UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

AFRODITE - Ambiente de Simulação Baseado em Agentes com Emoções

por
DIANA FRANCISCA ADAMATTI

Dissertação submetida à avaliação, como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Profª. Drª. Ana Lúcia Cetertich Bazzan
Orientadora

Porto Alegre, fevereiro de 2003.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Adamatti, Diana Francisca

AFRODITE – Ambiente de Simulação Baseado em Agentes com Emoções / por Diana Francisca Adamatti – Porto Alegre: PPGC da UFRGS, 2003

109 f: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2003. Orientadora: Bazzan, Ana Lúcia Cetertich.

1. Inteligência Artificial. 2. Sistemas Multiagentes. 3. Simulação Baseada em Agentes. 4. Modelos Cognitivos de Emoção. 5.Emoções. I. Bazzan, Ana Lúcia Cetertich. II. Título.

Segundo a Mitologia Grega, "Afrodite" era a deusa do amor e da beleza ....

(Thomas Bulfinch, O Livro de Ouro da Mitologia - Histórias de Deuses e Heróis)

Agradecimentos

A minha mãe Maria, minha irmã Luciana e minha madrinha Ires pelo apoio em todos os momentos de minha vida, desde o início e sempre. Com certeza, sem elas, nunca teria chegado ao final de mais esta etapa.

A meus sempre amigos: Luciana & Luciana (e sua gangue), Aline, Guilherme, André, Delcino, Vanessa, Cláudio, Emerson, Michele, Anelise e Eduardo. Vocês são pessoas fantásticas e tive muita sorte em conhecê-los! Tenho certeza que mesmo longe ficaremos sempre juntos. Quero dizer que somos muito mais que amigos, somos confidentes de alegrias e tristezas, e que este tipo elo nunca se rompe!

Um agradecimento especial a Regina Cantele, pelo enorme apoio da fase final do meu trabalho, e no início de uma nova etapa em minha vida!

A todos os mestres que tive durante toda esta longa jornada, e a confiança depositada por eles ao meu trabalho e esforço. Alguns dos nomes que sempre me recordo são: Profa. Maria de Lourdes Schenatto e Profa. Maria de Fátima, do meu primeiro grau; Profa. Maria Liege, Profa. Vera Lúcia e Profa. Roxane, do meu segundo grau; Prof. Ricardo Dorneles, Prof. Marcos Casa, Profa. Sandra Frigeri, Prof. Vanius Gava e Prof. Alexandre Ribeiro, da minha graduação. Todos eles estiveram presentes, incentivando, sendo verdadeiros professores. Tenho muito o que aprender com todos eles....

À minha orientadora, professora Ana Lúcia Cetertich Bazzan, pela confiança empregada em meu trabalho, disposição sempre presente em me auxiliar e por ser uma grande pesquisadora.

Aos bolsistas Thiago Ghilardi e Andrey Luis Tietbohl Palma pela grande ajuda na implementação deste trabalho.

Ao Roberto e a Eliane, pelo apoio dado nestes dois anos de curso, principalmente na chegada a Porto Alegre, quando eu não conhecia absolutamente nada! Não posso esquecer do seu Astro, que todos os dias passou uma mensagem de incentivo e disposição para continuar esta jornada!

Ao CNPq pelo auxílio financeiro, pois sem este seria impossível a realização deste trabalho.

E sobre tudo, a Deus, por ter permitido chegar a este objetivo, que nunca havia pensado conseguir alcançar algum dia, dando sempre esperança, saúde e força.

Sumário

Lista de Abreviaturas **8**

Lista de Figuras **9**

Lista de Tabelas **10**

Resumo **11**

Abstract **12**

1 Introdução **13**

2 Sistemas Multiagentes e Ambientes de Simulação Baseados em Agentes **16**

2.1 Inteligência Artificial Distribuída **16**
2.1.1 Agentes 16

2.2 Sistemas Multiagentes (SMA) **17**

2.3 Simulação **19**
2.3.1 Etapas da Simulação 19

2.4 Simulação Baseada em Agentes **20**

2.5 Ambientes de Simulação Baseados em Agentes **21**
2.5.1 SIEME 21
2.5.2 SWARM 22
2.5.3 SeSAm 23
2.5.4 SIMULA 24
2.5.4.1 SIMULA ++ 25

2.6 Comparativo entre Ambientes de Simulação Baseados em Agentes **25**
2.6.1 Linguagem de Programação 25
2.6.2 Paralelismo 26

2.6.3 Modelos de Simulação ... 26
2.6.4 Tipos de Agentes ... 27
2.6.5 Facilidades ... 27
2.6.6 Interface ... 27

3 Emoções e Modelo OCC ... 29

3.1 A natureza do problema ... 29

3.2 Modelo Proposto por Ortony, Clore e Collins (Modelo OCC) ... 30
3.2.1 Emoções Básicas ... 30
3.2.2 Modelo OCC ... 31
3.2.3 Sistema baseado em regras ... 35

3.3 Trabalhos Existentes ... 37
3.3.1 Modelos em nível de arquitetura ... 37
3.3.2 Modelos em nível de tarefas ... 39
3.3.3 Modelos em nível de mecanismo ... 40

3.4 Conclusões ... 41

4 Proposta de um Ambiente de Simulação Baseado em Agentes com Emoções ... 42

4.1 Aspectos considerados para o desenvolvimento do ambiente AFRODITE . 42

4.2 Arquitetura do ambiente AFRODITE ... 43

4.3 Primitivas pré-definidas do ambiente AFRODITE ... 45

4.4 Definição das regras de comportamento e emoções ... 48

4.5 Exemplo de funcionamento do cálculo da intensidade das emoções ... 50

5 Protótipo do Ambiente AFRODITE ... 52

5.1 Interface do Protótipo ... 52
5.1.1 Menu File ... 53
5.1.2 Menu Definition ... 55
5.1.3 Exemplo de como o protótipo disponibiliza os aspectos dos subgrupos do modelo OCC ... 58
5.1.4 Menu Simulation ... 60
5.1.5 Menu Help ... 61

5.2 Estruturas de Implementação ... 62

5.3 Estrutura Geral para Criação do Código para Execução da Simulação ... 64

5.4 Estrutura Principal para Execução do Código ... 65

6 Modelagem de Cenários Selecionados no Ambiente AFRODITE ... 66

6.1 IPD (Iterated Prisoner's Dilemma) ... 66
6.1.1 Modelagem do IPD no AFRODITE ... 67
6.1.2 Considerações sobre a Modelagem ... 72

6.2 Simulação de Multidões 73
6.2.1 Modelagem de Simulação de Multidões no AFRODITE..... 74
6.2.2 Considerações sobre a Modelagem 77

6.3 Venda de aparelhos celulares com serviço WAP 77
6.3.1 Modelagem da Venda de aparelhos celulares com serviço WAP no AFRODITE sem Emoção..... 79
6.3.2 Modelagem da Venda de aparelhos celulares com serviço WAP no AFRODITE com Emoção 82
6.3.3 Considerações sobre as Modelagens (sem emoção e com emoção)..... 84

6.4 Considerações Gerais sobre as Modelagens Realizadas 85

7 Conclusões e Trabalhos Futuros 86

Anexo 1 Categorias de Emoções [ELL2002] 89

Anexo 2 Código Gerado na Simulação 90

Código gerado para modelagem sem emoção 90

Código gerado para modelagem com emoção 91

Anexo 3 Exemplo de Arquivo de Saída Gerado pela Simulação 95

Anexo 4 Manual do Usuário 98

Anexo 5 Relação de Trabalhos Publicados ou Submetidos 102

Bibliografia 103

Lista de Abreviaturas

IA	Inteligência Artificial
IAD	Inteligência Artificial Distribuída
IPD	Iterated Prisoner's Dilemma
OCC	Modelo de Ortony, Clore & Collins
SESAM	Shell For Simulated Agent Systems
SIEME	Simulateur Evènementiel Multi-Entités
SIMULA	Sistema Multiagente Reativo
SMA	Sistema Multiagente
STI	Sistemas Tutores Inteligentes
WAP	Wairless Aplication Protocol

Lista de Figuras

FIGURA 2.1 - A abordagem SMA 18
FIGURA 2.2 - Etapas de um Processo de Simulação 20
FIGURA 2.3 - Sistema SWARM - Kernel e Bibliotecas 23
FIGURA 2.4 - Funcionalidades do ambiente SIMULA 24
FIGURA 3.1 - Estrutura das emoções no modelo OCC 34
FIGURA 4.1 - Arquitetura do ambiente AFRODITE 43
FIGURA 4.2 - Definição dos Agentes 44
FIGURA 4.3 - Definição das Regras de Comportamento e das Emoções 44
FIGURA 4.4 - Disposição do Ambiente 45
FIGURA 4.5 - Possíveis valores de retorno para a primitiva neighbor 47
FIGURA 5.1 - Interface Geral do Protótipo 53
FIGURA 5.2 - Menu *File* 53
FIGURA 5.3 - Tela *New Simulation* 54
FIGURA 5.4 - Tela *Open Simulation* 54
FIGURA 5.5 - Tela *Analisys* 54
FIGURA 5.6 - Menu *Definition* 55
FIGURA 5.7 - Tela *Agents Definition* 56
FIGURA 5.8 - Tela *Agent Characteristics* 56
FIGURA 5.9 - Tela *Rules Definition* 57
FIGURA 5.10 - Tela *Emotions Templates* 58
FIGURA 5.11 - Tela *Environment Definition* 59
FIGURA 5.12 - Alocação Randômica dos Agentes 59
FIGURA 5.13 - Alocação em Posição Desejada dos Agentes 60
FIGURA 5.14 - Menu *Simulation* 61
FIGURA 5.15 - Menu *Help* 61
FIGURA 5.16 - ER das tabelas para armazenamento dos dados 62
FIGURA 5.17 - Estrutura de dados do agente 63
FIGURA 6.1 - Portões e acessos do *Winbley Stadium* 74

Lista de Tabelas

TABELA 4.1 - Correspondência entre valores Definidos pelo Usuário e Discretização Armazenada pelo ambiente 49
TABELA 5.1 - Exemplo de Dados de Saída da Simulação 55
TABELA 5.2 - Relação entre os Conceitos e as Estruturas Utilizadas 62
TABELA 6.1 - Exemplo Típico de *payoff* do Dilema do Prisioneiro 67

Resumo

Este trabalho está relacionado às áreas de Sistemas Multiagentes, Simulação Computacional e Emoções. A partir do estudo destas áreas de pesquisa, foi proposto e desenvolvido um protótipo para um ambiente de simulação baseado em agentes com emoções.

Os sistemas multiagentes têm sido utilizados nas mais diversas áreas de pesquisa, não apenas para a área acadêmica, mas também para fins comerciais. Isso ocorre devido a características importantes que estes possuem, como flexibilidade e cooperação. Estas características são úteis para um grande número de aplicações, como para simulação de situações reais, pois os modelos de simulação desenvolvidos utilizando a tecnologia de agentes são muito eficazes e versáteis no estudo dos mais diferentes problemas.

Emoções vêm sendo estudadas há algum tempo, pois elas influenciam a tomada de decisão de todas as suas atividades. A tentativa de expressar emoções é algo complexo, dependendo de diversos fatores, tanto sociais como fisiológicos.

Objetivando a abrangência das pesquisas na área de sistemas multiagentes, este trabalho propõe o desenvolvimento de um protótipo para um ambiente de simulação baseado em agentes com emoções, utilizando como base para a estruturação das emoções o modelo OCC. Este novo ambiente é chamado AFRODITE.

De forma a melhor definir como o AFRODITE seria implementado, foram estudados quatro ambientes de simulação baseados em agentes existentes - SIEME, SWARM, SeSAm e SIMULA, e alguns aspectos destes foram utilizados na construção do novo ambiente.

Para demonstrar como o AFRODITE é utilizado, três exemplos de aplicações de áreas de conhecimentos diferentes foram modelados: o IPD (*Iterated Prisoner's Dilemma*), da área de Teoria dos Jogos; Simulação de Multidões, da área de Engenharia de Segurança; e Venda de aparelhos celulares com serviço WAP, da área de Telecomunicações.

Através dos três exemplos modelados foi possível demonstrar que o ambiente proposto é de fácil utilização e que a tarefa de inserção de emoções nas regras de comportamento pode ser realizada pelo usuário de forma transparente.

Palavras-chave: Inteligência Artificial, Inteligência Artificial Distribuída, Sistemas Multiagentes, Simulação Baseada em Agentes, Modelos Cognitivos de Emoção e Emoções.

TITLE: "AFRODITE – A FRAMEWORK FOR SIMULATION OF AGENTS WITH EMOTIONS"

Abstract

This work presents a study of Multi-agents Systems, Computational Simulation and Emotions. From the study of these three areas, a prototype of a framework for simulation of agents with emotions was developed.

Multi-agent systems have been used in the diverse research areas, not only academic but also in the commercial area. This happens due to some important agent's characteristics such as autonomy and flexibility. These characteristics are useful to a lot of applications, like simulation of real situations. Therefore, the development of simulations model using agent's technology are very efficient and versatile in the study of the most different problems.

Emotions have been studied for a long time, because they have influenced in the decision-making process. Express emotions is very complex due to diversity of factors, as social and physiological.

This work describes a framework developed to simulate agents with emotions. Such prototype is called AFRODITE and uses as base for the structure of emotions the OCC model.

In order to define how AFRODITE would be implemented, four frameworks of agents' simulation were studied - SIEME, SWARM, SeSAm and SIMULA -, and some aspects of these frameworks were used in the AFRODITE's construction.

To demonstrate as the AFRODITE works, three examples from different areas of knowledge were modeled: IPD (Iterated Prisoner's Dilemma), of the Games Theory area; Simulation of Multitudes, of the Security Engineer area; and Sale of Cellular Phone with WAP Service, of the Telecommunications area.

The objetive proposed by this work – development of a framework for simulations of agents with emotions in way that such task could be executed transparently – was reached with success. This could be observed through the modeled examples.

Keywords: Artificial Intelligence, Distributed Artificial Intelligence, Multiagentes Systems, Agents-based Simulations, Cognitive Models of Emotions and Emotions.

1 Introdução

O termo Inteligência Artificial (IA) nasceu, oficialmente, em meados da década de 50, quando jovens pesquisadores (John McCarthy, Marvin Minsky, Nathaniel Rochester e Claude Shannon) imaginaram que a IA poderia substituir a inteligência humana [BRA97]. Desde o início, este termo gerou polêmica, pois muitos o consideraram presunçoso, devido às promessas exageradas feitas pelos pesquisadores e pelos limites práticos e tecnológicos da época, e, conseqüentemente, gerando decepções [BIT98].

Desde a idéia inicial de se construir uma inteligência artificial comparável à do ser humano até objetivos menos complexos como de tornar os computadores mais úteis nas atividades das pessoas, a IA busca desenvolver ferramentas que auxiliem na realização das atividades humanas [BIT98]. Na prática, o principal objetivo da IA é fazer com que as máquinas possam realizar tarefas que antes eram executadas somente por pessoas. Segundo Bradshaw [BRA97], as pessoas ainda têm uma certa “norma” moral quanto a utilização de sistemas inteligentes na sociedade. Verifica-se um certo receio quanto à segurança de software onde o homem não possua o controle.

A construção de um software inteligente é algo bastante complexo. Primeiro, porque as aplicações são extremamente particulares, e para cada uma delas é necessário um estudo aprofundado antes do desenvolvimento. Depois, para cada aplicação é preciso decidir qual será a arquitetura de processamento, mecanismos de comunicação, conhecimento interno e dados representados. Estes sistemas têm se tornado cada vez mais complexos, existindo a necessidade de criação de uma entidade heterogênea e autônoma, na qual seja possível inserir ou remover funcionalidades de uma maneira prática e acessível. Assim, surge a idéia do agente, que é uma entidade autônoma, flexível e adaptável [FER99].

Com o passar do tempo, surgiu a necessidade de distribuir os sistemas de IA, para que os mesmos pudessem utilizar as facilidades de sistemas distribuídos. Com união das idéias de IA e sistemas distribuídos, no início da década de 70, surgiu a Inteligência Artificial Distribuída (IAD), que busca resolver os problemas de modo cooperativo e onde o foco está na integração de agentes, os quais cooperam entre si a fim de alcançar um objetivo comum. Uma das subáreas da IAD é a dos Sistemas Multiagentes (SMA), nos quais o foco está no agente em si, buscando o encapsulamento das funcionalidades de cada agente.

A utilização de sistemas que utilizem a tecnologia baseada em agentes vem aumentando, e está envolvendo várias áreas da ciência da computação, como engenharia de software e redes de computadores. Alguns exemplos de utilização da IA são: pesquisas em banco de dados, filtragem de informações na Internet, correio eletrônico, telecomunicações, projetos de interface, jogos de computadores e simulação de situações reais.

Este último exemplo, o de simulação de situações reais, é bastante compatível com a tecnologia baseada em agentes, pois na simulação multiagente o fenômeno real é decomposto em um conjunto de elementos e em suas interações. Cada elemento é modelado como um agente e o modelo geral é o resultado das interações entre estes agentes. A área de simulação é empregada como elemento auxiliar na tomada de decisões. Os modelos de simulação são muito eficazes e versáteis no estudo dos mais diferentes problemas. Utilizar a simulação baseada em computador é uma forma poderosa de projetar, planejar, controlar e avaliar novas alternativas em sistemas no mundo real [REB 99].

Outra área de estudo da IA são as emoções, pois nas organizações humanas, as atividades são realizadas por um grupo de pessoas que trabalha de modo cooperativo e deve tomar decisões conjuntas. Como as emoções são o centro das motivações humanas, delas depende a tomada de decisão das atividades humanas [GRA2001a]. As emoções são parte dos nossos relacionamentos, das histórias que contamos e dos planos que fazemos. Tentar expressar emoções em máquina é algo complexo, pois depende de diversos fatores, tanto sociais como fisiológicos. Vários pesquisadores propuseram modelos para estruturação de emoções, como Sloman [SLO2001], Cañamero *et al.*[CAN99], Moffat *et al.*[MOF2000], Ortony *et al.*[ORT88], entre outros.

Devido a compatibilidade da tecnologia baseada em agentes e da simulação, agentes têm sido empregados como base em vários ambientes de simulação, como SIEME [MAG2001a], SWARM [SWA95], SeSAm [KLÜ2002], SIMULA [FRO97], e outros. Porém, um ambiente de simulação baseado em agentes que faça uso de um modelo de estruturação de emoções ainda não foi desenvolvido. Simular emoções em agentes, como raiva ou amor, pode demonstrar reações diferentes, para diferentes situações. Um exemplo seria a não cooperação com outros agentes, como acontece em situações reais, tornando a análise dos resultados que o ambiente produz mais parecida com a realidade. Até o momento, existem apenas aplicações particulares para resoluções de problemas específicos, como o IPD (“*Iterated Prisoner’s Dilemma”*) ou “*Study of Social Norms*” [BAZ 2001][BAZ 2002].

O objetivo central deste trabalho é o desenvolvimento de um ambiente de simulação baseado em agentes com emoções. O modelo proposto por Ortony, Clore & Collins [ORT88], também chamado modelo OCC, foi escolhido como base para a estruturação das emoções, pois é baseado no princípio da diferenciação entre reações de valência positivas ou negativas e tenta simplificar o desenvolvimento de uma teoria focando vários aspectos cognitivos das emoções. Além disso, foi o primeiro modelo de estruturação de emoções desenvolvido visando sua implementação em máquina. As aplicações existentes que utilizam como base o modelo OCC, implementam partes do modelo, ou seja, disponibilizam apenas algumas das emoções propostas. Uma aplicação onde todas as emoções propostas pelo modelo possam ser utilizadas ainda não foi

implementada. O ambiente proposto, chamado AFRODITE, tem implementado, a disposição do usuário, todas as emoções propostas pelo modelo OCC.

Este trabalho esta organizado da seguinte maneira: o capítulo 2 apresenta uma visão geral de IAD, simulação baseada em agentes, alguns ambientes de simulação baseados em agentes existentes (SIEME, SWARM, SeSAm e SIMULA) e um estudo comparativo entre estes ambientes, de forma a verificar como foram implementados, e analisar pontos positivos e negativos dos mesmos para utilização no novo ambiente que está sendo proposto.

No capítulo 3 é realizada uma breve análise sobre emoções e é explicada a idéia de funcionamento e subdivisões do Modelo OCC. São apresentados, também, os trabalhos relacionados a esta área de estudo.

O capítulo 4 descreve o ambiente proposto, AFRODITE. Neste capítulo são apresentadas as características e funcionalidade que constituem o ambiente.

No capítulo 5 é explicado como foi desenvolvido o protótipo do AFRODITE e o capítulo 6 apresenta como três exemplos de áreas de conhecimento diferentes foram modelados neste ambiente, utilizando os recursos disponíveis.

O capítulo 7 apresenta as conclusões e idéias para trabalhos futuros.

2 Sistemas Multiagentes e Ambientes de Simulação Baseados em Agentes

Este capítulo apresenta uma visão geral de IAD (Inteligência Artificial Distribuída), Sistemas Multiagentes e o estudo da simulação em agentes. Também é realizado um estudo comparativo entre alguns ambientes de simulação baseados em agentes.

2.1 Inteligência Artificial Distribuída

A inteligência artificial clássica tem como modelo de inteligência o comportamento individual humano, cuja ênfase é a representação de conhecimento e métodos de inferência. A Inteligência Artificial Distribuída (IAD) é baseada no comportamento social, com a ênfase em ações e interações entre agentes.

Nas organizações humanas, as atividades são realizadas por um grupo de pessoas que trabalham de modo cooperativo. Essas atividades em grupo envolvem a tomada de decisões e a distribuição do conhecimento. A partir disso, a IAD começou a estudar a concepção de sistemas compostos por agentes artificiais e agentes humanos na resolução de tarefas de um grupo. Desta forma, um dos objetivos da IAD é construir sistemas capazes de solucionar problemas de forma cooperativa.

Em IAD o foco está na integração de agentes, os quais cooperam entre si a fim de alcançar um objetivo comum. Algumas das motivações da distribuição de sistemas inteligentes podem ser deduzidas facilmente, como: a redução de custos, o aumento de eficiência e velocidade, e a integração entre computadores e sistemas de diversas redes [BIT98].

2.1.1 Agentes

Para o desenvolvimento de sistemas complexos, existe a necessidade de criação de um novo paradigma de programação que contemple a distribuição, a autonomia, a flexibilidade e a heterogeneidade. Este paradigma deve possuir a facilidade de adicionar ou de remover funções, de forma extensível e flexível. O conceito de agente preenche estes requisitos e está sendo adotado como base para este novo paradigma. Dependendo do contexto onde os agentes estão inseridos, várias definições podem existir. Assim, não existe um consenso quanto a definição precisa do que é um agente [KNA98]. Algumas definições de agentes encontradas na literatura são:

- “Um agente pode perceber seu ambiente e, através de sensores e ações, atuar sobre este. Humanos têm seu corpo para atuar sobre o ambiente. Robôs têm câmeras com vários tipos de sensores para atuar sobre o ambiente. Um agente de software tem um “código” de bits para perceber e atuar sobre este ambiente.” [RUS95]
- “Agente de software: uma entidade computacional que desenvolve tarefas delegadas por usuários de forma autônoma”. [CAG97]
- “Um agente é uma entidade a qual pode-se associar uma identidade única, e que é capaz de realizar cálculos formais. Um agente pode ser considerado como um meio que produz um certo número de ações a partir dos conhecimentos e mecanismos que lhe são próprios.” [GAS 92]
- “Agente é uma entidade de software que executa uma determinada tarefa, empregando informações extraídas de seu ambiente para agir de forma adequada, no sentido de completar sua tarefa de modo bem sucedido. O software deve ser capaz de adaptar-se a eventuais modificações ocorridas em seu ambiente de maneira que o resultado pretendido seja independentemente alcançado.”[LIB 95]
- “Um agente computacional é aquele que possui uma identidade e realiza ações. São atores de comunicação para passagem de mensagens e realização de ações (tarefas) correntes.” [NWA 96]
- “Um agente é um sistema baseado em conhecimento que percebe seu ambiente (o qual pode ser físico, um usuário via interface gráfica, ou uma coleção de outros agentes), possui reações para interpretar percepções sobre o ambiente, resolve problemas, e determina ações a serem tomadas.” [TEC98]

2.2 Sistemas Multiagentes (SMA)

Uma linha de pesquisa existente na IAD é a de Sistemas Multiagentes (SMA), que estuda o comportamento de um conjunto de agentes autônomos com diferentes características, evoluindo em um ambiente comum. Estes agentes interagem uns com os outros, com o objetivo de realizar suas tarefas de modo cooperativo, compartilhando informações, evitando conflitos e coordenando a execução de atividades.

Segundo [ALV97], a abordagem em SMA pode ser observada conforme a Figura 2.1.

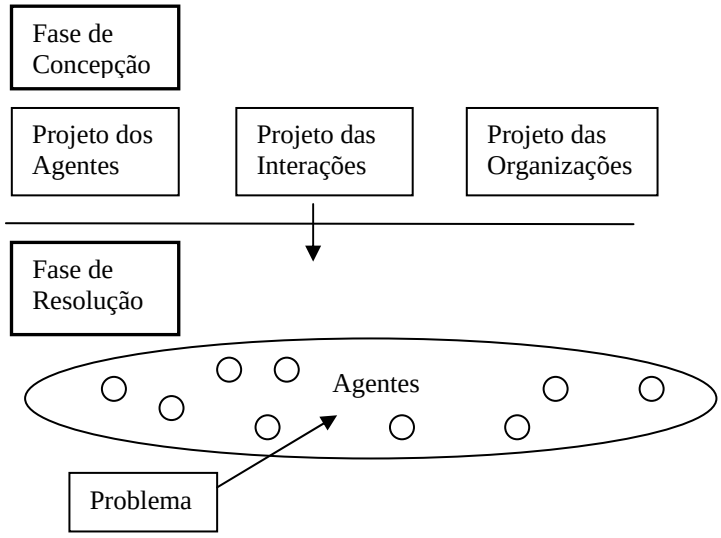


FIGURA 2.1 - A abordagem SMA [ALV97]

Na fase de Concepção, não existindo um problema a priori, os agentes são projetados com funcionalidades genéricas. Não há a preocupação na resolução de um problema específico. Na fase de Resolução, a partir dos agentes existentes, o problema é exposto e uma solução é encontrada.

Um SMA é algo bastante genérico, pois procura resolver um problema que surgiu após a construção da arquitetura do sistema. Sendo assim, o problema não existe a priori, pois o objetivo é estudar modelos genéricos, a partir dos quais se possam conceber agentes, organizações e interações, para depois resolver um problema específico [ALV97].

Nesse tipo de sistema, pressupõe-se que os agentes que formam uma sociedade são dotados de certa autonomia e inteligência para que, independentemente da existência dos outros agentes, consigam desenvolver o objetivo global, que é a solução de um problema qualquer [FER99].

O comportamento de um agente em um SMA é função da percepção que este possui de seu próprio estado, de suas interações com os outros agentes e de seu conhecimento.

Os SMA apresentam uma perspectiva interessante para projetar e desenvolver ambientes e ferramentas computacionais mais sofisticadas e poderosas de forma a atender o aumento da complexidade dos sistemas reais. Desta forma, permitem [CAG97][BIT98]:

- melhorar a robustez do sistema, evitando falhas;
- criar o apoio necessário para acelerar a execução da resolução de problemas pela paralelização de programas;
- integrar metodologias de resolução de problemas;
- combinar paradigmas de computação e fontes de conhecimento.

Geralmente, os agentes de um SMA são classificados em duas categorias:

- Reativos: funcionam segundo um modelo estímulo-resposta; não possuem uma representação explícita de seu ambiente; não possuem memória das ações executadas no passado e nem previsão das ações a serem executadas no futuro. Eles apenas reagem às modificações do ambiente e não existe relação social entre os agentes.
- Cognitivos: possuem uma representação explícita de seu ambiente, de outros agentes e deles mesmos. São baseados nos modelos humanos, possuem raciocínio e capacidade de planejamento de suas ações.

A IAD baseia-se no comportamento social, com ênfase nas ações e nas interações dos agentes. Um comportamento social inteligente pode surgir no caso de membros inteligentes da sociedade (agentes cognitivos) ou membros não inteligentes da sociedade (agentes reativos).

2.3 Simulação

O método de simulação é empregado com grande sucesso como elemento auxiliar na tomada de decisões, principalmente no planejamento a médio e longo prazo e em situações que envolvem custos e riscos elevados. Os modelos de simulação são muito eficazes e versáteis no estudo dos mais diferentes problemas. Seu emprego permite o exame de detalhes específicos com grande precisão [REB 99].

"*O propósito dos modelos de simulação é permitir a realização de estudos sobre os sistemas reais, analisando sua reação ante as influências externas e internas, ou sua abrangência no meio ambiente*." [STR 84]

Simulação baseada em computador é uma das ferramentas mais poderosas disponíveis nos dias de hoje para projetar, planejar, controlar e avaliar novas alternativas e/ou mudanças de estratégias em sistemas no mundo real. Sua utilização significa construir programas de computador (software) que 'representem' o sistema do mundo real em questão e 'imitem' seu funcionamento [REB 99].

Pode-se dizer que a simulação baseada em computador visa:
- descrever o comportamento do sistema;
- construir teorias e hipóteses que sigam as regras desse comportamento;
- utilizar o modelo para testar as hipóteses criadas a fim de prever comportamentos futuros;

2.3.1 Etapas da Simulação

A simulação pode ser dividida em três etapas [STR84]:
- Etapa da Modelagem: construir o modelo do fenômeno a ser estudado;
- Etapa de Experimento: aplicar variações sobre o modelo construído, alterando parâmetros que influam no processo de resolução;
- Etapa de Validação: comparar dados experimentais obtidos com o modelo e a realidade. É a análise de resultados.

Pela Figura 2.2, se podem verificar as etapas de um processo de simulação. A partir da realidade, é realizada uma modelagem do problema desejado e é construído um modelo. Para se construir este modelo, têm-se como base a teoria. Com o modelo, faz-se

a simulação e depois a avaliação do mesmo. Para se avaliar, utiliza-se os resultados obtidos pelo modelo e as observações da realidade [FRO97].

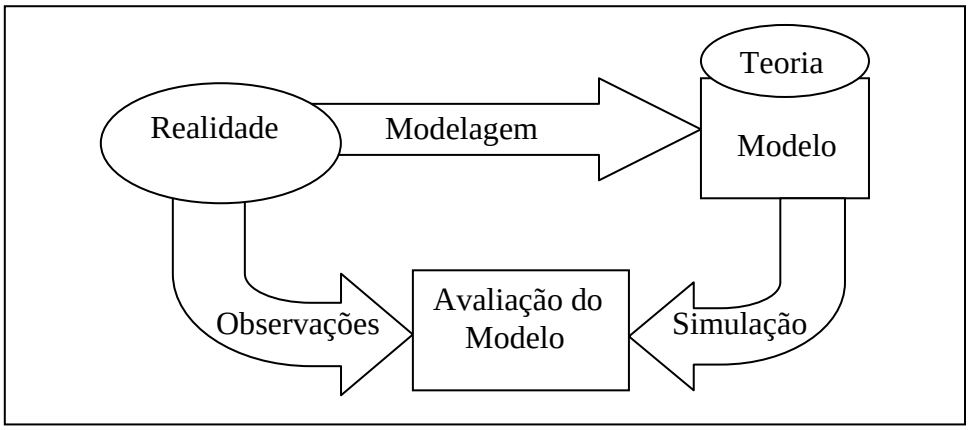


FIGURA 2.2 - Etapas de um Processo de Simulação [FRO97]

Pelo grande poder de avaliação, flexibilidade e versatilidade, a simulação computacional encontra aplicação em praticamente todas as áreas da atividade humana.

2.4 Simulação Baseada em Agentes

Muitas aplicações em SMA são desenvolvidas para simular alguma situação da realidade. Na simulação multiagente, o fenômeno real é decomposto em um conjunto de elementos e em suas interações. Cada elemento é modelado como um agente e o modelo geral é o resultado das interações entre estes agentes.

Modelar um fenômeno sob a perspectiva de um SMA pode ser visualizada nas seguintes etapas [FRO99]:

1ª) decompor o fenômeno em um conjunto de elementos autônomos;
2ª) modelar cada um dos elementos como um agente, definindo seu conhecimento, funções, comportamento e modos de interação;
3ª) definir o ambiente dos agentes;
4ª) definir quais agentes possuem a capacidade de ação e comunicação.

Segundo Feber [FER99] e Russel *et al.* [RUS95], o uso de agentes para a implementação de ambientes, inclusive de simulação, exige algumas características, que devem ser levadas em consideração. São elas:

- agentes têm tomada de decisão, ou seja, são autônomos;
- o comportamento dos agentes é representado em um alto nível de abstração;
- agentes são flexíveis, tendo características de comportamento pró-ativo e reativo;
- agentes podem executar tarefas que exijam performance de tempo real;
- agentes se encaixam em aplicações distribuídas;

- agentes possuem habilidades de trabalhar cooperativamente.

A implementação com agentes requer que alguns cuidados [KLÜ 2002]:

- identificar claramente os agentes, suas atividades e seus componentes dentro do ambiente; deve-se saber exatamente quais são suas atividades. Para selecionar as atividades deve-se observar os objetivos específicos que os agentes devem ter;
- lembrar que agentes possuem um comportamento não-trivial. Desta forma, agentes mais simples são implementados de forma mais eficiente em uma linguagem orientada a agentes;
- quanto mais simples a estrutura de cada agente, melhor para o sistema, pois o foco da modelagem deve ser o comportamento e as interações dos agentes, e não suas habilidades internas. Existem ferramentas de software orientadas a agentes, para facilitar seu desenvolvimento, como por exemplo o *Agent Builder* [AGE2000];
- deve haver uma descrição do sistema do agente e do ambiente em que o agente se posiciona, pois isso facilita a implementação e a solução de possíveis erros.

2.5 Ambientes de Simulação Baseados em Agentes

Em [ADA2001], foram estudados quatro ambientes de simulação baseados em agentes, SIEME, SWARM, SeSAm e SIMULA. Os aspectos mais relevantes de cada um destes ambientes são apresentados a seguir.

2.5.1 SIEME

O sistema SIEME – *Simulateur Evènementiel Multi-Entités* é um simulador baseado em agentes desenvolvido por Laurent Magnin [MAG2001a][SIE2001a][SIE2001b], do laboratório Laforia da Universidade Paris 6. SIEME utiliza a linguagem *SmallTalk*, de modo que uma aplicação desenvolvida nesta linguagem pode ser executada, sem modificações, em plataformas distintas, como PC, Macintosh ou Unix [MAG2001a]. A plataforma de SIEME utiliza o mesmo editor do *SmallTalk*. Devido a compatibilidade com a WEB, e visando tornar ainda mais portável o sistema, uma implementação foi desenvolvida utilizando Java. Nesta versão, o modelo de SIEME é duplicado dinamicamente para utilizar o estado corrente da simulação em diferentes computadores.

SIEME propõe um novo modelo de simulação, utilizando eventos, e baseado em tempo e espaço contínuos.

Segundo Magnin [MAG2001a], existem alguns fatores a serem analisados neste modelo de simulação:

- Entidades e Ações: uma ação que gera modificações no sistema realiza estas modificações sobre entidades. Porém as entidades não podem ser alteradas livremente. Por exemplo, um robô não pode alterar sua velocidade independente do resto do ambiente. Antes de haver qualquer alteração deve-

se verificar se todo o sistema continuará funcionando de forma correta após a alteração.

- Interações e Regras de Ambiente: quando se definem as entidades não se tem ainda a descrição de como ocorrerá a simulação no sistema. Para isso deve-se definir as regras e as interações entre as entidades. Em um simulador clássico, as interações são resultado das combinações de ações dos agentes. No modelo SIEME, tem-se uma nova concepção, as Regras de Ambiente, que são computadas apenas quando um conjunto de condições for verdadeiro.

- Algoritmo de Simulação: a simulação em SIEME envolve regras de ambiente geradas. O simulador determina, dependendo das equações de condição, quando as regras serão verdadeiras. Isso ocorre quando não houver prejuízo para o sistema em relação à sua execução. Após, os eventos correspondentes aos eventos das regras são inseridos em uma fila para execução.

Outro aspecto importante existente na implementação de SIEME é que cada agente possui embutidas as funcionalidades básicas de comunicação, interação, cooperação, etc. Desta maneira, o sistema não cria automaticamente agentes para executar as tarefas básicas, como normalmente acontece em sistemas que utilizam agentes. Esta abordagem é uma tentativa de simplificação da implementação, porque pode-se, a partir de cada agente, saber como ele trata cada funcionalidade. Porém, com esta abordagem, perde-se um pouco do aspecto da modularidade, muito importante no desenvolvimento para plataformas de agentes.

2.5.2 SWARM

SWARM foi desenvolvido na Universidade de Santa Fé, nos Estados Unidos. Este sistema é uma plataforma de software multiagente para a simulação de sistemas adaptativos [SWA95]. Fornece uma biblioteca orientada a objetos de componentes reutilizáveis para a construção de modelos, e para análise de experimentos a partir do ambiente.

O sistema SWARM está implementado em C. Todos os componentes do sistema são objetos. Para que estes objetos (agentes) possam se comunicar são enviadas mensagens entre os mesmos. Existe uma preocupação em desenvolver implementações paralelas. Para que isso possa ser possível, SWARM possui agentes genéricos em uma grande biblioteca de desenvolvimento e análise, e um *kernel* que possibilita esta finalidade.

SWARM provê o desenvolvimento, incluindo a implementação, testes e análise para sistemas multiagentes, a partir de um conjunto de bibliotecas ligadas a um *kernel*. O conjunto destas bibliotecas e do *kernel* são apresentadas na Figura 2.3.

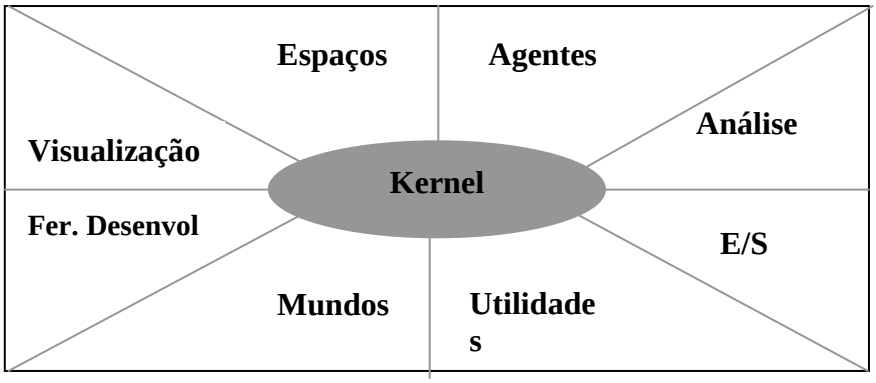


FIGURA 2.3 - Sistema SWARM – Kernel e Bibliotecas [SWA95]

Cada biblioteca tem uma função específica, sendo utilizada de acordo com a aplicação. Algumas bibliotecas são utilizadas em qualquer aplicação, como, por exemplo, agentes e ferramentas de desenvolvimento.

Um fator importante a salientar é que além da preocupação em desenvolver um ambiente que suportasse o paralelismo, SWARM buscou tornar eficiente as operações que envolvam arquiteturas de máquinas paralelas. Isso partiu da idéia de decompor problemas complexos em subproblemas.

2.5.3 SeSAm

O ambiente SeSAm (*Shell for Simulated Agent Systems*) foi desenvolvido no Departamento de Inteligência Artificial e Ciência da Computação Aplicada da Universidade de Wüzburg, Alemanha [KLÜ2002].

Em sua primeira versão, SeSAm foi implementado em Lisp, rodando apenas para Macintosh. Isso tornou o sistema muito pouco portável, trazendo complicações para o uso mais freqüente em diversos ambientes. Uma versão Java, rodando em Windows, Solaris, MacOS, Unix, Linux, AIX e HP-UX já está disponível.

A descrição do comportamento dos agentes é baseada na representação das primitivas básicas. Nesta representação, as primitivas devem estar fortemente conectadas. Para pré-definir estes símbolos utiliza-se a linguagem de programação, que executa um código correspondente ao símbolo. Para diferenciar declaração, representação explícita e código de programação, são utilizadas as descrições dos objetos que existem no modelo.

Para facilitar a implementação da simulação é definido um conjunto de primitivas para utilização. Isso é necessário para dar ao usuário um controle maior do sistema e , assim, a implementação é realizada em um nível maior de abstração. Um exemplo de primitiva é MOVE(), utilizada para que o agente se movimente um passo na direção especificada em sua sintaxe. Utilizando estas primitivas, a implementação se torna mais transparente e funcional, necessitando de apenas algumas noções básicas de programação para a implementação do sistema.

Quem modela o sistema pode determinar como será a visualização de cada passo durante a simulação. Para cada entidade é possível visualizar, através de uma janela, os valores internos da mesma. Manualmente, pode-se fixar pontos de parada, para análise até o ponto fixado, e posteriormente dar continuidade a simulação.

2.5.4 SIMULA

O ambiente SIMULA (Sistema Multiagente Reativo) foi desenvolvido no Instituto de Informática da Universidade Federal do Rio Grande do Sul [FRO97]. Utiliza a linguagem de programação C++ em sua primeira versão. Com a intenção de torná-lo ainda mais portável, seu código foi traduzido de C++ para Java.

SIMULA foi desenvolvido para atender usuários com conhecimento na tecnologia de agentes. O objetivo foi diminuir o esforço de programação do usuário na criação de aplicações. Também tenta estimular os usuários no projeto de novos sistemas com o uso de agentes.

A disponibilização de comportamentos pré-definidos têm como objetivo facilitar o desenvolvimento de SMA no ambiente SIMULA. Se o conjunto de comportamentos pré-definidos não for suficiente para execução de todos os comportamentos necessários, pode-se desenvolver um conjunto de aplicações específico. Para desenvolver este conjunto de novas aplicações, que não estão disponíveis, utiliza-se a programação convencional, exigindo conhecimento de detalhes de implementação e estrutura de dados da linguagem na qual o ambiente foi desenvolvido.

A Figura 2.4 apresenta um esquema de como o usuário deve proceder para desenvolver suas aplicações no SIMULA.

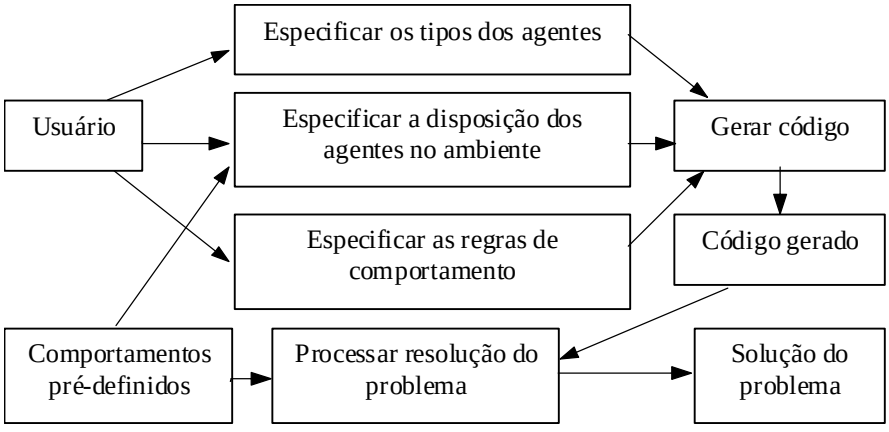


FIGURA 2.4 - Funcionalidades do ambiente SIMULA [FRO97]

2.5.4.1 SIMULA ++

No trabalho desenvolvido por Cordenonsi [COR99], é apresentada uma nova versão do ambiente SIMULA. Nesta versão, chamada SIMULA ++, há a possibilidade de utilização de regras de evolução, ou seja, o uso de algoritmos genéticos.

SIMULA ++ foi desenvolvido tendo como base o modelo proposto por [FRO97], possuindo algumas características principais de diferenciação. São elas:

- definição do comportamento: o sistema usa uma representação declarativa para o conjunto de regras que definem o comportamento do agente. As regras de comportamento não são estáticas, como SIMULA. Estas regras podem sofrer modificações dinâmicas, em tempo de execução. Para cada agente é definido um conjunto de regras, dependendo das características do mesmo.
- evolução e adaptabilidade: a característica principal do SIMULA ++ é a adaptabilidade que o conjunto de regras que o agente possui ao ambiente. Esta adaptabilidade é o resultado do uso de algoritmos de evolução, utilizados individualmente em cada conjunto de regras dos agente. Através de operações de recombinação e mutação, realizadas no conjunto inicial de regras de cada agente, são feitos testes dinâmicos no sistema.

SIMULA ++ também foi desenvolvido na linguagem Java, para que grande parte das definições realizadas em SIMULA pudessem ser utilizadas sem a necessidade de grandes alterações.

2.6 Comparativo entre Ambientes de Simulação Baseados em Agentes

Aspectos considerados importantes, não apenas à ambientes de simulação, mas para qualquer software (interface, por exemplo), são analisados nesta seção. O comparativo entre os ambientes de simulação baseados em agentes estudados visa verificar como cada implementação buscou solucionar problemas comuns.

2.6.1 Linguagem de Programação

Todos os ambientes, quando projetados, tentaram ser o mais portáveis possíveis. O ambiente SIEME, em sua primeira versão, foi desenvolvido em *SmallTalk*, e após, em Java para ser compatível com a WEB.

O SWARM foi desenvolvido em C. Já o SIMULA possui versões em C++ (primeira) e em Java (versão atual).

O ambiente SeSAm foi primeiramente implementado em Lisp para Macintosh. A nova versão foi implementada em Java, sendo altamente portável.

Todos os ambientes rodam em UNIX, LINUX, DOS e Macintosh, tornando-os de uma portabilidade bastante superior.

A linguagem de programação, ao ser escolhida, é um fator de extrema importância, pois além de portabilidade, deve garantir robustez e eficácia para tornar possível a implementação de outros aspectos do ambiente, como o paralelismo.

2.6.2 Paralelismo

Em sistemas de simulação de tempo real, a capacidade de realização de atividades de modo paralelo é necessária, visto que a execução de tais tarefas deve ser da forma mais rápida possível. Segundo [KLÜ 2002], o desenvolvimento de simulações assim exige o mapeamento de interações paralelas entre os agentes, tornando sua implementação muito mais complexa.

O ambiente SWARM tem agentes genéricos, pré-desenvolvidos, suas bibliotecas e seu *kernel* estão preparados para suportar interações concorrentes.

No caso dos ambientes SeSAm e SIMULA não houve, em um primeiro momento, preocupação com o desenvolvimento de um ambiente que permitisse o paralelismo. Nos documentos pesquisados destes softwares, não há referência direta quanto a este fator.

O SIEME, em sua versão para Java, tem suporte para interações paralelas, onde o modelo criado para a simulação é duplicado em todas as máquinas, de forma a ter-se um ambiente de simulação distribuído.

2.6.3 Modelos de Simulação

Para que a simulação seja realizada, o problema real é dividido em elementos menores e suas interações. Cada um destes elementos é modelado. O conjunto de todos estes elementos é chamado de modelo. A partir da definição do modelo de simulação, a implementação pode se tornar muito mais simples.

O ambiente SIEME propõe um modelo baseado em eventos. Sua simulação analisa entidades, ações, interações e regras de ambiente.

Para SeSAm, um aspecto importante para que a modelagem de modelos complexos seja realizada é o conteúdo do agente, ou seja, atributos, objetivos, atitudes, crenças, etc., que ele possui. Como o modelo consiste nas funcionalidades e interações dos agentes, ter uma definição correta das características de cada agente torna o modelo mais robusto.

No ambiente SIMULA, o usuário, ao criar o modelo, define os agentes, suas funcionalidades e interações e o ponto inicial da simulação.

Os ambientes de simulação têm que buscar a forma mais prática de tornar problemas reais em modelagens de simulação. O objetivo dos quatro ambientes analisados é fazer com que a implementação destes modelos possa ser realizada de forma amigável e com uma alta abstração de estruturas de dados existentes nos mesmos.

Pré-definir funcionalidades nos agentes torna a implementação da simulação mais fácil, pois coloca os pontos de programação em um nível de abstração maior. Ou

seja, o usuário do sistema não tem a preocupação com problemas quanto a linguagem de programação utilizada [KLÜ2002].

Em três dos ambientes analisados, SWARM, SeSAm e SIMULA, foram encontradas primitivas pré-definidas para as funcionalidades. Estas primitivas são conjuntos de comportamentos pré-definidos considerados pelos sistemas como normalmente necessários. Também fica a disposição do usuário, caso os comportamentos pré-definidos não satisfaçam as necessidades para criação da simulação, ferramentas para a implementação de novos comportamentos. Para os ambientes SWARM e SIMULA, quando da criação de novos comportamentos, o usuário deve ter domínio da linguagem de programação na qual o ambiente está implementado, como também da estrutura de dados existente para que sua implementação possa ser utilizada no ambiente sem causar erros na execução. O SeSAm disponibiliza uma linguagem para definição de novas funções, que facilita a definição de novas primitivas pelo usuário.

2.6.4 Tipos de Agentes

Todos os ambientes analisados fazem uso de agentes reativos. Segundo [FRO 97], no SIMULA, o processo de criação de regras em sistemas com agentes reativos é crucial, visto que sendo reativos, os agentes agem na dependência de suas regras. Desta forma, sendo os agentes reativos em todos os ambientes, a definição de comportamentos pré-definidos, citados na seção anterior, torna o processo de criação de uma simulação muito mais fácil.

Porém, na nova versão do ambiente SIMULA, o SIMULA ++ atribuiu adaptação aos agentes, onde, com o uso de algoritmos evolutivos, os mesmos sofrem alterações em suas regras automaticamente. Estas alterações são testadas dinamicamente, visando não trazer prejuízos aos objetivos que os agentes possuem.

2.6.5 Facilidades

O ambiente que apresenta uma grande quantidade de facilidades disponíveis é o SeSAm. Algumas das ferramentas disponíveis neste ambiente não foram encontradas com similaridade nos outros ambientes. Uma ferramenta muito interessante é a “Manipulação Direta”, que permite parar o sistema em um ponto desejado e realizar uma análise até o momento, facilitando o processo de criação, caso hajam erros na implementação realizada. Outra ferramenta muito prática é o módulo de “Estatísticas Básicas”, onde dados podem, primeiramente ser visualizados no ambiente e depois podem ser exportados para a realização de análises e amostragens da simulação.

2.6.6 Interface

Um aspecto bastante importante de análise em um ambiente de simulação é a sua interface. Como a idéia em ambientes de simulação é o seu uso em diversas áreas de interesse, seus usuários podem ser pessoas com conhecimento restrito na área de informática, fazendo com que a interface do ambiente seja um ponto crucial para sua utilização por um número maior de pessoas. A facilidade do uso do ambiente muito tem a ver com a amigabilidade que este possui, isto é, quanto seus usuários se sentem a vontade para utilizá-lo.

As interfaces têm outros aspectos a serem analisados, como a clareza para a definição dos agentes, de seus comportamentos, a forma como os agentes irão interagir uns com os outros, etc. Quando existirem os comportamentos pré-definidos, a interface pode ser mais facilmente manipulada, pois sabe-se a priori o que pode acontecer durante a definição dos comportamentos dos agentes.

Segundo [SIE2001], o ambiente SIEME apresenta uma interface de fácil utilização. O ambiente SWARM possui interface linha de comando, o que a torna mais complexa para o uso de pessoas menos adaptadas com o ambiente.

A utilização de uma interface gráfica torna o ambiente mais amigável. Em muitos casos, o uso de interface gráfica torna os sistemas mais lentos, porém para a maioria das aplicações vale a pena a perda de desempenho pela usabilidade que o sistema passa a ter.

No SIMULA, uma das preocupações iniciais foi o desenvolvimento de uma interface gráfica, sendo esta amigável e de fácil utilização. O SeSAm também possui uma interface gráfica muito poderosa capaz de dar suporte à sua grande quantidade de recursos.

Com uma visão geral, as interfaces construídas para os ambientes SeSAm e SIMULA apresentam características que as tornam melhores que os outros ambientes, devido a amigabilidade e clareza que possuem. Não há como comparar a interface de SWARM com as outras, devido ao fato de não ser gráfica. As interfaces de SIMULA e SIEME foram escritas (menus e comandos) na língua natural de seus desenvolvedores, ou seja, SIMULA em português e SIEME em francês, dificultando sua utilização por indivíduos de outras nacionalidades. SeSAm, em sua primeira versão (Lisp), foi escrito em alemão, porém, sua nova versão (em Java) está em inglês. Seria interessante que todos softwares possuíssem uma versão da interface em inglês, para que seu uso fosse universal.

3 Emoções e Modelo OCC

Existem muitos argumentos que sugerem que as emoções afetam a tomada de decisão nas atividades humanas. Para Gratch *et al.* [GRA2001b], emoções são a regra central de nossas vidas. O estado emocional tem grande impacto na tomada de decisões, ações, memória, atenção, etc. Existem diversas propostas para a estruturação das emoções, a fim de melhor explicar como estas funcionam e também realizar simulações em máquina.

Devido a uma série de fatores, tanto sociais quanto fisiológicos, a simulação de situações onde emoções têm um papel decisivo é algo complexo. Porém, com a utilização da tecnologia de agentes, vários pesquisadores vêm realizando trabalhos nesta área, como: Bates [BAT94], Sloman [99], Damásio [DAM94], Gratch *et al.* [GRA2001a,GRA2001b], Picard [PIC97,PIC2001], Bazzan *et al.* [BAZ2001,BAZ 2002] e Elliott [ELL94a,ELL94b]. Estes trabalhos realizaram estudos e apresentaram resultados onde a inserção de emoções pode alterar a tomada de decisão.

Neste capítulo é apresentado estudo direcionado para a finalidade do trabalho proposto sobre emoções e o modelo proposto por Ortony *et al*. [ORT88], que é o modelo escolhido como base teórica de estruturação das emoções para deste trabalho. A última seção apresenta alguns trabalhos que estão sendo realizados por grupos de pesquisa na área de emoções e IA .

3.1 A natureza do problema

Emoções são o centro das motivações humanas: elas são tanto precursor quanto resultado final em muitos empreendimentos. São parte dos nossos relacionamentos, das histórias que contamos, e dos planos que fazemos. Por isso, emoções têm sido objeto de estudo desde os tempos mais remotos, com estudos de Lao-Tzu e Sócrates [BER2001]. Porém, ainda há uma série de perguntas sem respostas. Por quê?

Primeiro, não existe um consenso quanto a definição de emoção. Segundo Reber: "*historicamente este termo tem provado total discordância para sua definição. Provavelmente não há na psicologia outro termo que não tenha uma definição e seja usado tão freqüentemente.*" [REB85] Del Nero [DEL97] afirma que a emoção é um processo consciente e que em conjunto com o pensamento e a vontade formam os protagonistas principais para o palco que é a mente. Para Damásio [DAM2000], a emoção é um rótulo que designa um conjunto de fenômenos ou comportamentos. Ele divide as emoções em primárias (medo, alegria, tristeza, raiva, etc.) e as secundárias (ciúme, culpa, orgulho, etc.). Moffat *et al.* [MOF2000] dizem que as emoções são funcionais, isto é, elas possuem um valor adaptativo para o organismo, contradizendo a convenção

existente de que as emoções não são racionais. Sloman [SLO2001] também conclui que não há uma definição única de emoção, pois esta depende de como se analisa quais são as concepções individuais dos seres humanos ou outros animais em relação ao assunto.

Em segundo lugar, deve-se considerar que alguns tratamentos das emoções são realizados tomando como base a fisiologia: emoções são coisas que sentimos. Separando isso dos aspectos cognitivos já teremos um modelo complicado. Damásio [DAM94] realiza estudos envolvendo três áreas de pesquisa: Filosofia, Psicologia e Biologia. A união destas três áreas é conhecida por neurociência cognitiva. Para ele: "*um sistema fisiológico complexo, com uma longa história evolucionária que existe em ambientes complexos, socialmente construídos, e cujas reações emocionais ao mundo são determinadas tanto por um caminho de resposta "built-in" como por mecanismos menos automáticos, mediados por significados e cognições*".

Em terceiro, deve-se considerar os estímulos para emoções, e quão complexas estas são. Cada ação é dependente de quando esta ocorreu, e é necessário saber quais eram os objetivos, os padrões e as preferências no momento da ocorrência do fato. Por exemplo, para mapear uma ação isolada, como a aquisição de um objeto de valor, podem estar escondidos vários estados emocionais, como alegria, medo, orgulho, satisfação, etc. As emoções têm uma natureza pessoal muito grande, pois cada um pode apresentar reações diferentes para a mesma situação.

Desta maneira, tem-se ainda um longo caminho a percorrer, pois o que foi estudado até agora é apenas uma pequena parte do todo. Expressões faciais, expressões corporais, mudanças de inflexões, escolha de palavras, interpretações subjetivas do comportamento são alguns dos trabalhos já realizados. Porém, estes trabalhos, na sua grande maioria, foram realizados isoladamente, e no ser humano estas reações ocorrem simultaneamente.

A fim de um melhor entendimento do funcionamento das emoções, foram propostos diversos modelos para a estruturação das emoções, cada uma visando aspectos diferentes do ser humano. Alguns de cunho psicológico, como percepção, sentimentos, experiências, cognição e comportamento [MOF2000][CAN99] [ORT88], e outros de cunho fisiológico, como batimentos cardíacos, pressão arterial e sudorese [SLO99] [DAM94] [PIC97].

3.2 Modelo Proposto por Ortony, Clore e Collins (Modelo OCC)

3.2.1 Emoções Básicas

Muitos pesquisadores afirmam que não existe um conjunto de emoções básicas, como Damásio [DAM94], Sloman [SLO99] e Ortony *et al.* [ORT88], porque muitas emoções podem ser distintas e igualmente básicas. Existem algumas perguntas a serem respondidas:

- Como sustentar que existam emoções básicas que possam ser universais?
- Estas emoções consideradas básicas formam emoções complexas, ou mistura de emoções?

- Emoções não básicas podem ter se desenvolvido antes das emoções básicas?

Conceituar emoções básicas é algo complicado, porque algumas emoções como alegria, tristeza, raiva e medo são perfeitamente conhecidas em todas as culturas. Algumas emoções são mais básicas que outras porque é dado um significado específico muito maior para elas, enquanto outras recebem uma especificação mais abstrata.

Uma forma de tratar a complexidade das emoções é determinar o grau de diferenciação de cada reação afetiva. Isto significa a distinção de um estado emocional para uma forma individual deste estado. Esta é uma proposta hierárquica, onde, no topo existem dois tipos de reação: as reações afetivas positivas e as reações afetivas negativas. Estas reações geram um valor, positivo ou negativo, que determinam a intensidade desta no organismo. Para esta valoração da reação, dá-se o nome de "reação de valência". As reações de valência são ingredientes importantes das emoções, pois todas as emoções envolvem alguma classificação de reações positivas ou negativas para algo. Desta forma, é melhor tratar as emoções em nível de diferenciação do que defini-las como básicas ou não.

3.2.2 Modelo OCC

O modelo OCC, assim chamado por ter sido desenvolvido por Ortony, Clore & Collins [ORT88], é baseado no princípio da diferenciação entre reações de valência positivas ou negativas.

No modelo OCC existem três aspectos que alteram as reações do mundo: os eventos, os agentes e os objetos. Os eventos são interessantes porque são analisadas suas conseqüências, os agentes porque são analisadas suas ações e os objetos porque são analisados aspectos e propriedades que estes possuem.

As emoções são reações, e as reações sempre agem sobre a perspectiva do mundo. É fácil perceber que diferentes organismos, ou pessoas em diferentes culturas têm reações diferentes sobre o mesmo assunto.

Dito de outra maneira, essa organização direciona o foco de interesse e determina a concepção das variáveis que regulam a intensidade da reação do indivíduo à situação tida como afetiva ou geradora de emoções. Uma das linhas centrais do modelo OCC é a idéia de que emoções são reações valoradas e que a intensidade dessas reações afetivas é o fator determinante à geração da emoção no indivíduo.

A estrutura geral do modelo baseia-se em três ramos principais, correspondendo aos três modos de reação que pode-se ter ao mundo. Esta estrutura apresenta uma descrição lógica, não uma descrição temporal dos fatos. Para que uma reação afetiva se transforme em emoção, deve-se determinar a intensidade que esta reação possui. É importante saber o que afeta a intensidade das emoções. A partir do estudo realizado por Ortony *et al.*, foram identificadas 22 emoções, porém no trabalho realizado por Elliott [ELL94], foram inseridas mais 4 emoções, sendo uma combinação de outras emoções já identificadas pelo modelo OCC. Estas 26 emoções estão divididas nos três ramos (eventos, agentes e objetos). Ver Anexo I – Categorias de Emoções.

Para que o modelo proposto não perca sua concepção inicial, os termos relacionados com as emoções serão mantidos em inglês, a fim de não haver distorções relacionadas às subdivisões propostas.

Os três grupos formam linhas de estudo sobre emoções e são classificados em subgrupos. O grupo que diz respeito às emoções que surgem das percepções do indivíduo sobre o **objeto**, como gostar, não gostar, inclui as emoções de *liking (*amizade) e *disliking* (antipatia). As emoções desse grupo são determinadas por atração em relação ao objeto ("*Atraction*").

O segundo grupo, o de **eventos**, relaciona emoções que são vistas como consequências de eventos. Esse grupo subdivide-se em:

- Subgrupo – *Consequences for Others* (conseqüências para os outros) ("*Fortunes-of-Others*"):
  - *Happy-for* (alegre por): a pessoa está feliz com a situação ou evento, pois acredita que as conseqüências deles são desejadas ou são boas e adequadas a outros;
  - *Gloating* (regozijo com a tristeza alheia): a pessoa está feliz com a situação ou evento, pois acredita que as conseqüências deles são desagradáveis ou indesejáveis a outrem.
  - *Resentment* (ressentimento): uma pessoa está triste e insatisfeita por acreditar que os efeitos do evento são agradáveis para outros;
  - *Jealousy* (ciúme): ressentimento sobre um objetivo exclusivo mutuamente desejado;
  - *Envy* (inveja): ressentimento sobre um objetivo que não é exclusivo, porém mutuamente desejado;
  - *Sorry-for* (tristeza por): pessoa fica triste porque um evento é desagradável ou indesejável para outros.

- Subgrupo - *Consequence for self* (conseqüências para si mesmo)
  - *Prospects Relevant* (Perspectiva relevante das consequências) ("*Prospect-Based*"):
    - *Hope* (esperança): uma pessoa está satisfeita com as conseqüências importantes que o evento pode trazer para si;
    - *Fear* (medo): uma pessoa está triste ou possui receios para com as conseqüências importantes que um determinado evento pode trazer para si;
  - Perspectiva relevante, onde se acredita que os eventos podem ser confirmados ou não ("*Confirmation*"):
    - *Satisfaction* (satisfação): a pessoa tem confirmada sua esperança nas consequências de uma determinada situação;
    - *Disappointment* (desapontamento): uma pessoa sofre por ver suas esperanças nas conseqüências de um evento não se realizarem;

- *Relief* (alívio): a pessoa observa a não realização das consequências importantes previstas para um determinado evento que lhe infundiu medo;
- *Fears-confirmed* (medo ou pavor confirmado): a pessoa observa a confirmação das consequências importantes previstas para um determinado evento que lhe infundiu medo.

▪ *Prospects Irrelevant* (Perspectiva irrelevante) ("*Well-Being*"):

- *Joy* (alegria): o indivíduo está muito alegre ou satisfeito com as consequências que o evento permitiu realizar;
- *Distress* (angústia): o indivíduo não está contente com as conseqüências que o evento permitiu realizar.

O terceiro grupo, o dos **agentes**, compreende as emoções com a propriedade de atribuição ("*Atribution*"):

- *Pride* (orgulho): quando um indivíduo aprova a conduta de um agente que está relacionado fortemente consigo ou é o próprio;
- *Admiration* (admiração): quando um indivíduo aprova a conduta de um outro agente;
- *Shame* (vergonha): quando um indivíduo não aprova a conduta de um agente que está relacionado fortemente consigo ou é o próprio;
- *Reproach* (censura): quando um indivíduo não aprova a conduta de um outro agente.

Adicionalmente, dois outros subgrupos são formados pela união de emoções que foram apresentadas anteriormente. A união da subdivisão *Well-Being* (eventos) com a subdivisão *Atribution* (agentes), tem por foco tanto a ação de um agente como as conseqüências de seus eventos ("*Atribution/Well-Being*"):

- *Remorse* (remorso): quando um indivíduo está triste e insatisfeito devido a sua própria conduta e ação (*shame* + *distress*);
- *Gratitude* (gratidão): quando um indivíduo aprova a ação de outrem e com o evento relacionado, sentindo-se agradecido (*admiration* + *joy*);
- *Anger* (raiva): quando um indivíduo desaprova radicalmente uma ação de outrem e está insatisfeito com uma situação ou evento relacionado à ação (*reproach* + *distress*);
- *Gratification* (gratificação): o indivíduo está satisfeito consigo mesmo e com sua ação e com o evento relacionado (*pride* + *joy*).

A união da subdivisão *Atraction* (objetos) com a subdivisão *Atribution* (agentes), tem por foco tanto a ação de um agente como os aspectos dos objetos ("*Atraction/Atribution*"):

- Love (amor): o indivíduo tem sobre os objetos e outros indivíduos um sentimento positivo, gostando das atitudes dos mesmos (*admirition* + *liking*);

- *Hate* (ódio): o indivíduo tem sobre os objetos e outros indivíduos um sentimento negativo, não gostando das atitudes dos mesmos (*reproach + disliking*)

Na Figura 3.1 pode-se verificar como estão estruturadas as oito subdivisões de emoções do modelo OCC [ORT88], juntamente com as alterações propostas por Elliott [ELL94], dependentes das reações de valência.

Ortony *el. at* complementam que este modelo é bastante simplificado, pois na realidade uma pessoa sente uma mistura de emoções, em diferentes situações. Porém, este modelo é importante pois tenta simplificar o desenvolvimento de uma teoria focando vários aspectos cognitivos das emoções.

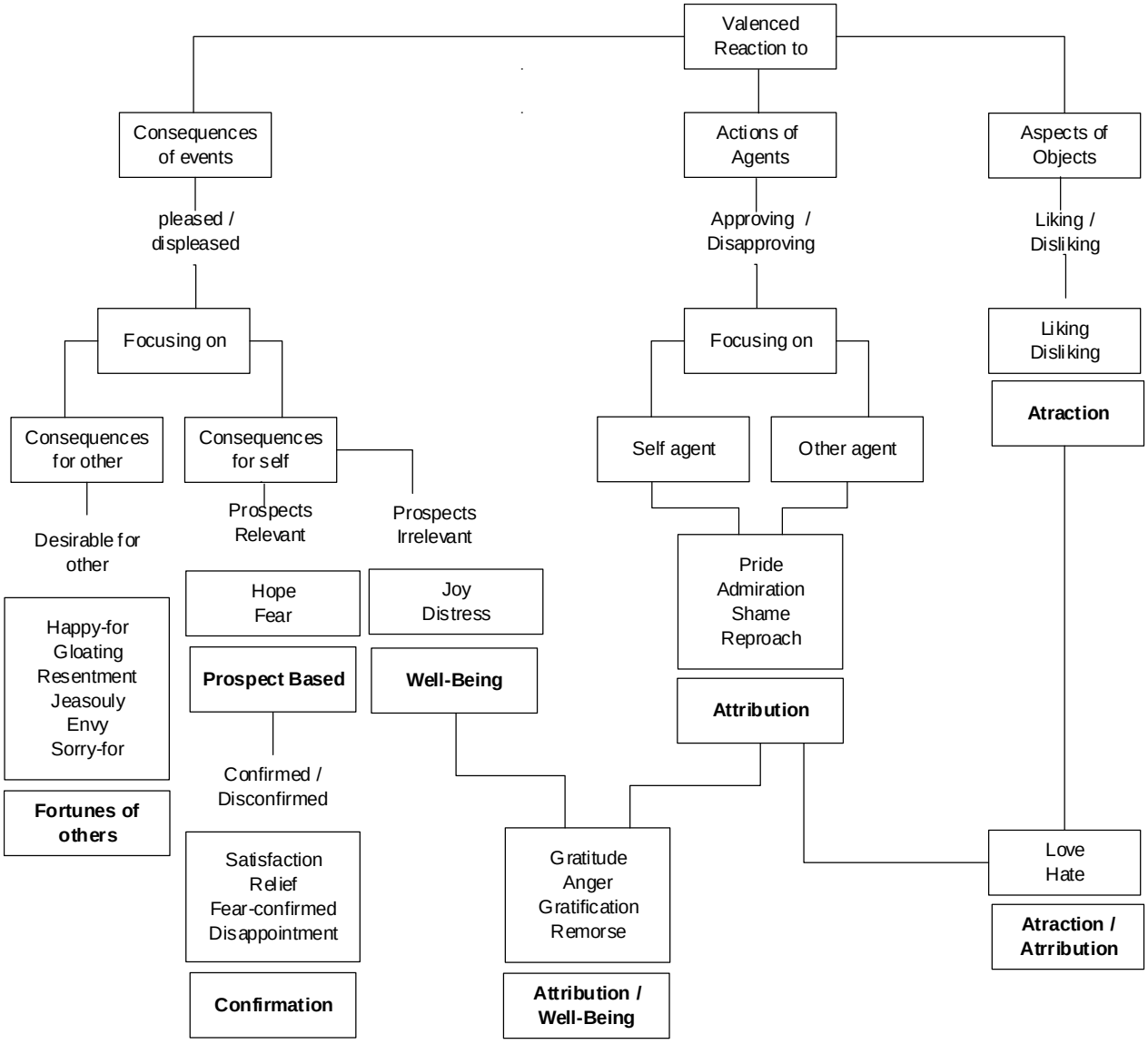


FIGURA 3.1 - Estrutura das emoções no modelo OCC [ORT88][ELL94]

Como dito anteriormente, a intensidade das reações afetivas é fator determinante na geração de emoção no indivíduo. O modelo OCC faz distinção entre variáveis locais

e globais de intensidade, para que a implementação do modelo ocorra de forma facilitada.

Variáveis globais afetam todos os tipos de emoções, e incluem:

- *Sense of reality* (senso de realidade): como cada indivíduo percebe uma situação;
- *Proximity* (proximidade): a qual proximidade de sentimento cada indivíduo está de uma situação;
- *Unexpectedness* (não esperado): como cada indivíduo é surpreendido em uma mesma situação;
- *Arousal* (despertar - estímulo): quanto um indivíduo é estimulado anteriormente para uma situação.

As variáveis locais afetam apenas os grupos de emoções:

- Emoções baseadas em eventos são afetadas por variáveis de desejo. O subgrupo de emoções *propects based* são afetadas por variáveis de *likelihood* (grau de crença que um evento ocorra) and *effort realisation* (grau para que um evento ocorra). O subgrupo de emoções *fortune of others* são afetadas por variáveis de *desirability-for others* (quanto se deseja que os objetivos dos outros se realizem), *liking* (afinidade com a outra pessoa) e *deservingness* (quanto você acha que a outra pessoa merece que aquilo aconteça).
- Emoções baseadas em agentes dependem de *praiseworthiness* (merecimento) ou *blameworthiness* (desprezo) a outro agente ou a si mesmo e pela *expectation-deviation* (quanto custa derivar as ações do agente para normas esperadas).
- Emoções baseadas em objetos são afetadas por *familiarity-of-the-object* (familiariade com o objeto) e por *appeal* (apelo) ao objeto.

3.2.3 Sistema baseado em regras

O modelo OCC pode ser traduzido para um sistema baseado em regras, o qual sintetiza e gera as emoções. Estas regras são do tipo estruturado (IF-THEN). A parte IF das regras testa o desejo (de uma conseqüência em um evento), ou merecimento (de uma ação em um agente), ou apelo (de um objeto). A parte THEN realiza o cálculo do potencial de geração para cada emoção.

Para formar as regras de cada uma das subdivisões do modelo (eventos, agentes e objetos) deve-se avaliar suas variáveis, no caso desejo, merecimento e apelo. Afora as variáveis locais, outros fatores influenciam na formação da regra: o potencial de geração de um estado para determinar a emoção ($P_{EMO);}$ as variáveis globais G; a intensidade da emoção ($I_{EMO);}$ um valor de limiar ($T_{EMO)}$ e uma função específica para a emoção (fn()). Como em cada subdivisão a formação da regra é a mesma (utiliza a mesma variável local) será exemplificada apenas uma emoção de cada subgrupo.

a) Grupo objeto → emoção LIKING:
   Para calcular o potencial de geração de uma emoção, a regra é a seguinte:
   IF A(p,o,t) > 0
      THEN set P_{LIK}(p,o,t) = fn (A(p,o,t), G)

onde:
A(p,o,t) = nível de apelo que uma pessoa *p* para um objeto *o* no tempo *t*.
P_{LIK}(p,o,t) = potencial que uma pessoa *p* tem para um objeto *o* no tempo *t*.
fn(A(p,o,t), G) = função específica que gera a emoção de LIKING, de acordo com o nível de apelo A e pelas variáveis globais G.

Para calcular a intensidade da emoção, na verificação se esta será manifestada ou não, a regra é a seguinte:

IF $P_{LIK}(p,o,t) > T_{LIK}(p,t)$
THEN set $I_{LIK}(p,o,t) = P_{LIK}(p,o,t) - T_{LIK}(p,t)$
ELSE set $I_{LIK}(p,o,t) = 0$

onde:
T_{LIK}(p,t) = valor limiar para que a pessoa *p* altere sua emoção no tempo *t*.
I_{LIK} (p,o,t) = intensidade que uma pessoa *p* tem para um objeto *o* no tempo *t*.

b) Grupo agente → emoção PRIDE:
Para calcular o potencial de geração de uma emoção a regra é a seguinte:
IF M(p,a,t) > 0
THEN set $P_{PRI}(p,a,t)$ = fn (M(p,a,t), G)

onde:
M(p,a,t) = nível de merecimento de uma pessoa *p* para um agente *a* no tempo *t*.
P_{PRI}(p,a,t) = potencial que uma pessoa *p* tem para um agente *a* no tempo *t*.
fn(M(p,a,t), G) = função específica que gera a emoção de PRIDE, de acordo com o nível de merecimento M e pelas variáveis globais G.

Para calcular a intensidade da emoção, na verificação se esta será manifestada ou não, a regra é a seguinte:
IF $P_{PRI}(p,a,t) > T_{PRI}(p,t)$
THEN set $I_{PRI}(p,a,t) = P_{PRI}(p,a,t) - T_{PRI}(p,t)$
ELSE set $I_{PRI}(p,a,t) = 0$
onde:
T_{PRI}(p,t) = valor limiar para que a pessoa *p* altere sua emoção no tempo *t*.
I_{PRI} (p,a,t) = intensidade que uma pessoa *p* tem para um agente *a* no tempo *t*.

c) Grupo eventos → emoção FEAR:
Para calcular o potencial de geração de uma emoção a regra é a seguinte:
IF D(p,e,t) > 0
THEN set $P_{FEA}(p,e,t)$ = fn (D(p,e,t), G)

onde:
D(p,e,t) = nível de desejo de uma pessoa *p* para um evento *e* no tempo *t*.
P_{FEA}(p,e,t) = potencial que uma pessoa *p* tem para um evento *e* no tempo *t*.
fn(D(p,e,t), G) = função específica que gera a emoção de FEAR, de acordo com o nível de desejo D e pelas variáveis globais G.

Para calcular a intensidade da emoção, na verificação se esta será manifestada ou não, a regra é a seguinte:

IF $P_{FEA}(p,e,t) > T_{FEA}(p,t)$

THEN set $I_{FEA}(p,e,t) = P_{FEA}(p,e,t) - T_{FEA}(p,t)$

ELSE set $I_{FEA}(p,e,t) = 0$

onde:

$T_{FEA(}p,t)$ = valor limiar para que a pessoa *p* altere sua emoção no tempo *t*.

$I_{FEA}(p,e,t)$ = intensidade que uma pessoa *p* tem para um evento *e* no tempo *t*.

O modelo OCC omite a maioria dos detalhes de implementação, sendo que o maior problema do sistema baseado em regras é a definição da função específica de cada emoção (fn()).

3.3 Trabalhos Existentes

Alguns grupos de pesquisa estudam emoções ligadas a Inteligência Artificial. Estes trabalhos foram subdivididos em três categorias, segundo os modelos propostos: em nível de arquitetura, em nível de tarefas e em nível de mecanismos [SAL2002]. Os modelos em nível de arquitetura buscam desenvolver ambientes baseados em modelos de emoções. O modelo OCC é a base de diversos trabalhos, pois é analisado como o modelo que pode ser expresso computacionalmente de forma mais direta e foi estudado e desenvolvido para este fim, porém ainda não foi implementado completamente [ORT88][BAT94].

Os modelos em nível de tarefa estão focados no desenvolvimento específico de alguma habilidade humana, como compreensão da linguagem natural, visão, etc., e a ligação das emoções transmitidas através destas habilidades.

Nos modelos em nível de mecanismos há preocupação com a implementação de novos modelos para as emoções. Vários modelos são propostos, analisando diferentes aspectos que interferem no processo cognitivo e emocional, determinando desta forma qual será o processo de modelagem para a teoria escolhida. O modelo OCC é um exemplo de modelo em nível de mecanismo.

3.3.1 Modelos em nível de arquitetura

- OZ Project - Carnegie Mellon University - Bates e Reilly [BAT94][REI92]: modelo de emoções como parte de um projeto maior cujo o objetivo é construir os agentes autônomos que habitam um micro mundo virtual. O objetivo não é fornecer uma teoria de emoções humana ou animal, mas sim desenvolver uma nova forma, mais simples e fácil capaz de produzir micro mundos com agentes que parecem ser emocionais. O modelo da emoção é encaixado em um módulo da arquitetura total e as emoções são utilizadas como meio para criação de um comportamento social mais verídico. O módulo da emoção, chamado Em, executa um subconjunto do modelo cognitivo de Ortony, de Collins e de Clore (Modelo OCC). Os objetivos atuais e o estado emocional apropriado se complementam, como por exemplo, ficar feliz quando o objetivo é estar satisfeito. Existe um mecanismo para determinar qual estímulo causa uma reação emocional incorporando mecanismos cumulativos de um ponto inicial. Dependendo da intensidade de um estímulo, uma ou outra reação

emocional pode ser acessada. As emoções têm constantes de deterioração e podem persistir por períodos de tempo, dependendo do tipo da emoção e da situação em que o ambiente se encontre.

- Cognition and Affection Project, Universidade de Birminghan, Reino Unido – Sloman [SLO99]: utiliza uma aproximação para explorar as emoções, que é análoga a teoria cognitiva de Newell [NEW89]. A idéia é que os mecanismos cognitivos não podem ser investigados isolados, e devem estar inseridos em uma arquitetura que inclua mecanismos de percepção, geração e gerenciamento de movimentos e estados afetivos, a partir de cada agente. Algumas emoções importantes são encontradas tipicamente em seres humanos, porém não em outros animais, tais como humilhação, antecipação, paixão, descoberta, desespero, etc; que são consideradas como propriedades emergentes dos sistemas de controle complexos que compreendem instrumentos cognitivos e não podem ser analisadas por estruturas separadas. Na verdade, Sloman não desenha uma linha entre processo cognitivo e o processo emocional. Seu trabalho coloca as emoções como uma "tendência disposicional", isto é, as emoções são fortemente acopladas a motivação e aos objetivos do organismo. As emoções podem ou não ser conscientes. A arquitetura cognitiva implementa estas idéias a partir de uma "hierarquia das disposições" e segue algumas das idéias a respeito do controle e da interrupção dos processos cognitivos estudados por Simon [SIM67], e definidos por uma teoria global da interrupção do sinal de Oatley e Johnson [OAT87].

- Elliot's Affective Reasoner System – Clark Elliott [ELL94a][ELL94b]: o foco deste trabalho está na emulação da entrada/saída no processo afetivo ou como deduzir o estado afetivo de outro agente. O sistema traça situações e variáveis de estado do agente em um conjunto de emoções específicas, e produz comportamentos correspondentes a elas. O "Affective Reasoner" é um modelo simbólico implementado a partir de uma versão prolongada do modelo de OCC, isto é, implementa um processo cognitivo de avaliação. O Affective Reasoner funciona em um mundo de agentes, onde cada agente possui um conjunto simbólico de avaliação que contém os objetivos, as preferências e estado atual do agente. Um comportamento emocional específico é determinado para uma situação particular, baseada neste conjunto de avaliação (por exemplo, se os objetivos do agente forem frustrados, a este pode ser atribuído um comportamento de irritação, raiva). Os agentes são capazes de indicar emoções, mudando seu estado emocional, e capazes de derivar emoções de outros agentes, a partir da observação do comportamento dos mesmos. O Affective Reasoner também apresenta uma interface multi-midia, com reconhecimento de fala (restrita), tradução de palavras, e apresentação de expressões faciais para uma variedade de emoções. Aplicações do Affective Reasoner podem ajudar na melhora da interação homem-máquina, análise de teorias de avaliação de emoções e o desenvolvimento de agentes mais interativos para jogos de computador.

- Eugénio Oliveira – Faculdade de Engenharia da Universidade do Porto – Portugal [OLI2002]: este projeto visa desenvolver uma arquitetura para agentes inteligentes com emoções. Em sua proposta, os mecanismos desenvolvidos utilizaram como base para as emoções modelos propostos por Damásio [DAM94], Picard [PIC97] e Sloman [SLO2001]. Esta arquitetura está subdividida em um módulo para definição das regras de emoção e um módulo para armazenamento de fatos acontecidos anteriormente (memória). As emoções seguem um padrão de "Valência Emocional",

a partir da realização de um cálculo, buscando dados da situação atual e informações anteriores armazenadas em sua memória. Isso permite que cada agente possua uma personalidade distinta.

- Jonathan Gratch e Stacy Marsella – Universidade da Califórnia do Sul – Estados Unidos [GRA2001a][GRA2001b]: este trabalho tem como objetivo desenvolver ambientes virtuais, onde os agentes participantes executam atividades decorrentes de alguma reação emocional. A arquitetura dos agentes destes ambientes utiliza um subconjunto do modelo cognitivo OCC. Um exemplo muito interessante utilizado para simulação foi uma missão de guerra, tendo com modelo virtual uma vila na Bósnia.

- Magda Bercht - Universidade Federal do Rio Grande do Sul - Brasil [BER2001]: sua tese de doutorado, intitulada “Em direção a Agentes Pedagógicos com dimensões afetivas”, apresenta o uso de aspectos afetivos como apoio à decisão de ação em Sistemas Tutores Inteligentes (STI). O trabalho leva em consideração fatores afetivos de modo a tornar mais flexível e adaptável a interação entre os “atores” envolvidos no processo de ensino-aprendizagem. Foi definida uma arquitetura para apoiar um STI de modo a reconhecer alguns dos fatores afetivos, representativos de estratégias de ação de agentes humanos em interação com agentes artificiais. Esse reconhecimento é realizado através de construções retiradas dos comportamentos observáveis do agente humano em contextos determinados. A arquitetura prevê um Sistema Multiagente para executar a percepção do reconhecimento de fatores afetivos e da conduta do aluno em interação, e de um agente pedagógico, representando o tutor.

3.3.2 Modelos em nível de tarefas

- Dyer’s BORIS e OpEd [DYE87]: estes dois sistemas são focados na compreensão de linguagem natural. Estes modelos implementam uma cognição "cold " (fria), onde o processo cognitivo não é afetado pela emoção. Ambos utilizam representações simbólicas das emoções para obter os estados e intenções emocionais do ambiente. BORIS é um programa da compreensão de linguagem natural que tem capacidade para compreender estados emocionais de caracteres e, em seguida, inferir um estado emocional ao texto de uma história. A base de conhecimento de BORIS consiste em estruturas simbólicas que representam emoções como “*five-slot frames*”. A base teórica que define este modelo vem da teoria cognitiva da avaliação de Sloman [SLO78], que define emoções como elementos específicos de um conjunto definido pelo cruzamento das crenças, dos objetivos e dos estados do organismo. OpEd é uma extensão de BORIS onde as crenças são os primeiros objetos da classe e o sistema OpEd pode melhor inferir sobre os estados de opinião do sistema.

- Allen [ALL93]: implementa aspectos da teoria de Frijda *et al.* [MOF93] sobre emoções em um ambiente distribuído para agentes. A teoria de Frijda atribui às emoções a observação sobre os objetivos do indivíduo e verificação se esse comportamento contribui para sua satisfação. A arquitetura do agente consiste em diversos componentes, os quais se comunicam através do “blackboard”. Os principais componentes envolvidos no processamento da emoção são o analisador,

que processa entradas dos sistemas perceptivo e cognitivo, e fornece dados a um comparador, que compara as entradas com os objetivos atuais. O objetivo com nível maior de igualdade de comparação é selecionado para a ação.

- MIT Media Laboratory Affective Computing - Picard [PIC97][PIC2001]: este grupo realiza uma série de estudos, analisando diferentes aspectos, como controle e personalização de interfaces e simulação de sistemas com emoções. O foco principal de pesquisa é o envolvimento das emoções nas interações Ser Humano-Computador. Para que as emoções possam ser percebidas pelos sistemas desenvolvidos, vários dispositivos físicos (monitores de respiração, sensor de pulsação, etc) são utilizados no usuário. A expressão “Computação Afetiva” foi criada por Picard [PIC97] e tem como base científica as ciências cognitivas, a psicologia, a filosofia, a neurologia e a biologia. Picard afirma que o desenvolvimento de computadores afetivos é somente uma questão de tempo.

3.3.3 Modelos em nível de mecanismo

- Dyer’s DAYDREAMER [DYE87]: modela estados emocionais e suas influências na memória, no aprendizado, no planejamento e na geração de pensamento. O modelo tenta representar a cognição "hot” (quente), permitindo um efeito da emoção no processamento cognitivo. O formalismo de representação básico é simbólico, consistindo em estruturas de dependência conceitual compostas de planos, objetivos e atos primitivos. Emoções diferentes são expressas em termos de uma escala representando um estado positivo ou negativo. O estado emocional completo é uma função de polaridade dos objetivos individuais onde pesos diferentes são associados a estes.

- Scherer – Geneva Emotion Research Group [SCH93a][SCH93b]: construiu e implementou um modelo da computação da teoria cognitiva de avaliação. Este modelo é essencialmente um sistema baseado em conhecimento que testa as entradas de uma situação descrita em termos de 15 “dimensões de avaliação” e as combina a uma emoção definida em termos de 14 “componentes emocionais prototipados”. Nas duas dimensões (avaliação e emoção) a representação é realizada em “vetores de características” e a distância Euclidiana é calculada entre o vetor de entrada e o vetor de emoções.

- Chwelos e Oatley [CHW94]: este trabalho é uma crítica ao modelo de Scherer, mas reconhece sua contribuição à pesquisa da emoção. Especificamente, discutem que os cálculos realizados podem não gerar nenhuma emoção, bem como a incapacidade de gerar diversas combinações de avaliação para uma única emoção. Discutem uma variedade de outras aproximações para executar o processo de avaliação, cada um baseado em um formalismo computacional diferente, incluindo árvores de decisão, regras, combinação de padrões, e modelos conexionista. Concluem que a aproximação do método conexionista é o mais promissor, capaz de generalizar e extrair testes de padrões aproximados aos humanos.

- ACRES – Moffat e Frijda e Phaf [MOF93]: implementaram um processo cognitivo de avaliação baseado na teoria de Frijda da dedução das emoções. Como todos os modelos da teoria cognitiva de avaliação, este modelo possui um mecanismo de

combinação das características de uma situação em um conjunto de emoções de saída. As emoções podem ser consideradas como manifestações de um sistema que tenha interesses múltiplos e que opere num ambiente incerto. Com as funções de realização de interesse num ponto inicial, os fenômenos principais das emoções seguem as considerações de propriedades em um subsistema. Os fenômenos principais são: a existência de sentimentos de prazer e de dor, a importância de variáveis cognitivas ou de avaliação, presença de inanimação, comportamentos pré-programados, como construção de planos complexos para alcançar os objetivos das emoções, e da ocorrência de interrupções, distúrbios e impulsos para algo que seja prioridade dos objetivos emocionais. As propriedades do sistema são facilidades para a detecção da relevância dos eventos no que diz respeito aos interesses múltiplos, como sinais da relevância que podem ser reconhecidos pelo sistema da ação, as facilidades para a precedência de controle, ou a ordenação da prioridade dos objetivos. O ACRES é construído a partir das especificações fornecidas pela teoria de Frijda para a emoção. A interação homem-máquina envolve a execução de uma tarefa de manipulação do conhecimento (o domínio do conhecimento acontece sobre emoções). ACRES responde emocionalmente quando um de seus interesses (por exemplo, ser mantido ocupado) é acionado. As respostas são sinais sociais, como interrupção no processamento de uma tarefa, ou recusa para aceitar instruções.

- Araújo - Universidade de Birminghan, Reino Unido [ARA93]: implementou um modelo conexionista de processamento emocional que emula dois fenômenos psicológicos observados: o efeito do estado emocional no desempenho e o efeito do estado emocional na memória e na recordação. O modelo consiste em duas redes de interação conexionista: uma para processamento emocional (EN) e um para processamento cognitivo (CN). EN é uma rede recorrente em três níveis que calcula a valência de cada estímulo. Uma característica que distingue este modelo é que o EN pode modificar os parâmetros que controlam o processamento na rede cognitiva, o que influencia a velocidade e a exatidão da recuperação do vetor. A vista teórica da emoção executada neste modelo separa os sistemas de interação que medem o processamento cognitivo e afetivo. Cada subsistema processa aspectos diferentes do estímulo e suas características distintas; o sistema afetivo é mais rápido, processa características dos estímulos relevantes à sobrevivência do organismo, e gera uma saída simples. O sistema cognitivo é mais lento, processa mais características, é capaz de um elevado grau de diferenciação, e relaciona características dos estímulos a outras informações, não limitadas aos objetivos do organismo.

3.4 Conclusões

Como pode ser verificado pelos trabalhos acima citados, o estudo das emoções vêm sendo realizado há um longo tempo. Existem diversos modelos e aspectos a serem analisados para simulação de situações que envolvem emoções. O estudo de trabalhos existentes é importante, pois é a partir destes que se pode entender melhor o que cada pesquisa atingiu e o que já foi realizado nesta área de estudo tão complexa.

Para o presente trabalho, este estudo foi muito interessante, pois foi possível verificar que o modelo OCC é base de vários outros trabalhos, provando ser um modelo com embasamento teórico, e desta forma, altamente estudado.

4 Proposta de um Ambiente de Simulação Baseado em Agentes com Emoções

Como visto na seção 2.5, foram propostos alguns ambientes de simulação baseados em agentes. Porém um ambiente de simulação baseado em agentes genérico que faça uso de emoções ainda não foi desenvolvido. Até o momento, foram realizadas simulações com emoções em agentes (usando o modelo OCC) para problemas específicos, como o IPD "*Iterated Prisoner's Dilemma*" [BAZ 2001] ou "*Study of Social Norms*" [BAZ 2002].

A simulação de emoções como raiva, amor, alegria, etc., pode demonstrar comportamentos relativos a estas, como, por exemplo, hostilidade à cooperação com outros agentes, tornando o ambiente mais parecido com a realidade.

O ambiente proposto, chamado de AFRODITE, é baseado em agentes com emoções, que utiliza como base para estruturação das emoções o modelo OCC e procura, através da utilização de emoções nos agentes, obter uma forma de raciocínio menos trivial, tornando os agentes mais complexos.

Para que seja possível simular qualquer aplicação no ambiente é necessário que o usuário defina primeiramente um modelo para sua aplicação. A definição do modelo é importante, pois é uma estruturação lógica do problema a ser simulado. A modelagem consiste em representar um problema ou uma situação real através de um grupo de agentes, que interagem entre si e com o ambiente no qual estão inseridos, visando atingir uma solução.

Desta maneira, neste capítulo é apresentada a modelagem para o ambiente de simulação baseado em agentes proposto, bem como foram implementadas as emoções propostas pelo modelo OCC.

4.1 Aspectos considerados para o desenvolvimento do ambiente AFRODITE

Após a realização de um estudo comparativo entre alguns ambientes existentes (seção 2.6), foi possível definir quais aspectos são prioritários para o desenvolvimento de um novo ambiente. Assim, o AFRODITE foi projetado da seguinte maneira:

a) linguagens de programação:
   A linguagem de programação escolhida foi Java, por ser altamente portável e suportar o paralelismo. Para que o armazenamento das simulações, foi utilizado o banco de dados MySQL, é compatível com Java e é um software livre;

b) modelos de simulação:
A modelagem utilizada foi a partir de primitivas pré-definidas para definição das regras de simulação;

c) tipos de agentes:
Os agentes são reativos, porém, com a utilização de emoções nos mesmos, estes podem ter comportamentos diferentes, mesmo tendo a mesma definição em suas regras;

d) facilidades:
A simulação pode ser realizada passo-a-passo (um ciclo de simulação por vez) ou apenas apresentando o resultado final da simulação proposta. Quando utilizada a simulação passo-a-passo, o usuário pode ver os valores de cada agente no ciclo em questão. Também pode ser definido um arquivo para que os dados gerados na simulação possam ser armazenados para análise posterior.

e) interface:
Utilização de uma interface gráfica, baseada em janelas e menus, visando dar ao usuário uma maior facilidade na utilização do ambiente. Para que o ambiente possa ser utilizado por um maior número de pessoas, toda a interface foi desenvolvida em inglês.

4.2 Arquitetura do ambiente AFRODITE

Como em qualquer ambiente de simulação baseado em agentes, é necessário que o usuário tenha um conhecimento mínimo de como funciona a tecnologia de agentes e como estes podem atuar em sistemas em geral. Assim, a definição dos agentes, suas características e suas interações devem ser definidas de forma correta no momento da modelagem do sistema. Para o ambiente proposto, uma característica de um agente é vista como um atributo que o agente possui. Para exemplificar, em uma linguagem orientada a objetos, quando se define uma classe, é necessário também definir suas variáveis (atributos), pois é a partir da manipulação destas que são criadas todas as estruturas de dados e a implementação é possível. Com as características dos agentes ocorre o mesmo, ou seja, cada agente possui atributos, que quando manipulados e testados, executam ações.

O ambiente AFRODITE foi subdividido em seis módulos, conforme Figura 4.1.

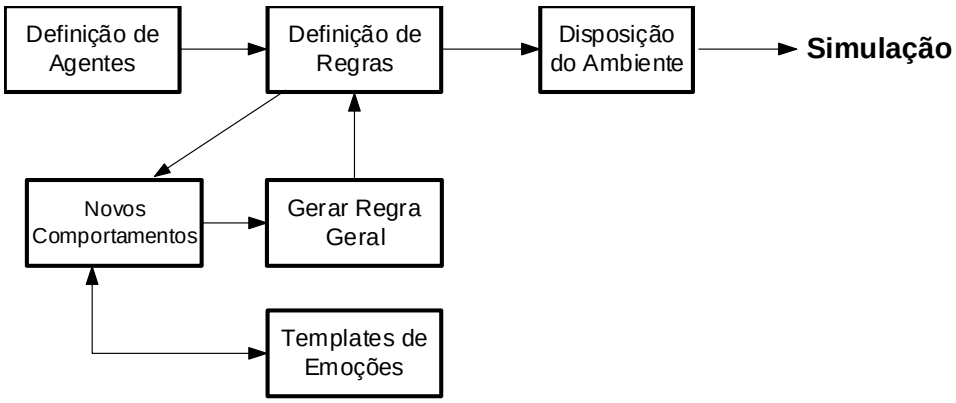


FIGURA 4.1 - Arquitetura do ambiente AFRODITE

- Definição de Agentes: módulo onde será criado o agente, definindo seu nome, quantidade deste tipo de agente na simulação e sua definição gráfica (figura .gif que irá identificar o agente na simulação gráfica). Os agentes podem ou não possuir características (se possuir características, será um agente do tipo *sujeito*, caso contrário, será um agente do tipo *objeto*). As características podem ser do tipo *integer, float, string, char* ou *date*, e podem receber um valor inicial quando criadas. A estrutura de como o agente pode ser definido é apresentada na Figura 4.2.

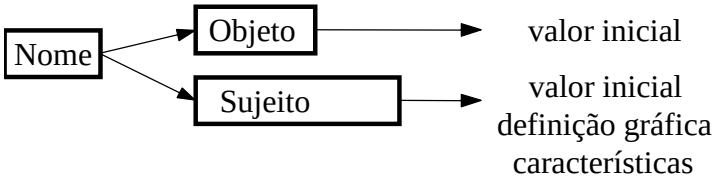


FIGURA 4.2 - Definição dos Agentes

- Definição de Regras: módulo onde serão criadas as regras de comportamento que os agentes possuem. Fazem parte deste módulo as primitivas pré-definidas do ambiente e as demais regras que serão desenvolvidas pelos usuários.
- *Templates* de Emoções: módulo onde cada subgrupo do modelo OCC terá uma classe específica, pois cada subgrupo possui testes de validação diferentes para suas emoções. A inserção das emoções as novas regras é realizada utilizando os agentes, as primitivas disponíveis no ambiente e valores reais. Não é necessário que todas as regras tenham emoções inseridas em seus comportamentos. A estrutura de como as regras de comportamento e emoções são definidas é apresentado na Figura 4.3.

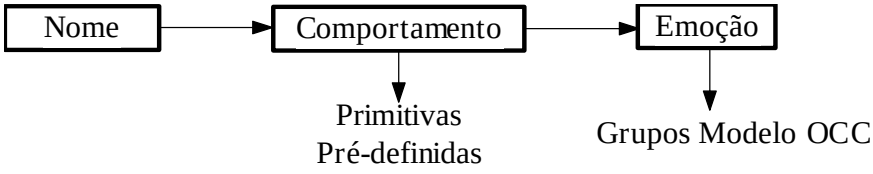


FIGURA 4.3 - Definição das Regras de Comportamento e das Emoções

- Disposição do Ambiente: módulo onde serão definidas as dimensões do ambiente (o espaço é sempre bidimensional, ou seja, trabalha com linhas (n) e colunas (m)), a quantidade de ciclos , a disposição dos agentes (linha e coluna) e a possibilidade de salvamento dos dados da simulação em arquivo. Quanto da disposição dos agentes, esta pode ser realizada de forma randômica, onde todos os agentes definidos no ambiente serão dispostos no *grid* bidimensional de forma aleatória, ou informando a posição desejada para cada agente no *grid* (esta forma de disposição dos agentes é menos utilizada, por ser menos realística, pois os agentes normalmente estão dispostos aleatoriamente no ambiente). A Figura 4.4 apresenta como o ambiente disponibiliza a definição das dimensões do ambiente e como os agentes são distribuídos. Na seção 5.1.2 será melhor explicado como os agentes são alocados no ambiente.

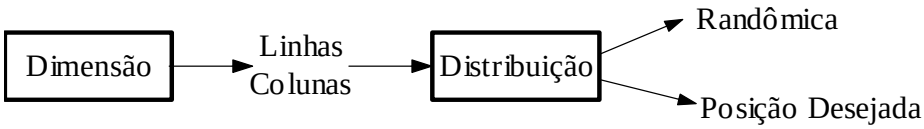


FIGURA 4.4 - Disposição do Ambiente

4.3 Primitivas pré-definidas do ambiente AFRODITE

As primitivas pré-definidas têm por objetivo facilitar a implementação de aplicações no ambiente. As primitivas abaixo apresentadas (em ordem alfabética) foram definidas com o intuito de serem altamente genéricas, ou seja, que aplicações de diversas áreas de conhecimento possam ser simuladas no ambiente.

- void addVal(*agent, characteristic, data type, value*)

Descrição: esta primitiva adiciona, a característica especificada, o valor passado como parâmetro ao valor existente no agente corrente na simulação. O valor da característica deve ser instanciado antes;
Parâmetro: nome do agente (*string*), nome da característica que receberá o valor (*string*), tipo do dado da característica (*string*) e valor a ser concatenado (*string*).

- void addVal(*agent, characteristic, row, col, data type, value*)

Descrição: esta primitiva adiciona, a característica especificada pela linha e coluna, o valor passado como parâmetro ao valor existente no agente corrente na simulação. O valor da característica deve ser instanciado antes;
Parâmetro: nome do agente (*string*), nome da característica que receberá o valor (*string*), linha e coluna do agente a receber o valor (*integer*), tipo do dado da característica (*string*) e valor a ser concatenado (*string*).

- String content(*row, col*)

Descrição: esta primitiva retorna o nome do agente na posição determinada pelo parâmetro. Caso a posição esteja vazia, retorna uma string vazia;
Parâmetro: linha e coluna da posição desejada (*integer*).

- void copyAgent(*agent, row, col*)

Descrição: esta primitiva copia um agente para a posição informada (linha, coluna). Caso a posição desejada já esteja preenchida por outro agente, este será sobreposto, ou seja, será apagado da simulação;
Parâmetro: nome do agente (*string*), linha e coluna para movimentação (*integer*).

- void create(*agent, row, col*)

Descrição: esta primitiva realiza a instanciação de um agente em determinada posição;
Parâmetro: nome do agente a ser instanciado (*string*), linha e coluna para instanciação (*integer*).

- int getCol()

Descrição: esta primitiva retorna a coluna atual do ciclo de simulação;
Parâmetro: não possui.

- int getCycle()

Descrição: esta primitiva retorna o ciclo de simulação atual;
Parâmetro: não possui.

- int getNeighborRow()

Descrição: esta primitiva só pode ser utilizada após a utilização da primitiva dos agentes *neighbor*, pois retorna a linha do último vizinho encontrado no atual ciclo de simulação. Caso não haja vizinhos com as características solicitadas na regra, retorna o valor -1;
Parâmetro: não possui.

- int getNeighborCol()

Descrição: esta primitiva só pode ser utilizada após a utilização da primitiva dos agentes *neighbor*, pois retorna a coluna do último vizinho encontrado no atual ciclo de simulação. Caso não haja vizinhos com as características solicitadas na regra, retorna o valor -1;
Parâmetro: não possui.

- int getRow()

Descrição: esta primitiva retorna a linha atual do ciclo de simulação;
Parâmetro: não possui.

- void incNeighbor(*characteristic, data type*)

Descrição: esta primitiva incrementa o valor de uma característica do vizinho encontrado pela primitiva neighbor(*agent*). Apenas para tipos de dado *integer* ou *float*;
Parâmetro: nome da característica que receberá o valor (*string*), tipo do dado da característica (*string*).

- void incVal(*agent, characteristic, data type*)

Descrição: esta primitiva incrementa o valor de uma característica do agente, de 1 em 1. Apenas para tipos de dado *integer* ou *float;*
Parâmetro: nome do agente (*string*), nome da característica que receberá o valor (*string*), tipo do dado da característica (*string*).

- void iniVal(*agent, characteristic, data type, value*)

Descrição: esta primitiva inicializa com um novo valor uma característica do agente.
Parâmetro: nome do agente (*string*), nome da característica que receberá o valor (*string*), tipo do dado da característica (*string*) e valor a ser inicializado (*string*).

- void kill(*agent*)

Descrição: esta primitiva retira um determinado agente da simulação, ou seja, este agente perde a sua instância no *grid*;
Parâmetro: nome do agente a ser retirado (*string*).

- void movAgent(*agent, row, col*)

Descrição: esta primitiva move um agente para a posição informada (linha, coluna). Caso a posição desejada já esteja preenchida por outro agente, o agente não se movimentará;
Parâmetro: nome do agente (*string*), linha e coluna para movimentação (*integer*).

- int neighbor(*agent*)

Descrição: esta primitiva retorna um valor *integer* para o primeiro agente informado por parâmetro encontrado em uma das oito posições vizinhas do agente corrente, conforme Figura 4.5 ("A" representa o agente corrente na simulação, que procura pelo agente especificado no parâmetro). Se não for localizado nenhum vizinho, o valor de retorno será 0;
Parâmetro: nome do agente a ser procurado nas posições vizinhas (*string*).

1	2	3
4	**A**	5
6	7	8

FIGURA 4.5 - Possíveis valores de retorno para a primitiva *neighbor*

- float perc(*agent, characteristic, data type*)

Descrição: esta primitiva retorna o valor percentual que uma característica tem sobre o número total de agentes.
Parâmetro: nome do agente (*string*), nome da variável a ser comparada (*string*) e tipo do dado da característica (*string*).

- void pos(*agent, row, col*)

Descrição: esta primitiva retorna nas variáveis do parâmetro (linha e coluna), a posição do agente especificado;
Parâmetro: nome do agente (*string*) e linha e coluna para retorno (*integer*).

- float random(*data type, inicial value, final value*)

Descrição: esta primitiva retorna um valor *float* randômico do tipo de dado determinado, dependendo dos valores determinados pelo intervalo de valores. Apenas para valores do tipo *integer* ou *float*;
Parâmetro: tipo do dado (*string*), início do intervalo e final do intervalo (*integer* ou *float*).

- String valContent(*agent, characteristic, row,col*)

Descrição: esta primitiva retorna o valor da característica de determinado agente, na linha e coluna específicas. Caso o agente da posição indicada seja diferente, retorna "erro".
Parâmetro: nome do agente (*string*), nome da característica (*string*), linha e coluna da posição desejada (*integer*).

- String valNeighbor(*characteristic, data type*)

Descrição: esta primitiva retorna o valor de uma característica do vizinho encontrado pela primitiva neighbor(*agent*);
Parâmetro: nome da característica que receberá o valor (*string*), tipo do dado da característica (*string*).

4.4 Definição das regras de comportamento e emoções

Tanto as regras de comportamento dos agentes quanto as regras que definem as emoções são do tipo IF-THEN, ou seja, "condição – ação". A ativação de um determinado agente, a partir de uma determinada situação, deve ser modelada nas regras de comportamento.

A criação das regras pode utilizar: os valores das características dos agentes, as primitivas pré-defindas e valores reais, fazendo relação a conectivos lógicos AND(&&), OR(||) ou NOT(!), e as comparações de igual (==), diferente (!=), maior (>) e menor (<).

Deve-se definir também a prioridade de cada regra de comportamento, porque em um ciclo de simulação mais do que uma regra de comportamento pode ser "verdadeira", e desta forma a simulação deve saber qual executar primeiro. A prioridade de cada regra é determinada por um valor numérico, definido pelo usuário. Quanto menor o valor, menor a prioridade. Por exemplo, uma regra com prioridade igual a 100 será executada depois que uma regra com prioridade igual a 200.

As regras de comportamento também seguem a estrutura IF-THEN. A diferença existente está em determinar a "intensidade" que uma emoção possui em determinada situação (seção 3.2.1). A fórmula definida no modelo OCC é a seguinte:

- Calcular o potencial de geração de uma emoção:
  IF variável local > 0
  THEN potencial de geração recebe valor da função definida pelo usuário[1].

- Calcular a intensidade de uma emoção:
  IF potencial de geração > limiar mínimo
  THEN intensidade recebe a diferença entre o potencial de geração e o limiar mínimo
  ELSE intensidade recebe o valor 0.

Para que o cálculo de intensidade de cada regra de emoção possa ser realizado, é necessário que o usuário especifique o valor de limiar mínimo para que a respectiva emoção se manifeste. Este limiar é expresso em um valor entre 0 e 1, equivalendo a um "percentual de aceitação".

Para cada condição da regra de emoção definida pelo usuário, também deve ser definido um "peso" para esta, que varia de 1 a 5 (o menor peso é 1 e o maior é 5). A partir destes pesos, será realizado o cálculo do "potencial de geração" da emoção, utilizando a Lógica Difusa [TAN97]. Foi escolhida a Lógica Difusa para realização

[1] As variáveis globais estão inseridas na função definida pelo usuário.

destes cálculos, pois é a definição matemática que mais se aproxima da realidade humana, visto que os valores expressos ficam entre 0 e 1. O mesmo não ocorre com a Lógica Booleana, pois os valores são 0 ou 1.

Com o intervalo [0;1], a Lógica Difusa pode definir um grau de "certeza" mais específico para uma determinada atividade. Os valores mais próximos de 1 indicam que a certeza é maior [TAN97]. Sua escolha para este ambiente ocorre pelo fato do ser humano nunca sentir uma emoção isoladamente, e assim pode-se calcular a intensidade de todas as emoções que cada agente tem definido, e após determinar qual será expressa. Segundo o modelo OCC, a emoção que se manifestará é a que possuir intensidade com maior valor.

A regra para determinar a intensidade de cada emoção (apresentada anteriormente), necessita que, primeiramente, seja calculado o potencial de geração. Os passos para realização deste cálculo são os seguintes:

1) Para cada condição da regra de emoção verdadeira (definida pelo usuário com valores entre 1 e 5), é realizada uma tradução destes valores para armazenamento em um conjunto de valores difusos, entre 0 e 1. Os valores definidos pelo usuário, entre 1 e 5, seguem a seguinte correspondência:

TABELA 4.1 – Correspondência entre valores Definidos pelo Usuário e Discretização Armazenada pelo ambiente

Valor Definido pelo Usuário	**Valor Discretizado Armazenado**
1	0.2
2	0.4
3	0.6
4	0.8
5	1
Se condição falsa	0

Esta tradução de valores, ou seja, sua discretização, foi definida para que, além de facilitar ao usuário a definição dos pesos de suas condições, fosse possível a implementação destes valores na interface do protótipo proposto (o qual será apresentado no capítulo 5).

2) Depois de testadas todas as condições definidas para a emoção, independente dos conectivos lógicos entre estas condições serem AND ou OR, é realizada uma soma de todos os valores armazenados neste conjunto. Não há diferenciação entre os conectores lógicos AND e OR para as condições pois ficaria difícil definir o quanto o peso de um conector AND é superior ao de um conector OR.
A soma de valores de um conjunto, na Lógica Difusa é a **operação de AND**. A soma dos valores calculados pela operação AND pode ser superior a 1, pois o usuário pode definir *n* condições, como diversos pesos, e todas podem ser verdadeiras.

Por exemplo, o usuário define um limiar de aceitação de *0.8* e três condições para uma regra de emoção, sendo que a primeira condição recebe como peso o valor *0.6*, a segunda condição o peso *0.8* e a terceira condição o peso *0.8.* Após realizados os testes, verificando quais condições são verdadeiras, os valores armazenados no conjunto foram: *[0.6, 0, 0.8]*, pois apenas a primeira e a terceira condições são verdadeiras. Assim, a partir da operação de ADD da Lógica Difusa, o valor do potencial de geração para esta emoção é *1.4.* Pela fórmula definida pelo modelo OCC, esta emoção teria uma intensidade diferente de 0, pois a intensidade recebe a diferença entre o potencial de geração (*1.4*) e o limiar de aceitação (*0.8*), ou seja, a intensidade da emoção para o exemplo seria *0.6 (1.4 – 0.8)*. Caso o valor do potencial de geração calculado tivesse sido menor do que o limiar mínimo de aceitação definido (*0.8*), o valor da intensidade para esta emoção seria *0*.

O cálculo da intensidade das emoções, apresentado acima, é realizado para todas as emoções definidas para cada agente e suas condições. A emoção que possuir maior intensidade no ciclo de simulação em questão, será expressa. Isto significa dizer que as ações definidas pelo usuário se determinada emoção fosse expressa serão executadas.

4.5 Exemplo de funcionamento do cálculo da intensidade das emoções

Para melhor entendimento do funcionamento do cálculo da emoção, é apresentado um exemplo simples (“ratos / queijos”), explicando apenas uma regra que gera uma emoção.

Tem-se a seguinte situação: em um *grid* 5x5, tem-se dois tipos de agente: rato e queijo. O número de ratos e queijos é 5 e 15, respectivamente. O objetivo do rato é procurar comida (no caso, queijo) e os queijos são simples agentes do tipo *objeto*, ou seja, não possuem características e, neste exemplo, nenhum objetivo definido. Os ratos e os queijos foram alocados no *grid* randomicamente.

Uma possível regra de emoção que pode ser definida para os ratos é que estes podem expressar alegria (*joy*) quando não há nenhum vizinho que seja um rato (este ‘outro’ rato está nas oito possíveis posições ao seu redor) e quando estes ratos vêem uma comida (também nas oito possíveis posições). Esta ação significa que primeiro o rato observa se há algum outro rato ao seu redor (“concorrência”) e depois se realmente há uma comida. A ação tomada pelo rato será de se mover em direção ao queijo para comê-lo. A regra a ser definida, no ambiente, para a emoção *joy* é a seguinte:

Se: **neighbor(“rato”) == 0 &&** *(condição 1)*
neighbor(“queijo”) != 0 *(condição 2)*

Então: **movAgent(“rato”,getNeighborRow(),getNeighborCol())** [2]

Limiar mínimo definido: *0.6*

[2] As primitivas *getNeighborRow()* e *getNeighborCol()* sempre se referem a última chamada da função *neighbor*, ou seja, o rato está se movendo em direção ao queijo.

Pesos para cada condição definida:

- condição 1: 2 (*0.4* pela tradução realizada internamente, conforme Tabela 4.1)
- condição 2: 4 (*0.8* pela tradução realizada internamente, conforme Tabela 4.1)

Os valores diferenciados para cada uma das condições da regra de emoção estão baseados no fato de que é mais importante encontrar um queijo do que encontrar um outro rato. Assim, o limiar mínimo de aceitação para esta emoção é *0.6*, pois se apenas a regra de que nenhum outro rato foi encontrado (que tem peso *0.4*) for verdadeira, a intensidade para esta emoção não será calculada. Porém, se for encontrado um queijo próximo do rato (que tem peso *0.8*), a intensidade será calculada.

Para demonstrar os cálculos necessários para definir a intensidade da emoção desta regra, suponha que as duas condições sejam verdadeiras e que o valor do potencial de geração é de *1.2* (*0.4 + 0.8*). A intensidade calculada para esta emoção será de *0.6*, ou seja, diferença entre potencial de geração (*1.2*) e o limiar mínimo (*0.6*).

Porém, não se pode afirmar que a emoção *joy* será expressa no ciclo de simulação testado, pois primeiro são testadas todas as emoções definidas para o cenário, e calculadas suas intensidades. A emoção com maior intensidade, que pode ser *joy*, será expressa.

5 Protótipo do Ambiente AFRODITE

Foi desenvolvido um protótipo do ambiente definido no capítulo anterior, com o objetivo de verificar se os aspectos contemplados (seção 4.1) são suficientes para o seu desenvolvimento. Outro objetivo deste protótipo é auxiliar a implementação de aplicações com agentes que façam uso de emoções, de forma transparente e concisa. Como dito anteriormente, o protótipo foi desenvolvido utilizando a linguagem Java e o banco de dados MySQL.

Este capítulo apresenta a interface completa do AFRODITE (menus e telas) e suas funcionalidades, na seção 5.1. Detalhes de implementação (estruturas de dados e tabelas do banco de dados) são apresentados na seção 5.2.

5.1 Interface do Protótipo

A interface é constituída por um menu principal com quatro opções, através das quais o usuário define os agentes, as regras de comportamento e emoções e as disposições do ambiente (cada uma destas opções e seus componentes serão explicados detalhadamente nas seções a seguir). No lado esquerdo da tela são apresentados os dados gerais da simulação, como o nome desta, os agentes que a compõem, as características dos agentes e as regras de comportamento dos agentes e suas emoções (em formato de árvore). Também é na tela inicial que se visualiza os ciclos da simulação, bem como os valores de cada agente. A Figura 5.1 apresenta uma visão geral da interface do protótipo.

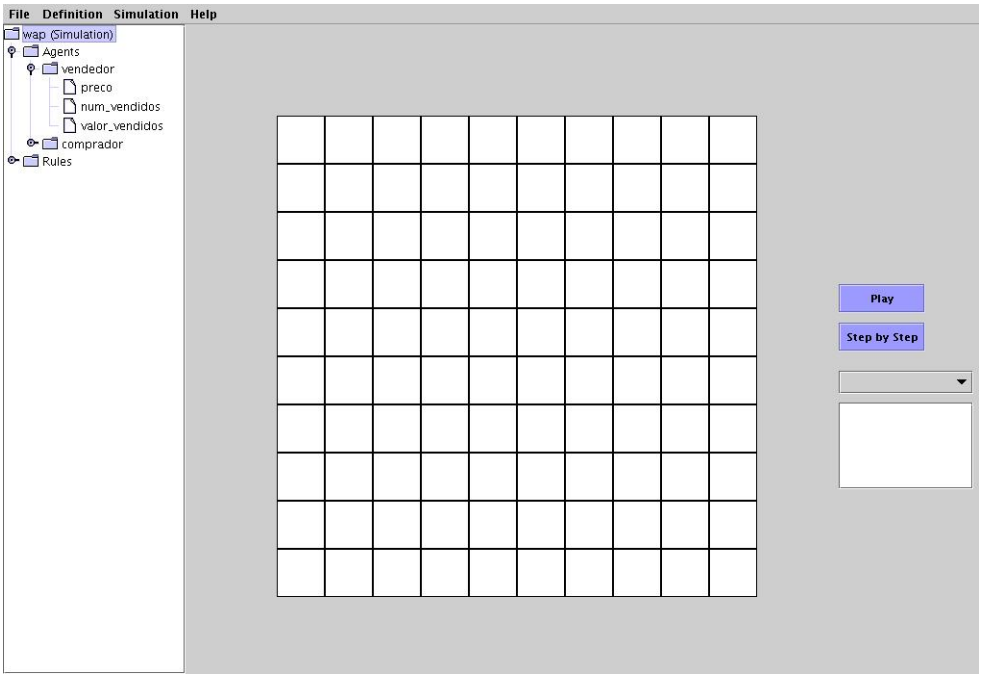


FIGURA 5.1 - Interface Geral do Protótipo

5.1.1 Menu *File*

Através das opções deste menu, o usuário executa as funções padrão de manipulação da simulação. Ele está dividido em quatro possíveis ações a serem tomadas: *New Simulation*, *Open Simulation*, *Analysis* e *Exit*. A Figura 5.2 apresenta o menu *File*.

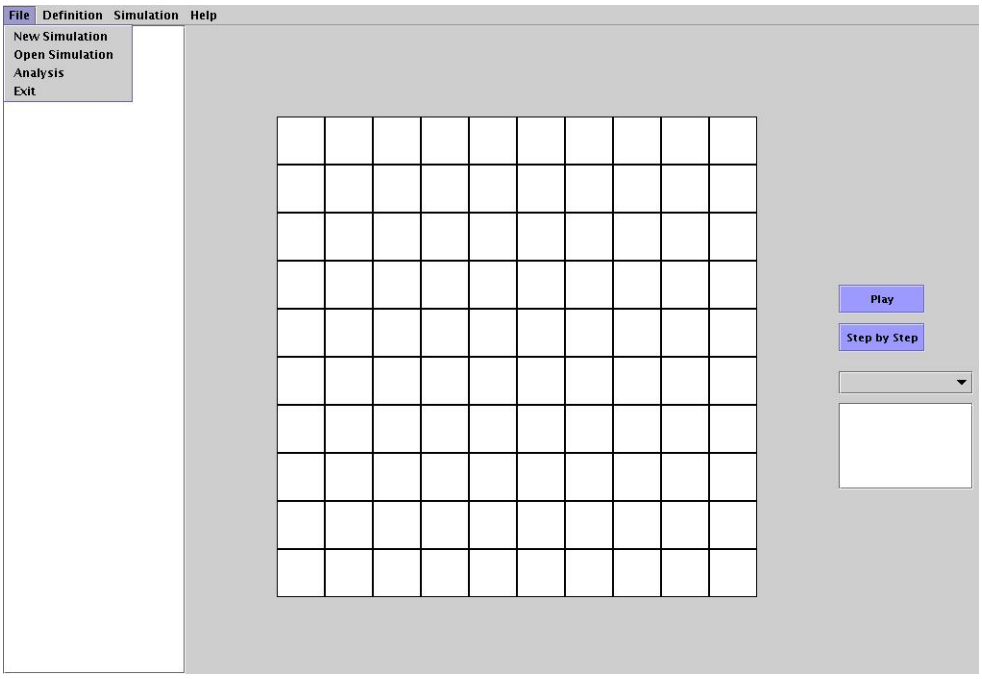


FIGURA 5.2 - Menu *File*

Na opção *New Simulation* (Figura 5.3) o usuário cria uma nova simulação, definindo um nome e o tamanho do *grid* (linhas e colunas).

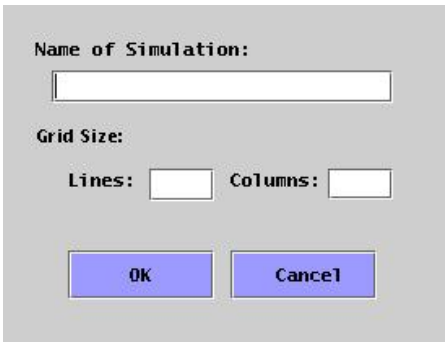


FIGURA 5.3 - Tela *New Simulation*

Na opção *Open Simulation* (Figura 5.4) são apresentadas todas as simulações existentes no sistema (criadas até o momento). O usuário pode apenas escolher qual das simulações existentes será novamente utilizada. Se deseja criar uma nova simulação, deve cancelar esta tela (botão *Cancel*) e escolher a opção *New Simulation.*

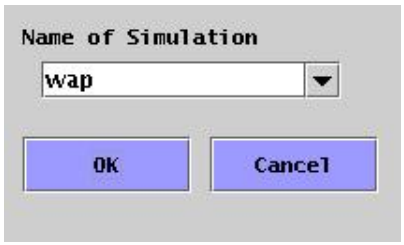


FIGURA 5.4 - Tela *Open Simulation*

Na opção *Analisys* (Figura 5.5), o usuário define um arquivo para saída dos dados, que conterá os valores das características definidas para cada um dos agentes alocados no *grid*. A Tabela 5.1 apresenta uma parte dos dados de um arquivo de saída (no Anexo III é apresentada parte de um arquivo de saída de dados gerados pelo AFRODITE). Este arquivo de dados gerado é utilizado para realização da análise de resultados da simulação. Caso o usuário não defina um nome de arquivo para saída dos dados, apenas serão mostrados os valores na tela.

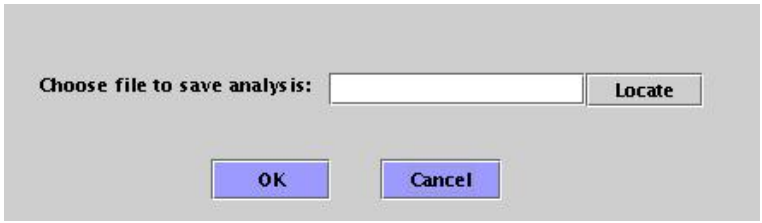


FIGURA 5.5 - Tela *Analisys*

TABELA 5.1 - Exemplo de Dados de Saída da Simulação

Cycle	vendedor@1.preco	vendedor@1.num_vendidos	vendedor@1.valor_vendidos
1	18	0	0
2	24	0	0
3	10	1	10
4	24	1	10
5	13	2	23
6	20	2	23
98	13	45	564
99	24	45	564

5.1.2 Menu *Definition*

Este menu está dividido em três módulos, um para a definição dos agentes e suas características (*Agents Definition*), um para definição das regras de comportamento e as emoções associadas (*Rules Definition*), e um terceiro para a definição de dimensões do *grid* e alocação dos agentes (*Environment Dimension*), conforme é apresentado na Figura 5.6.

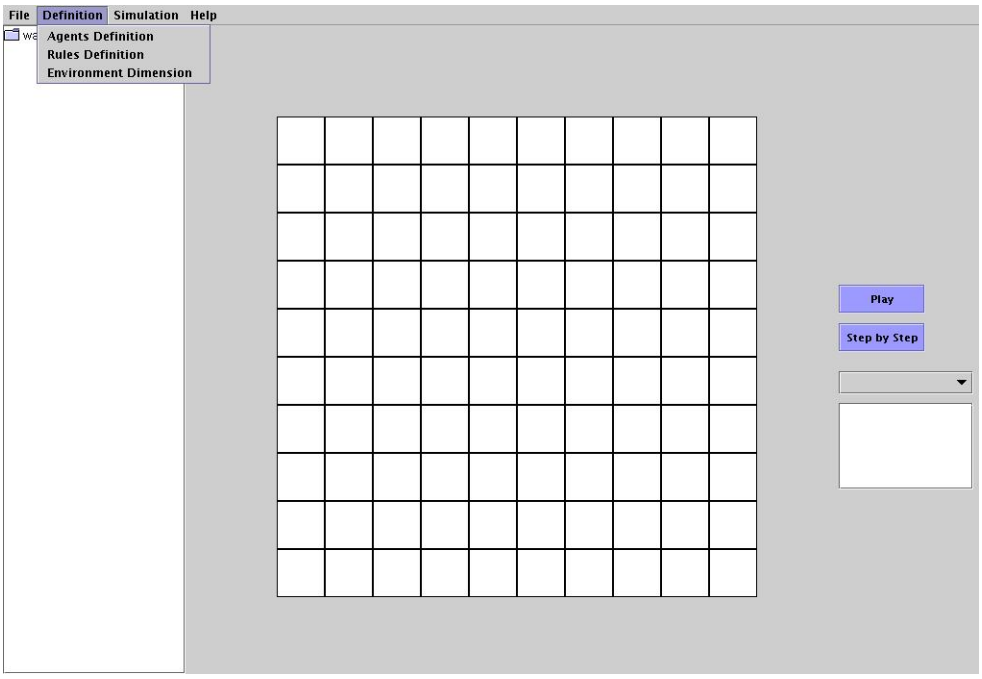


FIGURA 5.6 - Menu *Definition*

Na opção *Agents Definition*, o usuário pode criar, alterar ou excluir agentes. É necessário definir o nome do agente, a quantidade de agentes que a simulação terá e se é um agente do tipo objeto (que não possui características) ou do tipo sujeito (que possui características). Se o usuário escolher um agente do tipo objeto, ou seja, selecionar a opção *Type Object*, o botão *Characteristics* será desabilitado. A Figura 5.7 apresenta a tela para manipulação dos agentes, onde também podem ser vizualizados todos os

agentes existentes na simulação (opção *Existing Agents*), para que a manipulação destes seja facilitada.

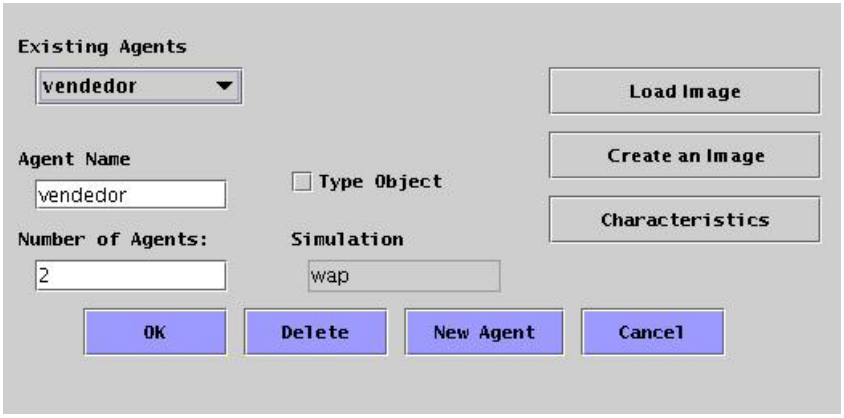


FIGURA 5.7 - Tela *Agents Definition*

Na opção *Agent Characteristics* o usuário pode criar, alterar ou excluir características (como definido na seção 4.2). É necessário definir o nome da característica, o tipo de dado desta (*integer*, *float*, *char*, *string* ou *date*), e um valor inicial. A Figura 5.8 apresenta a tela para manipulação das características, onde podem ser vizualizadas todas as características existentes na simulação (opção *Existing characteristics*). Nesta opção, para cada característica selecionada serão apresentados: tipo de dado da característica e valor inicial.

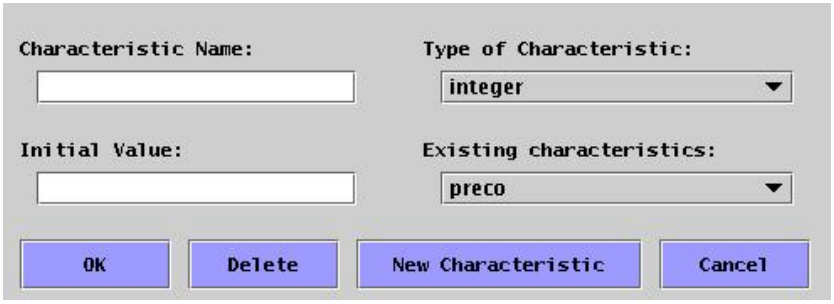


FIGURA 5.8 - Tela *Agent Characteristics*

Na opção *Rules Definition* (ver Figura 5.6) o usuário pode criar, alterar ou excluir regras. É necessário definir o nome da regra e a sua precedência (ordem que as regras serão executadas). Como o ambiente é baseado em um sistema de regras do tipo IF-THEN, ou "ação – condição", é necessário definir as condições para que determinada ação seja executada. As primitivas pré-definidas pelo ambiente, bem como todos os agentes e suas características são apresentados na interface (em cada um dos *ComboBox* disponibilizados, tanto para as condições quanto para as ações), a fim de facilitar a definição das regras. Para definição de uma emoção deve-se utilizar o botão *Emotions Templates* (ver Figura 5.10). A Figura 5.9 apresenta a tela para manipulação das regras, onde podem ser vizualizadas todas as regras já definidas na simulação (opção *Existing Rules*).

FIGURA 5.9 - Tela *Rules Definition*

Na opção *Emotions Templates* o usuário pode criar, alterar ou excluir as regras que disparam emoções. É necessário escolher a emoção que será definida (na Figura 5.10 a emoção escolhida foi "*fear*"), as condições e respectivos pesos, e o limiar mínimo. Ao definir o peso de cada condição, o usuário deve identificar qual a importância desta condição para a emoção definida. A forma que a intensidade da emoção é calculada foi explicada na seção 4.4.

Para cada uma das três divisões de emoções propostas pelo Modelo OCC (eventos, agentes e objetos), podem ser utilizados aspectos de acordo com as definições estabelecidas pelo próprio modelo, onde nos eventos são analisadas suas conseqüências, nos agentes suas ações e nos objetos suas propriedades. No ambiente de simulação baseado em agentes proposto, podem ser analisados três aspectos: o ambiente em si (suas dimensões espaciais/geográficas); os agentes do tipo sujeito, que possuem características; os agentes do tipo objeto, que não possuem características. Desta forma, pode-se dizer que o subgrupo eventos pode manipular aspectos tanto do ambiente, quanto de qualquer tipo de agente (tanto objeto, quanto sujeito); o subgrupo agentes pode manipular apenas agentes do tipo sujeito e aspectos do ambiente; e o subgrupo objeto pode manipular apenas agentes do tipo objeto e aspectos do ambiente.

Neste protótipo, a fim de facilitar a implementação das regras de emoção pelos usuários, após o usuário selecionar a emoção (*ComboBox "Emotion"*), é verificado a qual subgrupo esta pertence e ficam disponibilizados, em cada *ComboBox* para definição de condições e ações (Figura 5.10), os seguintes aspectos:

- para o subgrupo **Evento**: qualquer tipo de agente (sujeito ou objeto), primitivas pré-definidas e valores reais;
- para o subgrupo **Agente**: agentes do tipo sujeito, primitivas pré-definidas e valores reais;
- para o subgrupo **Objeto**: agentes do tipo objeto, primitivas pré-definidas e valores reais.

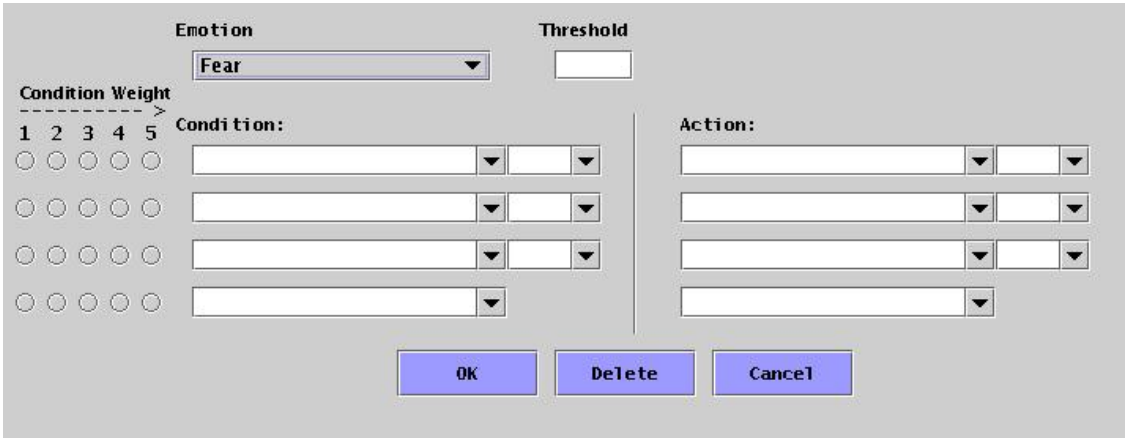


FIGURA 5.10 - Tela *Emotions Templates*

5.1.3 Exemplo de como o protótipo disponibiliza os aspectos dos subgrupos do modelo OCC

Na seção 4.4, foi apresentado um exemplo simples para exemplificar como funciona o cálculo da intensidade das emoções (ratos e queijos). Não é necessário novamente explicar o cenário, porém a regra definida para a emoção j*oy* (subgrupo - Eventos) é apresentada a seguir:

Se: **neighbor(“rato”) == 0 &&** *(condição 1)*
neighbor(“queijo”) != 0 *(condição 2)*

Então: **moveAgent(“rato”,getNeighborRow(),getNeighborCol())**

Como a emoção escolhida é *joy*, o usuário pode utilizar para a definição de suas regras qualquer tipo de agente (sujeito ou objeto), todas as primitivas disponíveis e valores reais. No exemplo acima foram apenas utilizadas primitivas pré-definidas. Porém, o subgrupo Eventos é o que disponibiliza mais aspectos para implementação de emoções no modelo OCC, pois analisa tanto os agentes, quanto o ambiente.

Na opção *Environment Definitions (*ver Figura 5.6), o usuário pode alterar a configuração inicial do tamanho do *grid*, definir a quantidade de ciclos que a simulação terá e alocar os agentes no *grid* (Figura 5.11). Por *default*, a quantidade de ciclos é 100. A alocação dos agentes pode ser randômica ou é possível alocar cada um dos agentes em uma determinada posição. Quando selecionado um tipo de alocação, randômica ou em posição determinada, esta será o tipo de alocação dos agentes, pois não é possível que uma parte dos agentes seja alocada randomicamente e outra em posição determinada.

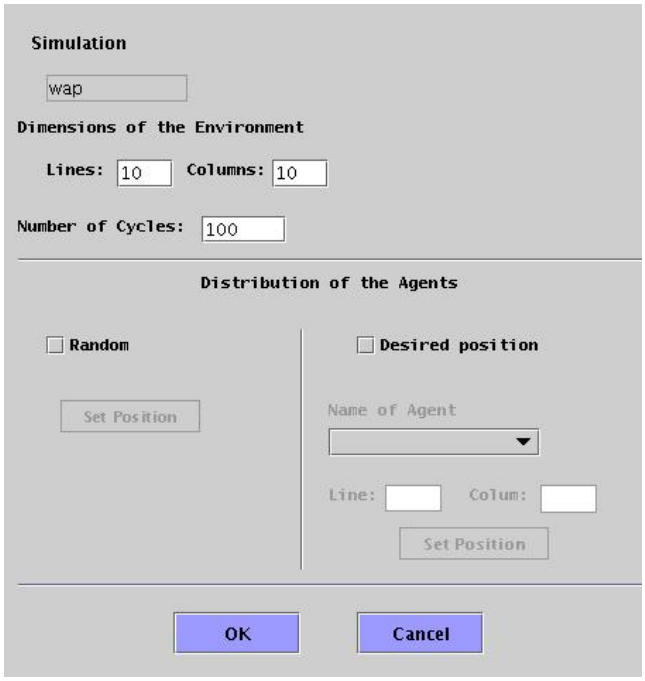


FIGURA 5.11 - Tela *Environment Definition*

Quando o usuário solicita que a alocação seja randômica, ele deve selecionar a opção *Random* e clicar no botão *Set Position* (Figura 5.12). Neste momento todos os agentes definidos pelo usuário serão alocados randomicamente no *grid*. Por exemplo, o usuário define o Agente1, que tem 10 instâncias e o Agente2, que tem 5 instâncias. A alocação randômica será realizada para os 10 agentes do tipo 1 e para os 5 agentes do tipo 2. Caso o tamanho definido para o *grid* seja menor que a quantidade de agentes definidos, o usuário receberá um aviso de erro na tela.

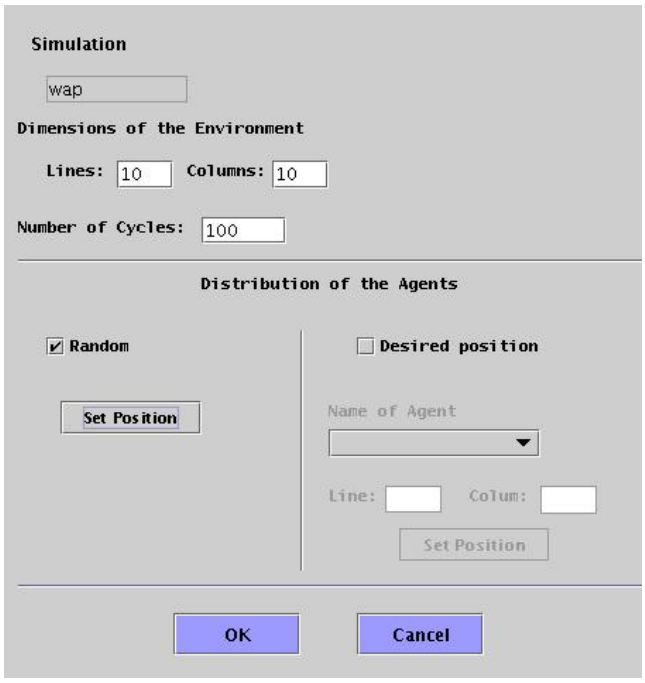


FIGURA 5.12 - Alocação Randômica dos Agentes

Quando o usuário solicita que a alocação seja em uma posição desejada, ele deve selecionar a opção *Desired position*, escolher o agente e indicar a linha e coluna para que este seja alocado. Após, deve-se clicar no botão *Set Position* (Figura 5.13). Este tipo de alocação de agentes é indicada para simulações com poucos tipos de agentes e em pequenas quantidades, pois é mais demorada para o usuário, e na maioria das simulações, a distribuição dos agentes é realizada de forma randômica. Também é testado aqui se o tamanho definido para o *grid* é menor que a quantidade de agentes definidos.

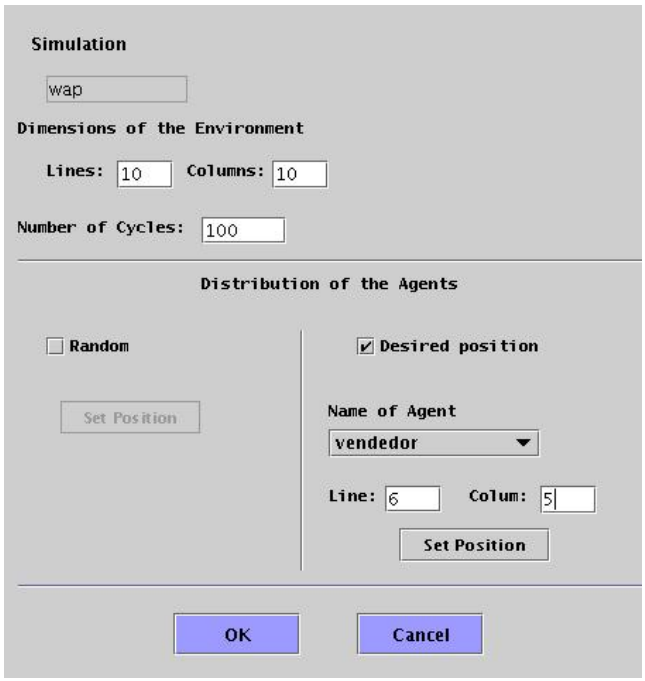


FIGURA 5.13 - Alocação em Posição Desejada dos Agentes

5.1.4 Menu *Simulation*

Esta opção dispara o processo de criação, em tempo de execução, de um arquivo .java (Figura 5.14). Este arquivo será compilado pelo compilador javac. A partir das definições realizadas pelo usuário, dos agentes e das regras de comportamento e emoções, o sistema monta um arquivo que será responsável pelo processo de simulação propriamente dito da aplicação, pois as abstrações realizadas pelo usuário para definição de agentes e de regras serão transformados em estruturas de dados para o sistema.

Após a criação deste arquivo .java, o sistema o compila (tornando-o em um arquivo .class) e o executa, realizando a simulação. Neste momento, a simulação será inicializada para a execução do primeiro passo, onde os agentes definidos já estarão instanciados para iniciar o processo de simulação. A partir deste momento o usuário escolhe na tela geral do protótipo se deseja executar todos os ciclos de simulação definidos de uma só vez (Botão *Play*) ou passo a passo (Botão *Step by Step*).

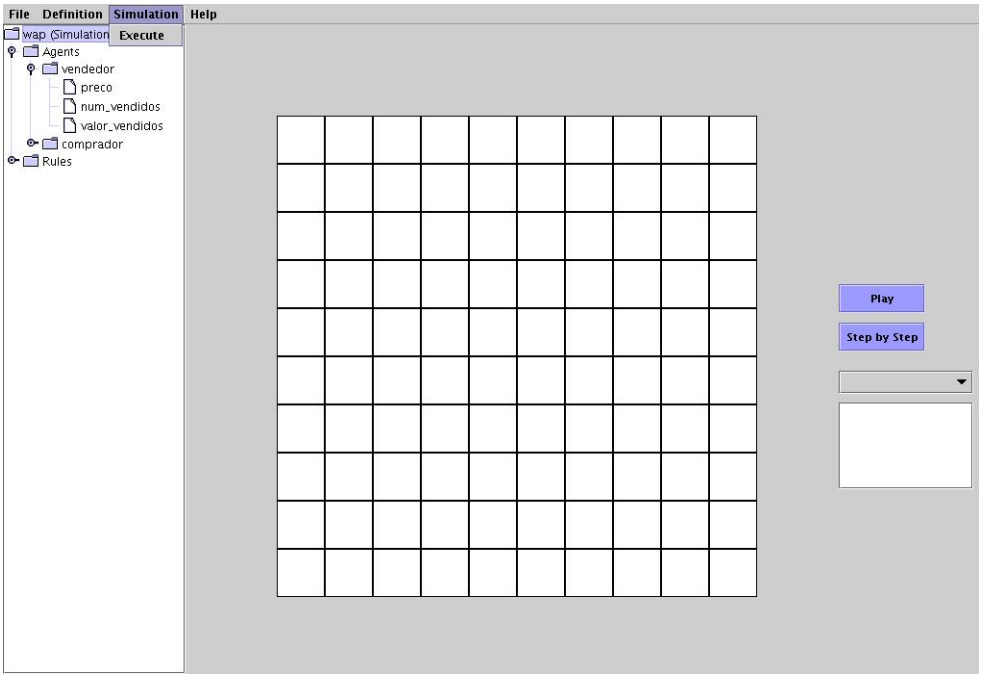


FIGURA 5.14 - Menu *Simulation*

5.1.5 Menu *Help*

Contém informações sobre o uso do ambiente, para o caso de posteriores dúvidas e informações sobre os desenvolvedores do protótipo. Este *Help* está em fase de construção.

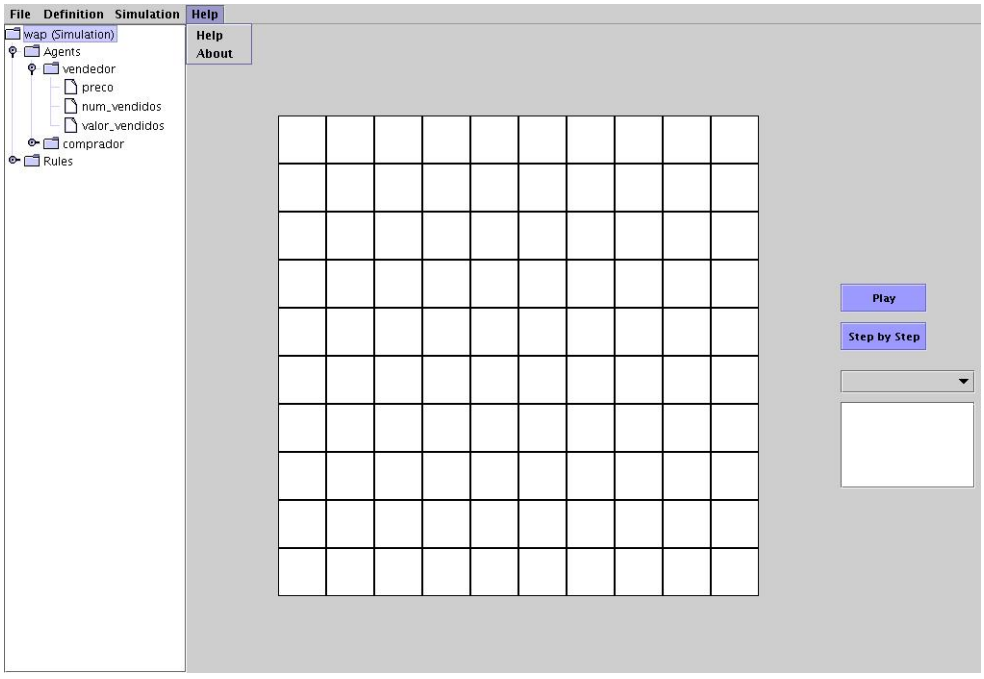


FIGURA 5.15 - Menu *Help*

Para uma melhor compreensão de como o usuário deve agir para definir sua aplicação no ambiente AFRODITE, o Anexo IV apresenta um Manual do Usuário, explicando passo-a-passo as informações necessárias para que a aplicação possa ser implementada.

5.2 Estruturas de Implementação

Esta seção apresenta como os conceitos de agentes e de regras de comportamento foram abordados na programação (principais estruturas de dados utilizadas no protótipo).

A Figura 5.16 apresenta o ER (Diagrama Entidade-Relacionamento) do banco de dados criado para armazenamento das simulações.

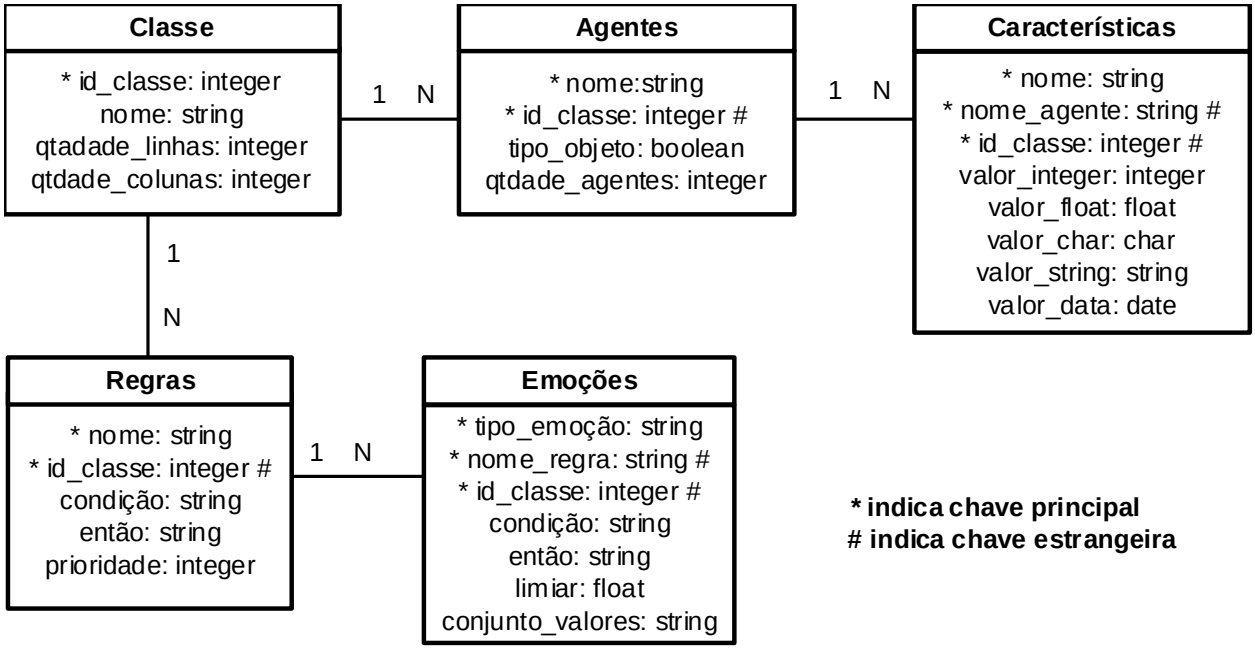


FIGURA 5.16 - ER das tabelas para armazenamento dos dados

Na tabela 5.2 é apresentada a relação entre os conceitos e as estruturas utilizadas.

TABELA 5.2 - Relação entre os conceitos e as estruturas utilizadas

Conceitos	Estruturas de Dados
Agentes	Estrutura em lista com todas as definições dos agentes. Esta lista tem como uma de suas variáveis a lista de características do agente.
Características	Estrutura em lista com todas as definições das características.
Regras de comportamento	Estrutura em lista com todas as definições das regras.
Emoções	Estrutura em lista com todas as definições das emoções.
Ambiente	Matriz bidimensional, onde cada posição é do tipo do agente.

Cada lista tem uma estrutura de dados com informações diferentes. As principais são as seguintes:

ESTRUTURA DE DADOS DO AGENTE
nome: *string*; // nome do agente
id_classe: *integer*; // simulação a qual pertence o agente
tipo_objeto: *boolean*; //se agente é do tipo objeto ou não
qtdade_agentes: *integer*; //quantidade de agentes para a simulação
lista_características: *características* [3] // lista das características definidas para o agente

ESTRUTURA DE DADOS DAS CARACTERÍSTICAS
nome: *string*;// nome da característica
nome_agente: *string* // nome do agente a qual característica pertence
id_classe: *integer*; // simulação a qual pertence a característica
valor_integer: *integer* // recebe valor se o tipo de dado for *integer*
valor_float: *float*; // recebe valor se o tipo de dado for *float*
valor_string: *string*; //recebe valor se o tipo de dado for *string*
valor_char: *char*; // recebe valor se o tipo de dado for *char*
valor_date: *string*; // recebe valor se o tipo de dado for *date*

Para uma melhor compreensão de como funciona a estrutura de dados do agente, a Figura 5.17, representa o tipo agente, em formato de lista, onde cada nodo é formado pelas estruturas de dados acima explicadas, incluindo a lista de características.

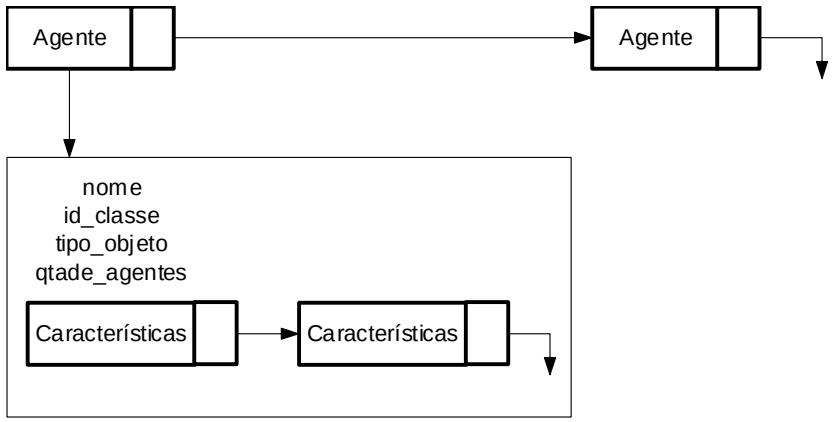


FIGURA 5.17 - Estrutura de dados do agente

ESTRUTURA DE DADOS DAS REGRAS
nome: *string*; //nome da regra
id_classe: *integer*;// simulação a qual pertence a regra
condição: *string*; // condição definida pelo usuário
então: *string*;// ação definida pelo usuário
precedencia: *integer*;// valor de prioridade que a regra terá para sua execução

ESTRUTURA DE DADOS DAS EMOÇÕES
tipo_emoção: *string*; // emoção escolhida

[3] Estrutura de dados apresentada a seguir.

nome_regra: *string*; // regra a qual pertence a emoção
id_classe: *integer*;// simulação a qual pertence a emoção
condição: *string*; // condição definida pelo usuário
então: *string*; // ação definida pelo usuário
limiar: *float*; // limiar mínimo de aceitação da emoção
conjunto_valores: *string*; // conjunto dos valores dos pesos de cada condição definida pelo usuário para as emoções

5.3 Estrutura Geral para Criação do Código para Execução da Simulação

Como explicado na seção 5.1.3, o código para execução da simulação é criado em tempo de execução. O usuário deve definir corretamente os agentes e as regras de comportamento, pois caso estas não estejam com a sintaxe correta, o arquivo gerado não será compilado.

Para um melhor entendimento, a seguir é apresentada uma visão geral da estruturação lógica para geração do código:

```
public void gerarArquivo(){
    <Path onde os outros arquivos .java  do ambiente estão instalados>/
Codigos.java");

// método para executar regras
    public void realizaTarefa(){

    //codigo das regras geradas
      for(m=0;m<prin.quantidadeRegras;m++){
          if (parserRegra(prin.lista_regras[m].condicao, 1)
              parserRegra(prin.lista_regras[m].entao, 2);

      ///codigo das emoções
       if(prin.quantidadeEmocoes > 0) {
              for(m=0;m<prin.quantidadeEmocoes;m++)
                    if(prin.lista_emocao[m].tipo_emocao.equals(prin.grid[i][j].tipo_e
              mocao)
                    if ("+parserRegra(prin.lista_emocao[k].condicao,1)
                          parserRegra(prin.lista_emocao[k].entao,2);

//método para calcular intensidade  das emoções definidas


    public void calculoEmocao(int i, int j)

       if(prin.quantidadeEmocoes > 0)
          parserRegra(prin.lista_condicoes[l].condicoes[k],1);

      // calcular potencial de geração
      potencial[l] = prin.lista_condicoes[l].pesos[k]+ potencial[l];
```

```
        nome_emocao[l] = prin.lista_condicoes[l].tipo_emocao;
        limiar[l] = prin.lista_condicoes[l].limiar;

       //calcular intensidade pelo potencial e limiar
        for(k=0;k<l;k++)\r\n")
                if(potencial[k]>limiar[k])
                   intensidade[k] = potencial[k]-limiar[k];
                else intensidade[k] = 0;

       //definir qual emocao possui maior intensidade e definir qual será expressa
        for(k=0;k<l;k++)
           for(int m=k;m<l;m++)
                if(intensidade[k]>=intensidade[m])
                    prin.grid[i][j].tipo_emocao = nome_emocao[k];

//executar arquivo .java  - geração do .class

Process compila=rt.exec("<Path onde os outros arquivos .java  do ambiente estão
instalados>Codigos.java");
compila.waitFor();
```

5.4 Estrutura Principal para Execução do Código

Depois da criação do código é necessário chamar uma nova classe (Codigos.class) para ser executada. A seguir, é apresentada como esta estrutura foi definida no ambiente:

```
gerarCodigo = new GerarCodigo(classe1, prin);
codigos = new Codigos(prin, primit);

//calcular emoção
    if(prin.quantidadeEmocoes > 0)
      for(i=0;i<prin.max_lin;i++)
         for(j=0;j<prin.max_col;j++)
            codigos.calculoEmocao(i,j);

//executar regras criadas
for(i=0;i<prin.max_lin;i++)
      for(j=0;j<prin.max_col;j++)
            codigos.realizaTarefa();
```

6 Modelagem de Cenários Selecionados no Ambiente AFRODITE

Para validar o funcionamento do ambiente foram modelados três exemplos de simulação em áreas de conhecimento diferentes. Os exemplos modelados nesta seção visam mostrar quão genérico é o ambiente que foi implementado. Os exemplos são os seguintes:

- IPD (*Iterated Prisoner's Dilemma*) – Teoria dos Jogos;
- Simulação de Multidões (cenário de evacuação em situações de risco, como incêndios) – Engenharia de Segurança;
- Venda de aparelhos celulares com o serviço WAP (*Wairless Aplication Protocol*) – Telecomunicações.

Primeiramente serão explicados os exemplos, ou seja, qual é o problema que se busca modelar. Em um segundo momento, será apresentado como estes exemplos foram modelados no AFRODITE. O primeiro exemplo (IPD) foi baseado em artigo anterior [BAZ2001], onde emoções foram inseridas. Esperava-se que a modelagem no ambiente AFRODITE fosse mais simples, pois não feita uma implementação *ad-hoc* para o novo problema que é gerado, a partir do emprego de emoções.

Os dois outros exemplos não possuem implementações com o uso de emoções. No caso da Simulação de Multidões foram propostas emoções, que se enquadrassem ao modelo OCC, a partir de idéias propostas no trabalho de Still [STI 2000]. Para a Venda de aparelhos celulares com o serviço WAP, foram consultados especialistas da área de telecomunicações, para entendimento da dinâmica de funcionamento da área, e de definição de possíveis reações que os agentes podem ter.

Como será apresentado em todos os exemplos, o valor de limiar mínimo para as regras de emoção, bem com o peso que cada uma das condições das emoções possuem o mesmo valor (*0.6*). Os pesos com valor *0.6* são indicados pelo usuário pelo valor *3*, como será mostrado nos exemplos a seguir. Foram utilizados valores iguais para limiar e pesos de condições, pois se pelo menos uma das condições das regras de emoção for verdadeira, o cálculo de intensidade seria realizado, sendo possível testar se este foi corretamente implementado.

6.1 IPD (*Iterated Prisoner's Dilemma*)

O formalismo conhecido como Dilema do Prisioneiro tem sido adotado como padrão para estudos da evolução da conduta de cooperação. É baseado em um jogo de dois participantes, onde cada um decide se irá ou não cooperar com o outro. Um exemplo típico de *payoff* pode ser apresentado na tabela 6.1, onde cada jogador recebe um valor, dependendo das escolhas feitas pelo jogador e por seu oponente.

TABELA 6.1 - Exemplo típico de *payoff* do Dilema do Prisioneiro

	Coopera	Não-coopera
Coopera	3/3	0/5
Não-coopera	5/0	1/1

Embora cooperação mútua seja a melhor solução em uma perspectiva conjunta dos jogadores, não-cooperar é a melhor escolha quando o interesse é individual. Assim, na maioria dos trabalhos envolvendo a cooperação no Dilema do Prisioneiro assume-se que as populações envolvidas para jogar o jogo não iterativo sempre convergem para não-cooperar. O Dilema do Prisioneiro Iterativo ocorre quando os jogadores jogam mais de uma vez entre si.

O trabalho de Nowak e May [NOW92] mostra uma estratégia para aumento do percentual de cooperadores, onde são considerados apenas dois tipos de jogadores: os que sempre cooperam (C) e os que nunca cooperam (D). Não é dada nenhuma atenção especial ao passado, ou a futuras jogadas, portanto nenhuma memória é utilizada. Em um espaço bidimensional (um *grid* n x n), onde cada espaço é ocupado por um cooperador ou não, estes jogam com seus vizinhos (oito posições possíveis do *grid*) e o resultado no *payoff* é a soma dos *payoffs* dos jogadores com os vizinhos. Na jogada seguinte, cada local da matriz n x n é ocupado pelo jogador com maior pontuação dentre os vizinhos que participaram da jogada. Com a utilização desta técnica, não há convergência para a não-cooperação, mas uma estabilidade no percentual de cooperação para 32%.

O trabalho de Bazzan e Bordini [BAZ2001] utiliza a estratégia proposta por Nowak e May [NOW92] e insere quatro emoções, baseadas no modelo OCC. As emoções escolhidas e suas regras são apresentadas abaixo:

- *joy*: ocorre quando o jogador tem um mínimo *y* de pontos ou um mínimo de vizinhos alegres;
- *distress*: onde o jogador tem um mínimo *y* de pontos ou um mínimo de vizinhos angustiados;
- *pity*: onde os vizinhos não têm um mínimo *y* de pontos;
- *anger*: onde o jogador não tem um mínimo *y* de pontos e seu oponente não coopera;

Com a inserção de emoções a taxa de cooperação dos jogadores ficou próxima de 50%, provando que as emoções alteram a tomada de decisão dos jogadores.

6.1.1 Modelagem do IPD no AFRODITE

Para este exemplo existe apenas um tipo de agente: jogador.

Os agentes *jogador* possuem cinco características:

- *coopera*: armazena se o agente está cooperando ou não - tipo de dado: *integer* (1= coopera; 2 = não coopera);
- *pontos*: armazena a quantidade de pontos de cada agente - tipo de dado: *integer*;
- *lin:* armazena a linha do vizinho com maior pontuação - tipo de dado: *integer*;
- *col:* armazena a coluna do vizinho com maior pontuação - tipo de dado: *integer*;
- *pontos_vizinho:* armazena a pontuação do maior vizinho - tipo de dado: *integer*;
- *media:* armazena a média dos vizinhos - tipo de dado: *float*;

O tamanho do grid é 10 x 10 e está totalmente ocupado, tendo-se 100 agentes do tipo jogador.

A modelagem dos agentes e das regras no AFRODITE é realizada como explicado na seção 5.2, utilizando as funcionalidades existentes.

Agente:

- nome: jogador
- tipo_objeto: false
- qtdade_agentes: 100
- lista_características:
    - Coopera valor_integer: -1
    - Pontos valor_integer: -1
    - Pontos_vizinho valor_integer: -1
    - Lin valor_integer: -1
    - Col valor_integer: -1
    - Media valor_float: -1

Regras:

- nome: *inicial*

  condição: getCycle == 1

  então: iniVal("jogador","pontos","integer",0);
  iniVal("jogador","coopera",random(“integer”,1,2));
  iniVal("jogador","lin","integer","0");
  iniVal("jogador","col","integer","0");
  iniVal("jogador","pontos_vizinho","integer","0");

  precedencia: 400

  Função: inicializar variáveis no primeiro ciclo de simulação;

- nome: *jogar_vizinho1C-C* [4]
  condição: valContent(“jogador”,”coopera”,getRow()-1, getCol()-1) == 1
  && jogador.coopera == 1

  então: addVal("jogador","pontos","integer", “3”);
  addVal(“jogador”,”pontos”,getRow()-1,getCol()-1,”integer”,”3”);

  precedencia: 300

  Função: verifica se o primeiro vizinho coopera e se o jogador também coopera. Adiciona o valor 3 para a jogada, tanto para o jogador quanto para o vizinho.

- nome: *jogar_vizinho1C-D*
  condição: valContent(“jogador”,”coopera”,getRow()-1, getCol()-1) == 2
  && jogador.coopera == 1

  então: addVal(“jogador”,”pontos”,getRow()-1,getCol()-1,”integer”,”5”);

  precedencia: 300

  Função: verifica se o primeiro vizinho não coopera e se o jogador coopera. Adiciona o valor 5 para a jogada ao vizinho.

- nome: *jogar_vizinho1D-C*
  condição: valContent(“jogador”,”coopera”,getRow()-1, getCol()-1) == 1
  && jogador.coopera == 2

  então: addVal("jogador","pontos","integer", “5”);

  precedencia: 300

  Função: verifica se o primeiro vizinho coopera e se o jogador não coopera. Adiciona o valor 5 para a jogada ao jogador.

- nome: *jogar_vizinho1D-D*
  condição: valContent(“jogador”,”coopera”,getRow()-1, getCol()-1) == 2
  && jogador.coopera == 2

  então: addVal("jogador","pontos","integer", “1”);
  addVal(“jogador”,”pontos”,getRow()-1,getCol()-1,”integer”,”1”);

  precedencia: 300

[4] As regras jogar_vizinho*N*C-C, jogar_vizinho*N*C-D, jogar_vizinho*N*D-C, jogar_vizinho*N*D-D são repetidas para as oito posições de vizinhança. Não há necessidade, para entendimento, que todas sejam expostas aqui.

Função: verifica se o primeiro vizinho não coopera e se o jogador também não coopera. Adiciona o valor 1 para a jogada, tanto para o jogador quanto para o vizinho.

- nome: *comparar_vizinho1*[5]

  condição: valContent("jogador","pontos",getRow()-1, getCol()-1)> jogador.pontos
  && valContent("jogador","pontos",getRow() -1, getCol() -1) > jogador.pontos_vizinho

  então: addVal("jogador","lin","integer", getRow() -1);
  addVal("jogador","col","integer",getCol() - 1);
  addVal("jogador","pontos_vizinho","integer",
  valContent("jogador","ponto",getRow()-1, getCol()-1));
  addVal("jogador","media","integer",
  valContent("jogador","ponto",getRow()-1, getCol()-1));

  precedencia: 200

  Função: se o primeiro vizinho tiver pontuação maior que a do jogador e maior que outro vizinho já comparado, armazena linha, coluna, pontuação e média do vizinho comparado (no exemplo, o vizinho comparado está na posição getRow() -1, getCol() -1, ou linha atual -1, coluna atual -1).

- nome: *copiar_melhor_vizinho*

  condição: valContent("jogador","pontos",lin, col) > jogador.pontos

  precedencia: 100

  Função: se o melhor vizinho tem uma pontuação maior que a do jogador, será verificada qual a emoção que este vizinho esta expressando, e a substituição será realizada dependendo da emoção. Desta maneira, esta regra não possui ações.

Regras de Emoções:

As emoções são inseridas na regra *copiar_melhor_vizinho*, porém não há como comparar qual emoção cada agente tem em determinado ciclo, pois primeiro é calculada a intensidade de cada emoção, a partir das condições da mesma, para depois saber qual será expressa. Assim, as regras que geram as emoções foram modificadas para avaliar apenas a pontuação dos vizinhos determinando um valor mínimo de pontos diferentes para cada uma das emoções, diferenciando-se do trabalho proposto em [BAZ2001] (ver seção 6.1). As regras propostas são apresentadas a seguir:

[5] Para as outras sete posições possíveis no grid (lin – 1, col; lin – 1, col +1; lin, col – 1; lin, col +1; lin +1, col –1; lin + 1, col; lin +1, col+ 1) são realizados os mesmos testes apresentados pela regra comparar_vizinho1. Não há necessidade que todas sejam repetidas, pois a mudança ocorre apenas na posição de linha e colunas testadas, e não nas ações que as regras irão executar.

- *joy*: se manifesta quando o jogador tem pelo menos um mínimo de pontos (90% da pontuação do seu maior vizinho);

- *distress*: se manifesta quando o jogador não tem pelo menos um mínimo de pontos (70% da pontuação do seu maior vizinho);

- *pity*: se manifesta quando o jogador não têm pelo menos um mínimo de pontos (50% da pontuação do seu maior vizinho);

- *anger*: se manifesta quando o jogador não tem pelo menos um mínimo de pontos (30% da pontuação do seu maior vizinho) e seu oponente não coopera;

- tipo_emoção: *joy*
  nome_regra: copiar_melhor_vizinho
  condição: jogador.pontos > = (jogador.media * 0.90)

  então: copyAgent(“jogador”,lin.col);
  iniVal("jogador","pontos_vizinho","integer","0");

  limiar: 0.6
  conjunto_valores: [3]

  Função: se a pontuação do jogador é igual ou superior a 90% a pontuação de seu maior vizinho, copia todos os dados do vizinho e inicializa a variável pontos_vizinho, para que no próximo ciclo possam ser realizadas novamente as comparações com os vizinhos.

- tipo_emoção: *distress*
  nome_regra: copiar_melhor_vizinho
  condição: jogador.pontos > = (jogador.media * 0.70)

  então: copyAgent(“jogador”,lin,col);
  iniVal("jogador","pontos_vizinho","integer","0");

  limiar: 0.6
  conjunto_valores: [3]

  Função: se a pontuação do jogador é igual ou superior a 70% a pontuação de seu maior vizinho, copia todos os dados do vizinho e inicializa a variável pontos_vizinho, para que no próximo ciclo possam ser realizadas novamente as comparações com os vizinhos.

- tipo_emoção: *pity*
  nome_regra: copiar_melhor_vizinho
  condição: jogador.pontos > = (jogador.media * 0.50)

  então: copyAgent(“jogador”,lin.col);
  iniVal("jogador","pontos_vizinho","integer","0");

limiar: 0.6
conjunto_valores: [3]

Função: se a pontuação do jogador é superior a 50% a pontuação de seu maior vizinho, copia todos os dados do vizinho e inicializa a variável pontos_vizinho, para que no próximo ciclo possam ser realizadas novamente as comparações com os vizinhos.

- tipo_emoção: *anger*
  nome_regra: copiar_melhor_vizinho
  condição: jogador.pontos > = (jogador.media * 0.30)
  && valContent(“jogador”,”coopera”,lin, col) == “2”

  então: copyAgent(“jogador”,lin.col);
  iniVal("jogador","pontos_vizinho","integer","0");

  limiar: 0.6
  conjunto_valores: [3, 3]

  Função: se a pontuação do jogador é igual ou superior a 30% da pontuação de seu maior vizinho e seu oponente não coopera, copia todos os dados do vizinho, e inicializa a variável pontos_vizinho, para que no próximo ciclo possam ser realizadas novamente as comparações com os vizinhos.

6.1.2 Considerações sobre a Modelagem

A modelagem realizada para o IPD no AFRODITE foi bastante extensa, devido a definição de várias regras, a fim de realizar a ”jogada” entre os agentes e verificar pontuação de cada vizinho. Isso prova que é necessário a definição de novas primitivas que realizem uma comparação/análise sistemática das características dos vizinhos que um agente possa ter. A primitiva *neigbhor* testa nas posições ao seu redor apenas por um tipo de agente. Porém, para o cenário apresentado, era necessário testar o conteúdos das características dos vizinhos, sendo necessário a definição de muitas regras.

Outro aspecto importante, que pode ser incorporado ao protótipo, é disponibilizar ao usuário primitivas de ambiente. Estas primitivas teriam o mesmo funcionamento de variáveis globais em um programa estruturado, ou seja, todos os agentes teriam acesso. Este aspecto foi analisado, pois se o usuário quiser modificar os valores da matriz de *payoff* terá que alterar todas as regras que especificam as jogadas com os vizinhos. A matriz apresentada na Tabela 6.1 é um exemplo típico e foi utilizada nesta modelagem, porém existem várias propostas com outros valores.

Apesar de ter sido necessário a definição de várias regras, o cenário com emoções pode ser modelado no AFRODITE e verificado o funcionamento dos agentes *jogador*, cooperando ou não, uns com os outros. Não foram realizadas análises comparativas, como é explicado na seção 6.4 .

6.2 Simulação de Multidões

"O estudo de como e onde multidões se formam e se movem a partir de uma densidade superior a uma pessoa por metro quadrado é chamada de 'Dinâmica de Multidões'." [STI2000]

O principal foco de estudo desta área é tentar antecipar problemas que possam ocorrer em uma situação de emergência, devido à grande quantidade de pessoas concentradas em um mesmo ambiente. Possíveis exemplos são problemas em estádios, estações de trem ou *shopping centers.*

Um dos cenários propostos por Still [STI2000] é entrada/saída do *Wembley Stadium*, visando manter a ordem e a segurança da multidão em situações de risco. O objetivo é verificar se os portões são suficientes para a quantidade de pessoas presentes. Para isso foram analisados vários aspectos:

a) o estádio tem capacidade total de 82.000 pessoas;
b) o tempo médio requerido para evacuação do estádio com capacidade total é relativamente curto – de 30 a 90 minutos, a fim de evitar problemas;
c) para medir o fluxo de entrada no estádio, tem-se como base o fluxo de pessoas por roleta. Os portões A, B, C, D, F, G, J, L e K tem 10 roletas cada e o portão M tem 8 roletas. Assim, o total de roletas do estádio é 98 (Figura 6.1);
d) o tempo de entrada também pode ser alterado porque as pessoas (antes de chegar ao portão escolhido) podem estar andando em direções diferentes, pois objetivam chegar a pontos diferentes. Um aspecto analisado foi a auto-organização das pessoas, formando "vias" de acesso aos pontos desejados. A multidão não possui uma inteligência coletiva, porém a auto-organização "emerge" naturalmente.

A idéia desta simulação é verificar como as pessoas se deslocam para sair do estádio de forma segura, analisando se a quantidade de roletas é suficiente a fim de não gerar tumulto. Na modelagem proposta não será analisada a auto-organização da multidão.

A velocidade de cada pessoa pode ser randômica (entre 1.0 e 4.0). A escolha do portão também é randômica, pois as pessoas têm acesso ao estádio por duas ruas, uma em frente e outra atrás do estádio.

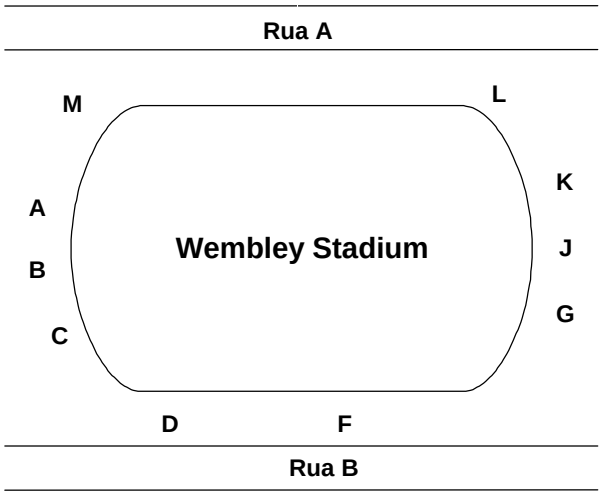


FIGURA 6.1 - Portões e acessos do *Winbley Stadium*

No trabalho realizado por Still não foram apresentados valores, porém foi concluído que se algum desastre ocorresse no estádio, as pessoas não conseguiriam sair em tempo hábil. Neste trabalho, Still afirma que aspectos emocionais são extremamente relevantes na tomada de decisão que cada pessoa manifesta em um desastre, porém não foram realizados experimentos. Desta maneira, a idéia é incluir algumas emoções nas pessoas que estão no estádio, simulando um desastre, a fim de mostrar quanto isso influencia a evacuação do mesmo.

As emoções propostas são:

- *fears-confirmed*: estar longe da saída e preso a um grupo sem poder sair;
- *distress*: estar preso a um grupo de pessoas sem poder sair;
- *hope*: estar próximo da saída;
- *fear*: estar longe da saída.

6.2.1 Modelagem de Simulação de Multidões no AFRODITE

Para este exemplo são propostos dois tipos de agentes: pessoa e portão de saída. Para facilitar a modelagem, os portões são o número de roletas do estádio, no caso 98.

Os agentes pessoa possuem duas características:

- *velocidade*: armazena a velocidade de cada pessoa, a fim de saber quando tempo esta demora para se deslocar na simulação - tipo de dado: *integer*;
- *portão:* armazena por qual portão a pessoa escolheu sair do estádio - tipo de dado: *integer*.

Os agentes portão possuem uma característica:

- *num_portão*: armazena o número do portão, a fim do agente pessoas saber por onde deve sair - tipo de dado: *integer*.

O tamanho do grid é de 300 x 300, ou seja, possui 90.000 “espaços” para ocupação e os agentes (pessoa + portão) ocupam 82.098 destes “espaços”, ficando a

movimentação difícil, devido ao grande número de agentes e a pequena quantidade de "espaços" livres.

Agentes:

- nome: *pessoa*
- tipo_objeto: false
- qtdade_agentes: 82.000
- lista_características:
    - velocidade valor_integer: -1
    - portão valor_integer: -1

- nome: *portão*
- tipo_objeto: false
- qtdade_agentes: 98
- lista_características:
    - num_portão valor_integer: -1

Regras:

- nome: *inicial*
  condição: getCycle == 1

  então: iniVal("pessoa","velocidade","integer",random("integer",1,4));
  iniVal("pessoa","portao","integer",random("integer",1,98));
  iniVal("portao","num_portao","integer","1");

  precedencia: 400

  Função: inicializar variáveis no primeiro ciclo de simulação;

- nome: *indicar_portao*
  condição: getCycle == 1
  && content(getRow(),getCol()) == "portão"

  então: incVal("portao","num_portao","integer");

  precedencia: 300

  Função: inicializar todos os portões, de 1 a 98;

- nome: *sair*
  condição: content(getRow(),getCol()) == "pessoa"
  && neighbor("portao") != 0
  && valNeighbor("portao","integer") == portão.num_portão

  então: kill("pessoa");

  precedencia: 200

Função: quando o agente for do tipo *pessoa*, e este está ao lado do seu portão escolhido para sair ele é retirado da simulação, ou seja, conseguiu sair do estádio.

- nome: *mover*
  condição: content(getRow(),getCol()) == "pessoa"

  precedencia: 100

  Função: testa se o agente é o tipo *pessoa*. Esta regra não possui então "ação" , pois a partir dos testes de emoções é que a movimentação do agente será definida. Como pode ser visto nas condições das emoções (mostradas a seguir), é realizado o mesmo teste para cada uma das emoções definidas, ou seja, '*content(getRow(),getCol()) == "pessoa"'* , porque apenas os agentes do tipo *pessoa* expressam emoções, e é necessário que uma regra seja definida e, a partir desta, as emoções possam ser definidas;

Regras de Emoções:

As emoções são inseridas na regra *mover*, pois o agente pessoa pode ter reações diferentes, dependendo da emoção expressada.

- tipo_emoção: *fears-confirmed*
  nome_regra: mover
  condição: content(getRow(),getCol()) == "pessoa"
  && neighbor("") == 0
  || neighbor("portão") == 0

  limiar: 0.6
  conjunto_valores: [3,3,3]

Função: o agente do tipo a *pessoa* não tem um espaço vazio para se movimentar ou o portão não está próximo. Não possui então "ação", porque o agente está desorientado.

- tipo_emoção: *distress*
  nome_regra: mover
  condição: content(getRow(),getCol()) == "pessoa"
  && neighbor("") == 0

  limiar: 0.6
  conjunto_valores: [3,3]

Função: o agente do tipo *pessoa* não tem um espaço vazio para se movimentar. Não possui então "ação", porque não tem como se movimentar, deve esperar para poder sair da posição.

- tipo_emoção: *hope*
  nome_regra: mover

condição: content(getRow(),getCol()) == “pessoa”
&& neighbor(“portão”) != 0

então: movAgent("pessoa", getNeighborRow(),getNeighborCol());

limiar: 0.6
conjunto_valores: [3,3]

Função: quando a *pessoa* está próxima do portão, se movimenta na direção do mesmo, a partir das posições de linha e coluna informadas pelas primitivas getNeighborRow() e getNeighborCol().

- tipo_emoção: *fear*
  nome_regra: mover
  condição: content(getRow(),getCol()) == “pessoa”
  && neighbor(“portão”) == 0
  && neighbor(“”) != 0

  então: movAgent("pessoa",getRow()+pessoa.velocidade,getCol()+
  pessoa.velocidade);

  limiar: 0.6
  conjunto_valores: [3, 3, 3]

  Função: quando a *pessoa* não consegue ver o portão de saída, mas há espaço vazio, o agente irá se movimentar. Esta movimentação depende da velocidade que cada agente *pessoa* possui.

6.2.2 Considerações sobre a Modelagem

O cenário proposto é simples de modelar, visto que são poucos agentes e características a definir. Porém, o tamanho do *grid* e a quantidade de instâncias de cada um dos agentes é enorme. Não é possível apresentar na tela do protótipo o *grid* de forma gráfica, devido ao tamanho deste (300x300). O arquivo de saída de dados possui 164.098 colunas (164.000 do agente *pessoa* devido a serem 82.000 instâncias e cada instância com duas características mais as 98 instâncias do agente *portão*, que possui apenas uma característica). A análise dos dados ficaria bastante complexa, devido ao tamanho do arquivo de saída.

A partir do problema encontrado para verificação dos dados gerados no arquivo de saída, concluiu-se que seria interessante que o usuário pudesse escolher apenas as características que fossem importantes para sua análise.

6.3 Venda de aparelhos celulares com serviço WAP

Este exemplo foi proposto pela empresa CPqD Telecom & IT Solutions (Campinas/SP) em setembro de 2002, na realização de um curso introdutório sobre modelagem e simulação em agentes. Os aspectos analisados nesta simulação são

específicos da área de telecomunicações e o foco deste trabalho não é estudar a terminologia desta área. A idéia é modelar o problema baseado na análise realizada em conjunto com os especialistas da empresa. A quantidade de agentes, o tamanho de grid e as ações que cada agente tem são baseados em explicações práticas dos especialistas. Não há bibliografia relativa a este assunto especificamente.

Este exemplo foi primeiramente modelado sem a utilização de emoções, e depois com a inserção de emoções em suas regras. No Anexo II é apresentado o código gerado pelo sistema para sua execução (conforme seção 5.1.3), e seria interessante que pudessem ser visualizadas as mudanças ocorridas no código quando emoções são inseridas na aplicação, bem como é realizado o cálculo de intensidade para cada emoção, a fim de definir a que será expressa no ciclo da simulação.

Na situação proposta, existem dois tipos de agentes, os compradores e os vendedores do serviço WAP (*Wairless Aplication Protocol*). O público-alvo para venda deste produto, no caso os compradores, já possuem aparelho celular. Deste público, 30% tem interesse em pagar por uma tarifa fixa de no máximo de R$ 10,00; 40% tem interesse por uma tarifa fixa de no máximo de R$ 15,00; e 30% não possuem interesse algum em adquirir o serviço. O tamanho do mercado é de 40 compradores e 2 vendedores, em um espaço (*grid*) de 10x10.

Os vendedores podem oferecer o serviço da seguinte maneira:

- se os compradores têm interesse pela tarifa fixa de R$ 10,00, o valor oferecido fica no intervalo de R$ 8,00 a R$ 15,00;
- se os compradores têm interesse pela tarifa fixa de R$ 15,00, o valor oferecido fica no intervalo de R$ 10,00 a R$ 25,00.

Os vendedores podem oferecer o serviço com um valor superior ao que os compradores estão dispostos a pagar, sendo que neste caso, a venda não é realizada. A cada tentativa, até a compra do serviço, o número de abordagens para venda são armazenados, para analisar quanto o comprador precisa “pesquisar” para encontrar o produto que deseja. Nesta simulação os vendedores são parados, ou seja, eles estão em ”pontos de venda”. Os vendedores abordam os compradores que passam pelos pontos venda.

O objetivo desta simulação é descobrir o preço médio de venda do serviço (tarifa fixa 10 ou fixa 15) e o número médio de contatos que o comprador realizou até adquirir o serviço.

Foram definidas quatro situações, onde emoções são incluídas, utilizando o Modelo OCC:

- compradores impulsivos, que compram a primeira oferta feita pelo vendedor, independente de quanto possa pagar. A emoção escolhida para esta situação foi *Satisfaction*, pois o comprador sente uma realização em adquirir o produto;
- compradores que mesmo tendo interesse em adquirir o produto, e o preço ofertado pelo vendedor está dentro de suas condições, não adquire o serviço. A emoção escolhida para esta situação foi *Fear*, pois o comprador tem medo de estar realizando um mau negócio;

- vendedores oferecem o serviço ao comprador, e este é superior ao que este pode pagar. O vendedor realiza uma nova oferta, baixando seu preço, na tentativa de realizar a venda. A emoção escolhida para esta situação foi *Disappointment*, pois os vendedores se sentem desapontados por ter que vender o produto por um valor inferior ao pretendido;
- vendedores que realizam pesquisa de mercado com os outros vendedores, a fim de verificar se o número de vendas efetuadas pelos outros é superior a sua e se os valores oferecidos são inferiores, obrigando os mesmos a baixar seus preços. A emoção escolhida para esta situação foi *Envy*, pois os vendedores estão sentindo inveja do trabalho realizado pelos outros vendedores.

6.3.1 Modelagem da Venda de aparelhos celulares com serviço WAP no AFRODITE sem Emoção

Os agentes compradores possuem seis características:

- *contato:* para armazenar quantas vezes o serviço foi oferecido até ser adquirido - tipo de dado: *integer*;
- *serviço:* para saber se o comprador quer adquirir o serviço do tipo tarifa *fixo* 10 ou tarifa fixo 15 - tipo de dado: *string*;
- *percentual_fixo_10:* para saber qual é o percentual de compradores que querem o serviço com tarifa *fixo* 10 - tipo de dado: *float*;
- *percentual_fixo_15:* para saber o percentual de compradores que querem o serviço com tarifa *fixo* 15 - tipo de dado: *float*;
- *total_percentual_fixo10:* para saber se a quantidade de compradores que receberam tarifa *fixo*10 atingiu o percentual desejado - tipo de dado: *float*;
- *total_percentual_fixo_15:* para saber se a quantidade de compradores que receberam tarifa *fixo* 15 atingiu o percentual desejado - tipo de dado: *float*.

Os agentes vendedores possuem três características:

- *preço:* para armazenar o preço gerado (randomicamente) ao comprador no momento da venda - tipo de dado: *float*;
- *num_vendidos:* para armazenar a quantidade de serviços vendidos - tipo de dado: *integer*;
- *valor_vendidos:* para armazenar o valor pelo qual o serviço foi efetivamente vendido - tipo de dado: *float*.

Agentes:

- nome: *comprador*
- tipo_objeto: false
- qtdade_agentes: 40
- lista_características:
    - contato — valor_integer: -1
    - servico — valor_string: nenhum
    - percentual_fixo10 — valor_float: 30
    - percentual_fixo15 — valor_float: 40

- total_percentual_fixo10 valor_float: -1
- total_percentual_fixo15 valor_float: -1

- nome: *vendedor*
- tipo_objeto: false
- qtdade_agentes: 2
- lista_características:
  - preço valor_float: -1
  - num_vendidos valor_integer: -1
  - valor_vendidos valor_float: -1

Regras:

- nome: *inicial*

  condição: getCycle == 1

  então: iniVal("vendedor","num_vendidos","integer","0");
  iniVal("vendedor","valor_vendidos","float","0");
  iniVal("comprador","contato","integer","0");
  iniVal("comprador"," total_percentual_fixo10","float","0");
  iniVal("comprador"," total_percentual_fixo15","float","0");

  precedencia: 400

  Função: inicializar variáveis no primeiro ciclo de simulação;

- nome: *indicar_servico_fixot10*

  condição: getCycle == 1 &&
  content(getRow(),getCol()) == "comprador" &&
  comprador.servico == "nenhum" &&
  comprador.total_percentual_fixo10 < perc("comprador",
  "percentual_fixo10","float")

  então: iniVal("comprador","servico","string","fixo10");
  incVal("comprador","total_percentual_fixo10", "float");

  precedencia: 300

  Função: inicializar a variável serviço com o valor “fixo 10”, do agente *comprador*, comparando se o percentual desejado já foi atingido, ou seja 30%;

- nome: *indicar_servico_fixo15*

  condição: getCycle == 1 &&
  content(getRow(),getCol()) == "comprador" &&
  comprador.servico == "nenhum" &&

comprador.total_percentual_fixo15 < perc("comprador",
"percentual_fixo15","float")

então: iniVal("comprador","servico","string","fixo15");
incVal("comprador","total_percentual_fixo15", "float");

precedencia: 300

Função: inicializar a variável serviço com o valor “fixo 15”, do agente *comprador*, comparando se o percentual desejado já foi atingido, ou seja 40%;

- nome: *contato_vendedor1*
  condição: content(getRow(),getCol()) == "vendedor" &&
  neighbor("comprador") != 0 &&
  valNeighbor(“serviço”,string”) == "fixo10"

  então: addVal("vendedor","preco","float",random("float",8,15));
  incNeighbor("contato","integer");

  precedencia: 200

  Função: o agente *vendedor* procura em sua “vizinhança” algum comprador para ofertar o serviço do tipo tarifa fixo10. Neste momento o vendedor inicializa sua variável preço e o comprador incrementa sua variável contato;

- nome: *contato_vendedor2*
  condição: content(getRow(),getCol()) == "vendedor" &&
  neighbor("comprador") != 0 &&
  valNeighbor(“serviço”,”string”) == "fixo15"

  então: addVal("vendedor","preco","float",random("float",10,25));
  incNeighbor("contato","integer");

  precedencia: 200

  Função: o agente *vendedor* procura em sua “vizinhança” algum comprador para ofertar o serviço do tipo tarifa fixo15. Neste momento o vendedor inicializa sua variável preço e o comprador incrementa sua variável contato;

- nome: *venda1*
  condição: valNeighbor("servico","string") == "fixo10" &&
  vendedor.preco <= 10

  então: incVal("vendedor","num_vendidos","integer");
  addVal("vendedor","valor_vendidos","float","vendedor.preco");

  precedencia: 100

  Função: o agente *comprador* com tarifa fixo10 compara o preço ofertado pelo vendedor com o que está disposto a pagar. Se o valor for igual ou menor que o

valor estipulado, o vendedor incrementa a quantidade de serviços vendidos e armazena o valor pelo qual o serviço foi vendido;

- nome: *venda2*
  condição: valNeighbor("servico","string") == "fixo15" &&
  vendedor.preco <= 15

  então: incVal("vendedor","num_vendidos","integer");
  addVal("vendedor","valor_vendidos","float","vendedor.preco");

  precedencia: 100

  Função: o agente *comprador* com tarifa fixo15 compara o preço ofertado pelo vendedor com o que está disposto a pagar. Se o valor for igual ou menor que o valor estipulado, o vendedor incrementa a quantidade de serviços vendidos e armazena o valor pelo qual o serviço foi vendido;

- nome: *mover_comprador*
  condição: content(getRow(),getCol()) == "comprador"

  então: movAgent("comprador",random("integer",0,9), random("integer", 0, 9));

  precedencia: 50

  Função: o agente comprador se move para linha e coluna gerados randomicamente, desde que a esta posição esteja livre, ou seja, nenhum outro agente esteja alocado nesta posição;

6.3.2 Modelagem da Venda de aparelhos celulares com serviço WAP no AFRODITE com Emoção

Na modelagem com emoções, as variáveis não foram modificadas, porém as regras de comportamento sofreram alterações. As regras *inicial, indicar_serviço_fixo10, indicar_serviço_fixo15 e mover_comprador* não foram alteradas (verificar seção 6.3.1). As regras *venda1 e venda2* foram excluídas, porque a venda depende agora de uma emoção do comprador ou do vendedor. Estas emoções foram inseridas nas regras *contato_vendedor1 e contato_vendedor2*, que não sofrem alterações em suas ações e condições, apenas têm inseridas as emoções.

Regras de Emoção:

- tipo_emoção: *satisfaction*
  nome_regra: contato_vendedor1

  condição: (valNeighbor(“serviço”,”string”) == “fixo10”
  || (valNeighbor(“serviço”,”string”) == “fixo15”

  então: incVal(“vendedor”,num_vendidos”,”integer”);
  addVal(“vendedor”,”valor_vendidos”,”float”,”vendedor_preço”);

limiar: 0.6
conjunto_valores: [3, 3]

Função: independente do valor oferecido pelo *vendedor*, o *comprador* sempre compra.

- tipo_emoção: *fear*
  nome_regra: contato_vendedor1
  id_classe: 4
  condição: (valNeighbor(“serviço”,”string”) == “fixo10”
  && vendedor.preço < = 10)
  ||(valNeighbor(“serviço”,”string”) == “fixo15”
  && vendedor.preço < = 15)

  então: incNeighbor("contato","integer");

  limiar: 0.6
  conjunto_valores: [3, 3, 3, 3]

  Função: independente do valor oferecido pelo *vendedor*, o *comprador* nunca compra. Não há então “ação” para esta emoção.

- tipo_emoção: *disappointment*
  nome_regra: contato_vendedor1
  id_classe: 4
  condição: (valNeighbor(“serviço”,”string”) == “fixo10”
  && vendedor.preço < = 10)
  ||(valNeighbor(“serviço”,”string”) == “fixo15”
  && vendedor.preço < = 15)

  então: iniVal(“vendedor”,”preço”,”float”,random(“float”,8,25));

  limiar: 0.6
  conjunto_valores: [3, 3, 3, 3]

  Função: se o valor oferecido for maior do que o *comprador* possa pagar, o *vendedor* faz uma nova oferta.

- tipo_emoção: *envy*
  nome_regra: contato_vendedor1
  condição: content(getRow(),getCol()) == “vendedor”
  && neighbor(“vendedor”) != 0
  &&vendedor.num_vendidos<
  valNeighbor(“num_vendidos”,”float”)
  && vendedor.preço < valNeighbor(“preço”,”float”)

  então: iniVal(“vendedor”,”preço”,”float”,random(“float”,8,25));

  limiar: 0.6

conjunto_valores: [3, 3, 3, 3]

Função: *vendedor* procura por outro vendedor ao seu redor, para realizar pesquisa de mercado e modificar seus valores.

6.3.3 Considerações sobre as Modelagens (sem emoção e com emoção)

A modelagem deste cenário, que se refere a um assunto específico e pouco trabalhado até o momento, foi realizada a partir de experiências práticas dos especialistas da área. Primeiramente, foi realizada a modelagem sem a utilização de emoções, e depois com a inserção destas.

Sabendo que os objetivos da simulação deste cenário eram descobrir o preço médio de venda do serviço (tarifa fixa 10 ou fixa 15) e a média de contatos que o comprador realizou até adquirir o serviço, os resultados obtidos são apresentados abaixo.

Para o primeiro caso, sem a inserção de emoções nas regras:

- Preço médio de venda: 12.6
- Média de contatos por comprador: 6.1

Para o segundo caso, com emoções inseridas as regras de comportamento:

- Preço médio de venda: 10.4
- Média de contatos por comprador: 8.8

Para chegar a estes resultados, cada simulação foi repetida 10 vezes e uma média foi calculada. Isso foi realizado para que não houvessem dúvidas sobre os valores gerados, pois como os agentes são alocados randomicamente, e os resultados obtidos poderiam ser decorrência deste fator.

A distribuição das emoções foi a seguinte:

- Compradores: 52% *fear* e 48% *satisfaction*;
- Vendedores: 69% *disappointment* e 31% *envy*.

Com esta distribuição para as emoções pode-se perceber que as emoções alteram a tomada de decisão dos compradores em relação ao preço final de venda do serviço e ao número de contatos que um vendedor necessita fazer até que o produto seja vendido. Isso é derivado ao fato de que o número de vendedores desapontados é superior aos com inveja, e o número de compradores com medo de realizar um mau negócio é superior aos que sentem satisfação em comprar o serviço independente do valor.

6.4 Considerações Gerais sobre as Modelagens Realizadas

Os exemplos utilizados no protótipo tiveram o objetivo de mostrar que o ambiente proposto atende aos requisitos desejados e explicados no capítulo 4, como a utilização de primitivas pré-definidas. Houve a possibilidade de modelagem de exemplos de diferentes áreas de pesquisa, mostrando que o ambiente é bastante genérico.

Em cada uma das modelagens foram apresentadas considerações específicas para cada um dos exemplos, porém, independente de alguns aspectos que devem ser acrescentados ao protótipo, o principal objetivo desejado com a modelagem destes exemplos foi mostrar que AFRODITE implementa aplicações baseadas em agentes com emoções de forma transparente e concisa. Não há necessidade do usuário definir variáveis específicas para as emoções. O usuário apenas deve fornecer as condições e as ações para que emoções se manifestem e quais as ações decorrentes para a emoção definida.

Não foram realizadas análises comparativas dos resultados obtidos com estas modelagens, pois no primeiro exemplo (IPD), foram realizadas algumas alterações nas regras de emoção, e os dois outros exemplos não haviam trabalhos realizados com valores a serem comparados. O último exemplo modelado, venda de aparelhos celulares com o serviço WAP, pode ter alguns aspectos analisados, pois foram realizadas simulações sem regras de emoção e com regras de emoção. Este exemplo foi utilizado com base em um de nossos artigos ("*AFRODITE – A Framework for Simulation of Agents with Emotions*", submetido ao *Agent-Based Simulation* 4 - Montpellier - França) - verificar Anexo V.

Outro fator da não comparação de resultados obtidos deve-se ao fato de que a proposta de cálculo realizada neste trabalho, utilizando a Lógica Difusa, é diferenciada de outras implementações específicas de problemas com emoção. Possíveis trabalhos futuros seriam a comparação e análise detalhada dos resultados obtidos no AFRODITE e em outros ambientes.

Cada um dos exemplos apresentou aspectos que poderiam ser alterados, visando melhorar as modelagens realizadas no ambiente. Alguns destes aspectos são: a criação de primitivas que analisem características dos agentes em sua vizinhança, definição de primitivas de ambiente, inicialização automática dos agentes e escolha de quais dados são importantes para análise no arquivo de saída.

7 Conclusões e Trabalhos Futuros

Pelo fato dos ambientes computacionais atuais serem extremamente heterogêneos, torna-se necessário o desenvolvimento de ferramentas capazes de atender não apenas ao interesse das aplicações de cunho acadêmico, mas também para uso em sistemas reais.

A utilização da tecnologia de agentes vem crescendo, devido a diversos fatores que estes possuem, como flexibilidade e cooperação. Este aumento progressivo se deu graças ao interesse da comunidade científica, que realiza uma série de pesquisas na área.

Na abordagem de SMA, as funcionalidades ficam encapsuladas nos agentes, fornecendo assim sistemas modulares e reutilizáveis. Estes sistemas oferecem uma perspectiva interessante para projetar e desenvolver ambientes sofisticados e poderosos, pois aumentam a robustez do sistema e as atividades podem ser paralelizadas, visto que cada agente pode ser analisado isoladamente.

A utilização da simulação como elemento auxiliar na tomada de decisões é muito eficaz, pois seu emprego permite o exame de detalhes específicos com grande precisão. Na simulação computacional, a idéia é realizar uma descrição do comportamento do problema, definir regras deste comportamento e analisar, a partir de testes, como este se comporta.

Diversas aplicações em SMA são desenvolvidas para simular alguma situação da realidade, pois esta abordagem se adapta aos objetivos necessários da simulação. As emoções são uma das áreas de estudo da IA que há alguns anos vem sendo pequisada a fim de provar sua influência nas atividades humanas. Ainda não existe uma definição precisa do que são emoções, pois são muitos os fatores que fazem com que estas se manifestem. Porém, trabalhos específicos da área de simulação de emoções demonstram que estas alteram a tomada de decisão que os seres humanos têm, para as mais diversas situações.

O objeto central de estudo deste trabalho foi o desenvolvimento de um ambiente de simulação baseado em agentes com emoções. A partir de um estudo de alguns ambientes baseados em agentes existentes (SIEME, SWARM, SeSAm e SIMULA), foi possível analisar como estes foram implementados e definir como o novo ambiente poderia ser projetado. AFRODITE incorpora alguns dos aspectos dos ambientes acima citados, como escolha de definição de primitivas pré-definidas, linguagem de programação e interface gráfica.

Além de realizar simulações de situações reais, o ambiente proposto tinha como objetivo a inclusão de emoções em suas regras de comportamento. Para isso, deveria ser escolhido um modelo de estruturação de emoções, que fosse de fácil entendimento para o usuário e que pudesse ser implementado. O modelo OCC foi escolhido, primeiro porque é baseado em regras IF-THEN (ação-condição), podendo ser implementado em máquina de uma forma simplificada, e em segundo lugar, porque ele é base de diversos trabalhos existentes, como [BAT94] [ELL94a] [GRA2001a] [BAZ2001] [BAZ2002] [GMY2002]. Isso prova que o modelo tem sido largamente estudado e analisado, e possui um excelente embasamento teórico.

O ambiente AFRODITE, como todos os outros ambientes estudados, exige que o usuário tenha conhecimento da tecnologia de agentes e que a modelagem seja realizada de forma correta, pois como em qualquer implementação, pequenos detalhes, como inicializações, ou incrementações, alteram o resultado final. Mas a grande vantagem de ambientes de simulação é que o usuário não necessita conhecer integralmente uma linguagem de programação para desenvolver suas aplicações. É necessário apenas conhecer as facilidades que o ambiente oferece, no caso, as primitivas pré-definidas, e ter noções de conectores lógicos (AND, OR e NOT).

Como o AFRODITE visa o desenvolvimento de aplicações que façam uso de emoções, é necessário que no momento em que o usuário for definir as regras que gerarão as emoções, ele defina o limiar mínimo para que estas sejam executadas e determine qual é o peso de cada condição que compõem a regra da emoção. Ou seja: quanto cada condição afeta as atitudes do agente para a emoção escolhida. Desta maneira, primeiramente, é necessário que o usuário compreenda como funciona a modelagem de emoções e quais emoções existem no modelo OCC.

Três exemplos de modelagens foram realizados no AFRODITE, permitindo mostrar a sua adequação ao objetivo proposto, ou seja, a utilização de emoções em um ambiente de simulação e como isso pode ser realizado de forma facilitada. Nos outros ambientes estudados, como por exemplo no SeSAm, seria necessário definir uma variável para cada emoção e implementar uma função específica para calcular a intensidade desta, a fora a definição de cada regra para a geração da emoção desejada. O AFRODITE realiza a tarefa de definição da emoção de forma transparente.

Estes exemplos foram escolhidos por serem de áreas de conhecimento diferentes e tentando demonstrar que este ambiente pode ser utilizado por pesquisadores de diversas áreas, sendo o mais abrangente possível. É importante lembrar que, quando utilizada a tecnologia baseada em agentes, várias modelagens podem ser propostas. Nas modelagens realizadas, houve uma tentativa de utilização da maioria das primitivas pré-definidas pelo ambiente, a fim de mostrar porque as mesmas foram criadas.

Assim, o estudo realizado neste trabalho pôde mostrar aspectos da abordagem de SMA e como sua utilização pode ser aplicada nas mais diversas áreas de pesquisa, como na simulação.

A utilização da abordagem de SMA em conjunto com simulação deve ser analisada para testes em diversas áreas de pesquisa, pois a visualização é modularizada e o custo/benefício de sua implementação são muito maiores, fazendo com que, além de desenvolver com maior facilidade, também tenha-se uma maior agilidade.

Após a realização das modelagens foram analisados alguns aspectos, que podem facilitar a utilização do ambiente (ver capítulo 6). Primeiramente, com a modelagem do IPD foi considerado que há necessidade de criação de primitivas pré-definidas para análise das características da vizinhança do agente e a definição de primitivas de ambiente, para visualização de todos os agentes (como variáveis globais). No cenário da Simulação de Multidões foi considerada a possibilidade do usuário definir o que quer que seja gerado no arquivo de saída, ou seja, apenas as características que sejam necessárias para sua análise.

Além destes aspectos analisados nas modelagens, outras idéias podem ser propostas, objetivando melhorias do protótipo. Uma primeira sugestão é realizar análise sintática logo após a criação de cada uma das regras, ou seja, se a regra estiver formulada de forma incorreta, o erro será constatado já no momento da criação desta. No protótipo atual, o usuário define todas suas regras, e no final o compilador java gera o código executável. Caso o usuário tenha definido algo incorretamente, o erro só aparecerá após a compilação.

Outro aspecto será melhorar a interface gráfica em relação a inserção de regras de comportamento e emoções, a fim de que o usuário possa visualizar na íntegra todas as condições que definir. A idéia é criação de uma caixa de edição, onde cada condição é adicionada, alterada ou removida isoladamente.

Outras duas idéias, que podem melhorar o protótipo, e que exigem algumas alterações em relação a estrutura de dados proposta são:

- em relação a performance: paralelizar as atividades dos agentes. Por ter sido desenvolvido em java, esta tarefa seria realizada a partir da utilização de *threads*. Como cada agente executa suas atividades separadamente, o processamento seria muito mais rápido;
- em relação a criação de simulações mais realísticas utilizando emoções: permitir que cada agente tenha uma “memória” do que aconteceu nos ciclos de simulação anteriores, a fim de estabelecer uma relação entre o que já aconteceu, e se isso afeta a ação a ser tomada no ciclo em questão. Esta tarefa alteraria a estrutura de dados existente, pois seria necessário definir tanto no banco de dados (tabela específica para esta informação), como nas estruturas em lista do protótipo maneiras de armazenar estes dados. Outro ponto a ser analisado é saber qual seria um tamanho razoável de memória que um agente poderia ter (por exemplo, últimos 10 ciclos de simulação). Esta proposta de inserção de memória nos agentes da simulação não é um trabalho trivial para ser realizado, mas traria muitos benefícios as modelagens realizadas, pois os seres humanos sempre tomam decisões baseadas em experiências anteriores.

Além de melhorias na implementação do protótipo, seria interessante a realização de análises comparativas de um mesmo exemplo, entre resultados obtidos no AFRODITE e outros ambientes, como por exemplo, SeSAm ou SIMULA. Estas análises além de comparar os resultados, deveriam comparar o grau de dificuldade de implementação das emoções nos outros ambiente (quantidade de variáveis e regras de comportamento necessárias).

Anexo 1 Categorias de Emoções [ELL2002]

Clark Elliott, 1998
after Ortony, et al., 1988

Group	Specification	Category Label and Emotion Type
Well-Being	appraisal of a situation as an event	joy: pleased about an event distress: displeased about an event
Fortunes-of-Others	presumed value of a situation as an event affecting another	happy-for: pleased about an event desirable for another gloating: pleased about an event undesirable for another resentment: displeased about an event desirable for another jealousy*: resentment over a desired mutually exclusive goal envy*: resentment over a desired non-exclusive goal sorry-for: displeased about an event undesirable for another
Prospect-based	appraisal of a situation as a prospective event	hope: pleased about a prospective desirable event fear: displeased about a prospective undesirable event
Confirmation	appraisal of a situation as confirming or disconfirming an expectation	satisfaction: pleased about a confirmed desirable event relief: pleased about a disconfirmed undesirable event fears-confirmed: displeased about a confirmed undesirable event disappointment: displeased about a disconfirmed desirable event
Attribution	appraisal of a situation as an accountable act of some agent	pride: approving of one's own act admiration: approving of another's act shame: disapproving of one's own act reproach: disapproving of another's act
Attraction	appraisal of a situation as containing an attractive or unattractive object	liking: finding an object appealing disliking: finding an object unappealing
Well-being/ Attribution	compound emotions	gratitude: admiration+joy anger: reproach+distress gratification: pride+joy remorse: shame+distress
Attraction/ Attribution	compound emotion extensions	love:admiration+liking hate:reproach+disliking

*Non-symmetric additions necessary for some stories.

Anexo 2 Código Gerado na Simulação

Este anexo apresenta como o ambiente AFRODITE gera as abstrações da tela em código executável. O código apresentado a seguir é referente ao terceiro exemplo testado (Venda de aparelhos celulares com o serviço WAP). Primeiramente é mostrado o código gerado, onde não foram definidas regras para as emoções (seção 6.3.1).

A segunda parte deste anexo apresenta o código gerado com regras para as emoções (seção 6.3.2). O texto em *itálico* é referente as modificações para inclusão de emoções e como é realizado o cálculo de intensidade das emoções do modelo OCC, proposto neste trabalho.

Código gerado para modelagem sem emoção

```
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.math.*;
import java.sql.*;
import java.util.Date;

public class Codigos {
principal prin;
acessosBD acesso = new acessosBD();
ResultSet consulta;
primitivas primit;
int i, j, m;

public Codigos(principal prin_par, primitivas primit_par){
        prin = prin_par;
        primit = primit_par;
        }

public void realizaTarefa(){
        if ( primit.getCycle() == 1) {
                primit.iniVal("comprador","contato","integer","0");
        }
        if ( primit.getCycle() == 1 {
                primit.iniVal("vendedor","num_vendidos","integer","0");
                primit.iniVal("vendedor","valor_vendidos","float","0");
                primit.iniVal("comprador","tipo","integer",primit.random("integer",1,10));
        }
        if ( primit.getCycle() == 1
        &&(prin.grid[prin.lin][prin.col].nome.equals("comprador"))
        &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[4].tipo_integer<=5))
        {
                primit.iniVal("comprador","servico","string","fixo10");
        }
        if ( primit.getCycle() == 1
        &&(prin.grid[prin.lin][prin.col].nome.equals("comprador"))&&
        (prin.grid[prin.lin][prin.col].lista_caracteristicas[4].tipo_integer<=10))
        {
                primit.iniVal("comprador","servico","string","fixo15");
```

```
		}
		if ( primit.content( primit.getRow(),primit.getCol()).equals("vendedor")
		&& primit.neighbor("comprador") != 0
		&& primit.neighbor("comprador") != 0
		&&primit.valNeighbor("servico","string").equals("fixo10"))
		 {
			primit.iniVal("vendedor","preco","float",primit.random("float",8,15));
			primit.incNeighbor("contato","integer");
			}
		if ( primit.content( primit.getRow(),primit.getCol()).equals("vendedor")
		&& primit.neighbor("comprador") != 0
		&&primit.valNeighbor("servico","string").equals("fixo15"))
		 {
			primit.iniVal("vendedor","preco","float",primit.random("float",10,25));
			primit.incNeighbor("contato","integer");
		}
		if (primit.valNeighbor("servico","string").equals("fixo10")
		&&(prin.grid[prin.lin][prin.col].nome.equals("vendedor"))
		&&(prin.grid[prin.lin][prin.col].lista_caracteristicas[0].tipo_float<=10)){
			primit.addVal("vendedor","num_vendidos","integer","1");
			primit.addVal("vendedor","valor_vendidos","float","vendedor.preco");
		}
		if(primit.valNeighbor("servico","string").equals("fixo15")
		&&(prin.grid[prin.lin][prin.col].nome.equals("vendedor"))
		&&(prin.grid[prin.lin][prin.col].lista_caracteristicas[0].tipo_float<=15)){
			primit.addVal("vendedor","num_vendidos","integer","1");
			primit.addVal("vendedor","valor_vendidos","float","vendedor.preco");
		}
		if( primit.content( primit.getRow(), primit.getCol()).equals("comprador")){
		primit.movAgent("comprador", rimit.random("integer",0,9),primit.random("integer",0,9));
		}
}
}
```

Código gerado para modelagem com emoção

```
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.math.*;
import java.sql.*;
import java.util.Date;

public class Codigos {
principal prin;
acessosBD acesso = new acessosBD();
ResultSet consulta;
primitivas primit;
int i, j, m;

public Codigos(principal prin_par, primitivas primit_par){
		prin = prin_par;
		primit = primit_par;
		}

public void realizaTarefa(){
		if ( primit.getCycle() == 1) {
			primit.iniVal("vendedor-emo","num-vendidos-emo","integer","0");
			primit.iniVal("vendedor-emo","valor-vendidos-emo","float","0");
```

```
                    for(m=0;m<prin.quantidadeEmocoes;m++){
                    }
          }
          if ( primit.getCycle() == 1) {
                    primit.iniVal("comprador-emo","contato-emo","integer","0");
                    primit.iniVal("comprador-emo","tipo-
emo","integer",primit.random("integer",1,10));
                    primit.iniVal("comprador-emo","nao-compra-emo","integer","0");
                    for(m=0;m<prin.quantidadeEmocoes;m++){
                    }
          }
          if ( primit.getCycle() == 1
          &&(prin.grid[prin.lin][prin.col].nome.equals("comprador-emo"))
          &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[2].tipo_integer<=5)){
                    primit.iniVal("comprador-emo","servico-emo","string","fixo10");
                    for(m=0;m<prin.quantidadeEmocoes;m++){
                    }
          }
          if ( primit.getCycle() == 1
          &&(prin.grid[prin.lin][prin.col].nome.equals("comprador-emo"))
          &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[2].tipo_integer<=10)){
                    primit.iniVal("comprador-emo","servico-emo","string","fixo15");
                    for(m=0;m<prin.quantidadeEmocoes;m++){
                    }
          }
          if ( primit.content( primit.getRow(), primit.getCol()).equals("vendedor-emo")
          && primit.neighbor("comprador-emo") != 0
          &&(prin.grid[prin.lin][prin.col].nome.equals("comprador-emo"))
          &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[1].tipo_string.equals("fixo10"))){
                    primit.iniVal("vendedor-emo","preco-emo","float",primit.random("float",8,15));
                    primit.incNeighbor("contato-emo","integer");
                    for(m=0;m<prin.quantidadeEmocoes;m++){
                              if ((prin.grid[prin.lin][prin.col].nome.equals("vendedor-emo"))
                              &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[2].tipo_float
                                                                        >8)){
                                        primit.addVal("vendedor-emo","num_vendidos-
                                                                        emo","integer","1");
                                        primit.addVal("vendedor-emo","valor-vendidos-
                                                  emo","float","vendedor-emo.preco-emo");
                              }
                              if (primit.valNeighbor("servico-emo","string").equals("fixo10")
                              &&(prin.grid[prin.lin][prin.col].nome.equals("vendedor-emo"))
                              &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[2].tipo_float
                                                                        >10)){
                                        primit.iniVal("vendedor-emo","preco-emo","float",
                                                            primit.random("float",8,10));
                              }
                              if(primit.valNeighbor("servico-emo","string").equals("fixo10")
                              &&(prin.grid[prin.lin][prin.col].nome.equals("vendedor-emo"))
                              &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[2].tipo_float
                                                                        <=10)
                              ||primit.valNeighbor("servico-emo","string").equals("fixo15")
                              &&(prin.grid[prin.lin][prin.col].nome.equals("vendedor-emo"))
                              &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[2].tipo_float
                                                                        <=15)){
                                        primit.incNeighbor("nao-compra-emo","integer");
                              }
                              If ( primit.content( primit.getRow(), primit.getCol()).equals("vendedor-
                                                                        emo")
                              && primit.neighbor("vendedor-emo") != 0) {
```

```
                                        primit.iniVal("vendedor-emo","preco-emo","float",
                                                                primit.random("float",9,20));
                                }
                        }
                }
                if ( primit.content( primit.getRow(), primit.getCol()).equals("vendedor-emo")
                && primit.neighbor("comprador-emo") != 0
                &&(prin.grid[prin.lin][prin.col].nome.equals("comprador-emo"))
                &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[1].tipo_string.equals("fixo15"))){
                        primit.iniVal("vendedor-emo","preco-emo","float",primit.random("float",10,25));
                        primit.incNeighbor("contato-emo","integer");
                        for(m=0;m<prin.quantidadeEmocoes;m++){
                                if (primit.valNeighbor("servico-emo","string").equals("fixo15")
                                &&(prin.grid[prin.lin][prin.col].nome.equals("vendedor-emo"))
                                &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[2].tipo_float
                                                                                        >15)){
                                primit.iniVal("vendedor-emo","preco-emo","float",
                                                                primit.random("float",10,25))
                                }
                        }
                }
}
public void calculoEmocao(int i, int j){
                double []intensidade;
                double []potencial;
                String []nome_emocao;
                double []limiar;
                int l = 0;
                int k = 0;
                intensidade = new double[100];
                potencial = new double[100];
                limiar = new double[100];
                nome_emocao = new String[100];
                l = 0;
                while(!prin.lista_condicoes[l].tipo_emocao.equals("")){
                        for(k=0;k<3;k++)
                                if ((prin.grid[prin.lin][prin.col].nome.equals("vendedor-emo"))
                                &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[2].tipo_float>8)){
                                        potencial[l] = prin.lista_condicoes[l].pesos[k]+ potencial[l];
                                        nome_emocao[l] = prin.lista_condicoes[l].tipo_emocao;
                                        limiar[l] = prin.lista_condicoes[l].limiar;
                                }
                                if(primit.valNeighbor("servico-emo","string").equals("fixo10")){
                                        potencial[l] = prin.lista_condicoes[l].pesos[k]+ potencial[l];
                                        nome_emocao[l] = prin.lista_condicoes[l].tipo_emocao;
                                        limiar[l] = prin.lista_condicoes[l].limiar;
                                }
                                if ((prin.grid[prin.lin][prin.col].nome.equals("vendedor-emo"))
                                &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[2].tipo_float
                                                                                        >10)){
                                        potencial[l] = prin.lista_condicoes[l].pesos[k]+ potencial[l];
                                        nome_emocao[l] = prin.lista_condicoes[l].tipo_emocao;
                                        limiar[l] = prin.lista_condicoes[l].limiar;
                                }
                                if(primit.valNeighbor("servico-emo","string").equals("fixo10")){
                                        potencial[l] = prin.lista_condicoes[l].pesos[k]+ potencial[l];
                                        nome_emocao[l] = prin.lista_condicoes[l].tipo_emocao;
                                        limiar[l] = prin.lista_condicoes[l].limiar;
                                }
                                if ((prin.grid[prin.lin][prin.col].nome.equals("vendedor-emo"))
```

```
                        &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[2].tipo_float
                                                                        <=10)){
                                potencial[l] = prin.lista_condicoes[l].pesos[k]+ potencial[l];
                                nome_emocao[l] = prin.lista_condicoes[l].tipo_emocao;
                                limiar[l] = prin.lista_condicoes[l].limiar;
                        }
                        if (primit.valNeighbor("servico-emo","string").equals("fixo15")){
                                potencial[l] = prin.lista_condicoes[l].pesos[k]+ potencial[l];
                                nome_emocao[l] = prin.lista_condicoes[l].tipo_emocao;
                                limiar[l] = prin.lista_condicoes[l].limiar;
                        }
                        if ( primit.content( primit.getRow(), primit.getCol()).equals("vendedor-
                                                                        emo")){
                                potencial[l] = prin.lista_condicoes[l].pesos[k]+ potencial[l];
                                nome_emocao[l] = prin.lista_condicoes[l].tipo_emocao;
                                limiar[l] = prin.lista_condicoes[l].limiar;
                        }
                        if ( primit.neighbor("vendedor-emo") != 0){
                                potencial[l] = prin.lista_condicoes[l].pesos[k]+ potencial[l];
                                nome_emocao[l] = prin.lista_condicoes[l].tipo_emocao;
                                limiar[l] = prin.lista_condicoes[l].limiar;
                        }
                        if(primit.valNeighbor("servico-emo","string").equals("fixo15")){
                                potencial[l] = prin.lista_condicoes[l].pesos[k]+ potencial[l];
                                nome_emocao[l] = prin.lista_condicoes[l].tipo_emocao;
                                limiar[l] = prin.lista_condicoes[l].limiar;
                        }
                        if ((prin.grid[prin.lin][prin.col].nome.equals("vendedor-emo"))
                        &&(prin.grid[prin.lin][prin.col].lista_caracteristicas[2].tipo_float
                                                                        >15)){
                                potencial[l] = prin.lista_condicoes[l].pesos[k]+ potencial[l];
                                nome_emocao[l] = prin.lista_condicoes[l].tipo_emocao;
                                limiar[l] = prin.lista_condicoes[l].limiar;
                        }
                l++;
                }
                for(k=0;k<l;k++)
                        if(potencial[k]>limiar[k])
                                intensidade[k] = potencial[k]-limiar[k];
                        else intensidade[k] = 0;
                for(k=0;k<l;k++){
                        for(int m=k;m<l;m++)
                                if(intensidade[k]>=intensidade[m])
                                        prin.grid[i][j].tipo_emocao = nome_emocao[k];
                }
        }
}
```

Anexo 3 Exemplo de Arquivo de Saída Gerado pela Simulação

Este anexo apresenta como o ambiente AFRODITE gera o arquivo de saída de dados, citado na seção 5.1.2. Os dados apresentados abaixo são uma parte dos dados gerados para o terceiro exemplo testado (Venda de aparelhos celulares com o serviço WAP).

Os dados abaixo mostram com a primeira instância do agente "Vendedor" e como suas características são apresentadas no arquivo.

Cycle	vendedor@1.preco	Vendedor@1.num_vendidos	vendedor@1.valor_vendidos
1	18	0	0
2	24	0	0
3	10	1	10
4	24	1	10
5	13	2	23
6	20	2	23
7	22	2	23
8	13	3	36
9	14	4	50
10	13	5	63
11	19	5	63
12	13	6	76
13	20	6	76
14	16	6	76
15	22	6	76
16	10	7	86
17	12	8	98
18	23	8	98
19	22	8	98
20	12	9	110
21	12	10	122
22	15	11	137
23	22	11	137
24	15	12	152
25	21	12	152
26	15	13	167
27	10	14	177
28	14	15	191
29	17	15	191
30	21	15	191
31	16	15	191
32	20	15	191
33	19	15	191
34	22	15	191
35	12	16	203
36	18	16	203

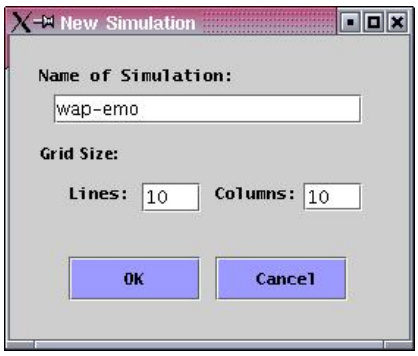
37	11	17	214
38	16	17	214
39	20	17	214
40	21	17	214
41	12	18	226
42	23	18	226
43	12	19	238
44	18	19	238
45	13	20	251
46	13	21	264
47	22	21	264
48	21	21	264
49	10	22	274
50	11	23	285
51	15	24	300
52	11	25	311
53	17	25	311
54	21	25	311
55	15	26	326
56	24	26	326
57	24	26	326
58	13	27	339
59	20	27	339
60	14	28	353
61	19	28	353
62	16	28	353
63	20	28	353
64	15	29	368
65	21	29	368
66	10	30	378
67	13	31	391
68	21	31	391
69	16	31	391
70	21	31	391
71	12	32	403
72	21	32	403
73	15	33	418
74	24	33	418
75	10	34	428
76	16	34	428
77	21	34	428
78	10	35	438
79	23	35	438
80	10	36	448
81	22	36	448
82	10	37	458
83	15	38	473
84	24	38	473
85	14	39	487
86	22	39	487
87	22	39	487
88	18	39	487

89	13	40	500
90	14	41	514
91	14	42	528
92	17	42	528
93	22	42	528
94	13	43	541
95	21	43	541
96	10	44	551
97	24	44	551
98	13	45	564
99	24	45	564

Anexo 4 Manual do Usuário

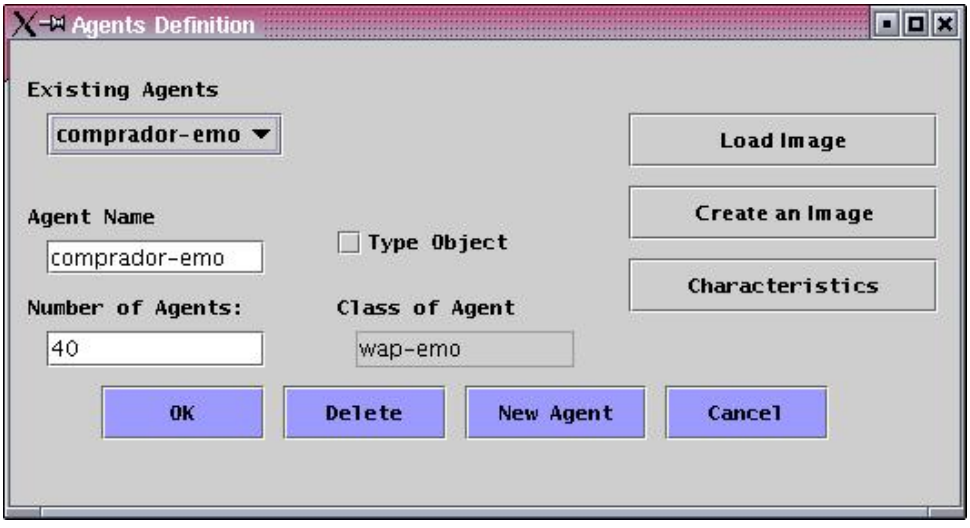
Este manual apresenta passo a passo como o usuário deve proceder para utilizar o ambiente AFRODITE. Como já dito anteriormente, o usuário deve ter conhecimento da tecnologia de agentes para desenvolver sua modelagem.

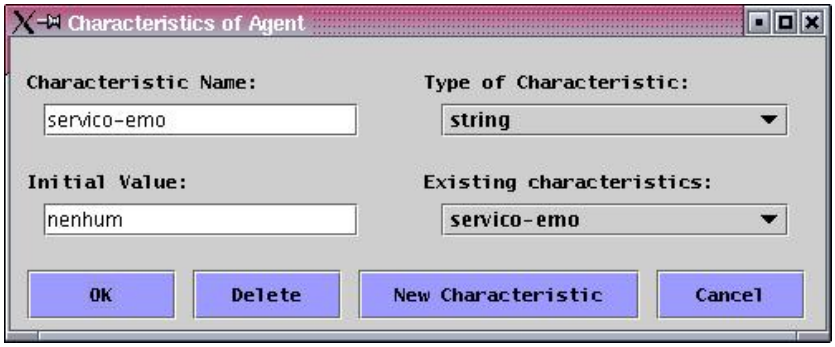
Como exemplo, será utilizado o terceiro exemplo testado - Venda de aparelhos celulares que tenham o serviço WAP (seção 6.3.2). Primeiramente o usuário deve criar uma nova simulação, no menu *File*, opção *New Simulation*, definindo o nome da simulação e tamanho do grid. No exemplo, no nome da simulação é **wap-emo** e o tamanho do grid é de **10 x 10**.



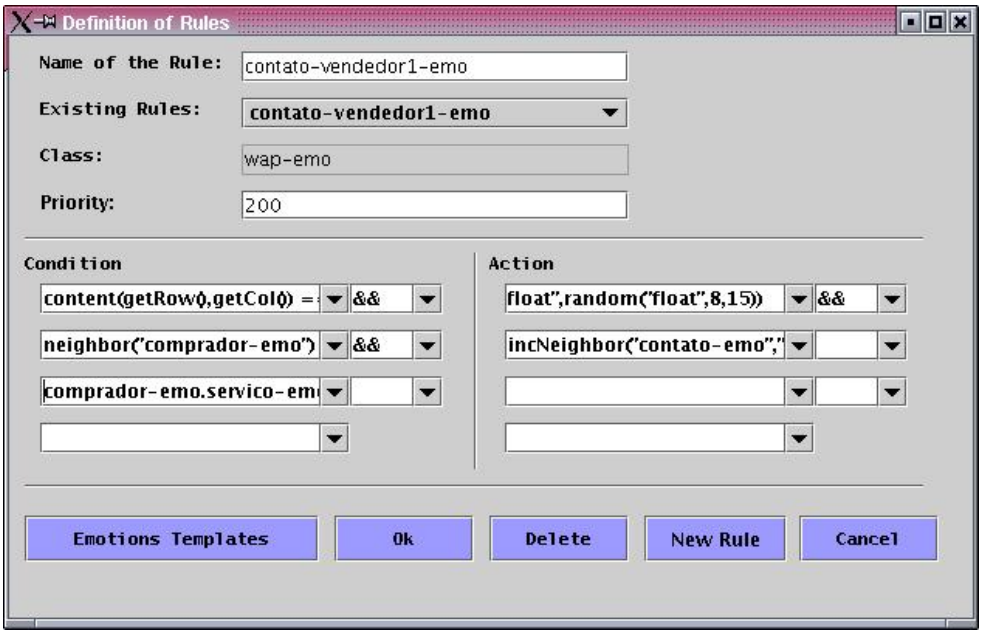
Depois o usuário deve definir os agentes que fazem parte da simulação. Para isso, deve utilizar o menu *Definition*, opção *Agents Definition* (definir nome do agente, quantidade de agentes na simulação, se é do tipo objeto e sua definição gráfica - Botões *Load Image* ou *Create an Image*).

Caso o agente seja do tipo sujeito, devem ser definidas as características (nome, valor inicial e tipo de dado), clicando no botão *Characteristics* da tela *Agents Definition*. Para exemplificar, serão apresentados a criação um agente e uma característica (agente **Comprador-emo,** característica **serviço-emo**).

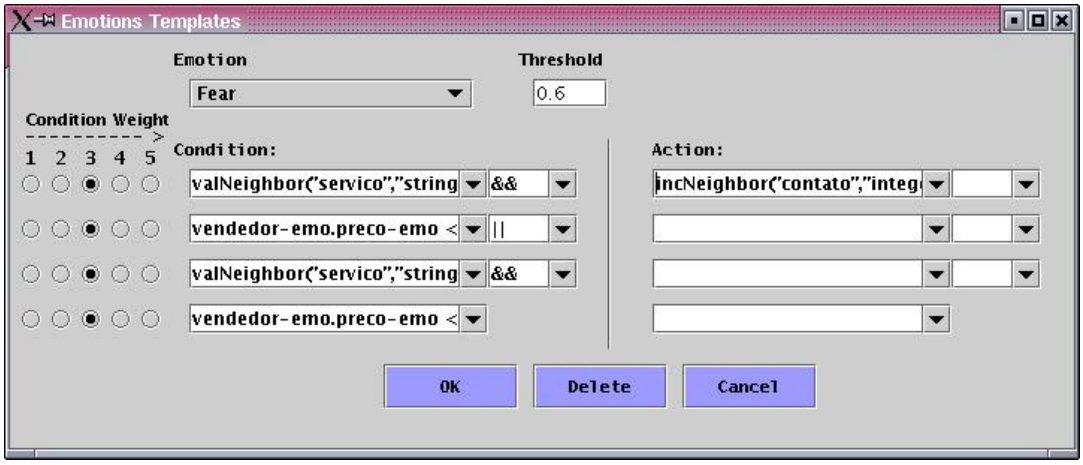




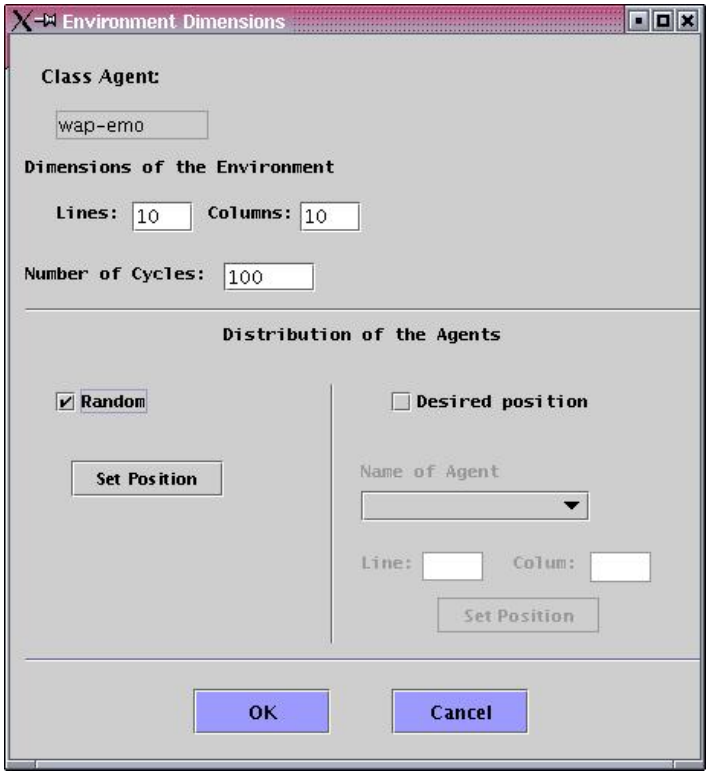
Após deve-se definir as regras que fazem parte da simulação (nome, precedência, condições e ações). Utilizar menu *Definition*, opção *Rules Definition*. A regra apresentada a seguir é **contato_vendedor1-emo**. São apresentadas as primitivas pré-definidas (seção 4.3), bem como agentes definidos com suas respectivas características, a fim de auxiliar a criação das regras.



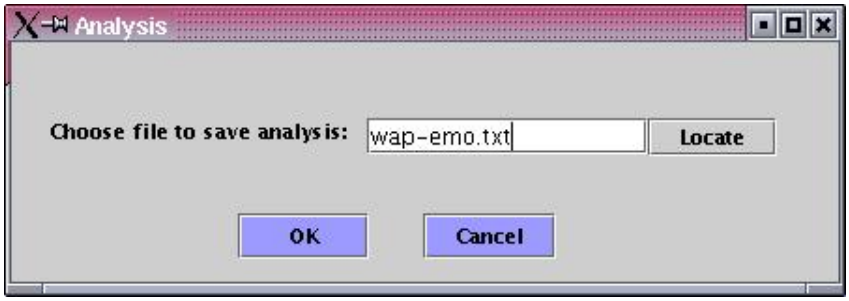
Caso a regra tenha emoções inseridas, clicar no botão *Emotions Templates*. O usuário deve selecionar a emoção desejada, definir o limiar mínimo para que a emoção se manifeste, suas condições e suas ações. Para cada uma das condições deve-se definir um valor que vai de 1 a 5 (sendo 1 o menor e 5 o maior). Este valor determina o "peso" que cada condição, quando verdadeira, possui para a geração da emoção proposta. O cálculo para geração desta emoção está explicado na seção 4.4. A emoção apresentada a seguir é *fear*.



Depois que todos os agentes e regras de comportamento e emoções forem definidos, deve-se alocar os agentes no grid. Para isso, utilizar o menu *Definition*, opção *Environment Demension*. Para o exemplo, os agentes são alocados randomicamente, então seleciona-se a opção *Random* e clica-se no botão *Set Position*. Caso o usuário deseje alterar o tamanho do grid e o número de ciclos total da simulação (por *default*, o número de ciclos apresentados é 100), o usuário também pode fazer nesta opção.



Se o usuário desejar que os dados sejam armazenados em um arquivo de saída, para futuras análises, deve utilizar o menu *File*, opção *Analysis*.



Para iniciar a simulação, o usuário deve executar seu código, utilizando o menu *Simulation*, opção *Execute*. Neste passo, o AFRODITE gera o código para execução da simulação. Em seguida, o usuário escolhe nos menus na tela principal se deseja executar a simulação toda de uma vez (botão *Play*) ou passo a passo (botão *Step by step*).

Anexo 5 Relação de Trabalhos Publicados ou Submetidos

- **Regional Center And Grid Development In Brazil.**
  Cláudio F. R. Geyer, Adenauer Yamin, Luciano C. Silva, Patrícia K. Vargas, Marko Petek, Diana F. Adamatti, Iara Augustin, Jorge L. V. Barbosa.
  Lishep 2002 Grid Workshop, UERJ, Rio de Janeiro / RJ.

- **Sistemas Multiagentes em Ambientes de Aprendizagem**
  Diana Francisca Adamatti
  CBComp 2002 – Agent's Day – Itajaí / SC

- **Extending the Computacional Study of Social Norms with a Systematic Model of Emotions**
  Ana L. C. Bazzan, Diana F. Adamatti e Rafael H. Bordini
  SBIA 2002 – Recife / PE

- **A Framework for Simulation of Agents with Emotions**
  Diana F. Adamatti e Ana L.C. Bazzan
  WORKCOMP 2002 – ITA – São José dos Campos / SP

- **Using the OCC Model to Simulate Agents with Emotions: Case Studies and a Proposal for a Framework**
  Ana L. C. Bazzan, Diana F. Adamatti e Rafael H. Bordini
  Submetido ao RASTA 2002 (capítulo de livro)

- **AFRODITE – A Framework for Simulation of Agents with Emotions**
  Diana F. Adamatti e Ana L. C. Bazzan
  Aceito ao Agent-Based Simulation 4 - Montpellier – França

Bibliografia

[ADA2001] ADAMATTI, D.F. **Estudo Comparativo de Ambientes de Simulação Baseados em Agentes.** 2001. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[ALL93] ALLEN, S. Resolving Conflicts using Emotional Behaviour in the Blocks World. In: WORKSHOP ON ARCHITECTURES UNDERLYING MOTIVATION AND EMOTION, WAUME, 1993. **Proceedings…** Birmingham, UK: The University of Birmingham, 1993.

[ALV97] ALVARES, L. O. C.; SICHMAN, J. S. Introdução aos Sistemas Multiagentes. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 16., 1997, Brasília. **Anais...** Brasília: UnB, 1997. p.1-37.

[AND99] ANDRÉ, E. et al. Integrating Models of Personality and Emotions into Lifelike Characters. In: WORKSHOP ON AFFECT IN INTERACTIONS - TOWARDS A NEW GENERATION OF INTERFACES, 1999. Siena, Itália. **Proceedings…** [S.l. : s.n.], 1999. p. 136-149.

[ARA93] ARAUJO, A.F.R. Emotions Influencing Cognition: Effect of Mood Congruence and Anxiety upon Memory. In: WORKSHOP ON ARCHITECTURES UNDERLYING MOTIVATION AND EMOTION, WAUME, 1993. **Proceedings…** Birmingham, UK: The University of Birmingham, 1993.

[BAT92] BATES, J.; LOYALL, A.B.; REILLY, W.S. Integrating Reativity, Goals and Emotion in a Broad Agent. In: ANNUAL CONFERENCE OF THE COGNITIVE SCIENCE SOCIETY, 14., 1992. **Proceedings…** [S.l. : s.n.], 1992.

[BAT94] BATES, J. **The Role of Emotion in Believable Agents.** Pittsburgh, PA: School of Computer Science, Carnegie Mellon University,1994. (Technical Report CMU-CS-94-136).

[BAZ97] BAZZAN, A. L. C. **An Evolutionary Game-Theoretical Approach for Coordination of Traffic Signal Agents**. 1997. Tese (Doutorado) - Fakultat fur Informatik, Universität Karlsruhe, Karlsruhe.

[BAZ98] BAZZAN, A. L. C.; BORDINI, R. H.; CAMPBELL, J. A. Moral sentiments in multi-agents systems. In: INTERNACIONAL WORKSHOP ON AGENT THEORIES, ARCHITECTURES AND LANGUAGES, ATAL, 5., 1998, Paris. **Inteligent Agents V:** proceedings. Paris: Computer Science Laboratory / University Pierre et Mane Curie, 1998.

[BAZ2001] BAZZAN, A. L. C.; BORDINI, R. H. A Framework for the Simulations of Agents with Emotions: Report on Experiments with the Interated Prisioner's Dilemma. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS SYSTEMS, AGENTS, 5., 2001. **Proceedings.....** [S.l. : s.n.], 2001.

[BAZ2002] BAZZAN, A.L.C.; ADAMATTI, D. F.; BORDINI, R. H; Extending the Computacional Study of Social Norms with the Use a Systematic Model of Emotions**.** In: BRAZILIAN SYMPOSIUM ON ARTIFICIAL INTELLIGENCE, SBIA, 16., 2002, Porto de Galinhas. **Advances in Artificial Intelligence**: proceedings. Berlin: Springer-Verlag, 2002. p.108-117. (Lecture Notes in Artificial Intelligence, v. 2507)

[BER2001] BERCHT, M. **Em direção a Agentes Pedagógicos com dimensões afetivas.** 2001. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[BIT98] BITTENCOURT, G. Inteligência Artificial Distribuída. In: WORKSHOP DE COMPUTAÇÃO, WORKCOMP, 1., 1998, São José dos Campos. **Anais...** São José dos Campos: CTA/ITA, 1998.

[BRA97] BRADSHAW, J. M. **Software Agents**. Menlo Park: AAAI Press / The MIT Press, 1997.

[CAN99] CAÑAMERO, D.; VAN DE VELDE, W. Emotionally Grounded Social Interaction. In: DAUTENHAHN, K. (Ed.). **Human Cognition and Social Agent Technology.** Amsterdam: John Benjamins, 1999.

[CAG97] CAGLAYAN, A.; HARRISON, C. **Agent SourceBook - A complete Guide to Desktop, Internet and Intranet Agents**. New York: Wiley Computer Publishing, 1997. 349 p.

[CAS95] CASTELFRANCHI, C.; CONTE, R. Understanding the effects of norms in social groups through simulation. In: GIBERT, G.N.; CONTE, R. (Ed.). **Artificial Societies**: the computer simulation of social life. London: UCL Press, 1995. p. 252-267.

[CHW94] CHWELOS, G.; OATLEY, K. Appraisal, Computational Models, and SchererÍs Expert System. **Cognition and Emotion**, London, v. 8, n. 3, p. 245-257, 1994.

[COR99] CORDENONSI, A. Z.; ALVARES, L. O. SIMULA ++: a Tool Evolutionary Reactive Multi-Agent Systems. In: ARGENTINE

SYSMPOSIUM ON ARTIFICIAL INTELLIGENCE, 1999, Buenos Aires. **Proceedings....** Buenos Aires: Sadro, 1999.

[DAM2000] DAMÁSIO, A. R. **O Mistério da Consciência**: do corpo e das emoções ao conhecimento de si. São Paulo: Companhia das Letras, 2000. 474 p.

[DEL97] DEL NERO, H.S. **O sítio da mente**: pensamento, emoção e vontade no cérebro humano**.** São Paulo: Collegium Cognitio, 1997. 510 p.

[DYE87] DYER, M. Emotions and their computations: Three Computer Models. **Cognition and Emotion**,London, v. 1, n. 3, p. 323-347, 1987.

[ELL94a] ELLIOTT, C. Research problems in the use of a shallow Artificial Intelligence model of personality and emotion. In: AAAI SPRING SYMPOSIUM ON BELIEVABLE AGENTS, 1994, Stanford. **Proceedings….** [S.l. : s.n.], 1994.

[ELL94b] ELLIOTT, C. Multi-Media Communication with Emotion-driven "Believable Agents". In: AAAI SPRING SYMPOSIUM ON BELIEVABLE AGENTS, 1994, Stanford. **Proceedings….** [S.l. : s.n.], 1994.

[ELL97] ELLIOTT, C. Affective Reasoner Personality Models for Automated Tutoring Systems. In: WORKSHOP PEDAGOGICAL AGENTS, 5., Kobe, JP. **Proceedings….** [S.l. : s.n.], 1997. p. 33- 39.

[ELL2002] ELLIOTT, C. **Affective Reasoning Project.** Disponível em: <http://condor.depaul.edu/~elliott/ar.html>. Acesso em: fev. 2002.

[FER99] FERBER, J. **Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence**. London: Addison-Wesley, 1999. 509 p.

[FRO97] FROZZA, R. **SIMULA**: Ambiente para Desenvolvimento de Sistemas Multiagentes Reativos. 1997. 116f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[FRO99] FROZZA, R. **Estudo sobre Coordenação de Agentes em Ambientes Multiagentes.** 1999. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[GAS92] GASSER, F.J.; HOFFMANN, D. A Multi-Agent Architecture for Oparation and Maintenance of Telecommunicatons Networks**.** In: INTERNACIONAL COFERENCE ON AI, EXPERT SYSTEMS AND NATURAL LANGUAGE, 12., 1992. **Proceedings…** [S.l. : s.n.], 1992. p. 427-436.

[GMY2002] GMYTRASIEWICZ, P.J.; LISETTI, C. L. Emotions and Personality in Agent Design. In: INTERNATIONAL CONFERENCE ON

AUTONOMOUS AGENTS AND MULTI AGENTS SYSTEMS, AAMAS, 2002. **Proceedings…** New York: ACM Press, 2002. p. 360-33.

[GRA2001a] GRATCH, J.; MARSELHA, S. Tears and Fears: Modeling emotions and emotional behaviors in synthetic agents. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS SYSTEMS, AGENTS, 5., 2001. **Proceedings.....** [S.l. : s.n.], 2001.

[GRA2001b] GRATCH, J.; MARSELHA, S. Modeling Emotions in the Mission Rehearsal Exercise. In: CONFERENCE ON COMPUTER GENERATED FORCES AND BEHAVIORAL REPRESENTATION, 10., 2001. **Proceedings…**[S.l. : s.n.], 2001.

[IWA2001] IWAZUME, M.; KATO, Y.; KANAI, A. A Multi-agent-based Model for Analyzing Human Cognitive Process of Advertising Information. In: WORLD MULTI-CONFERENCE ON SYSTEMICS, CYBERNETICS AND INFORMATICS, SCI, 5., 2001. **Proceedings…** [S.l. : s.n.], 2001.

[KAP2001] KAPOOR, A.; MOTA, S.; PICARD, R. Towards a Learning Companion that Recognizes Affect. In: EMOTIONAL AND INTELLIGENT II: THE TANGLED KNOT OF SOCIAL COGNITION, AAAI FALL SYMPOSIUM, 2001. **Proceedings…** [S.l. : s.n.], 2001.

[KLÜ2002] KLÜGL, F. et al. Multi-Agent Modelling in Comparison to Standard Modelling. In: BARROS, F. J.; GIAMBIASI, N. (Ed.). **Artificial Intelligence, Simulation and Planning in High Autonomy Systems.** Lisboa: Publishing House, 2002. p. 105-110.

[KNA98] KNAPIK, M.; JOHNSON, J. **Developing Intelligent Agents for Distributed Systems**: Exploring architecture, technologies & applications. New York: McGraw-Hill, 1998. 389 p.

[LIE95] LIEBERMAN, H. **Attaching Interface Agent Software to Applications**. Cambridge, MA: Media Laboratory, Massachussets Institute of Technology – MIT, 1995.

[MAG2001a] MAGNIN, L. **SIEME**: An Interactions Based Simulation Model. Ibaraki, Japão: Electrotecnical Laboratory, 2001.

[MAG2001b] MAGNIN, L. **Rectangles and Circles**: Towards Realistic Simulation of Robots Playing Soccer. Ibaraki, Japão: Electrotecnical Laboratory, 2001.

[MIN86] MINSKY, M. **The Society of Mind.** New York : Simon and Schuster, 1986.

[MOF93] MOFFAT, D.; FRIJDA, N.H.; PHAF, R.H. Analysis of a model of emotions. In: PROSPECTS FOR ARTIFICIAL INTELLIGENCE, 1993. **Proceedings…** Washington, DC: IOS Press, 1993.

[MOF2000] MOFFAT, D. C.; FRIJDA, N. Functional models of emotion. In: HATANO, G.; OKADA, N.; TANABE, H. (Ed.). **Affective minds.** Amsterdam: Elsevier Science, 2000. p. 59-68.

[NEW89] NEWELL, A.; ROSENBLOOM, P.S.; LAIRD, J.E. Symbolic architectures for cognition. In: POSNER, M. (Ed.). **Foundations of cognitive science.** Cambridge: MIT Press, 1989.

[NOW92] NOWAK, M.; MAY, R. Evolutionary games and spatial chaos. **Nature**, New York, v. 359, p.826-829, 1992.

[NWA96] NWANA, H. S. et al. **ZEUS**: A Toolkit for Building Distributed Multi-Agent Systems. London: Applied Research & Technology – BT Laboratories, 1996.

[OAT87a] OATLEY, K. Cognitive science and the understanding of emotions. **Cognition and Emotion**, London, v.1, n. 3, p. 209-216, 1987.

[OAT87b] OATLEY, K.; JOHNSON-LAIRD, P.N. Towards a cognitive theory of the emotions. **Cognition and Emotion**, London, v. 1, n. 1, p. 29-50, 1987.

[OLI2002] OLIVEIRA, E.; SARMENTO, L. Emotional Valence-ased Mechanisms and Agent Personality. In: BRAZILIAN SYMPOSIUM ON ARTIFICIAL INTELLIGENCE, SBIA, 16., 2002, Porto de Galinhas. **Advances in Artificial Intelligence**: proceedings. Berlin: Springer-Verlag, 2002. p.152-162. (Lecture Notes in Artificial Intelligence, v. 2507).

[ORT88] ORTONY, A.; CLORE, G.; COLLINS, A. **The Cognitive Structure of Emotions.** Cambridge: Cambridge University Press, 1988. 207 p.

[PER97] PEREIRA, A. S. **Um Estudo de Aplicações de Ensino na Internet Orientadas a Agentes**. 1997. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[PIC97] PICARD, R. **Affective Computing**. Cambridge,MA: MIT Press, 1997. 292p.

[PIC2001] PICARD, R. What does it mean for a computer to “have” emotions? In: TRAPPL, R.; PETTA, P.; PAYR, S. (Ed.). **Emotions in Humans and Artifacts**. Cambridge, MA: MIT Press, 2001.

[REB85] REBER, A. S. **Dictionary of psychology**. London: Penguin, 1985.

[REB99] REBONATTO, M. T. **Um Estudo sobre Simulação Paralela**. 1999. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[REI92] REILLY, W. S.; BATES, J. **Building Emotional Agents.** Pittsburgh, PA: School of Computer Science, Carnegie Mellon University,1992. (Technical Report CMU-CS-92-143)

[ROS95] RUSSEL, S.; NORWIG, P. **Artificial Intellingence - A Modern Approach**. New Jersey: Prentice Hall, 1995. 932p.

[SAL2002] THE SALK INSTITUTE FOR BIOLOGICAL STUDIES**. Emotion Home Page**. Disponível em: <http://emotion.salk.edu/>. Acesso em: fev. 2002.

[SES2001] SESAM**. The Multi-Agent Simulation Environment**. Disponível em: <http://www.simsesam.de>. Acesso em: jun. 2001.

[SIE2001a] SIEME. **Simulateur d'environnement pour systèmes multi-agents**. Disponível em: <http://www-poleia.lip6.fr/~magnin/these/sieme.html>. Acesso em: jun. 2001.

[SIE2001b] SIEME. **Modélisation et simulation de l'environnement dans les systèmes multi-agents. Application aux robots footballeurs.** Disponível em: <http://www-poleia.lip6.fr/~magnin/these/rapport/index.html>. Acesso em: jun. 2001.

[SCH93a] SCHERER, K. R. Studying the emotion-antecedenet appraisal process: An expert system approach. **Cognition and Emotion**, London, v. 7, p. 1-141, 1993 .

[SCH93b] SCHERER, K. R. Studying the emotion-antecedenet appraisal process: An expert system approach. **Cognition and Emotion**, London, v. 7, p. 325-355, 1993.

[SIM67] SIMON, H.A. Motivational and emotional controls of cognition. In: MODELS OF THOUGHT, 1967. **Proceedings…** New Haven: Yale University Press, 1967. p. 29-38.

[SLO78] SLOMAN, A. **The Computer Revolution in Philosophy**. Hassocks, Sussex: Harvester Press and Humanities Press, 1978.

[SLO99] SLOMAN, A. Architectural Requirements for Human-like Agentes Both Natural and Artificial (What sorts of machines can love?). In: DAUTENHAHN, K. (Ed.). **Human Cognition and Social Agent Technology.** Amsterdam: John Benjamins, 1999.

[SLO2001] SLOMAN, A. Beyond Shallow Models of Emotions. In: COGNITIVE PROCESSING, 2001. **Proceedings…** [S.l. : s.n.], 2001. p. 177-198.

[STA2001] STALLER, A.; PETTA, P. Introducing Emotions into the Computational Study of Social Norms: A First Evaluation. **Journal of Artificial Societies and Social Simulation – JASSS**, London, v. 4, n.1, 2001.

[STI2000] STILL, G. K. **Crowd Dynamics**. 2000. Tese (Doutorado) - University of Warwick, Warwick.

[STR84] STRACK, J. **GPSS:** modelagem e simulação de sistemas**.** Rio de Janeiro: LTC, 1984. 174 p.

[SWA94] SWARM. **An overview of the Swarm simulation system.** Disponível em: <www.swarm.org>. Acesso em: abril 2001.

[SWA95] SWARM. **The Swarm Simulation System – A tool for studying complex systems**. Disponível em: <www.swarm.org>. Acesso em: abril 2001.

[SWA2001] SWARM. **Development Group**. Disponível em: <www.swarm.org>. Acesso em: jun. 2001.

[TAN97] TANAKA, K. **An introduction to fuzzy logic for practical applications.** New York : Springer-Verlag, 1997. 138 p.

[TEC98] TECUCI, G**. Building Intelligent Agents - An Aprrenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies**. London: Academic Press, 1998. 320 p.