (cinco de cada) para patrulhar cada um dos seis mapas anteriormente citados, e o impacto deste tempo na coleta das métricas do patrulhamento. Conforme as discussões apresentadas, os agentes CR – reativos – apresentaram melhor desempenho que os agentes CC e HPCC, contrariando os resultados apresentados por Santana (2004), Almeida et al. (2004) e Chevaleyre (2004). Nestes trabalhos, os simuladores utilizados não consideravam o tempo gasto pelos agentes para decidir suas próximas ações, na medida da ociosidade dos vértices do grafo a ser patrulhado.

De posse dos experimentos e resultados aqui apresentados, mais as informações constantes nos capítulos anteriores, conclusões e trabalhos futuros são apontados no próximo capítulo.

Capítulo 6

Conclusões e Trabalhos Futuros

Este trabalho discutiu o estabelecimento do patrulhamento como um *benchmark* para SMA, e apresentou um simulador de SMA construído especificamente para a tarefa de patrulhamento, cujo nome convencionou-se chamar *SimPatrol*. Em decorrência de tais atividades, houve contribuições para o campo de estudos do patrulhamento (expostas na seção 6.1), e a identificação de novos horizontes de pesquisa na área (apontadas na seção 6.2).

6.1. Contribuições para o Patrulhamento

As contribuições desta dissertação para o problema do patrulhamento foram as seguintes:

- Revisão da definição do problema do patrulhamento, que ficou restrita à questão da minimização do intervalo de tempo compreendido entre duas visitas a uma mesma região (considerando-se todas as regiões de interesse do espaço), independentemente da natureza do evento patrulhado (proteção contra intrusos, supervisão de salas, etc.);
- Revisão dos parâmetros de entrada do problema do patrulhamento e redefinição dos seus respectivos domínios, atividade essencial para o levantamento dos requisitos funcionais do novo simulador;
- Apresentação do patrulhamento como um bom *benchmark* para SMA, apontando-se a necessidade da consolidação de um *testbed* para que este processo ocorra (o *SimPatrol* constituiu justamente um primeiro passo neste sentido);
- Resumo do estado-da-arte de *benchmarks* para SMA, representado pelas competições *RoboCup* e TAC, destacando-se características importantes dos seus *testbeds*, que nortearam a definição de requisitos não-funcionais para o *SimPatrol* (em especial a arquitetura cliente-servidor);

- Especificação e implementação do *SimPatrol*, *software* que sana a necessidade identificada por Menezes (2006) de se aprimorar as versões do simulador de patrulhamento anteriores, de modo a dotá-las da capacidade de emular ambientes heterogêneos (i.e. com prioridades de visitação distintas para os vértices) e dinâmicos (i.e. com sociedades abertas, e vértices e arestas passíveis de desaparecimento em tempo de simulação);
- Avaliação em tempo real do desempenho de parte das principais arquiteturas de agentes patrulheiros, abrindo-se espaço para a resolução das críticas mais relevantes sobre as pesquisas anteriores, que não consideravam o tempo de raciocínio gasto pelos agentes, na contagem da ociosidade dos vértices do grafo a ser patrulhado.

Na seqüência, trabalhos futuros são apontados para o estudo do patrulhamento.

6.2. Trabalhos Futuros

O problema do patrulhamento é um assunto que vem sendo estudado pelo grupo de pesquisas em IA do Centro de Informática (CIn) da UFPE há algum tempo. Além dos trabalhos de Machado (2002), Almeida (2003), Santana (2004) e Menezes (2006), que correspondem a dissertações de mestrado, há ainda vários artigos publicados, a exemplo dos textos de Machado et al. (2002a, 2002b), Almeida et al. (2003, 2004), Santana et al. (2004), Menezes, Tedesco e Ramalho (2006), e Moreira et al. (2007).

Contemporaneamente a esta dissertação, há ainda uma outra de autoria do pesquisador Luiz Josué da Silva Filho que, sob orientação das professoras Patrícia Tedesco e Liliâne Salgado, desenvolve novas arquiteturas de agentes capazes de particionar o grafo a ser patrulhado em porções menores e conseqüentemente mais fáceis de serem percorridas.

Nestes termos, partindo de todo este cenário, fica claro o fato de que esta dissertação faz parte de um corpo de pesquisas maior e cabe, neste momento, enfatizar quais as lacunas e quais as portas que se abrem com a sua conclusão. Em especial, dado o produto final deste trabalho – o *SimPatrol* – há uma forte esperança

de que ele se consolide como *testbed* para o patrulhamento multiagente e garanta aos pesquisadores vindouros a possibilidade de se focar no problema em si, deixando a encargo do *software* representar o território a ser patrulado, efetuar transmissões de mensagens, manter a coerência nas posições dos patrulheiros, calcular as ociosidades dos vértices e limitar as ações e percepções dos agentes com base em configurações.

Identifica-se aqui, portanto, quatro tipos de trabalhos futuros possíveis, decorrentes das contribuições desta dissertação:

- Aprimoramento do *testbed* constituído pelo *SimPatrol*;
- Desenvolvimento de novas arquiteturas de agentes patrulheiros;
- Reavaliação do desempenho das arquiteturas existentes, levando-se em consideração o tempo de raciocínio gasto pelos patrulheiros;
- Desenvolvimento de soluções para patrulhar ambientes dinâmicos.

Sobre o primeiro item listado, relacionado ao próprio *SimPatrol* em si, este se refere a extensões e melhorias do programa, dando-se atenção especial a questões como adição de modelos de ruídos nas percepções, ações e comunicações dos agentes, ou mesmo a construção de ferramentas que venham a reforçar o seu papel de *testbed* para o patrulhamento. São elas: visualizadores em tempo real da simulação (nos moldes do *Soccermonitor*, da *RoboCup*), *plotters* dos valores coletados para as ociosidades em tempo real, ferramentas de configuração dos parâmetros de entrada do problema do patrulhamento, etc.

O segundo item, por sua vez, refere-se a técnicas que ainda não foram aplicadas no desenvolvimento de novas arquiteturas de agentes, a exemplo de *otimização por colônias de formigas* (Dorigo e Stützle, 2004) ou mesmo *algoritmos genéticos* (Michalewicz, 1996). Acredita-se que, com o *SimPatrol*, estas pesquisas sejam facilitadas, já que os pesquisadores não precisarão mais desenvolver seus próprios simuladores.

Sobre o terceiro item, este tem a ver com a capacidade do simulador em coletar as ociosidades dos vértices em valores reais de contagem tempo. Há a necessidade de se reavaliar todas as arquiteturas já propostas para que se estabeleça um novo consenso sobre quais estratégias apresentam, de fato, os melhores desempenhos, e em que condições.

Por fim, o quarto item diz respeito às novas possibilidades oferecidas pelo *SimPatrol* para a configuração de parâmetros de entrada do problema do

patrulhamento. Com o oferecimento de cenários nunca antes estudados na área em questão, como elaborar e avaliar arquiteturas que considerem os parâmetros de dinamicidade e heterogeneidade do território a ser patrulhado, de vitalidade dos patrulheiros, e dinamicidade das sociedades, todos apontados nesta dissertação?

Apêndice A

DTD de objetos XMLáveis no SimPatrol

```

<!--
ACTION

type = 0: teleport
* parameter "vertex_id" is the id of the vertex to where the agent
  shall be teleported (required in this case);
* parameter "edge_id" is the id of the edge to where the agent shall
  be teleported (optional in this case, its default value is null);
* parameter "elapsed_length" is the length already trespassed by the
  agent on the edge (optional in this case, its default value is 0).

type = 1: go to
* parameter "vertex_id" is the id of the vertex to where the agent
  shall go to (required in this case);
* parameter "initial_speed" contains the speed of the agent at
  the beginning of the movement (its default value is -1);
* parameter "acceleration" contains the acc of the movement
  (its default value is -1).

type = 2: visit
* no parameters required.

type = 3: broadcast message
* parameter "message" contains the message to be broadcasted
  (its default value is null);
* parameter "message_depth" contains the depth the message reaches
  on the graph of the simulation (its default value is -1).

type = 4: stigmatize
* no parameters required.

type = 5: recharge
* parameter "stamina" contains the value of stamina to be added to the
  agent (its default value is 0);

type = 6: atomic recharge
* parameter "stamina" contains the value of stamina to be added to the
  agent (its default value is 0);
-->

<!ELEMENT action (EMPTY)>
<!ATTLIST action type (0 | 1 | 2 | 3 | 4 | 5 | 6) #REQUIRED>
<!ATTLIST action vertex_id CDATA "null">
<!ATTLIST action edge_id CDATA "null">
<!ATTLIST action elapsed_length CDATA "0">
<!ATTLIST action initial_speed CDATA "-1">
<!ATTLIST action acceleration CDATA "-1">
<!ATTLIST action message CDATA "null">
<!ATTLIST action message_depth CDATA "-1">
<!ATTLIST action stamina CDATA "0">

```

```

<!--
CONFIGURATION

type = 0: environment's creation
* parameter attribute ignored;
* the body is an environment;

or

* the parameter is the path of a file containing the environment;
* body ignored.

type = 1: agent's creation
* the parameter attribute is the id of the society where the agent must
be added;
* the body is an agent.

type = 2: metric's creation
* parameter attribute is the duration, in seconds, of a cycle of
measurement of the metric (its default value is 1 sec);
* the body is a metric.

type = 3: simulation's start
* parameter attribute is the time of simulation;
* body ignored.

type = 4: agent's death
* the parameter attribute is the id of the agent that shall die;
* body ignored.

type = 5: event's collecting
* parameter attribute ignored;
* body ignored.
-->
<!ELEMENT configuration (environment | agent | metric | EMPTY)>
<!ATTLIST configuration type (0 | 1 | 2 | 3 | 4 | 5) #REQUIRED>
<!ATTLIST configuration parameter CDATA #IMPLIED>

<!-- ORIENTATION -->
<!ELEMENT orientation (ort_item*)>
<!ATTLIST orientation message CDATA #IMPLIED>

<!-- ORT ITEM -->
<!ELEMENT ort_item EMPTY>
<!ATTLIST ort_item agent_id CDATA #REQUIRED>
<!ATTLIST ort_item socket CDATA #REQUIRED>

```

```

<!-- ENVIRONMENT -->
<!ELEMENT environment (graph, society+)>

<!-- GRAPH -->
<!ELEMENT graph (vertex+, edge*, stigma*)>
<!ATTLIST graph label CDATA #REQUIRED>

<!-- VERTEX -->
<!ELEMENT vertex (etpd, etpd)?>
<!ATTLIST vertex id ID #REQUIRED>
<!ATTLIST vertex label CDATA #REQUIRED>
<!ATTLIST vertex priority CDATA "0">
<!ATTLIST vertex visibility CDATA "true">
<!ATTLIST vertex idleness CDATA "0">
<!ATTLIST vertex fuel CDATA "false">
<!ATTLIST vertex is_enabled CDATA "true">

<!-- EDGE -->
<!ELEMENT edge (etpd, etpd)?>
<!ATTLIST edge id ID #REQUIRED>
<!ATTLIST edge emitter_id IDREF #REQUIRED>
<!ATTLIST edge collector_id IDREF #REQUIRED>
<!ATTLIST edge oriented CDATA "false">
<!ATTLIST edge length CDATA #REQUIRED>
<!ATTLIST edge visibility CDATA "true">
<!ATTLIST edge is_enabled CDATA "true">
<!ATTLIST edge is_in_dynamic_emitter_memory CDATA "false">
<!ATTLIST edge is_in_dynamic_collector_memory CDATA "false">

<!-- SOCIETY -->
<!ELEMENT society (agent*)>
<!ATTLIST society id ID #REQUIRED>
<!ATTLIST society label CDATA #REQUIRED>
<!ATTLIST society is_closed CDATA "true">

<!-- AGENT -->
<!ELEMENT agent (etpd?, allowed_perception*, allowed_action*)>
<!ATTLIST agent id ID #REQUIRED>
<!ATTLIST agent label CDATA #REQUIRED>
<!ATTLIST agent state CDATA "0">
<!ATTLIST agent vertex_id IDREF #REQUIRED>
<!ATTLIST agent edge_id IDREF #IMPLIED>
<!ATTLIST agent elapsed_length CDATA "0">
<!ATTLIST agent stamina CDATA "1.0">
<!ATTLIST agent max_stamina CDATA "1.0">

<!-- ALLOWED PERCEPTION -->
<!ELEMENT allowed_perception (limitation*)>
<!ATTLIST allowed_perception type (0 | 1 | 2 | 3 | 4) #REQUIRED>

<!-- ALLOWED ACTION -->
<!ELEMENT allowed_action (limitation*)>
<!ATTLIST allowed_action type (0 | 1 | 2 | 3 | 4 | 5 | 6) #REQUIRED>

<!-- LIMITATION -->
<!ELEMENT limitation (lmt_parameter+)>
<!ATTLIST limitation type (0 | 1 | 2 | 3) #REQUIRED>

<!-- LMT PARAMETER -->
<!ELEMENT lmt_parameter EMPTY>
<!ATTLIST lmt_parameter value CDATA #REQUIRED>

```

```
<!-- ETPD - Event Time Probability Distribution -->
<!ELEMENT etpd (pd_parameter+)>
<!ATTLIST etpd seed CDATA #REQUIRED>
<!ATTLIST etpd next_bool_count CDATA "-1">
<!ATTLIST etpd type (0 | 1 | 2 | 3) #REQUIRED>
```

```
<!-- PD PARAMETER -->
<!ELEMENT pd_parameter EMPTY>
<!ATTLIST pd_parameter value CDATA #REQUIRED>
```

```
<!-- METRIC -->
<!ELEMENT metric (EMPTY)>
<!ATTLIST metric type (0 | 1 | 2 | 3) #REQUIRED>
<!ATTLIST metric value CDATA #IMPLIED>
```

```
<!-- PERCEPTION -->
<!ELEMENT perception (graph | agent* | stigma*)>
<!ATTLIST perception type (0 | 1 | 2 | 3 | 4) #REQUIRED>
<!ATTLIST perception message CDATA #IMPLIED>
```

```
<!-- STIGMA -->
<!ELEMENT stigma (EMPTY)>
<!ATTLIST stigma vertex_id IDREF #IMPLIED>
<!ATTLIST stigma edge_id IDREF #IMPLIED>
```

REFERÊNCIAS BIBLIOGRÁFICAS

- ALMEIDA, A. **Patrulhamento Multiagente em Grafos com Pesos**. 2003. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-graduação em Ciência da Computação, Centro de Informática, Universidade Federal de Pernambuco, Recife, 2003.
- ALMEIDA, A.; CASTRO, P.; MENEZES, T.; RAMALHO, G. **Combining Idleness and Distance to Design Heuristic Agents for the Patrolling Task**. In: 2nd Brazilian Workshop on Games and Digital Entertainment. Salvador. 2003. *Proceedings...* [S.l.]: [S.n.]. 2003.
- ALMEIDA, A.; RAMALHO, G.; SANTANA, H.; TEDESCO, P.; MENEZES, T.; CORRUBLE, V.; CHEVALEYRE, Y. **Recent Advances on Multi-agent Patrolling**. In: XVII Brazilian Symposium on Artificial Intelligence. 2004. São Luís. *Proceedings...* Berlin: Springer-Verlag. 2004. p. 474-483.
- CHEN, M.; FOROUGH, E.; HEITZ, F.; HUANG, Z.; KAPETANAKIS, S.; KOSTIADIS, K.; KUMMENEJE, J.; NODA, I.; OBST, O.; RILEY, P.; STEFFENS, T.; WANG, Y.; YIN, X. **Robocup Soccer Server: Users Manual**. [S.l.], 2002. Disponível em <<http://sserver.sourceforge.net/docs/manual.pdf>>. Acesso em 17 ago. 2007.
- CHEVALEYRE, Y. **Theoretical Analysis of the Multi-agent Patrolling Problem**. In: IEEE / WIC / ACM International Conference on Intelligent Agent Technology. 2004. Beijing. *Proceedings...* [S.l.]: [S.n.]. 2004.
- CHEVALEYRE, Y. **Le probleme multi-agents de la patrouille**. In: Annales du LAMSADE n. 4. 2005. Paris. *Annales...* Paris: [S.n.]. 2005.
- COOK, W. **The Traveling Salesman Problem**. [S.l.], 2008. Disponível em <<http://www.tsp.gatech.edu/>>. Acesso em 12 abr. 2008.
- DEITEL, H.; DEITEL, P. **Java: Como Programar**. 4 ed. São Paulo: Bookman. 2003.
- DEITEL, H.; DEITEL, P.; NIETO, T.; LIN, T.; SADHU, P. **XML: How to Program**. New Jersey: Prentice Hall. 2001.
- DORIGO, M.; STÜTZLE, T. **Ant Colony Optimization**. Cambridge, Massachusetts: “A Bradford book”, The MIT Press, 2004.
- DROGOUL, A., LANDAU, S., MUÑOZ, A. **RoboCup: un benchmark pour l’IAD?** Não publicado. 2007.
- ERIKSSON, H.; PENKER, M.; LYONS, B.; FADO, D. **UML 2 Toolkit**. Indianapolis: Wiley Publishing. 2004.
- GAREY, M.; JOHNSON, D. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. Murray Hill: Bell Telephone Laboratories. 1979.

GOSS, S.; BECKERS, R.; DENEUBOURG, J.; ARON, S.; PASTEELS, J. **How Trail Laying and Trail Following Can Solve Foraging Problems For Ant Colonies: Behavioural Mechanisms of Food Selection.** In: NATO ASI Series (G20), 1990. [S.l.]. Series... [S.l.]: NATO Scientific Affairs Division. 1990. p.661-678.

GOULD, R. **Graph Theory.** Menlo Park: The Benjamin/Cummings Publishing Company. 1988.

HANKS, S.; POLLACK, M.; COHEN, P. **Benchmarks, Testbeds, Controlled Experimentation and the Design of Agent Architectures.** 1993. Technical Report – Department of Computer Science and Engineering, University of Washington, Seattle, 1993.

HÜBNER, J.; BORDINI, R.; VIEIRA, R. **Introdução ao Desenvolvimento de Sistemas Multiagentes com Jason.** In: XII Escola de Informática da Sociedade Brasileira de Computação. 2004. Guarapuava. *Anais...* Guarapuava: Editora da UNICENTRO. 2004. cap. 2, p. 51 – 89. Disponível em <<http://www.inf.furb.br/~jomi/pubs/2004/Hubner-eriPR2004.pdf>>. Acesso em 11 nov. 2008.

JENNINGS, R.; WOOLDRIDGE, M. **Agent Technology: foundations, applications, and markets.** London: Springer-Verlag. 1998.

KUROSE, J.; ROSSI, K. **Computer Networking: A Top-Down Approach Featuring the Internet.** 3 ed. New York: Addison Wesley. 2004.

LEVY, R.; ROSENSCHEIN, J. **A game theoretic approach to the pursuit problem.** In: 11th International Distributed Artificial Intelligence Workshop, 1992. [S.l.]. *Proceedings...* [S.l.]:[S.n.]. 1992.

MACHADO, A. **Patrulhamento Multiagente: uma Análise Empírica e Sistemática.** 2002. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-graduação em Ciência da Computação, Centro de Informática, Universidade Federal de Pernambuco, Recife, 2002.

MACHADO, A.; ALMEIDA, A.; RAMALHO, G.; ZUCKER, J.; DROGOUL, A. **Multi-Agent Movement Coordination in Patrolling.** In: 3rd International Conference on Computers and Games Workshop on Agents in Computer Games, Workshop on Agents in Computer Games. 2002. Edmonton. *Proceedings...* [S.l.]: [S.n.]. 2002a.

MACHADO, A.; RAMALHO, G.; ZUCKER, J.; DROGOUL, A. **Multi-Agent Patrolling: an Empirical Analysis of Alternative Architectures.** In: 3rd International Workshop on Multi-agent Based Simulation. 2002. Bologna. *Proceedings...* [S.l.]: [S.n.]. 2002b.

MARTELLO, S.; TOTH, P. **Knapsack Problems: Algorithms and Computer Implementations.** West Sussex: John Wiley & Sons. 1990.

MENEZES, T. **Negociação em Sistemas Multiagente para Patrulhamento**. 2006. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-graduação em Ciência da Computação, Centro de Informática, Universidade Federal de Pernambuco, Recife, 2006.

MENEZES, T.; TEDESCO, P.; RAMALHO, G. **Negotiator agents for the patrolling task**. In: 10th Ibero-American Conference on AI, 18th Brazilian Symposium on AI, International Joint Conference. Ribeirão Preto. 2006. *Proceedings...* [S.l.]: Springer-Verlag. 2006. p. 23-27.

MICHALEWICZ, Z. **Genetic algorithms + data structures = evolution programs**. 3. ed. New York: Springer, 1996.

MOREIRA, D.; SILVA FILHO, L.; TEDESCO, P.; RAMALHO, G. **SimPatrol: Establishing a Testbed for Multi-agent Patrolling**. In: SBGames 2007 – VI Brazilian Symposium on Computer Games and Digital Entertainment, 2007. São Leopoldo. *Proceedings...* São Leopoldo: [S.n.]. 2007.

PEREIRA, L. **Inteligência Artificial: Mito e Ciência**. [S.l.], 2001. Disponível em <<http://www.geocities.com/revistaintelecto/iamc.html>>. Acesso em 30 set. 2008.

PRESSMAN, R. **Software Engineering: A Practitioner's Approach**. 5 ed. New York: McGraw-Hill. 2001.

ROBOCUP. **Robocup**. [S.l.], 2008. Disponível em <<http://www.robocup.org>>. Acesso em 21 mai. 2008.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 2 ed. New Jersey: Prentice Hall. 2003.

SANTANA, H. **Patrulha multiagente com aprendizagem por reforço**. 2004. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-graduação em Ciência da Computação, Centro de Informática, Universidade Federal de Pernambuco, Recife, 2004.

SANTANA, H.; RAMALHO, G.; CORRUBLE, V.; RATITCH, B. **Multi-Agent Patrolling with Reinforcement Learning**. In: 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems. New York. 2004. *Proceedings...* [S.l.]: [S.n.]. 2004.

STEELS, L. **Cooperation between Distributed Agents through Self-Organisation: Decentralized A.I**. In: E. Werner and Y. Demazeau, 1989. Amsterdam. *Proceedings...* Amsterdam: Elsevier. 1989.

STEPHENS, L.; MERX, M. **The effect of agent control strategy on the performance of a DAI pursuit problem**. In: 10th International Workshop on Distributed Artificial Intelligence, 1990. Bandera. *Proceedings...* Bandera: [S.n.]. 1990.

STONE, P. **Multiagent Competitions and Research: Lessons from RoboCup and TAC**. In: RoboCup 2002. 2002. Fukuoka. *Robot Soccer World Cup VI, Lecture Notes in Artificial Intelligence*. Berlin: Springer-Verlag. 2003. p. 224 – 237.

SUN. **Javadoc Tool**. [S.l.], 2008. Disponível em <<http://java.sun.com/j2se/javadoc/>>. Acesso em 07 jul. 2008.

TAC. **Trading Agent Competition**. [S.l.], 2008. Disponível em <<http://www.sics.se/tac/page.php?id=1>>. Acesso em 23 mai. 2008.

TAC CLASSIC. **TAC Classic Agent Protocol: Version 1.0**. [S.l.], 2008. Disponível em <<http://www.sics.se/tac/docs/protocol/>>. Acesso em 27 mai. 2008.

TIOBE COMPANY. **TIOBE Programming Community Index for November 2008**. [S.l.], 2008. Disponível em <<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>>. Acesso em 30 set. 2008.