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

FABIO YOSHIMITSU OKUYAMA

Descrição e Geração de Ambientes para Simulações com Sistemas Multiagentes

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

Prof. Dr. Rafael Heitor Bordini
Orientador

Porto Alegre, fevereiro de 2003

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Okuyama, Fabio Yoshimitsu

Descrição e Geração de Ambientes para Simulações com Sistemas Multiagentes / Fabio Yoshimitsu Okuyama. – Porto Alegre: Programa de Pós-Graduação em Computação, 2003.

119 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2003. Orientador: Rafael Heitor Bordini.

1. Inteligência artificial distribuída. 2. Sistemas multiagente. 3. Simulação social. 4. Comunicação entre agentes. 5. Ambientes em sistemas multiagente. I. Bordini, Rafael Heitor. II. Título.

AGRADECIMENTOS

Agradeço à minha família que me ajudaram e incentivaram na realização do curso de mestrado: Julio Okuyama, Maria Okuyama, Vilma Yuri Okuyama, Ricardo Massami Okuyama e Shirley Lika Okuyama.

Agradeço à família Aires (Alfredo, Adelaide e Hannah) por me acolherem como membro da família e a Cristian J. Salaine, Raquel da Silveira, Ben Hur e Fernanda Pimentel pelo companheirismo e amizade que ajudaram a obter equilíbrio que precisava para realizar este curso.

Agradeço aos colegas do laboratório 214 (Luciana F. Schroeder, Diana F. Adamatti, Luciana Foss, Guilherme Drehmer, Marcelo Azambuja, Daniel Basso e Rafael Jannone) pela boa convivência durante estes 2 anos e também por tirarem minhas dúvidas desde gramática à programação.

Agradeço aos colegas veteranos Eidy L. T. Guandeline, Cristiane R. Y. Mashuda e César Ossamu Ida por terem aberto o caminho e por me ajudarem na adaptação em uma cidade desconhecida.

Agradeço à Shinobu Takeuchi pelo carinho, paciência e o incentivo para a realização deste trabalho.

Gostaria de agradecer ao orientador, Dr. Rafael Heitor Bordini, pela orientação em todo o curso de mestrado e pelo incentivo e dedicação à orientação deste trabalho. À Denise de Oliveira pelo desenvolvimento da Simulação de Crescimento Urbano, utilizado como estudo de caso nesta dissertação.

Agradeço Marko Petek e ao prof. Dr. Cláudio Fernando Resin Geyer por permitirem e realizarem a transmissão via internet da defesa desta dissertação.

Gostaria de agradecer aos membros da banca, Dr. Antônio Carlos da Rocha Costa, Dr. Álvaro Freitas Moreira e Dra. Renata Vieira por permitirem a rápida realização da defesa de mestrado. Agradeço também a Dra. Ana Lúcia Certetich Bazzan por presidir a banca devido a impossibilidade da presença do orientador. Agradeço aos membros da banca pelas valiosas correções e sugestões a este trabalho.

Agradeço aos professores, técnicos e discentes do Instituto de informática que dão valor à este curso, fazendo com que valesse a pena todo o esforço envolvido na realização deste mestrado.

Agradeço à CAPES pelo auxílio financeiro à este trabalho, sem o qual este não seria possível.

Agradeço a todos que me ajudaram direta ou indiretamente na formação da pessoa que sou hoje. Apesar de ser uma lista de agradecimentos extensa, sou muito agradecido a todos.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

BDI	Belief-Desire-Intention
CC	Consumidor Comercial
CR	Consumidor Residencial
DTD	Documment Type Definition
ELMS	Enviroment description Language for Multiagent Simulations
FRC	Função de Revisão de Crenças
IA	Inteligência Artificial
IAD	Inteligência Artificial Distribuída
JVM	Java Virtual Machine
KQML	Knowledge Query and Manipulation Language
MABS	Multi-Agent Based Simulation
MAS	Multi-Agent System
PI	Produtor Imobiliário
PROPUR	Programa de Pós-Graduação em Planejamento Urbano e Rural
RDP	Resolução Distribuída de Problemas
SACI	Simple Agent Communication Infrastructure
SMA	Sistemas Multiagente
UFC	Unidade de Forma Construída
XML	eXtensible Markup Language

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho situa-se na área de Sistemas Multiagente, que é uma sub-área da Inteligência Artificial Distribuída. Em particular, o problema abordado nesta dissertação é o da modelagem de ambientes, um aspecto importante na criação de simulações baseadas em sociedades de agentes cognitivos, no entanto pouco tratado na literatura da área. A principal contribuição deste trabalho é a concepção de uma linguagem, chamada ELMS, própria para a definição de ambientes multiagente, e a implementação de um protótipo de interpretador para esta linguagem. O resultado da interpretação é um processo que simula o ambiente descrito em alto nível, e é apropriado para a interação com os agentes cognitivos que irão compartilhar o ambiente. Esta linguagem foi desenvolvida no contexto do projeto MASSOC, que tem como objetivo a criação de simulações sociais com agentes cognitivos. A abordagem deste projeto dá ênfase ao uso da arquitetura BDI para agentes cognitivos, à comunicação inter-agente de alto nível (ou seja, baseada em atos de fala) e à modelagem de ambientes com a linguagem ELMS, que é proposta neste trabalho. Os ambientes e agentes que podem ser usados na criação de simulações, bem como a comunicação entre eles utilizando a ferramenta SACI, são definidos ou gerenciados a partir de uma interface gráfica, que facilita a criação e controle de simulações com a plataforma MASSOC. Além de apresentar a linguagem ELMS e seu interpretador, esta dissertação menciona ainda, como breve estudo de caso, uma simulação de aspectos sociais do crescimento urbano. Esta simulação social auxiliou na concepção e avaliação da linguagem ELMS.

Palavras-chave: Inteligência artificial distribuída, sistemas multiagente, simulação social, comunicação entre agentes, ambientes em sistemas multiagente.

Description and Generation of Environments for Multi-Agent Based Simulations

ABSTRACT

This work is situated in the area of Multi-Agent Systems, a sub-area of Distributed Artificial Intelligence. In particular, the problem approached in this dissertation is that of environment modelling, which is an important aspect in the creation of simulations based on societies of cognitive agents, but one that is seldom discussed in the multi-agent systems literature. The main contribution of this work is the conception of a language, called ELMS, that is suitable for the definition of multi-agent environments, and the implementation of a prototype interpreter for that language. The result of such interpretation is a process that simulates the environment described at high-level, and that is appropriate for the interaction with the cognitive agents that will share the environment. The language presented here was developed in the context of the MASSOC project, which aims at the creation of social simulations with cognitive agents. The project's approach emphasizes the use of the BDI architecture for cognitive agents, the high-level inter-agent communication (i.e., based on speech acts), and environment modelling with ELMS, as proposed in this work. The environment and agents that can be used for creating simulations, as well as their communication using the SACI toolkit, are defined or managed through a graphical user interface, which facilitates the creation and control of simulations with MASSOC. Besides presenting the ELMS language and its interpreter, this dissertation also mentions, as a brief case study, a simulation of social aspects of urban growth. This social simulation was helpful in the conception and assessment of the ELMS language.

Keywords: distributed artificial intelligence, multi-agent systems, social simulation, agent communication, environments in multi-agent systems.

1 INTRODUÇÃO

A Inteligência Artificial Distribuída (IAD), segundo Weiss (1999, pág.1), é o estudo, construção e aplicação de sistemas multiagente, que são sistemas nos quais vários agentes inteligentes interagem para realizar um conjunto de objetivos ou tarefas. Segundo Gasser (1987), a IAD se divide basicamente em Sistemas Multiagente (SMA) e Resolução Distribuída de Problemas(RDP). Conforme esta classificação, RDP é o estudo de técnicas para divisão do trabalho para a resolução de um problema particular entre vários módulos que cooperam pela interação e compartilhamento de conhecimento sobre o problema e o desenvolvimento da solução, enquanto a pesquisa em SMA busca o comportamento inteligente em uma coleção de agentes inteligentes autônomos que podem trabalhar para um único objetivo global ou trabalhar em objetivos individuais que interagem. Em (BORDINI; VIEIRA; MOREIRA, 2001), os autores afirmam que o enfoque principal dos SMA é prover mecanismos para criação de sistemas computacionais a partir de entidade de software autônomas,denominadas agentes, que interagem através de um ambiente compartilhado por todos os agentes de uma sociedade e sobre o qual atuam, alterando seu estado. Como os agentes possuem um conjunto específico e limitado de capacidades, freqüentemente os agentes precisam interagir para atingir seus objetivos. Assim, é possível para os projetistas de sistemas computacionais a criação de sistemas computacionais complexos e dinâmicos de forma naturalmente distribuída e *bottom-up*.

A especificação de um SMA pode ser feita através da modelagem do ambiente, agentes, interações e organizações envolvidas. Um aspecto pouco explorado é a modelagem do ambiente, que é parte importante, principalmente para simulações sociais onde é necessário um ambiente virtual onde os agentes irão interagir. Desta forma, quando o SMA é um sistema computacional completo (não situado em um ambiente real), a modelagem do ambiente compartilhado entre os agentes é fundamental.

Este trabalho está inserido no contexto do projeto MASSOC, que será detalhado no Capítulo 6. Este projeto tem como objetivo o desenvolvimento de uma abordagem (e plataforma) para a criação de simulações baseadas em SMA que incorporam tecnologias para especificação e execução de agentes cognitivos. Porém, faltavam meios para a especificação do ambiente onde os agentes estariam situados nesse SMA. Quando um SMA é um sistema totalmente computacional, isto é não interage com o mundo real, a definição do ambiente é uma importante parte da modelagem de SMA que não é explicitada na literatura. A modelagem de ambientes possui características que diferem da especificação de agentes. Por este motivo era necessário uma linguagem específica para a especificação de ambientes. Esta foi a motivação para o desenvolvimento da linguagem ELMS. A linguagem foi criada para auxiliar a descrição de ambientes para simulações multiagente, incluindo também uma representação "física" para os agentes dentro destes ambientes.

1.1 Objetivos

Esta dissertação visa propor uma linguagem de alto nível na qual o ambiente a ser compartilhado pelos agentes de um SMA possa ser descrito. Esta linguagem deve abranger diversos aspectos envolvidos na modelagem de ambientes para SMA, tais como as propriedades dos objetos que são alteradas pelas ações dos agentes, as propriedades que são perceptíveis a cada agente, que ações são permitidas a cada agente e sob que condições estas podem ser executadas. Complementando este trabalho de elaboração da linguagem para definição de ambientes, foi implementado um protótipo de um programa que, a partir da interpretação de um ambiente especificado nesta linguagem, gera um processo que simula o ambiente definido. Ou seja, foi implementado, para a linguagem definida, um gerador de ambientes, que são processos que interagem com os agentes, provendo-lhes percepções e executando suas ações básicas.

Inicialmente, esta dissertação apresenta uma caracterização dos sistemas multiagente, de maneira mais específica os SMA cognitivos e agentes baseados na arquitetura BDI. Serão apresentados aspectos como comunicação entre agentes, modelagem de ambientes para SMA, organizações, alguns aspectos de simulação social baseada em agentes e o projeto MASSOC (capítulo 6). Estes assuntos são abordados para que se possa ter uma visão do ELMS dentro do contexto onde este está inserido.

Além disto, é apresentado um breve estudo de caso, que ajudou na concepção da linguagem ELMS tanto apontando falhas na linguagem (que foram corrigidas ao longo do tempo), quanto mostrando em que aspectos a expressividade da linguagem deveria ser aumentada para facilitar a elaboração do estudo de caso. Este estudo de caso foi uma simulação social sobre o crescimento urbano elaborada no grupo de pesquisa do projeto MASSOC (OLIVEIRA, 2002). Esta é uma simulação relativamente sofisticada, e portanto um bom parâmetro para avaliar a adequação do ELMS.

1.2 Estrutura

Esta dissertação encontra-se estruturada conforme descrito a seguir.

No capítulo 2 é apresentada uma visão geral de agentes e arquiteturas de agentes. Será enfatizada a arquitetura de agentes BDI (*Belief-Desire-Intention*) de maneira prática, evitando seus aspectos formais e as questões de âmbito filosófico. É apresentada também a linguagem de programação orientada a agentes AgentSpeak(XL), que permite a especificação do comportamento de agentes BDI.

No capítulo 3 é apresentada uma visão geral dos Sistemas Multiagentes. Neste capítulo são discutidos alguns aspectos da simulação social baseada em sistemas multiagente, que é uma das áreas onde os sistemas multiagente cognitivos têm grande importância, e uma das áreas de aplicação pretendidas para a plataforma MASSOC.

No capítulo 4 são abordados a comunicação entre agentes, tratando a comunicação como meio de interação entre agentes cognitivos; aspectos da linguagem KQML (FININ, T. et al., 1994b; FININ; FRITZSON; MCKAY, 1992; FININ, T. et al., 1994a; MAYFIELD; LABROU; FININ, 1996). Serão vistas também algumas características do SACI (HÜBNER; SICHMAN, 2000), ferramenta que possibilita a comunicação entre agentes e controle de agentes distribuídos.

No capítulo 5 são abordados os ambientes multiagente e o ELMS, a linguagem proposta neste trabalho com o objetivo de ser utilizada no projeto MASSOC para a descrição e criação de ambientes para simulações multiagente.

No capítulo 6 é apresentada a abordagem MASSOC para criação de simulações baseadas em agentes. Será apresentado um estudo de caso onde foi utilizada a plataforma MASSOC. Trata-se de uma simulação social cujo objetivo é estudar aspectos sociais do crescimento urbano, como por exemplo o papel das classes sociais neste processo.

No capitulo 7 são apresentadas algumas conclusões da pesquisa descrita nesta dissertação.

2 AGENTES

Apesar do grande uso do termo e da quantidade de artigos sobre agentes, não existe um consenso à respeito do significado do termo agente. Existem diversas visões e concepções de agentes. O termo "agente" tem sido usado de maneiras tão diversas que não tem um significado claro se não for associado a uma concepção específica de agência. Em um extremo, agentes são considerados entidades cognitivas munidas, assim como humanos, de percepções, motivações e sentimentos, por exemplo. No outro extremo, agentes são meramente autômatos e comportam-se exatamente como foram programados (HUHNS; SINGH, 1998). Os conceitos mais genéricos seriam que agentes são entidades que percebem um ambiente através de sensores e agem no mesmo através de efetuadores (RUSSEL; NORVIG, 1995). Um conceito também genérico seria que agentes são sistemas computacionais situados em um ambiente, e que são capazes de ações autônomas neste ambiente para realizar seus objetivos (WOOLDRIDGE, 1999). Em Shoham (1993), afirma-se que um agente é uma entidade cujo estado é visto como sendo constituído de componentes tais como crenças, capacidades, escolhas e compromissos. Nesta concepção, o que torna um hardware ou software um agente é o fato do projetista ter escolhido analisá-lo e controlá-lo este através destes termos mentais.

A figura 2.1,traduzida de Wooldridge (1999), apresenta um modelo abstrato de agente. Na maioria dos casos, o agente faz parte do ambiente no qual interage recebendo percepções e executando ações.

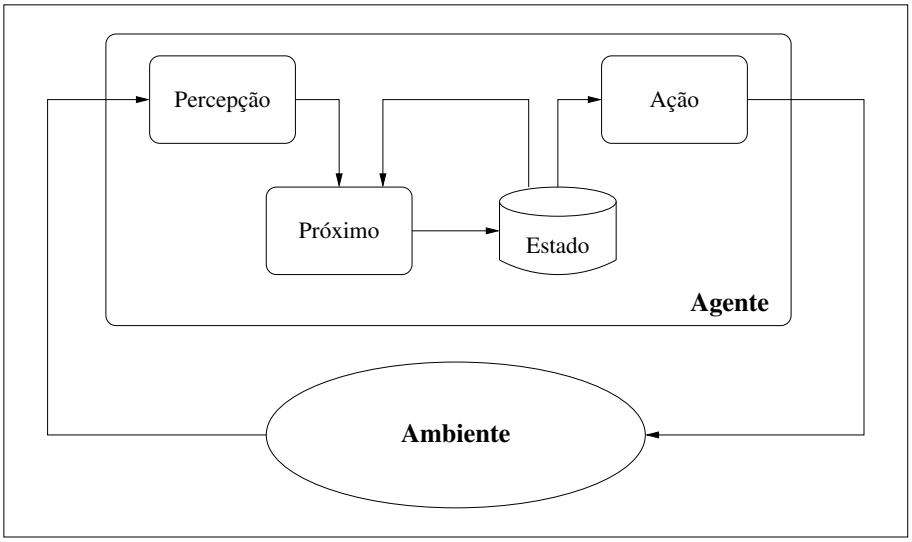

Figura 2.1: Modelo Geral de Agente

Segundo Wooldridge e Jennnings (1995) para um sistema ser considerado um agente ele deveria apresentar as seguintes características:

- **autonomia:** operação sem a intervenção direta de humanos ou outros agentes, e capacidade de controlar seu estado interno;
- **habilidade social:** interação com outros agentes ou humanos através de algum tipo de linguagem de comunicação entre agentes;
- **reatividade:** agentes percebem o ambiente e respondem rapidamente a mudanças que ocorrem;
- **pró-atividade:** agentes devem ser capazes de tomar inciativas para atingir um objetivo.

Mesmo nas características tratadas nos conceito citados de agentes existem divergências. Por exemplo, de acordo com Luck e d'Inverno (1995) para que um agente apresente autonomia, este deve ter um conjunto de motivações, onde motivação seria qualquer desejo ou preferência que pode levar a geração e a adoção de objetivos que afetam a saída do raciocínio ou comportamento para satisfação dos objetivos. Outras características algumas vezes encontradas nas definições de agentes seriam:

- **capacidade de representação:** o agente deve possuir uma representação simbólica do que acredita ser verdade no ambiente no qual está inserido;
- **capacidade de deliberação:** dada uma motivação e o estado do ambiente, o agente deve ser capaz de decidir quais objetivos serão seguidos por ele (BORDINI; VIEIRA; MOREIRA, 2001);
- **mobilidade:** capacidade de movimentação através de uma rede;
- **veracidade:** assume-se que os agentes não irão comunicar informações falsas propositalmente;
- **benevolência:** assume-se que se os agentes não possuem objetivos conflitantes, eles tentarão fazer o que lhes foi pedido;
- **racionalidade:** assume-se que um agente irá agir para alcançar seus objetivos, e não irá agir de maneira a prevenir que seu objetivo seja alcançado (WOOLDRIDGE; JENNINGS, 1995).

Wooldridge e Jennings (1995) afirmam que, para determinados grupos de pesquisadores, que trabalham com IA, o conceito de agentes teria um conceito mais forte e mais específico do que os já citados. Para estes, agentes seriam sistemas computacionais que além das propriedades citadas, seriam concebidos ou implementados usando conceitos normalmente aplicados a seres humanos. Porém Shoham (1993) afirma que é possível associar qualidades mentais para qualquer tipo de máquina. Contudo, isto seria útil apenas quando a associação ajudasse a compreensão da estrutura da máquina, seu comportamento passado ou futuro, ou como corrigir ou melhorá-la.

2.1 Agentes e Objetos

Um sistema baseado em agentes é um sistema onde a principal abstração utilizada é a de agente (JENNINGS; SYCARA; WOOLDRIDGE, 1998). Desta forma, um sistema baseado em agentes não precisa ser implementado com estruturas de software que correspondem a agentes, da mesma forma que um programa pode ser projetado utilizando orientação a objetos e ser implementado em uma linguagem funcional. Como vários dos programas orientados a agentes são implementados em linguagens orientadas a objetos, alguns tendem a confundir os termos e as diferenças conceituais.

Apesar da existência de similaridades, existem grandes diferenças entre agentes e objetos, entre estas se destaca a autonomia. Na programação orientada a objeto tem-se a noção de encapsulamento, no qual os objetos teriam controle sobre seu estado interno. Em linguagens como JAVA e Object Pascal, faz parte da boa prática de programação declarar as variáveis como sendo privadas, sendo as privadas acessíveis apenas pelo próprio objeto. Desta forma, segundo Wooldridge (1999), poderia se dizer que um objeto teria autonomia sobre seu estado, mas não apresenta autonomia sobre seu comportamento, pois um objeto precisa disponibilizar métodos para outros objetos que podem invocá-los a qualquer momento, desta forma o objeto não tem controle sobre a execução de seus métodos. Outras diferenças seriam as características tais como reatividade, pró-atividade e habilidade social, características não previstas na idéia básica de orientação a objetos (WOOLDRIDGE, 1999). Além disso um agente deve ser concebido como tendo sua própria linha de execução (*thread*) enquanto que no modelo orientado a objetos o sistema tem uma linha de execução, sendo múltiplas *threads* algo opcional.

2.2 Arquiteturas de Agentes

Assim como definições para agentes, existem inúmeras arquiteturas para implementação e modelagem de agentes. Em (WOOLDRIDGE, 1999) as arquiteturas de agentes são separadas em quatro grupos:

- Agentes baseados em lógica: nos quais a tomada de decisões é feita através de deduções lógicas.
- Agentes reativos: a tomada de decisão é implementada na forma de um mapeamento de situações e ações.
- Agentes Belief-Desire-Intention (BDI): onde a tomada de decisão depende da manipulação de estruturas de dados representando crenças, desejos e intenções. Esta arquitetura será mais detalhada na seção 2.3.
- Arquiteturas de camadas: onde a tomada de decisões é feita em várias camadas de software, em que cada camada analisa o ambiente em diferentes níveis de abstração.

Agentes baseados em lógica Esta seria a abordagem que tem um maior embasamento na IA simbólica. *"Ela sugere que um comportamento inteligente pode ser gerado em um sistema através de uma representação simbólica do ambiente, do comportamento desejado para o agente e a manipulação sintática desta representação"* (WOOLDRIDGE, 1999). Nestes agentes o seu estado interno poderia ser projetado como um base de fórmulas da lógica de primeira ordem. O banco de dados seria o conjunto de informações que o agente tem sobre o ambiente.

Arquiteturas Reativas As arquiteturas reativas tiveram origem na busca por alternativas ao uso da IA simbólica, visto o problema da intratabilidade de alguns problemas através desta. De maneira genérica, uma arquitetura reativa seria uma na qual não existe uma representação simbólica centralizada de um modelo de mundo, e que não faz uso de raciocínio simbólico complexo. Serão abordadas mais características dos agentes reativos quando forem abordados os sistemas multiagente reativos.

Arquiteturas em camadas Baseados na idéia de decompor o comportamento em reativo e pró-ativo, separando-os em subsistemas, ou camadas de software, as arquiteturas de camadas apresentam uma hierarquia entre camadas que interagem. A organização das camadas pode ser horizontal ou vertical. Na organização horizontal, todas as camadas tem acesso a entrada sensorial e a saída de ações. Na organização vertical, apenas uma das camadas tem acesso as informações sensoriais, que é processada e passada para as camadas superiores, até chegar a camada que tem o controle sobre a saída de ações. Na figura 2.2 são mostradas as arquiteturas de camadas horizontal e vertical, traduzida de (JENNINGS; SYCARA; WOOLDRIDGE, 1998).

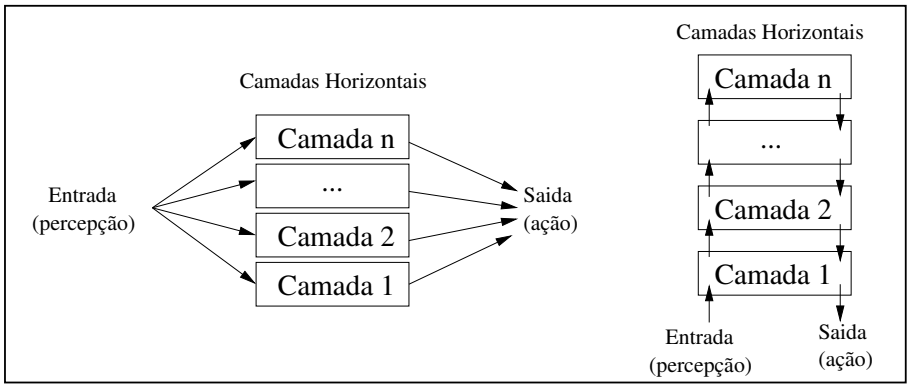

Figura 2.2: Arquitetura de Camadas

2.3 Agentes BDI

A arquitetura BDI tem suas origens no estudo de *atitudes mentais*, em particular crenças, desejos e intenções que representam, respectivamente, estados informativos motivacionais e deliberativos de um agente. Estas atitudes mentais determinam diretamente o comportamento do sistema e são cruciais para se conseguir desempenhos adequados em situações onde a deliberação é limitada pelos recursos (BAILEY; GEORGEFF; KEMP; KINNY, 1995). As crenças seriam as informações que os agentes têm sobre o ambiente, os desejos seriam os mundos possíveis com os quais o agente pode escolher se comprometer a tornar realidade, e intenções representam estados que o agente pretende alcançar e que tem recursos comprometidos para este fim, conforme afirmado em (JENNINGS; SYCARA; WOOLDRIDGE, 1998).

Os fundamentos filosóficos da arquitetura BDI podem ser encontrados em (BRATMAN; ISRAEL; POLLACK, 1988) onde são discutidos aspectos como a necessidade de compromisso em relação as intenções, questões sobre o projeto de agentes BDI e um modelo abstrato para um interpretador de agentes BDI. A arquitetura BDI tem grande aceitação no meio científico, porém alguns pesquisadores da teoria da decisão clássica e

planejamento questionam a necessidade de todas as três atitudes, enquanto pesquisadores de sociologia e IAD (Inteligência Artificial Distribuída) questionam a adequação de apenas três atitudes (RAO; GEORGEFF, 1995).

O processo de raciocínio prático de um agente BDI, segundo Wooldridge (1999), seria como mostra a figura 2.3 adaptada de Wooldridge (1999).

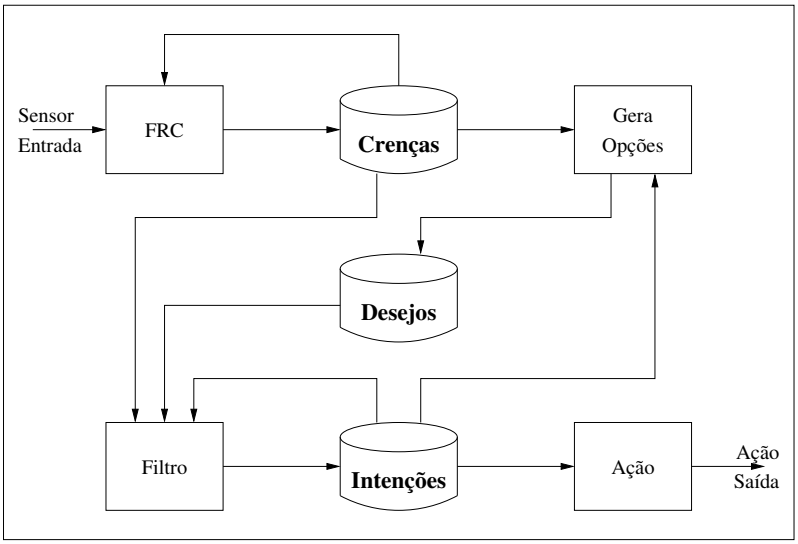

Figura 2.3: Arquitetura BDI Genérica

Os componentes mostrados na figura são:

- Crenças: um conjunto de crenças, nas quais estão representados os fatos que o agente acredita ser verdade no ambiente corrente.

- Função de Revisão de Crenças (FRC): o revisor de crenças tem como objetivo captar a percepção dos sensores, analisá-las em conjunto com as crenças atuais e gerar um novo conjunto de crenças consistentes.

- Gerador de opções: a função de geração de opções determina quais opções (de estados de mundo atingíveis) estão disponíveis para o agente com base em suas crenças atuais e intenções existentes.

- Desejos: Conjunto atual de desejos do agente, representa os estados de mundo que o agente deseja atingir em algum momento não específico. Inclui os desejos não atingidos e os desejos criados para atender outros desejos.

- Filtro: o filtro representa o processo de deliberação do agente, que determina as intenções do agente com base em suas crenças, desejos e intenções anteriores.

- Intenções: um conjunto de intenções, estados de mundo que o agente deseja tornar verdadeiros, com as quais o agente está comprometido a tentar atingir (através de suas ações no ambiente).

- Seletor de ação: determina a ação a ser executada com base nas intenções correntes.

2.3.1 Crenças (*Beliefs*)

Um conjunto de crenças de um agente contém sua informação atual sobre o estado de mundo e também, possivelmente, alguns aspectos de seu estado interno. Crenças são tipicamente observações sobre o ambiente ou conclusões derivadas pelo sistema através destas observações, que podem mudar com o passar do tempo (BAILEY et al., 1995). Em um sistema multiagente um agente pode precisar manter várias crenças sobre outros agentes, incluindo crenças sobre os estados mentais destes agentes.

O conhecimento contido no conjunto de crenças é normalmente representado como predicados da lógica de primeira ordem. Crenças sobre estados internos de um sistema são chamadas crenças de meta-nível. Estas tipicamente descrevem as crenças, objetivos e intenções do sistema, assim como outras importantes informações de controle. Por exemplo, uma crença de que uma intenção em particular foi suspensa pode ser representada como:

estadoIntencao(intencao123, suspensa)

2.3.2 Objetivos (*Goals*)

Na literatura de filosofia, desejos podem ser inconsistentes e o agente pode não saber os meios para atingir estes desejos. Desejos têm a tendência de "puxar" o agente para diferentes direções. Os desejos são entradas no processo de deliberação do agente que tem como saída um conjunto de desejos que são consistentes e atingíveis (SINGH; RAO; GEORGEFF, 1999), geralmente denominados *objetivos* (*goals*). Um subconjunto de desejos consistentes e atingíveis são chamados de objetivos. Uma simplificação feita nas implementações de interpretadores BDI é a substituição dos desejos por objetivos.

Objetivos são expressos como condições a serem atingidas sobre uma seqüência de estados de mundo e são descritos com a aplicação de operadores temporais à descrições de estado. Isto permite a representação de uma grande variedade de objetivos, incluindo objetivos a serem atingidos, testados, mantidos e a espera por determinadas condições (BAILEY et al., 1995). Por exemplo, um objetivo é testar se a pressão de um tanque é menor que 200 psi, outro seria aguardar a pressão tornar-se maior que 285 psi, e um terceiro seria fazer com que a pressão torne-se igual a 350 psi. Estes objetivos poderiam ser representados da seguinte forma:

teste $pressao(tanque, X)$ **and** $X < 200$

aguarde $pressao(tanque, X)$ **and** $X > 285$

atinja $pressao(tanque, X)$ **and** $X < 350$

A descrição de objetivos não é restrita à especificação de comportamentos desejados do ambiente externo, mas também pode caracterizar o estado interno ou comportamento do agente. Tais descrições são chamadas especificação de objetivos de meta-nível (BAILEY et al., 1995).

2.3.3 Planos

Os planos de um agente são parte de sua base de conhecimentos que descrevem como o agente deve se comportar quando certos fatos são adicionados à sua base de crenças ou quando novos objetivos são instanciados (BAILEY et al., 1995). Estes novos fatos podem chegar devido à eventos no ambiente sendo diretamente percebido por sensores ou sendo

inferimos pelo agente. Novos objetivos podem ser fornecidos por um usuário ou serem sub-objetivos (*subgoals*) de objetivos já existentes.

Cada plano consiste de uma condição de invocação, que especifica em quais eventos o plano deve ser ativado, um contexto, que especifica em quais condições o plano se aplica, e um corpo, que especifica os passos do procedimento a ser executado.

A condição de invocação é uma expressão de evento que descreve o que precisa ocorrer para que a execução do plano seja considerada.

O contexto é uma expressão de estado que descreve as crenças que devem estar presentes para que o plano seja executado. Como esta expressão poderá conter variáveis livres, de maneira que a condição possa ser satisfeita de mais de uma maneira, caberá ao interpretador selecionar a instância do plano a ser executada.

O corpo de um plano pode ser representado por um grafo que tem um nodo inicial e possivelmente múltiplos nodos finais. Os arcos no grafo são rotulados com ações a serem realizadas ou objetivos parciais a serem atingidos na execução do plano. Para que um plano seja executado com sucesso, todos os objetivos parciais e ações que nomeiam os arcos do nodo inicial até um nodo final devem ser realizados (KINNY; GEORGEFF, 1996).

Os planos de um agente típico não é constituído apenas de conhecimento procedural a respeito de um domínio específico, mas também incluem *metalevel plans* que são conhecimento sobre a manipulação das crenças desejos e intenções de um agente.

2.3.4 Estrutura de Intenções

A estrutura de intenções contém todas as tarefas, chamadas intenções, que o sistema escolheu para a execução, imediata ou posteriormente. Uma intenção é uma instância de um plano e todos os sub-planos que foram instanciados para realizar o plano.

A estrutura de intenções captura, de maneira conceitual, as decisões que um agente tomou sobre os meios para a realização de um objetivo (BAILEY et al., 1995). Sendo uma intenção um processo algumas vezes de longa duração, o agente se compromete a realizar a alcançar estes objetivos. Quando as ações falham ou o ambiente se altera de maneira inesperada, o agente precisa reconsiderar estes compromissos.

O balanço correto entre compromissos e reconsiderações é um aspecto importante no projeto do agente, que depende sensivelmente do ambiente onde o agente existe. A reconsideração da escolha de uma ação a cada passo é potencialmente muito custoso, por outro lado, o compromisso incondicional com um curso de ação pode resultar na falha do sistema em cumprir objetivos (RAO; GEORGEFF, 1995).

2.4 AgentSpeak(XL)

O AgentSpeak(XL) é uma extensão à linguagem AgentSpeak(L) apresentada em (BORDINI et al., 2002). AgentSpeak(L) é uma linguagem de programação para definição do comportamento de um agente BDI, apresentada por Rao em (RAO, 1996). Esta linguagem de alto nível tem como objetivo facilitar a aplicação das teorias BDI, visto a complexidade das lógicas modais BDI.

2.4.1 AgentSpeak(L)

Em Rao (1996), o autor apresenta a linguagem e a operação de um interpretador abstrato para esta. A linguagem foi mais tarde formalizada em (D'INVERNO; LUCK, 1998) na linguagem de especificação formal Z. Porém somente em (MACHADO; BOR-

DINI, 2001) foi apresentado um primeiro protótipo de um interpretador para a linguagem. Outros trabalhos como (MOREIRA; BORDINI, 2002; BORDINI; MOREIRA, 2002) também tratam da formalização e validação da linguagem.

A seguir é apresentado a sintaxe básica da linguagem AgentSpeak(L), resumida de (BORDINI et al., 2002). Para uma descrição mais completa veja (D'INVERNO; LUCK, 1998; MOREIRA; BORDINI, 2002).

O comportamento de um agente definido em AgentSpeak(L), é descrito através da especificação de um conjunto de crenças iniciais e um conjunto de planos. As definições a seguir introduzem as noções necessárias para a especificação de tais conjuntos. Existem várias similaridades com a sintaxe do Prolog, incluindo a convenção de uso de iniciais maiúsculas para os nomes de variáveis, e aspectos relacionados à forma dos predicados e unificação.

Um *átomo de crença* é simplesmente um predicado em sua notação usual, e átomos de crença ou suas negações são *literais de crença*. O conjunto inicial de crenças é apenas uma coleção de átomos de crença que não dependem de variáveis.

No AgentSpeak(L) é possível definir dois tipos de objetivos (*goals*): objetivos de realização (*achievement goals*) e objetivos de teste (*test goal*). Objetivos de realização são predicados, assim como as crenças, precedidos com o operador '!' enquanto os objetivos de teste são precedidos do operador '?'. Objetivos de realização declaram que o agente quer alcançar um estado de mundo onde o predicado associado é verdadeiro. Objetivos de teste declaram que o agente quer testar se o predicado associado é verdadeiro, isto é, se este pode ser unificado com a base de crenças do agente.

Os eventos disparadores (*triggering events*) definem quais eventos podem iniciar a execução de planos. No AgentSpeak(L) são definidos dois tipos de eventos: aqueles relacionados à adição, precedidos pelo operador '+', e aqueles relacionados à remoção, precedidos pelo operador '-', de atitudes mentais(crenças ou objetivos).

De acordo com o modelo de agente, apresentado anteriormente na figura 2.1 de Wooldridge (1999), para realizar ações no ambiente, os planos devem fazer referências à *ações básicas* que o agente é capaz de realizar no ambiente onde está inserido. Estas ações são referenciadas como predicados, apenas que são símbolos predicativos especiais utilizados para este propósito.

Um plano AgentSpeak(L) possui um cabeçalho que é formado por um evento disparador que denota o propósito do plano e uma conjunção de literais de crença que forma um contexto que deve ser satisfeito para a execução do plano (o contexto deve ser conseqüência lógica da base de crenças do agente). Um plano também possui um corpo, que é formado por uma seqüência de ações básicas, objetivos e sub-objetivos de realização ou teste.

Definição de Plano: se e é um evento disparador, $b_1, \ldots, b_m$ são literais de crença, e $h_1, \ldots, h_n$ são objetivos ou ações, então e `:` b_1 `&` $\ldots$ `&` b_m `<-` h_1 `;` $\ldots$ `;` h_n`.` é um *plano*.

A expressão à esquerda da seta é referida como o *cabeçalho* do plano e a expressão à direita da seta é referida como o *corpo* do plano. A expressão à direita da ponto no cabeçalho do plano é referido como *contexto*. O corpo de um plano ou um contexto vazio é representado pela expressão "$true$".

Definição de Agente: Um agente AgentSpeak(L) é definido pela tupla $\langle E, B, P, I, A, \mathcal{S}_\mathcal{E}, \mathcal{S}_\mathcal{O}, \mathcal{S}_\mathcal{I} \rangle$, onde E é um conjunto de eventos, B é um conjunto de crenças iniciais, P é um conjunto de planos, I é um conjunto de intenções

e A é um conjunto de ações. A função de seleção $\mathcal{S}_\mathcal{E}$ seleciona um evento do conjunto E; a função de seleção $\mathcal{S}_\mathcal{O}$ seleciona uma opção ou um plano aplicável de um conjunto de planos aplicáveis; e $\mathcal{S}_\mathcal{I}$ seleciona uma intenção do conjunto I.

Intenções são cursos de ações com as quais o agente se comprometeu a fim de realizar algum objetivo específico. No interpretador, cada *intenção* é uma pilha de *planos parcialmente instanciados*, ou seja, planos onde algumas variáveis foram instanciadas. Um evento, que pode iniciar a execução de planos, pode ser *externo*, quando originado da percepção do ambiente, ou *interno* quando gerado pela própria execução de um plano (a adição ou remoção de um objetivo ou crença dentro do corpo de um plano). Planos escolhidos por eventos internos serão empilhados sobre a intenção que os gerou. Enquanto planos escolhidos por eventos externos geram uma nova intenção em I, representando diferentes focos de atenção do agente no ambiente. As funções de seleção ($\mathcal{S}_\mathcal{E}, \mathcal{S}_\mathcal{O}, \mathcal{S}_\mathcal{I}$) que são consideradas partes específicas de cada agente, apesar de não terem sido definidas[1] nas especificações da linguagem, são parte importante de um interpretador AgentSpeak(L).

A figura 2.4, traduzida de Machado e Bordini (2001), descreve o interpretador é de grande ajuda para a compreensão do interpretador AgentSpeak(L) proposto por Rao. Nesta figura, conjuntos (de crenças, planos, eventos, planos e intenções) são representados por retângulos; círculos representam processamento envolvido; e losangos representam a seleção de um elemento em um conjunto.

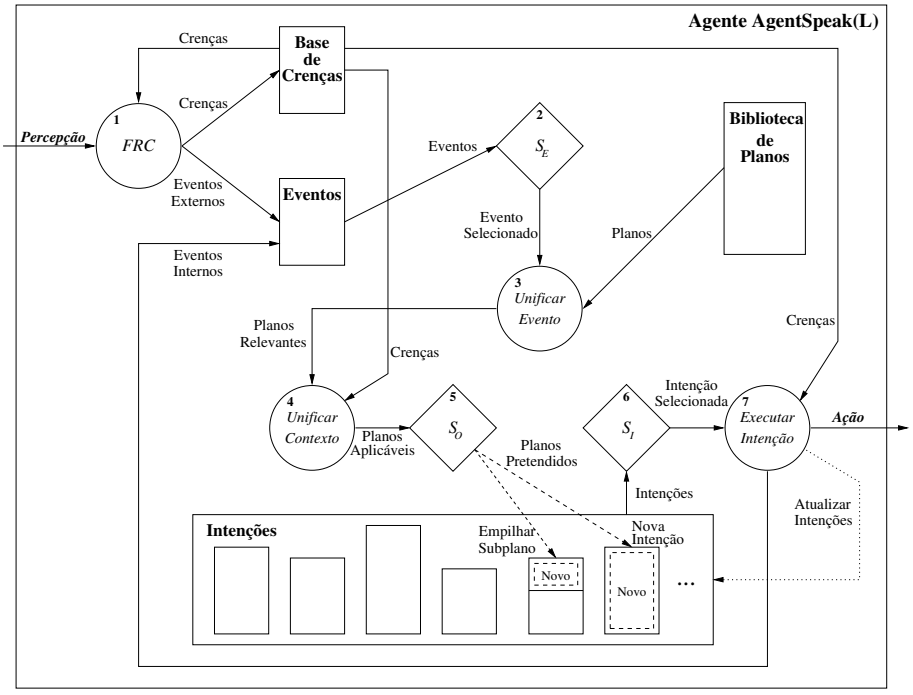

Figura 2.4: Interpretação de Programa AgentSpeak(L)

A cada ciclo de interpretação, a lista de eventos é atualizada, que podem ser gerados através da percepção ambiente ou pela execução de intenções. Note que foi introduzido uma Função de Revisão de Crenças (FRC) no que é implícita no interpretador de Rao, mas que normalmente é explícita na arquitetura BDI genérica, como visto na figura 2.3. Isto assume que as crenças são atualizadas a partir das percepções e mudanças nas crenças dos agentes, o que implica na inserção de eventos no conjunto de eventos.

[1] Com exceção do $\mathcal{S}_\mathcal{I}$ que é abordado na linguagem estendida AgentSpeak(XL).

Como visto anteriormente, agente AgentSpeak(L) é definido através de crenças iniciais e planos, porém $\mathcal{S}_{\mathcal{E}}$,$\mathcal{S}_{\mathcal{O}}$ e $\mathcal{S}_{\mathcal{I}}$ são parte da definição do agente que deveriam ser selecionados na execução do interpretador. Após o $\mathcal{S}_{\mathcal{E}}$ selecionar um evento, o interpretador tem que unificar este evento com os eventos disparadores dos cabeçalhos dos planos, gerando uma conjunto de todos os *planos relevantes*. Quando unificado o contexto do cabeçalho dos planos com a base de crenças, é determinado o conjunto de *planos aplicáveis* (planos que podem ser utilizados para o tratamento do evento). Então o $\mathcal{S}_{\mathcal{O}}$ seleciona um único plano aplicável e empilha o plano na intenção que gerou o evento, caso seja um evento interno, ou cria uma nova pilha de intenção, caso seja um evento externo. A seguir, o $\mathcal{S}_{\mathcal{I}}$ seleciona uma das intenções do agente e pega o comando que está no corpo do plano. Este comando poderá ser uma ação básica, ação do agente sobre o ambiente, ou a geração de um evento interno. O comando será executado, com todas as operações e atualizações envolvidas, finalizando um ciclo de interpretação AgentSpeak(L).

2.4.2 Extensões ao AgentSpeak(L)

AgentSpeak(XL) estende o AgentSpeak(L) em diversas maneiras, tal como o tratamento de falhas de planos, adição e remoção de crenças no corpo do plano, uma nova forma de construção chamada *ações internas* que permite a extensão da linguagem. Outra característica da linguagem estendida é a possibilidade de expressar as relações entre os planos, assim como um critério quantitativo para sua execução. Isto aumentou de forma considerável a expressividade da linguagem, facilitando a programação de determinados tipos de aplicação onde o raciocínio quantitativo ou priorização de tarefas são necessárias. Isto permitiu um maior controle sobre as intenções do agente, o que não era possível no AgentSpeak(L), a não ser que fosse implementada uma função de seleção de intenções específica. Para maiores detalhes sobre as extensões, veja (BORDINI et al., 2002).

3 SISTEMAS MULTIAGENTES

Os Sistemas Multiagente baseiam-se na idéia de que um comportamento inteligente seria possível através das interações de seus múltiplos agentes. Apesar de um sistema de agentes distribuídos poder ter um equivalente centralizado, que otimizado pode ser mais eficiente que um distribuído, os sistemas distribuídos são, na maioria das vezes, muito mais simples para a compreensão e fáceis de desenvolver (HUHNS; LARRY; STEPHENS, 1999), especialmente quando a resolução do problema é de natureza distribuída.

Ao contrário das abordagens da IA tradicional, onde a metáfora da inteligência é baseada em um *comportamento individual humano* com ênfase na representação de conhecimentos e métodos de inferência, a metáfora usada na IAD é baseada em *comportamento social* com ênfase nas ações e interações. Um comportamento social "inteligente" pode surgir de membros "inteligentes" da sociedade, chamados agentes cognitivos, ou de membros "não-inteligentes" da sociedade, chamados agentes reativos (SICHMAN; DEMAZEAU; BOISSIER, 1992).

Os problemas tratados em SMA, diferentemente da RDP, tratam de agentes que trabalham autonomamente e cada qual em seus objetivos particulares. Estes objetivos podem interagir, complementando-se ou contrapondo-se. Porém mesmo que não haja interação direta entre seus objetivos, existirá alguma forma de interação entre os agentes, visto que estes encontram-se em um ambiente compartilhado.

Algumas das razões para o crescente interesse em SMA seriam, segundo (JENNINGS; SYCARA; WOOLDRIDGE, 1998):

- capacidade de prover robustez e eficiência;
- possibilitar interoperação com sistemas legados (antigos);
- capacidade de resolução de problemas onde dados, conhecimento e controle podem estar distribuídos.

Na área de SMA, busca-se criar sistemas onde diversos agentes possam interagir para que cada agente atinja seus objetivos. Em (BORDINI; VIEIRA; MOREIRA, 2001), os autores afirmam que o enfoque principal dos SMA é prover mecanismos para criação de sistemas computacionais a partir de entidade de software autônomas, denominadas agentes, que interagem através de um ambiente compartilhado por todos os agentes de uma sociedade e sobre o qual atuam, alterando seu estado. Como os agentes possuem um conjunto específico e limitado de capacidades, freqüentemente os agentes precisam interagir para atingirem seus objetivos. Assim, é possível para os projetistas de sistemas computacionais a criação de sistemas computacionais de forma naturalmente distribuída e *bottom-up*.

3.1 Sistemas Multiagente Reativos

Os SMA reativos baseiam-se na idéia de que o comportamento inteligente deve emergir da interação de diversos agentes simples. Normalmente, um sistema reativo apresenta um grande número de agentes, podendo ser baseados no comportamento de colônias de insetos. Algumas das principais características dos SMA reativos, citados em (ÁLVARES; SICHMAN, 1997) são:

- Não há representação explícita de conhecimento: o conhecimento do agente é implícito e se manifesta através de seu comportamento;
- Não há representação do ambiente: o seu comportamento se baseia no que é percebido a cada instante no ambiente mas sem uma representação explícita dele;
- Não há memória das ações: os agentes reativos não mantém um histórico de suas ações, de forma que uma ação passada não exerce nenhuma influência sobre suas ações futuras, excetuando as mudanças percebidas no ambiente causadas por ações passadas sobre o mesmo;
- Organização etológica: a forma de organização dos agentes reativos é similar a sociedades de insetos, mais simples se comparada à organização social dos sistemas cognitivos;
- Grande número de membros: os SMA reativos têm, em geral, um grande número de agentes, da ordem de dezenas, centenas ou mesmo milhões de agentes.

3.2 Sistemas Multiagente Cognitivos

Nos SMA cognitivos o comportamento inteligente também deve ter origem nas interações dos agentes, porém devido a maior complexidade destes agentes e o fato de que os SMA cognitivos agrupam um menor número de agentes, há uma maior responsabilidade do agente em originar o comportamento inteligente. No desenvolvimento destes, utiliza-se uma noção mais forte de agência, onde os agentes apresentam de maneira mais clara características como autonomia e capacidades de deliberação. As características dos SMA apresentadas em (JENNINGS; SYCARA; WOOLDRIDGE, 1998) são:

- cada agente tem capacidades e informações incompletas para solução do problema, de forma que cada agente tem um ponto de vista limitado do problema;
- não há um sistema de controle global;
- os dados são descentralizados;
- a computação é assíncrona.

Segundo Demazeau (1995), os componentes básicos de um SMA seriam os agentes, as interações, o ambiente e as organizações. Neste modelo os agentes poderiam ser desde simples autômatos a complexos sistemas baseados em conhecimento. O ambiente seria dependente do domínio da aplicação, porém na maioria dos casos seriam ambientes espaciais. As estruturas de interação compreenderiam desde interações físicas até os atos de fala. Ainda nesta visão, as organizações poderiam ser baseadas em estudos biológicos

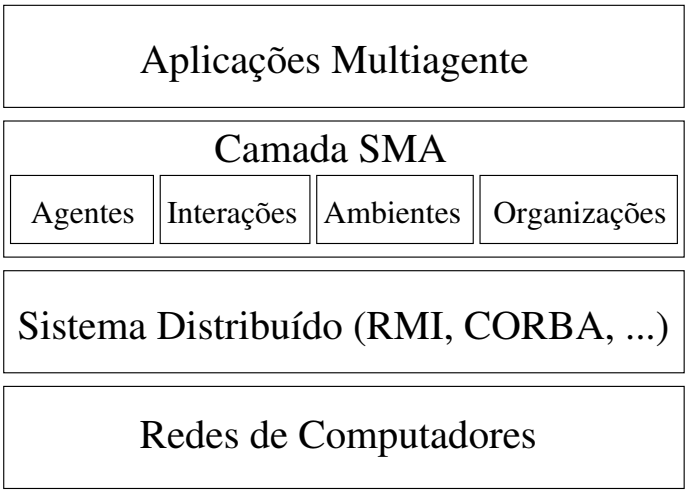

Figura 3.1: Estrutura de um SMA

ou guiadas pelas leis sociais baseadas nos estudos de ciências sociais. Desta forma, a construção de um SMA deve ser baseada nos seguintes pressupostos:

Seguindo estas idéias, dado um problema ou uma simulação a realizar, o usuário selecionaria os agentes, um modelo de ambiente, modelos de interações e organizações a serem instanciados, gerando desta forma um sistema multiagente para a resolução de seu problema. A figura 3.1, adaptada de Demazeau (1995), mostra esta concepção de SMA.

Equação declarativa: um SMA é composto de vários agentes, um conjunto de interações possíveis e pelo menos uma organização;

Equação funcional: a funcionalidade de um SMA é a soma das funções individuais de cada um dos agentes e a função coletiva, obtida através das interações;

Princípio da recursão: um SMA pode ser considerado, em um nível de abstração mais elevado, como sendo um agente.

3.3 Simulação Social e os SMA

Os SMA têm grande influência das ciências sociais, da mesma forma, simulações baseadas em SMA têm crescente importância para simulações sociais. Simulações sociais é uma das áreas onde SMA é a abordagem mais adequada para sua realização, por ser um problema de resolução descentralizada e os agentes envolvidos serem basicamente autônomos.

A maior preocupação da simulação social é o estudo experimental e a modelagem de conjuntos de efeitos imprevisíveis de uma população de agentes em um ambiente comum, o que seria o estudo da complexidade, emergência, auto-organização e dinâmica originada do comportamento dos agentes (CASTELFRANCHI, 1998). A simulação computacional consiste da análise de propriedades de modelos teóricos com o objetivo de explicar ou prever fenômenos naturais (FERBER, 1999, pág. 35). A simulação computacional de fenômenos social é um campo de pesquisa promissor, que se encontra na intersecção entre ciências sociais, matemática e ciência da computação. Estas simulações computacionais têm aplicação nas diversas ciências sociais, abrangendo da sociologia à economia, da psicologia social às teorias de organização política, e da demografia à antropologia e

arqueologia (CONTE; NIGEL; SICHMAN, 1998). Atualmente, a simulação computacional de processos e fenômenos sociais pode ser considerada uma de pesquisa área bem estabelecida. Nas últimas décadas a simulação computacional foi capaz de se beneficiar de um grande número de facilidades como linguagens de alto nível e desenvolvimento de sistemas inteligentes.

Neste contexto, a IAD, e em particular os SMA, forneceram as arquiteturas para desenvolvimento de agentes relativamente autônomos, fundamentais neste tipo de simulação. Os SMA trouxeram uma nova solução ao conceito de modelagem e simulação de ciências ambientais, oferecendo a possibilidade de representar diretamente indivíduos, seu comportamento e suas interações. A simulação com SMA se baseia na idéia que é possível representar de maneira computadorizada o comportamento de entidades que são ativas no mundo, e através disto representar um fenômeno como o produto de interações de um conjunto de agentes que possuem autonomia operacional (FERBER, 1999).

3.3.1 Simulação Social e Simulação Baseada em SMA

A abordagem baseada em agentes aumentou as potencialidades da simulação computacional como uma ferramenta para teorização sobre questões científicas sociais. Se o campo de SMA pode ser caracterizado pelo estudo de sociedades de agentes autônomos artificiais, a simulação social baseada em agentes pode ser definida como o estudo das sociedades artificiais de agentes autônomos (CONTE; NIGEL; SICHMAN, 1998).

Os SMA e a simulação social diferem em termos de formalismos utilizados, os SMA baseiam-se em lógica e IA enquanto a simulação social baseia-se na matemática (CONTE; NIGEL; SICHMAN, 1998). Apesar da teoria das decisões e teoria dos jogos terem influenciado ambos, existem diferenças teóricas entre os dois campos.

Segundo (CONTE; NIGEL; SICHMAN, 1998) os SMA herdaram da IA e das ciências cognitivas um grande embasamento teórico o que resultou em:

- longa experiência com projeto e implementação de arquiteturas integradas em oposição a autômatos elementares;
- forte ênfase no agente como um todo em oposição a apenas ações;
- grande atenção ao processo de construção de planos, não apenas a tomada de decisão e escolha;
- familiaridade com a normalização e implementação de estados mentais e comportamentais de agentes;
- tendência de prover o agente social com capacidades específicas para responder pedidos e tarefas sociais ao invés de modelar processos sociais como meras propriedades emergentes da interação dos agentes.

A área de simulação social tem maior influência das ciências sociais em relação aos SMA. Os fatores que contribuíram para o desenvolvimento da área foram:

- tendência de uso da simulação computacional para a testes de hipóteses teóricas ao invés de eficiência computacional do sistema;
- maior familiaridade com a interpretação de fenômenos sociais da vida real;
- produção de grandes quantidades de dados relativos à populações artificiais de grande escala.

Todas estas características resultaram na consolidação da reputação da metodologia científica da simulação computacional, reduzindo a característica *toy-world* de suas aplicações.

3.3.2 Objetivos da Simulação Social

No campo da simulação social, objetivos variam em função das disciplinas de interesse. Objetivos mais práticos prevalecem em ciências econômicas, gerenciamento e política e a ciência das organizações. Outras disciplinas como arqueologia e antropologia são claramente direcionadas ao crescente conhecimento científico através de formulação e testes de modelos interpretativos de fenômenos existentes através da reconstrução computacional (CONTE; NIGEL; SICHMAN, 1998).

A maior razão para o crescimento de interesse dos cientistas sociais na simulação computacional é o seu potencial para descoberta e formalização. Cientistas sociais podem construir modelos simples com foco em algum aspecto específico do mundo social e descobrir as conseqüências de suas teorias na "sociedade artificial" que construíram (GILBERT; TROITZSCH, 1999).

Nos SMA, objetivos científicos são utilizados para produção de software para várias aplicações como robótica para manufatura, controle de tráfego aéreo e defesa militar. Todavia, este campo já contribuiu para aumentar a compreensão da inteligência social e modelagem de agentes através da abordagem arquitetônica (CONTE; NIGEL; SICHMAN, 1998).

3.3.3 Organizações

Tanto para a simulação social e simulação baseada em SMA, as organizações e o estudo das organizações são aspectos importantes, seja para o estudo científico das organizações ou indiretamente na busca das melhores estruturas organizacionais ou na busca de melhores resultados ou performance.

Uma organização pode ser definida como um conjunto de relacionamentos entre componentes ou indivíduos que compõem uma unidade, dotada com qualidades não apresentadas no nível de componentes ou indivíduos (FERBER, 1999, pág. 88).

Segundo Carley e Gasser, não existe um grande consenso sobre o conceito de organizações, estes autores listam em (CARLEY; GASSER, 1999) algumas características que normalmente estão associadas a este conceito:

- constituídas de múltiplos agentes (humanos ou artificiais);
- estão associados à tecnologias de resolução de problemas de grande escala;
- estão comprometidas com uma ou mais tarefas; organizações são sistemas altamente ativos;
- capaz de afetar e ser afetada pelo ambiente;
- possui conhecimento, cultura, memórias, história e capacidades distintas a um agente simples.

As características que tornam o estudo das organizações uma tarefa complexa e bastante pesquisadas é que as organizações são sistemas complexos, dinâmicos, adaptativos que evoluem.

O estudo de organizações dentro dos SMA pode ser classificado, de acordo com Ferber (1999), da seguinte maneira:

- **nível micro-social:** onde há interesse essencialmente nas interações entre agentes e nas várias formas de relacionamentos que existem entre pequenas quantidades de agentes;

- **nível de grupos:** onde há interesse nas estruturas intermediárias que compõem organizações completas. Enfocando nas diferenciações de papéis e atividades dos agentes, emergência de estruturas organizacionais entre agentes e problemas gerais de agregação de múltiplos agentes na constituição de organizações;

- **nível de sociedades globais:** onde o interesse é concentrado na dinâmica de um grande número de agentes, junto à estrutura geral do sistema e sua evolução.

Alguns conceitos ou fenômenos importantes abordados no estudo de organizações seriam:

Emergência: fenômeno no qual estruturas não-existentes no nível micro "emergem" no nível macro, através da interação das estruturas micro.

Emergência cognitiva: fenômeno que ocorre quando os agentes tomam consciência através de uma "conceitualização" do fenômeno da emergência, influenciando o comportamento destes agentes (CASTELFRANCHI, 1998).

Imergência: processo através do qual a estrutura emergente no nível macro ocasiona mudanças no nível micro, remodelando o comportamento das estruturas micro (agentes) (CASTELFRANCHI, 1998).

Emergência de segunda ordem: processo através do qual as mudanças realizadas no nível micro através da imergência ocasionam alteram as estruturas emergentes já existentes.

O comportamento organizacional é resultado da interações entre uma variedade de agentes adaptativos (humanos ou artificiais), estruturação emergente em resposta a processos não-lineares e interações entre um grande número de outros fatores (PRIETULA; CARLEY; GASSER, 1998). Por esta razão, a análise computacional torna-se uma importante ferramenta para construção de teorias pois permite gerar um conjunto de proposições teóricas mesmo quando existem interações complexas de diversos fatores.

3.3.4 Questões em Aberto

Apesar dos resultados obtidos tanto na simulação social e nas simulações baseadas em SMA, o potencial do estudo computacional de fenômenos sociais ainda não foi totalmente explorado. A seguir são citadas algumas questões em aberto, citadas em (CONTE; NIGEL; SICHMAN, 1998):

- Como pode se combinar um modelo e projeto mais sofisticado de agentes, computacionalmente mais custosos, em uma simulação qualitativa e quantitativa de um fenômeno social? Enquanto SMA agregam um pequeno número de unidades computacionais, simuladores sociais obtém dados sobre populações de grande escala. Seria possível obter um volume significante de dados em, por exemplo, uma plataforma BDI?

- Como o paradigma da emergência e o estudo da inteligência social podem ser relacionados?

- Como explicar o elo micro-macro? O paradigma da emergência é insuficiente, pois só relaciona de maneira unidirecional, do comportamento às estruturas sociais. Este paradigma não menciona a difusão de propriedades da chamada emergência de segunda ordem, ou seja, representações mentais das estruturas sociais. Como conciliar este processo com a autonomia dos agentes?

- Agentes fazem escolhas racionais; eles decidem entre alternativas de modo consistente com seu critério interno (racionalidade, maximização de utilidade ou outro critério). Como formam estas alternativas? Como integrar modelos de tomada de decisões sociais com planejamento e resolução de problemas?

- O paradigma da emergência objetiva explicar a convergência de certas regularidades comportamentais. Todavia, se o aprendizado social justifica alguns mecanismos responsáveis pela difusão de conhecimento, este não aparenta fornecer inovações. Como podem novas regras e convenções substituir outras? Como pode se conciliar estabilidade e inovação? Algoritmos genéticos ligam estes aspectos pela introdução de mutação. Porém, a visão de que a inovação seria uma mutação acidental não faz justiça ao papel ativo dos agentes no estabelecimento de convenções.

- A dinâmica social não é apenas um efeito do aprendizado e evolução. Agentes sociais modificam seu comportamento e de seus companheiros através de argumentação, planejamento e influência. Como combinar a flexibilidade e reatividade do agente com sua autonomia e inteligência?

4 COMUNICAÇÃO ENTRE AGENTES

Um dos aspectos mais importantes em SMA são as interações baseadas em comunicação. A comunicação é a maneira pela qual os agentes cognitivos interagem, tendo um papel fundamental para a coordenação e outras formas de interação de agentes em um sistema multiagente.

Agentes se comunicam através de protocolos de comunicação que permitem a troca de mensagens entre agentes. Essa comunicação é realizada para obter um melhor desempenho no cumprimento de seus objetivos ou da sociedade em que estão inseridos (HUHNS; LARRY; STEPHENS, 1999). Estes objetivos, da sociedade e dos outros agentes, podem não ser conhecidos explicitamente pelos agentes. A comunicação pode possibilitar que os agentes coordenem suas ações e comportamentos, resultando em sociedades mais coerentes. Coerência é o quão bem uma sociedade de agentes se comporta como uma unidade. Um problema dos SMA é como manter uma coerência global sem um controle global explícito.

Segundo (MAYFIELD; LABROU; FININ, 1995) um sistema de informações que opere em ambientes altamente distribuídos, dinâmicos, heterogêneos e com grande número de nodos, tal como a Internet, têm diversos problemas, como por exemplo:

- A arquitetura predominante na Internet, o modelo Cliente-Servidor, é muito restritiva. Alguns nodos podem agir como cliente ou servidor dependendo com quem estiverem interagindo.

- Heterogeneidade: Diferentes formatos de dados, diferentes plataformas, diferentes padrões aumentam a complexidade do sistema.

- Muitas tecnologias de software, como o raciocínio baseado em conhecimento e o processamento de linguagem natural entre outras, estão prontas para participar e contribuir em um ambiente como a Internet. Porém, existe uma multiplicidade de técnicas e ferramentas para a construção de servidores inteligentes e softwares baseados em agentes. Esta grande variedade de opções, muitas vezes incompatíveis entre si, afastam as grandes aplicações destas tecnologias.

Uma solução para estes problemas seria uma comunidade de agentes inteligentes. Quando menciona-se agentes, faz se referência às suas habilidades de serem pró-ativos, comunicarem-se usando um linguagem expressiva, cooperarem para atingirem objetivos complexos e usarem informação e conhecimento para a gerência de recursos e lidar com pedidos de agentes externos.

Nesta seção que trata da comunicação entre agentes, nas primeiras subseções serão brevemente discutidas as interações entre agentes. Na seqüência, serão apresentados os

aspectos necessários a uma linguagem para comunicação entre agentes e a infra-estrutura projetada para o uso de KQML (*Knowledge Query and Manipulation Language*) em particular. Será apresentado também o SACI (Simple Agent Communication Infrastructure) (HÜBNER; SICHMAN, 2000, 2001), ferramenta que possibilita a comunicação baseada em KQML, além de fornecer outras facilidades para criação e controle de agentes distribuídos.

4.1 Interações entre Agentes

Estruturas de interação variam desde interações físicas a atos de fala. Modelos de interações físicas como forças eletrostáticas, por exemplo, permitem expressar interações simples como atração e repulsão, modelos muito utilizados como base para modelagem de interações entre agentes reativos (DEMAZEAU, 1995). Os protocolos de interação governam a troca de séries de mensagens entre agentes, especificando a seqüência de uma conversação. Como os agentes reativos não possuem controle deliberativo, nem raciocínio explícito, não é esperado destes que sejam capazes de realizar conversações estruturadas baseadas em atos de fala. O enfoque desta seção será sobre os agentes cognitivos e comunicação baseada em atos de fala.

4.2 Dimensões do Significado

O significado é uma combinação da semântica (o que os símbolos denotam) e da pragmática (como os símbolos são interpretados por indivíduos). Para que os agentes se comunicarem, compreender e serem compreendidos, é importante considerar as diferentes dimensões do significado que estão associados com a comunicação. A seguir, estão resumidas algumas das dimensões citadas em (HUHNS; LARRY; STEPHENS, 1999):

- Significado pessoal e convencional: um agente pode ter sua própria compreensão sobre uma mensagem, porém esta pode ser diferente ao significado convencionado por outros agentes. Por isto, SMA devem optar por significados convencionais, especialmente se o ambiente for aberto, onde novos agentes podem entrar a qualquer momento.

- Perspectiva do emissor, receptor e da sociedade: independente do significado convencional de uma mensagem, a mensagem pode ser expressa de acordo com o ponto de vista do emissor, receptor ou outros observadores.

- Contextualidade: mensagens não podem ser compreendidas de maneira isolada, mas deve ser interpretada levando-se em consideração os estados mentais dos agentes, o presente estado do ambiente e seu histórico. A interpretação é diretamente afetada por mensagens e ações prévias dos agentes.

- Identidade: quando ocorre a comunicação entre agentes, o significado depende das identidades e dos papéis dos agentes envolvidos.

- Cardinalidade: uma mensagem enviada de maneira privada para um agente poderia ser interpretada de maneira diferente se a mesma mensagem fosse enviada publicamente para todos os agentes.

4.3 Atos de Fala

A comunicação falada entre humanos é utilizada como modelo para comunicação entre agentes computacionais. Uma base para analisar a comunicação humana é a *Teoria dos Atos de Fala*, formulada por Searle (1969). A teoria dos atos de fala trata a linguagem natural como ações no mundo. Segundo (HUHNS; LARRY; STEPHENS, 1999), um ato de fala tem três aspectos:

- Locução: o que foi pronunciado pelo falante;
- Ilocução: o significado pretendido pelo falante;
- Perlocução: a ação resultante da locução.

Na linguagem KQML é usado o termo *performativa* para identificar a força ilocucionária de uma sentença. A força ilocucionária pode ser classificada como afirmativas, diretivas (comandos em uma estrutura mestre-escravo), declarativas e expressivas (expressão de emoções). Exemplos de forças ilocucionárias podem ser: pedidos, sugestões, comprometimento e respostas. Desta forma, a teoria dos atos de fala ajuda a definir o tipo da mensagem utilizando-se a força ilocucionária, o que determina a semântica do ato de comunicação. A intenção do emissor em seu ato comunicativo é claramente definida, de forma que o receptor não tem dúvida sobre o tipo da mensagem recebida (HUHNS; LARRY; STEPHENS, 1999). Esta declaração explícita da intenção do emissor simplifica o projetos da parte de comunicação dos agentes.

4.4 Ontologias

Para possibilitar a interação através de mensagens em um grupo de agentes heterogêneo, desenvolvidos com diferentes técnicas de programação e programadores, é necessário que eles usem uma conceitualização comum do domínio. Uma conceitualização nomeia e descreve as entidades que podem existir no domínio, descrevendo também o relacionamento entre essas entidades (BORDINI; VIEIRA; MOREIRA, 2001). Tais especificações explícitas de um domínio são chamadas ontologias e são essenciais para o desenvolvimento e uso de sistemas inteligentes assim como para possibilitar sua interoperabilidade.

4.5 Requisitos para Linguagens de Comunicação entre Agentes

De acordo com (MAYFIELD; LABROU; FININ, 1996), pode-se dividir os requisitos de um linguagem de comunicação entre agentes em sete categorias: forma, conteúdo, semântica, implementação, *networking*, ambiente e confiabilidade. Acredita-se que a linguagem que satisfizer tais exigências é suficientemente sofisticada. Porém, em alguns momentos, alguns desses requisitos podem ser conflitantes. Por exemplo, uma linguagem que pode ser lida claramente pode não ser concisa o suficiente. Segundo (MAYFIELD; LABROU; FININ, 1996), no projeto de uma linguagem de comunicação entre agentes devem ser ponderados os seguintes ítens:

- Forma: uma linguagem de comunicação entre agentes deve ser de fácil compreensão, sintaticamente simples, declarativa e extensível. Para transmitir uma mensagem de um agente para outro, a mensagem deve passar por meio de uma cadeia de

bits do mecanismo de transporte e por isso a linguagem também deve ser linear ou facilmente linearizável.

- Conteúdo: deve-se diferenciar a linguagem de comunicação, a qual expressa atos comunicativos, e a linguagem de conteúdo, a qual expressa fatos sobre o domínio. A linguagem de comunicação deve conter um conjunto de atos comunicativos. No entanto, esse conjunto poderia ser extensível, para garantir a funcionalidade da linguagem em diversos sistemas.

- Semântica: deve ter embasamento teórico (atos de fala, por exemplo) e não pode ser ambígua, o que se espera da semântica de qualquer linguagem computacional. Deve garantir que a similaridade de significado tenha similaridade de representação. Já que a linguagem de comunicação poderá ser utilizada para interação entre aplicações espacialmente dispersas, localização e tempo devem ser cuidadosamente considerados pela semântica. A descrição semântica deve prover um modelo de comunicação.

- Implementação: deve ser eficiente tanto em velocidade quanto em largura de banda utilizada. A linguagem deve permitir o uso de um subconjunto de primitivas básicas de atos de comunicação, já que agentes simples talvez não precisem de todas as primitivas. A interface deve ser de fácil utilização, detalhes de baixo nível devem ser transparentes para o usuário.

- *Networking*: Deve suportar todas as conexões básicas: ponto-a-ponto, *multicast* e *broadcast*. Conexões síncronas e assíncronas devem ser suportadas. Deve conter um conjunto de primitivas rico o suficiente para servir de base para que linguagens e protocolos de alto nível possam ser implementados. Tais protocolos de alto nível devem ser implementados como camadas de software independentes das camadas inferiores relacionadas à comunicação.

- Ambiente: Tendo em vista que o ambiente no qual os agentes vão interagir é distribuído, heterogêneo e dinâmico, a linguagem deve suportar interatividade com outras linguagens e protocolos, suportar descoberta de conhecimento em grandes redes e possibilitar o aproveitamento de sistemas legados.

- Confiabilidade: deve prover segurança e confiabilidade na comunicação entre agentes. Deve haver meios de autenticação dos agentes, tolerância a falhas, garantindo que o sistema não seja abalado por mensagens incompletas ou impróprias.

4.6 A Linguagem KQML

***K**nowledge **Q**uery and **M**anipulation **L**anguage* (FININ, T. et al., 1994a; MAYFIELD; LABROU; FININ, 1995, 1996; FININ, T. et al., 1994b) é uma linguagem projetada para permitir a interação entre agentes. No projeto desta linguagem, tentou-se satisfazer a maioria dos ítens básicos, citados anteriormente.

Quando usa-se KQML, o agente transmite o conteúdo das mensagens, composto na linguagem de sua escolha, encapsulada dentro de uma mensagem KQML. O conteúdo da mensagem pode ser expresso em qualquer linguagem de representação e escrito tanto em texto puro quanto em uma notação binária. As implementações de KQML ignoram o conteúdo da mensagem, precisando apenas reconhecer o início e o fim desta.

A sintaxe baseia-se em uma lista com parênteses balanceados. O elemento inicial da lista é uma primitiva da linguagem (performativa) e os argumentos restantes são pares de campos e valores. O conjunto de primitivas formam o "núcleo" da linguagem e determinam os tipos de interações que um agente usando KQML pode ter. A função básica de uma primitiva de linguagem é identificar o modo que uma mensagem deve ser tratada e de fornecer o "ato de fala" que indica a intenção do remetente ao enviar a mensagem. A primitiva indica se o conteúdo é um comando, uma pergunta, uma declaração ou qualquer outra ilocução reconhecida mutuamente. Também pode explicitar como o remetente gostaria que a resposta fosse enviada, por exemplo, o protocolo a ser seguido.

Por exemplo, uma mensagem representando uma pergunta sobre o preço de uma ação de uma empresa ABCD poderia ser escrita do seguinte modo (adaptado de (MAYFIELD; LABROU; FININ, 1996)):

(ask-one

: content (PREÇO ABCD ?preço)

: receiver servidor-acoes

: language PROLOG

: ontology BOLSA-NY)

Nesta mensagem , a primitiva KQML é *ask-one* (requisita uma única resposta como resultado), a ontologia assumida por essa pergunta é identificada por *BOLSA-NY*, o receptor da mensagem é o servidor identificado como *servidor-acoes* e a pergunta foi escrita usando a linguagem PROLOG. Diversas outra primitivas também poderiam ter sido usadas no lugar de *ask-one*, dependendo do objetivo da mensagem.

A seguir são descritas algumas das principais primitivas da linguagem KQML, a mesma seleção apresentada em (BORDINI; VIEIRA; MOREIRA, 2001). Estas performativas são divididas em diversas categorias. Nas descrições abaixo, *E* denota o agente que envia a mensagem (*Emissor*), e *R* denota o agente que recebe a mensagem (*receptor*).

- Performativas de Informação

  tell: *E* informa para *R* que o conteúdo da mensagem é verdadeiro para *E*, ou seja o conteúdo da mensagem está na base de conhecimentos de *E*;

  untell: o conteúdo da mensagem não faz parte da base de conhecimentos de *E*.

  deny: *E* informa para *R* que a negação do conteúdo da mensagem é verdadeira para *E*;

- Performativas de Consulta

  ask-if: *E* quer saber se o conteúdo da mensagem é verdadeiro para *R*;

  ask-all: *E* quer saber de *R* todas as respostas que *R* considera verdadeiras para a consulta contida no conteúdo da mensagem (uma resposta por mensagem).

- Respostas Básicas

  error: *E* indica a *R* que não compreendeu a mensagem recebida anteriormente;

sorry: *E* informa a *R* que compreendeu a mensagem, mas não pode prover uma resposta.

- Performativas de Bases de Dados

  insert: *E* pede para *R* acrescentar o conteúdo da mensagem na base de conhecimentos de *R*;

  delete: *E* pede para *R* retirar o conteúdo da mensagem na base de conhecimentos de *R*;

- Definição de Capacidades

  advertise: *E* quer que *R* saiba que *E* pode e processará mensagens do tipo que está no conteúdo desta.

- Performativas de Efetuação

  achieve: *E* solicita a *R* que tente fazer com que o conteúdo da mensagem torne-se verdadeiro;

  unachieve: *E* quer reverter a ação de um *achieve* enviado previamente;

- Performativas de Rede

  register: *E* anuncia para *R* (facilitador, ver seção 4.6.1) sua presença e nome simbólico associado ao endereço na rede;

  unregister: cancelar um *register* feito anteriormente;

  transport-address: *E* anuncia a *R* seu novo endereço na rede;

  forward: *E* solicita a *R* que passe a mensagem para outro agente especificado no campo "to";

  broadcast: *E* pede a *R* que repasse a mensagem para todos os agentes que *R* conhece.

- Performativas de facilitação

  broker-one: *E* pede para *R* achar uma resposta para a executiva que está no conteúdo da mensagem;

  recommend-one: *E* pede a *R* para sugerir um agente que possa processar o conteúdo da mensagem.

Embora exista um conjunto pré-definido de primitivas reservadas, este não é um conjunto mínimo necessário ou fechado. Um agente que utilize KQML pode escolher um subconjunto qualquer de primitivas para comunicar-se. O conjunto de primitivas é extensível. No entanto, se uma implementação de KQML que implemente uma das primitivas reservadas deve implementá-la do modo padrão (MAYFIELD; LABROU; FININ, 1995).

Como o KQML não foi definido por um único grupo de pesquisa para um projeto particular, não há uma arquitetura particular de sistema e vários sistemas diferentes podem

estar envolvidos. É possível que duas implementações do KQML, uma em C e outra em Common Lisp, sejam são totalmente interoperáveis e utilizadas de maneira conjunta.

Na proposta do KQML, há também uma infraestrutura para a realização da comunicação entre agentes. Nesta arquitetura estão presentes os "facilitadores" que são agentes que tem conhecimento sobre quais agentes estão acessíveis e quais são as habilidades de cada um dos agentes de uma sociedade.

4.6.1 Facilitadores KQML

Em uma sociedade de agentes, para que os agentes comuniquem-se com outros cujo endereço na rede é desconhecido, os agentes dependem dos facilitadores. Para que a presença de um agente seja conhecida por outros agentes basta que este registre-se no facilitador. Ele mantém um registro de nomes de serviços e encaminha mensagens com endereços incompletos através do nome de um serviço registrado. Na verdade os facilitadores são agentes de rede que têm capacidades para manipular mensagens KQML (FININ, T. et al., 1994a). Tipicamente, há um facilitador para cada grupo local de agentes, mas também pode haver facilitadores redundantes para tolerância a falhas. Quando cada agente inicia, este deve se anunciar ao facilitador local que o registra em seus banco de dados, e quando o mesmo termina, o facilitador o remove do banco de dados. Isto torna possível que as sociedades de agentes funcionem de forma dinâmica, ou seja, que agentes possam entrar e sair da sociedade livremente. Deste modo, as aplicações podem interoperar sem terem de manter uma lista de agentes presentes na sociedade.

4.6.2 Interfaces para Programação

Para facilitar a programação dos agentes, a comunicação normalmente é implementada como uma biblioteca, que deve prover as funções para manipulação das mensagens. É necessário haver uma interface de programação tornando transparente as funções de comunicação de baixo nível. Esta ficaria inserida na aplicação e forneceria acesso às ferramentas da aplicação para analisar seu conteúdo.

A comunicação pode estar muito ligada à aplicação e ser implementada na mesma linguagem de programação da aplicação. Porém, não é necessário inserir completamente a comunicação no código da aplicação. Uma interface deve prover basicamente duas funções: *send-kqml-message* e *receive-kqml-message*. Dependendo das capacidades da implementação, as mensagens que estão chegando podem ser classificadas de acordo com sua primitiva, ou outro aspecto, e encaminhada para diferentes funções de manipulação.

4.7 SACI

Como já visto, a comunicação tem um papel muito importante dentro dos SMA. Nesta seção será apresentado o SACI (Simple Agent Communication Infrastructure) (HÜBNER; SICHMAN, 2000, 2001), ferramenta que permite a comunicação, criação e gerência de agentes distribuídos que utilizam KQML, além de ferramentas de suporte à agentes distribuídos. O SACI é implementado em JAVA, utilizando-se como base para a comunicação a estrutura RMI (Remote Method Invocation). As principais características, apresentadas em (HÜBNER; SICHMAN, 2000), do SACI são:

- Os agentes utilizam KQML para se comunicar. Há funções para compor, enviar e receber mensagens KQML. Apesar disso, o uso do KQML é facultativo.
- Os agentes são identificados por um nome. As mensagens são transportadas

utilizando-se somente o nome do receptor, afastando o projetista do agente de questões relativas à comunicação de baixo nível.

- Um agente pode conhecer os outros por meio de um serviço de páginas amarelas. Agentes podem registrar seus serviços no facilitador e consultá-lo sobre que serviços são oferecidos por quais agentes.

- Os agentes podem ser iniciados remotamente.

Estrutura de Funcionamento

As principais funções realizadas pelo SACI são o controle e suporte da troca de mensagens entre os agentes, o controle de agentes presentes em uma sociedade de agentes e o controle dos serviços de páginas amarelas (consulta de agentes por serviço) e páginas brancas (consulta de agentes por nome).

A entrada e saída de agentes em uma sociedade é controlada pelo facilitador. Conforme sugerido da proposta do KQML, cada sociedade possui um facilitador que mantém em sua estrutura a identidade, localização física e os serviços oferecidos pelos agentes da sociedade. Ao entrar na sociedade, o agente tem que contatar o facilitador e registrar um nome. O facilitador verificará a unicidade do nome e irá associá-lo a localização do agente. Ao sair da sociedade o agente deve avisar o facilitador.

O envio e a recepção de mensagens é feita basicamente pelo componente MBox que serve de interface entre o agente e a sociedade. A finalidade do MBox é tornar transparente o envio e recebimento de mensagens, encapsulando funções de composição de mensagens, envio síncrono e assíncrono, e o recebimento de mensagens.

O anúncio de habilidades é feito através do facilitador. Os agentes podem anunciar ao facilitador suas habilidades, que poderá ser consultado como um serviço de páginas amarelas. Desta forma quando um agente precisa de um serviço e não conhece o nome de um agente capaz de realizá-lo, pode requisitar ao facilitador uma lista com os agentes que realizam determinado serviço.

O processo de inicialização de uma sociedade com um agente *APLusServer* e outro *SampleSaciAg*, é mostrado na figura 4.1 traduzida de Hübner e Sichman (2001).

Os passos contidos na figura são descritos a seguir:

1. O menu do SACI é iniciado;

2. O menu do SACI inicia um *Launcher Demon* automaticamente;

3. O menu do SACI inicia um *Society Launcher* automaticamente;

4. O *Society Launcher* pede ao *Launcher Demon* para criar uma nova sociedade;

5. O *Launcher Demon* inicia um facilitador para a nova sociedade;

6. O menu do SACI inicia um *Agent Launcher* automaticamente;

7. O *Agent Launcher* pede ao *Launcher Demon* para criar um novo agente "APLusServer" a partir da classe "PlusServer";

8. O *Launcher Demon* inicia este agente;

9. O agente "SampleSaciAg" é iniciado por uma JVM;

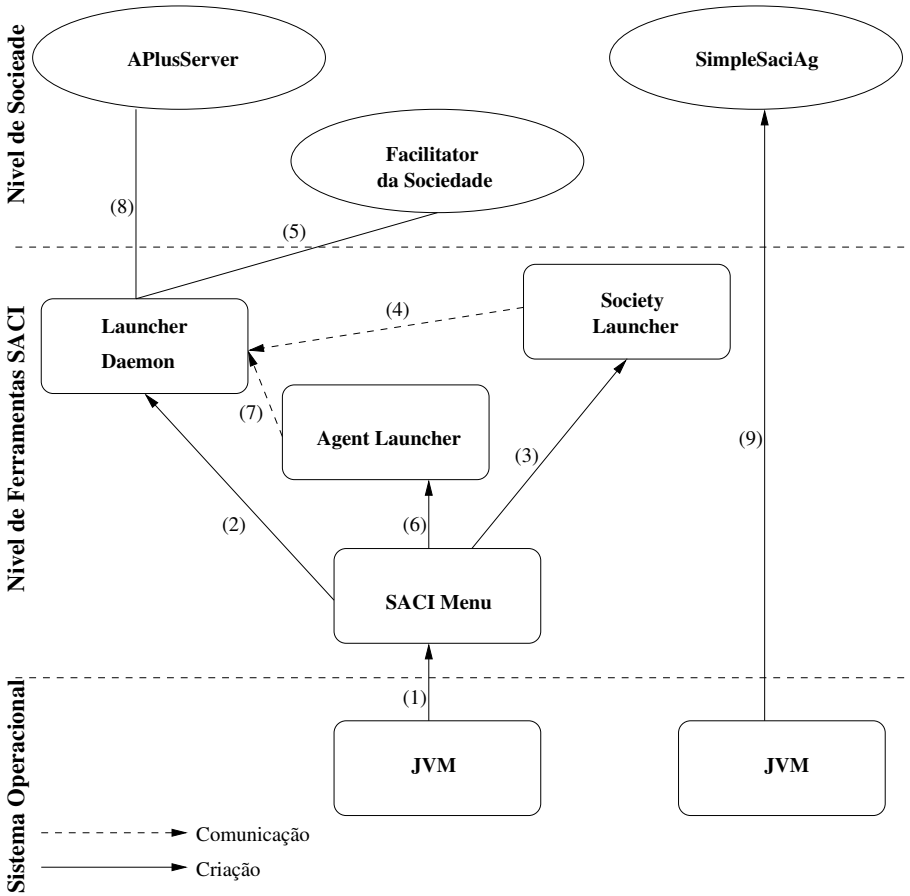

Figura 4.1: Estrutura do SACI

5 AMBIENTES MULTIAGENTES E O ELMS

De acordo com Wooldridge (1999), agentes são sistemas computacionais situados em um ambiente, e capazes de ações autônomas para realização dos objetivos para os quais foram projetados. Os agentes percebem e interagem entre si através de um ambiente, alterando o estado deste para alcançar estados onde seus objetivos sejam atingidos. Desta forma, ambientes são parte importante em um SMA, seja o mundo real, a Internet ou um ambiente virtual. Em algumas modelagens, ambiente também é considerado como parte importante na comunicação, por exemplo na comunicação falada, o falante cria ondas sonoras que irão se propagar pelo ar até chegar ao ouvinte, utilizando o ambiente como meio para realização da comunicação.

A modelagem de ambientes é uma questão importante no desenvolvimento de simulações multiagentes, onde os agentes não agem diretamente em um ambiente real, tal como robôs com sensores e efetores reais. Todavia, a literatura de sistemas multiagentes poucas vezes explicita esta parte da modelagem de sociedades de agentes, considerando os ambientes como dados. Em alguns casos onde simplesmente não existe ambiente, o que Ferber chama de "SMA puramente comunicante" (*purely communicating MAS*) onde não existe ambiente e os agentes não fazem outras ações além de se comunicar diretamente através de mensagens (FERBER, 1999, pág. 11). Porém, neste caso o ambiente pode ser considerado a rede através da qual o agente se comunica.

Nos sistemas multiagentes reativos, o ambiente tem um papel importante. Partindo do princípio que os agentes reativos não possuem memória, é apenas a percepção do ambiente que permite a eles tomarem decisões sobre suas ações. No outro extremo, agentes cognitivos possuem uma representação interna do ambiente no qual estão situados, e realizam decisões, adoção de objetivos, mudanças no curso de ações, baseados em mudanças que a percepção do ambiente causa em sua representação. Portanto, a modelagem de ambientes é igualmente importante em ambas as classes de sistemas multiagentes, quando estas são sistemas de software.

Neste capítulo, que apresenta os aspectos principais da linguagem criada para a especificação de ambientes que é compartilhado entre os agentes em um sistema multiagente, que denominamos ELMS (**E**nvironment Description **L**anguage for **M**ulti-Agent **S**imulations). Com esta linguagem, é possível especificar ambientes que são, do ponto de vista dos agentes, inacessíveis, não determinísticos, não-episódicos, dinâmicos, porém devem ser discretos (estas características serão apresentadas a seguir e discutidos na seção 5.7). Apesar de não ser possível a definição de ambientes contínuos, acreditamos que é possível a definição de ambientes complexos o suficiente para suportar uma grande variedade de simulações multiagente.

5.1 Classificação de Ambientes

Em (RUSSEL; NORVIG, 1995) é fornecida um classificação para os possíveis ambientes, como visto a seguir:

Acessível ou inacessível. Se os sensores de um agente podem dar acesso ao estado completo do ambiente, então podemos dizer que o ambiente é acessível para este agente. Um ambiente é efetivamente acessível se os sensores detectam todos os aspectos relevantes à escolha da ação.

Determinístico ou não-determinístico. Se o próximo estado do ambiente é completamente determinado pelo estado corrente e as ações selecionadas pelos agentes, então podemos dizer que o ambiente é determinístico. A princípio, um agente não precisa se preocupar com a incerteza em um ambiente acessível e determinístico. Porém se o ambiente é inacessível, então este pode parecer não-determinístico, do ponto de vista do agente.

Episódico ou não-episódico. Em um ambiente episódico, as experiências dos agentes são divididas em episódios. Em cada episódio, o comportamento do agente consiste apenas em perceber e agir, pois estas ações não irão interferir nos episódios subseqüentes.

Estático ou dinâmico. Se o ambiente pode mudar enquanto o agente está deliberando, então dizemos que o ambiente é dinâmico para este agente, caso contrário o ambiente é estático. Se o ambiente não muda com a passagem do tempo, mas o desempenho do agente é medido pelo seu tempo de deliberação, então dizemos que o ambiente é semidinâmico.

Discreto ou contínuo. Se existe um número determinado e limitado de percepções e ações, podemos dizer que o ambiente é discreto. Um jogo de xadrez é um ambiente discreto, pois há um número fixo de possíveis movimentos para jogada, enquanto dirigir é contínuo, pois a velocidade e a localização dos outros veículos variam de maneira contínua.

A tabela 5.1, traduzida de Russell e Norvig (1995), apresenta alguns exemplos de ambientes e sua classificação segundo os critérios acima.

5.2 Modelagem de Ambientes

Agentes são sistemas computacionais situados em um ambiente, e que são capazes de ações autônomas neste ambiente para realizar seus objetivos (WOOLDRIDGE, 1999). Considerando um agente situado em um ambiente, quaisquer outros agentes serão, do ponto de vista deste, parte importante do ambiente, especialmente nos SMA, onde podem existir múltiplos agentes. Desta forma a modelagem dos ambientes é um aspecto importante nas simulações multiagente. Mesmo quando não simulam um mundo real ou não permitam a resolução de problemas específicos, a construção de "mundos sintéticos" tem grande importância na pesquisa de sistemas multiagentes pois torna possível analisar certos mecanismos de interação de maneira mais detalhada que em uma aplicação real (FERBER, 1999).

Tabela 5.1: Exemplos de Ambientes e Características

	Acessível	**Determ.**	**Episódico**	**Estático**	**Discreto**
Xadrez com relógio	Sim	Sim	Não	Semi	Sim
Xadrez sem relógio	Sim	Sim	Não	Sim	Sim
Pôquer	Não	Não	Não	Sim	Sim
Gamão	Sim	Não	Não	Sim	Sim
Dirigir automóvel	Não	Não	Não	Não	Não
Sistema de diagnóstico médico	Não	Não	Não	Não	Não
Sistema de análise de imagem	Sim	Sim	Sim	Semi	Não
Robô carregador de blocos	Não	Não	Sim	Não	Não
Controlador de refinaria	Não	Não	Não	Não	Não
Tutor interativo de inglês	Não	Não	Não	Não	Sim

Nos SMA reativos, o ambiente tem papel importante pois como estes não possuem memória, as alterações que realizam no ambiente possibilitarão que ações tomadas anteriormente tenham influência nas ações futuras, permitindo um seqüenciamento de ações. Já os agentes cognitivos, que na maioria das vezes possuem uma representação interna do ambiente, podem tomar decisões baseadas em *mudanças* percebidas no ambiente (por comparação com a memória dos estados anteriores).

Na modelagem de ambientes para simulações com SMA, deve-se levar em consideração os aspectos relevantes do ambiente à simulação, que podem ser características físicas perceptíveis dos agentes, objetos, forças eletrostáticas, feromônios e várias outras características, dependendo do enfoque desejado para a simulação.

5.2.1 Exemplo MANTA

O MANTA (Modelling ANTnest Activity) é uma aplicação de SMA reativos apresentada em (DROGOUL; FERBER, 1992). Aqui serão apresentados apenas alguns dos aspectos da modelagem do ambiente desta simulação.

O objetivo desta simulação é modelar os comportamentos de uma sociedade de formigas comuns (*Ectatomma ruidum*). O ambiente reproduz um ninho similar aos utilizados em laboratórios. Esta colônia pode ser modificada ao longo da simulação para testar a influência da topologia na população.

O ambiente é definido como um grande conjunto de entidades que representam espaços, dispostos bidimensionalmente, não havendo espaços sobrepostos. Espaços são quadrados de mesmo tamanho, de granularidade regulável. Os espaços "sabem" a todo momento quais agentes estão presentes neles, além de sua posição (x, y) absoluta no espaço. Os espaços são divididos em espaços livres e obstáculos, a diferença entre eles é

que os obstáculos não podem ser ocupados e não propagam estímulos.

Como estes agentes são reativos e não possuem representação do ambiente nem de outros agentes, estes não realizam comunicação direta com outros agentes. Os agentes interagem deixando "sinais" no ambiente, alterando propriedades dos espaços. Estes "sinais" seguem a idéia dos feromônios utilizados pelos insetos para se comunicarem.

Para completar o ambiente foram inseridos agentes "ambientais" para representar fatores ambientais como luz e umidade. Estes agentes propagam seus estímulos alterando propriedades dos espaços. Por exemplo, os agentes "luz" propagam e alteram a intensidade de seus estímulos para que as formigas percebam a diferença entre dentro e fora do ninho e além do dia e da noite.

Com estes fatores, os agentes formigas têm os dados ambientais para escolher suas ações e executá-las.

O ambiente desta simulação seria, segundo a classificação de ambientes de Russel e Norvig (1995), não-acessível, não-determinístico (do ponto de vista do agente), não-episódico (apesar de tratado de maneira episódica pelos agentes), estático e discreto. Sendo, desta forma, um ambiente relativamente complexo devido a quantidades de agentes e fatores envolvidos.

5.2.2 Exemplo *Soccer Server*

O Soccer Server é um simulador de jogo de futebol, utilizado na RoboCup (CORTEN, E. et al., 1999; NODA, I. et al., 1998). A RoboCup é uma tentativa de estimular os pesquisadores de IA e de robótica a pesquisarem um problema padrão comum, onde diversas tecnologias possam ser integradas, examinadas, testadas e comparadas. A descrição abaixo refere-se a versão sem robôs da RoboCup, onde todos os fatores, exceto as decisões dos jogadores são controlados e calculados pelo Soccer Server.

O Soccer Server permite que uma partida seja jogada entre dois times de jogadores-programas, possivelmente implementados em diferentes linguagens e plataformas. Uma partida no Soccer Server é controlada com comunicação cliente-servidor. O Soccer Server fornece um campo de futebol virtual e simula os movimentos dos jogadores e da bola (NODA, I. et al., 1998). Um programa cliente pode fornecer o "cérebro" de um jogador, conectando-se ao Soccer Server via rede. Como retorno o cliente recebe as informações dos sensores do jogador.

O campo de futebol e todos os objetos, incluindo os jogadores, são bidimensionais. Os jogadores e a bola são tratados como círculos e a movimentação dos objetos é calculada pelo Soccer Server seguindo os comandos enviados pelos clientes jogadores. Existem outros objetos no campo, como linhas, bandeiras e traves que são pontos fixos para referência da posição, orientação e velocidade do jogador. Os clientes não recebem a posição absoluta do jogador, apenas a distância relativa aos objetos presentes, de forma que objetos fixos são pontos de referência para os cálculos dos clientes.

Para refletir a natureza do mundo real, e também reforçar a importância de um comportamento robusto, o servidor introduz vários tipos de incerteza na simulação:

- Ruído é acrescentado ao movimento dos objetos: a quantidade de ruído aumenta com a velocidade do objeto;
- Ruído é adicionado aos parâmetros dos comandos: são adicionados pequenos números randômicos aos parâmetros de comandos enviados pelos clientes;
- Execução limitada de comandos. O servidor executa apenas um comando por ciclo

para cada jogador. Se um cliente envia mais de um comando em um ciclo, não é possível a este saber qual comando será executado.

- Sensores inexatos. Quanto mais longe um objeto, menor a confiabilidade da informação sensória retornada pelo servidor.

O ambiente que o Soccer Server fornece para simulação de uma partida de futebol é, segundo a classificação de Russel e Norvig (1995), não-acessível, não-determinístico, não-episódico, dinâmico e contínuo. Sendo, desta forma, um ambiente extremamente complexo.

5.3 A Modelagem de Ambientes Utilizando ELMS

A descrição de um ambiente utilizando ELMS pode ser feita através da especificação de propriedades do ambiente que são modeladas como conjuntos de: objetos (referidos como *recursos*) do ambiente; agentes (a representação "física" de um agente que é visível a outros agentes no ambiente); ações que cada agente pode realizar no ambiente, e tipos de percepções disponíveis para cada tipo de agente.

Os recursos ou objetos que estão presentes em um ambiente podem ser modelados como um conjunto de propriedades e reações que estes podem realizar em resposta a estímulos externos ao objeto. Em outras palavras, os objetos podem reagir, apenas agentes são pró-ativos. Agentes podem ser considerados componentes do ambiente, visto que, do ponto de vista de um agente, qualquer outro agente é parte do ambiente. Desta forma, agentes são definidos como uma lista de propriedades (que define os aspectos perceptíveis dos agentes situados neste ambiente), uma lista de ações que estes são capazes de realizar (pró-ativamente), e uma lista de percepções a que eles têm acesso. Do ponto de vista do ambiente, as atividades de deliberação de um agente não são relevantes, pois são internas a este, e não observáveis para os outros agentes. Como mencionado antes, os aspectos internos dos agentes são descritos com o uso de AgentSpeak(XL), visto na seção 2.4.

Em relação à grade, se o projetista do ambiente optar por utilizá-la, suas posições podem ser acessadas por coordenadas absolutas ou relativas. Coordenadas relativas são precedidas por '$+$' ou '$-$', então $(+1, -1, +0)$, por exemplo, se refere a posição na diagonal superior à direita da posição atual do agente (na mesma profundidade).

Para a definição dos tipos de percepção que cada tipo de agente tem acesso, é necessário definir quais propriedades do ambiente, agentes e objetos são perceptíveis para cada tipo de percepção. Também, as condições associadas com cada propriedade perceptível podem ser especificadas (as condições sob as quais uma propriedade é informada a um agente quando a percepção do agente é enviada a este). Na definição de percepções é possível especificar atributos de células, conteúdo de células, atributos de recursos ou agentes e variáveis de controle do ambiente. O ELMS possibilita que agentes tenham como percepção algumas variáveis de controle do ambiente, como por exemplo o passo corrente da simulação, que pode servir para dar aos agentes uma noção abstrata de "tempo". Porém, variáveis de controle da simulação normalmente não são acessadas pelo agente. Contudo o projetista pode dar acesso a estas variáveis de controle, se isto for útil em algum caso específico.

Uma ação é definida como uma seqüência de atribuições nas propriedades (do ambiente, recursos ou agentes) que esta causa, e as precondições que devem ser satisfeitas para que a ação possa ser realizada no ambiente.

5.4 Construções da Linguagem ELMS

A linguagem ELMS usa uma sintaxe XML[1], porém a especificação de ambientes é feita através da interface da plataforma MASSOC, que será abordada na seção 6.1. Todavia, a especificação do ambiente pode ser feita em XML em um simples editor de texto, ou alguma outra ferramenta, se o usuário assim preferir.

Uma especificação de ambiente em ELMS pode ser formada de nove tipos de definições. Existem construções específicas da linguagem para cada uma dessas definições, sendo que algumas delas podem ser repetidas várias vezes, porém não misturadas[2]. A ordem usual das definições é como segue:

1. Opções da grade – Definição da grade (opcional) para representação espacial do ambiente.

2. Recursos – Definição das classes de recursos (objetos) que podem estar presentes no ambiente.

3. Agentes – Definição das classes agentes que podem estar presentes no ambiente da simulação.

4. Percepções – Definição dos tipos de percepções que podem ser realizadas no ambiente definido para a simulação.

5. Ações – Definição das ações que os agentes podem realizar no ambiente.

6. Reações – Definição das reações que os recursos podem ocasionar no ambiente.

7. Observáveis – Definição das propriedades que serão enviadas como resultado de um passo ou da simulação.

8. Valores da Simulação – Definição de valores correntes das propriedades dos elementos instanciados.

9. Inicialização – Definição de comandos que serão executados no ambiente antes do início da simulação.

Os ítens listados acima serão detalhados nas subseções que seguem.

5.4.1 Opções da Grade

A existência de uma grade, tal como sua definição, é opcional. A grade pode ser bidimensional ou tridimensional, sendo os parâmetros as dimensões dos eixos X, Y e Z. Ainda na definição da grade do ambiente, uma lista de atributos pode ser definida: os atributos definidos nesta seção serão replicados para cada célula da grade. A declaração de um atributo possui um nome, o tipo de propriedade (integer, float, string ou boolean) e um valor inicial opcional. Nas expressões de inicialização (ver Seção 5.4.11) dos atributos de cada célula podem ser utilizadas as palavras reservadas *X*, *Y* e *Z* para referenciar a posição da célula.

[1] A definição da linguagem ELMS (DTD) pode ser visto no Anexo A.

[2] Lembre-se que as definições são editadas em qualquer ordem em uma interface gráfica, que irá gerar a seqüência de definições na ordem apropriada.

O exemplo abaixo mostra a definição de uma grade onde o eixo X tem dimensão 10 (pode assumir os valores de 0 a 9), e os eixos Y e Z dimensões 7 e 8, respectivamente. A dimensão do eixo Z é opcional. Cada uma das células da grade definida abaixo terá como atributos uma variável inteira e outra booleana. A variável inteira tem como nome `indiceCor` e valor inicial igual a zero. A variável booleana tem como nome `ocupado` e tem `falso` como valor inicial. Cada célula possui duas reações possíveis, `reacao1` e `r2`, a serem definidas posteriormente (ver Seção 5.4.6).

```
<DEFGRID SIZEX="10" SIZEY = "8" SIZEZ = "9">
    <INTEGER NAME = "indiceCor"> 0 </INTEGER>
    <BOOLEAN NAME = "ocupado"> "FALSE" </BOOLEAN>
    <REACTIONS LIST ="reacao1 r2">
</DEFGRID>
```

5.4.2 Recursos

Na seção de definição de recursos, são definidas as classes de recursos ou objetos, que podem ter diversas instâncias alocadas na simulação. Uma definição de classe de recurso possui um nome, uma lista de atributos e uma lista de reações. Os atributos são definidos da mesma maneira que os atributos das células (definindo nome, tipo e valor inicial). As reações que uma classe de recursos pode ter é dada por uma lista de nomes que identificam as reações (ver seção 5.4.6). Note que uma mesma reação pode aparecer em diferentes classes de recursos.

O exemplo abaixo mostra a definição de um recurso que teria como nome `recurso` e seria capaz de executar as reações `reacao1` e `reacao2`, que são executadas quando suas precondições forem satisfeitas. Nas expressões de inicialização (ver seção 5.4.11) dos atributos de cada recurso, podem ser utilizadas as palavras reservadas `SELF` e `SELFCLASS`, que representam o índice[3] de uma instância específica e o nome da classe de recurso, respectivamente. O recurso do exemplo abaixo tem como atributos duas propriedades do tipo string e uma propriedade inteira. A primeira propriedade *string* tem nome `nome` e valor inicial `agua`, a segunda *string* tem nome `tipo` e valor inicial igual a `mineral`, a propriedade inteira tem nome `qtde` e valor inicial igual a 10.

```
<RESOURCE NAME="recurso">
    <STRING NAME = "nome" VALUE = "agua">
    <STRING NAME = "tipo" VALUE ="mineral"/>
    <INTEGER NAME = "qtde">10</INTEGER>
    <REACTIONS LIST="reacao1 reacao2"/>
</RESOURCE>
```

5.4.3 Agentes

Nesta parte da definição de ambiente, são definidas as classes de agentes que podem participar das simulações em que este ambiente seja usado. Uma especificação de uma classe de agente contem seu nome, uma lista de atributos[4], uma lista de ações e uma lista

[3] Quando uma instância de um recurso é criada ou um agente entra na sociedade, um índice é atribuído a ele. Por exemplo, a décima instância de uma classe de recurso a ser criada ter índice 9 (a primeira tem índice 0).

[4] Propriedades dos agentes que são perceptíveis ou características dos agentes enquanto parte do ambiente.

de percepções permitidas para este agente. É necessário especificar uma lista de nomes ações que os agentes desta classe têm permissão para executar no ambiente. O conjunto de percepções é uma lista de nomes de percepções que estão disponíveis para uma classe de agente (o tipo de informação que o ambiente irá enviar para estes agentes a cada ciclo de percepção/ação na simulação). Assim como as reações dos recursos, os mesmos nomes de percepções e ações podem aparecer em várias definições, de forma que estes podem ser reaproveitados em diferentes classe de agentes. Da mesma forma que com os recursos, nas expressões de inicialização (ver seção 5.4.11) dos atributos de cada recurso podem ser utilizadas as palavras reservadas `SELF` e `SELFCLASS` que representam respectivamente o índice de uma instância específica e o nome da classe do agente.

O exemplo abaixo apresenta uma definição de uma classe de agentes de nome `agente` que é capaz de realizar as ações `acao1` e `andar`, de acordo com suas precondições. Este agente também é capaz de realizar as percepções `visao` e `tato`, também dependentes de suas precondições. Essa classe de agente tem como atributo uma variável inteira de nome `id` que será inicializada com o número da instância da classe de agente. Outro atributo desta classe é o valor booleano de nome `acordado` que tem verdadeiro como valor inicial. As percepções e ações serão detalhadas nas seções que seguem.

```
<AGENT NAME="agente">
    <INTEGER NAME="id"> "SELF" </INTEGER>
    <BOOLEAN NAME ="acordado"> "TRUE" </BOOLEAN>
    <ACTIONS LIST="acao1 andar"/>
    <PERCEPTIONS LIST = "visao tato"/>
</AGENT>
```

5.4.4 Percepções

Nesta seção são especificadas as percepções referidas nas definições de agentes. Uma definição de percepção é formada por um nome, uma lista opcional de precondições, e uma lista de nomes de propriedades. Estas propriedades podem ser quaisquer dos atributos associados com definições de recursos, agentes, células da grade ou valores de controle da simulação. Se todas as precondições são satisfeitas, então os valores das propriedades listadas serão enviadas para o agente como o resultados de sua percepção do ambiente. Note que a percepção pode ser baseada na posição espacial do agente na grade, mas isso não é obrigatório. Existem várias possibilidades que de definições que podem ser utilizadas de acordo com as necessidades do projetista do ambiente.

No exemplo que segue, está definida uma percepção de nome `visao`, cuja precondição é que o atributo `acordado` tenha valor igual a `TRUE`. O conteúdo de todas as células é enviado para cada agente que realiza este tipo de percepção e tem a precondição satisfeita. Além das palavras reservadas `SELF` e `SELFCLASS`, podem ser utilizadas as palavras reservadas `ALL` e `CONTENTS`. A palavra reservada `ALL` pode referenciar todas os índices de instâncias de uma classe ou todas as posições de um eixo da grade. A palavra reservada `CONTENTS` referencia o conteúdo de uma célula.

```
<PERCEPTION NAME="visao">
   <PRECONDITION>
     <EQUAL>
        <OPERAND>
            <ELEMENT_ATT NAME      = "SELFCLASS"
```

```
                                ATTRIBUTE = "acordado">
                        <INDEX>"SELF"</INDEX>
                    </ELEMENT_ATT>
                </OPERAND>
                <OPERAND> "TRUE" </OPERAND>
            </EQUAL>
        </PRECONDITION>
        <CELL_ATT ATTRIBUTE = "CONTENTS">
            <X> "ALL" </X>   <Y> "ALL" </Y>
        </CELL_ATT>
    </PERCEPTION>
```

Para definir percepções com variáveis de controle do ambiente, deve-se inserir da definição da percepção uma chave para atributo de elemento, colocando no campo para nome da classe a palavra reservada `ENV` e em atributo o nome da variável de controle, conforme o exemplo que segue:

```
<ELEMENT_ATT NAME = "ENV"  ATTRIBUTE = "STEP"/>
```

No exemplo acima, será enviado ao agente o conteúdo da variável de controle de ambiente `STEP` que indica o passo atual da simulação, no caso de uma simulação síncrona (ver Seção 5.6).

5.4.5 Ações

Nesta parte da definição do ambiente são definidas as ações são referenciadas nas definições de agentes. Uma definição de ação tem um nome, uma lista opcional de parâmetros, uma lista opcional de precondições, e uma seqüência de comandos que determinam quais mudanças no ambiente a ação causa. A lista de parâmetros informa quais parâmetros serão recebidos do agente para a execução deste tipo de ação. A seqüência de comandos pode ser formada por atribuições de valores para atributos; instanciação ou remoção de recursos; além de alocação ou reposicionamento de instâncias de agentes ou recursos na grade. Os comandos estão detalhados na seção 5.4.13 Se as precondições forem totalmente satisfeitas, então todos os comandos da seqüência de comandos serão executados, na ordem em que foram definidos, ocasionando mudanças no ambiente. É importante enfatizar que as ações devem ser atômicas[5], apesar disto não ser obrigatório; porém, ao projetar o ambiente desta forma, a simulação se torna mais consistente. Os parâmetros são acessados pelos nomes e devem ser enviados pelo agente, na ordem em que foram definidos.

O exemplo abaixo define uma ação chamada `moverDireita`. Esta ação recebe como parâmetro uma valor inteiro que é referenciado pelo rótulo `CASAS`. Esta ação será executada, caso o parâmetro `CASAS` tenha um valor menor do que três, condição que está explicitada na seção `PRECONDITION` do código abaixo. A ação irá mover o agente que deseja realizar a ação para a célula à direita da célula atual do agente, dependendo do parâmetro que representa a posição relativa à direita para a qual o agente irá se mover.

```
<ACTION NAME="moverDireita">
    <PARAMETER NAME="CASAS" TYPE="INTEGER"/>
```

[5]Como o raciocínio, planejamento e o encadeamento de ações são atividades internas dos agentes, as ações definidas sobre o ambiente devem ser, em princípio, curtas e simples.

```
    <PRECONDITION>
        <LESSTHAN>
            <OPERAND> "CASAS" </OPERAND>
            <OPERAND> 3 </OPERAND>
        </LESSTHAN>
    </PRECONDITION>
    <MOVE>
        <ELEMENT NAME = "SELFCLASS">
                <INDEX>"SELF"</INDEX>
        </ELEMENT>
        <FROM>
            <CELL>
                <X>+0</X> <Y>+0</Y>
            </CELL>
        </FROM>
        <TO>
            <CELL>
                <X>"CASAS"</X> <Y>+0</Y>
            </CELL>
        </TO>
    </MOVE>
</ACTION>
```

5.4.6 Reações

Esta seção define as possíveis reações dos recursos no ambiente. Para cada reação é definido um nome, uma lista de precondições, e uma seqüência de comandos. Os comandos são definidos da mesma forma que para as ações, conforme visto na seção anterior. Todas as expressões na lista de precondições devem ser satisfeitas para a execução da respectiva reação. De maneira diferente das ações, onde apenas uma ação é selecionada pelo agente, todas as reações associadas aos recursos que tenham suas precondições satisfeitas serão executadas. As reações são executadas por recursos, na ordem em que aparecem na lista de reações dos recursos.

5.4.7 Observáveis

Nesta seção são definidas quais propriedades dos agentes, recursos e ambiente serão enviadas como resultados da simulação. A freqüência de saída destes resultados parciais depende do modo de execução e de parâmetros definidos pelo usuário (ver seção 5.6). As propriedades listadas podem ser aquelas associadas com qualquer instância de recursos, agentes, células da grade e valores de controle da simulação.

No exemplo abaixo, a cada vez que forem geradas as saídas para o usuário, esta terá como conteúdo o valor da propriedade `indiceCor` e o conteúdo de todas as células da grade. O conteúdo de uma célula é referido pela palavra reservada `CONTENTS`.

```
<OBSERVABLE>
    <CELL_ATT ATTRIBUTE = "indiceCor">
        <X> "ALL" </X>  <Y> "ALL" </Y>
    </CELL_ATT>
```

```
    <CELL_ATT ATTRIBUTE = "CONTENTS">
       <X> "ALL" </X>  <Y> "ALL" </Y>
    </CELL_ATT>
</OBSERVABLE>
```

5.4.8 Valores da Simulação

Esta seção define os valores correntes das variáveis de controle do ambiente e dos valores das propriedades de cada instância dos agentes e recursos. Também nessa seção, a posição dos agentes e recursos na grade são definidos, caso exista uma grade. Os valores das variáveis de controle do ambiente são definidos com nomes pré-definidos. Os valores para propriedades de agentes e recursos são acessados pelo nome do elemento (agente ou recurso), índice da instância específica e o nome da propriedade. As posições dos agentes e recursos na grade são determinados por construções específicas da linguagem, como nos exemplos abaixo:

```
<INSTANCE NAME = "agente" INDEX = "3">
    <PROPERTY NAME = "cor" VALUE = "AZUL"/>
</INSTANCE>
```

O exemplo acima define o valor corrente da propriedade `cor` como sendo `AZUL`, para a instância de índice 3 da classe `agente`.

```
<GRID X="1" Y="2">
    <PROPERTY NAME= "ocupado" = "TRUE">
    <CONTENT NAME = "agente" INDEX = "3"/>
</GRID>
```

O código acima indica que, na posição $(1, 2)$ da grade, a propriedade `ocupado` possui valor ”TRUE”. Este trecho indica também que a instância de índice 3 da classe `agente` está presente na célula.

5.4.9 Inicialização

Nesta seção, recursos do ambiente podem ser instanciados e posicionados na grade (recursos podem ser criados dinamicamente no ambiente). Todos os comandos desta seção são executados antes do início da simulação. Os comandos possíveis são apresentados na seção 5.4.13.

5.4.10 Declaração de Atributos

Como mencionado anteriormente, os tipos dos atributos possíveis são: booleano (BOOLEAN), inteiro (INTEGER), ponto flutuante (FLOAT)[6] e texto (STRING). A declaração dos atributos é composta pela *tag* do tipo do atributo que contém o nome e valor inicial da propriedade. Com exceção dos atributos do tipo texto, é possível inicializar as propriedades com expressões (vistos a seguir).

[6]Este tipo de atributo deve ser evitado dentro das percepções, pois não é suportado pela versão corrente do AgentSpeak(XL).

5.4.11 Expressões

As expressões possíveis na linguagem ELMS são formadas por operadores matemáticas, lógicos e relacionais. Também é possível utilizar os comandos RAND e RANDOM. Entre os operadores relacionais estão incluídas a comparação de igualdade (EQUAL), desigualdade (UNEQUAL), maior que (GREATERTHAN) e menor que (LESSTHAN). Entre as operações matemáticas estão as operações de soma (ADD), subtração (SUBTRACT), multiplicação (MULTIPLY), divisão (DIVIDE), resto de divisão (MOD), somatório (SUM) e produtório (PROD). Entre os operadores lógicos, estão disponíveis conjunção (AND), disjunção (OR) e negação (NOT).

Os operadores relacionais possuem dois operandos, sendo que cada operando pode ser uma operação, uma constante ou um atributo de elemento ou célula. As operações lógicas de conjunção e disjunção aceitam dois operandos, onde cada operando pode ser uma outra operação, uma constante ou um valor de atributo ou célula. A operação de negação é unária.

As operações matemáticas de soma, subtração, multiplicação, divisão e resto de divisão aceitam dois operandos, onde cada operando pode ser outra operação, uma constante ou o valor de um atributo de elemento ou célula. As operações de somatório e produtório aceitam como operador um atributo de uma classes de recurso ou agentes, tendo como resultado a soma ou produto dos valores do atributo em todas as instâncias desta classe. As operações de somatório e produtório também aceitam como parâmetro um atributo da grade, sendo possível definir-se os eixos que serão percorridos na realização da soma ou produto. Em versões futuras, planeja-se possibilitar o uso de intervalos na realização destas operações.

No exemplo a seguir é realizada a soma do atributo total_a de todas as células da grade.

```
<SUM>
    <CELL_ATT ATTRIBUTE = "total_a">
        <X>"ALL"</X> <Y>"ALL"</Y>
    </CELL_ATT>
</SUM>
```

O comando RAND gera um número randômico entre 0 e 1, enquanto o comando RANDOM tem como parâmetros um valor mínimo (inclusive) e um valor máximo (exclusivo), gerando um número randômico neste intervalo. Estes comandos podem ser utilizados em quase todas as seções do código, com exceção da seção de "valores da simulação". No exemplo abaixo, será gerado um valor randômico entre 0 e 9.

```
<RANDOM MIN="0" MAX="10"/>
```

5.4.12 Precondições

As precondições tanto para ações, reações e percepções são definidas através de um conjunto de operações cujos resultados devem ser TRUE. Caso não seja definido explicitamente um operador lógico (entre os mencionados na seção anterior), será realizada uma operação AND com os resultados destas operações. Por exemplo, o seguinte código:

```
<PERCEPTION>
    <PRECONDITION>
        <EQUAL>.........</EQUAL>
```

```
        <GREATERTHAN>....</GREATERTHAN>
    </PRECONDITION>
    ...
</PERCEPTION>
```

Tem o mesmo significado que o código abaixo:

```
<PERCEPTION>
    <PRECONDITION>
        <AND>
            <OPERAND>
                <EQUAL>.........</EQUAL>
            </OPERAND>
            <OPERAND>
                <GREATERTHAN>....</GREATERTHAN>
            </OPERAND>
        </AND>
    </PRECONDITION>
    ...
</PERCEPTION>
```

5.4.13 Comandos

Os comandos possíveis na linguagem ELMS são a atribuição (`ASSIGN`), inserção de elemento[7] na grade (`IN`), o comando de inserção randômica (`IN_RAND`), remoção de elementos da célula (`OUT`), realocação de elementos (`MOVE`), instanciação de elementos (`NEW`) e exclusão de instância (`DELETE`).

A atribuição tem como parâmetros um atributo que receberá o valor e uma expressão cujo resultado será atribuído. Os comandos de inserção e remoção de elementos na grade têm como parâmetro o índice de um elemento já instanciado e uma posição na grade onde o elemento será alocado ou removido. Há também o comando de inserção randômica, que possui como parâmetros o elemento a ser inserido (nome da classe e o índice) e a precondição que deve ser atendida pela célula para que a inserção (em uma célula aleatória mas que satisfaça a precondição) seja realizada.

O comando de realocação possui como parâmetros um elemento, uma posição de origem e outra de destino. Deve-se lembrar que um elemento pode ocupar mais de uma posição na grade e os elemento têm como ponto de referência[8] a primeira célula onde foi inserido. Com o comando de realocação, estará sendo realizada a movimentação de todo o elemento, através da mudança do ponto de referência. Tanto a especificação do índice dos elementos como as posições na grade aceitam expressões, constantes ou atributos de células ou elementos.

O comando de instanciação possui como parâmetros a classe, o número de instâncias a ser criado, uma posição (opcional) onde as instâncias serão posicionas e um rótulo (opcional) para uma variável do tipo inteira à qual será atribuído o índice da primeira instância criada. No exemplo abaixo são instanciados dois recursos do tipo `recurso1` que serão alocados na posição (3,4) da grade. Será armazenado em uma variável inteira chamada `newRef` o índice da primeira instância criada. Finalmente, o comando de exclusão de

[7] Refere-se como "elemento" tanto objetos (recursos) como agentes.

[8] O ponto de referência usado para cálculo de posições relativas.

instância possui como parâmetro o nome da classe e uma expressão que indica o índice da instância a ser excluída.

```
<NEW NAME= "recurso1" REFERENCE="newRef">
    <N>2</N>
    <CELL>
         <X>3</X>
         <Y>4</Y>
    </CELL>
</NEW>
```

5.5 Formato das Percepções ELMS

As percepções que o ambiente envia para os agentes seguem o formato decrenças AgentSpeak(L). Nesta seção será descrito o formato destas percepções, que são reconhecidas pelo interpretador AgentSpeak(XL), visto na seção 2.4.

As percepções são enviadas para os agentes no seguinte formato:

`@set(world(`*percepção1*`,`*percepção2*`,...));`

As percepções que compõem o conjunto de percepções podem ser atributos de células, conteúdo de células, atributos de recursos, atributos de agentes ou valores das variáveis de controle do ambiente. Cada átomo de crença, que representa uma informação da percepção, é separado do seguinte por vírgula. O formato de cada tipo de percepção é explicitado nas seções a seguir.

5.5.1 Atributos de Células

O formato de uma percepção de um atributo de célula segue a seguinte forma:

`cell(position(`X`,`Y`,`Z`),`*atributo*`(`*Valor*`))`

Onde X,Y e Z representam a posição da célula, *atributo* representa o nome do atributo (propriedade) da célula e *Valor* representa o valor da propriedade. Caso a grade seja bidimensional não será incluído o valor de Z. A coordenada da célula terá valores absolutos se a percepção foi definida com valores absolutos e terá valores relativos à posição do agente caso contrário. Caso a propriedade seja do tipo booleana, será enviado `t` para verdadeiro[9] e `f` para falso.

5.5.2 Atributos de Elementos

No caso de percepções de atributos de elementos, o formato do átomo de crença é:

classe`(instance(`N`),`*atributo*`(`*Valor*`))`

Onde *classe* denota a classe do elemento, N representa o índice da instância da classe, *atributo* indica a propriedade do elemento e *Valor* indicando o valor da propriedade.

[9] "true" é palavra reservada do interpretador AgentSpeak(XL).

5.5.3 Conteúdo de Célula

O conteúdo de uma célula é enviado para o agente conforme o seguinte formato:

classe`(instance(`N`)position(`X,Y,Z`))`

Onde *classe* representa a classe do elemento, N indica o índice da instância e X, Y e Z denotam a posição, sendo que Z será enviado somente se a grade seja tridimensional.

5.5.4 Variáveis de Controle do Ambiente

As percepções contendo valores das variáveis de controle do ambiente seguem o seguinte formato:

atributo (*Valor*)

Onde *atributo* indica o nome da variável e *Valor* representa o valor da propriedade.

5.6 Execução de Ambientes ELMS

Em uma simulação que faça uso da linguagem ELMS, o ambiente é controlado por um processo que é gerado a partir da especificação do ambiente feita nesta linguagem. Este processo controla o acesso e as mudanças em uma estrutura de dados que representa o ambiente (apenas o controlador do ambiente tem acesso à estrutura de dados). A estrutura de dados que representa o ambiente é gerada pelo interpretador da linguagem ELMS a partir da especificação feita nesta linguagem. A linguagem possui construções que permitem a geração de uma descrição não apenas do estado inicial do ambiente de uma simulação, mas também a descrição instantânea de uma simulação em execução. O processo que controla a execução do ambiente pode gerar uma descrição instantânea do ambiente (na linguagem ELMS) a partir das estruturas de dados que representam o ambiente. Esta característica permite ao usuário salvar a simulação para uma execução posterior, ou fazer mudanças neste ambiente (através da interface ou diretamente na especificação ELMS). Isto também pode ser útil para gerar formas mais complexas de visualização de simulações multiagentes.

É possível selecionar a execução do ambiente no modo síncrono e assíncrono, selecionado de acordo com a necessidade do projetista da simulação. A simulação síncrona, nesta abordagem, tem como característica a execução em forma de passos, de forma a realizar uma execução "justa", com a execução das ações dos agentes em uma ordem aleatória. Em contrapartida, no modo de execução assíncrono, é priorizado o agente que responde mais rápido às percepções, seja por possuir melhores recursos computacionais ou pela performance de seu código[10].

Enquanto no modo de execução síncrono a duração da simulação é definida em "passos", no modo assíncrono é definido um tempo limite para toda a simulação. Tanto o tempo limite da simulação assíncrona como o número de passos da simulação síncrona são definidos pelo usuário através da interface gráfica.

No modo de execução síncrono, o processo que controla o ambiente, envia para todos os agentes da simulação a cada ciclo de execução os dados das percepções as quais estes têm acesso (da forma como foi especificado com o ELMS). Neste modo de execução o ambiente irá aguardar que todos os agentes escolham a ação para execução[11] em um

[10] A ordem de execução é importante, pois a execução de uma ação pode bloquear a execução de outras.

[11] É possível definir um tempo limite (*timeout*) que o ambiente irá aguardar as ações.

dado passo da simulação, e somente então as ações são executadas. Isto permite que a performance da rede e da CPU não interfiram nos resultados da simulação síncrona.

No modo síncrono a execução é feita da seguinte forma:

1. são executados os comandos na seção de inicialização antes do início da simulação;
2. são verificadas quais percepções da lista de percepções dos agentes estão realmente disponíveis;
3. são enviadas as percepções resultantes (cujas condições foram satisfeitas) para os respectivos agentes;
4. o ambiente aguarda até receber de todos os agentes as ações que cada um escolheu[12] executar naquele ciclo da simulação;
5. a fila de ações é "embaralhada" para garantir que cada agente tenha a chance de executar sua ação primeiro;
6. verificar se a ação que está na primeira posição da fila satisfaz a sua respectiva precondição para execução;
7. executar a ação, caso a precondição tenha sido satisfeita;
8. caso a ação não satisfaça a precondição, uma mensagem com conteúdo "`@fail`" é enviado para o agente;
9. a primeira ação da fila é removida;
10. caso ainda existam ações na fila, retorna-se ao passo 6;
11. são verificadas as reações definidas no ambiente que serão executadas caso suas precondições tenham sido atendidas;
12. são enviados os valores das propriedades observáveis[13] para a interface ou para o arquivo de saída escolhido;
13. caso a contagem de passos não tenha atingido o valor definido pelo usuário, retorna-se ao passo 2.

A simulação assíncrona é executada da seguinte forma:

1. são executados os comandos na seção de inicialização antes do início da simulação;
2. são verificadas quais percepções da lista de percepções dos agentes estão realmente disponíveis;
3. são enviadas as percepções resultantes (cujas condições foram satisfeitas) para os respectivos agentes;
4. aguarda-se a recepção de ações dos agentes e armazenando-as em ordem de chegada em uma fila;

[12] O agente deve mandar uma mensagem com conteúdo "`true`" caso não queira executar nenhuma ação.
[13] Conjunto de dados de saída, definido para cada ambiente, ver seção 5.4.7.

5. concorrentemente à recepção de ações, é executada a primeira ação da fila, caso sua precondição seja satisfeita;

6. caso a ação não satisfaça a sua precondição, uma mensagem com conteúdo "`@fail`" é enviado para o agente;

7. são verificadas as reações definidas no ambiente que serão executadas caso suas precondições tenham sido atendidas;

8. é enviada uma nova percepção para o agente[14] cuja ação foi processada;

9. a primeira ação da fila é removida;

10. a cada n (n definido pelo usuário) ações executadas, são enviados os valores das propriedades observáveis para a interface ou para o arquivo de saída escolhido;

11. caso a condição de término definida pelo usuário não tenha sido alcançada, retorna-se ao passo 4.

Em ambos os casos acima, é realizada de maneira concorrente com a execução do ambiente o controle da entrada e saída de agentes da simulação. As percepções são enviadas através do SACI em mensagens KQML que contêm uma lista de átomos de crença AgentSpeak(L).

5.7 Expressividade da Linguagem ELMS

No início do Capítulo 5 afirmou-se que, com esta linguagem, é possível especificar ambientes que são, do ponto de vista dos agentes, inacessíveis, não-determinísticos, não-episódicos, dinâmicos, porém devem ser discretos. Nesta seção, mostra-se como cada uma destas caracteristicas de ambientes podem ser modeladas em ELMS.

Inacessível: um ambiente acessível é um ambiente onde o agente tem acesso ao estado completo do ambiente e um ambiente inacessível é onde o agente ter acesso restrito ao estado do ambiente. Desta forma, na modelagem com o ELMS, o agente tem acesso apenas às propriedades do ambiente que foram escolhidas pelo projetista do ambiente, na definição das percepções.

Não-determinístico: Como os ambientes ELMS podem ser inacessíveis e também devido ao fato de que podem existir vários agentes realizando ações simultaneamente, do ponto de vista de um agente, os ambientes ELMS podem parecer não-determinísticos.

Não-episódico: nos ambientes ELMS, o estado corrente do ambiente é o produto das ações realizadas pelos agentes no estado anterior. Porém como tratam-se de agentes cognitivos, as ações tomadas por um agente poderão interferir em suas ações futuras, já que é um processo interno ao agente que determina como ele reaje a cada ciclo de execução.

[14] Se o agente desejar uma nova percepção, mas não queira realizar uma ação, deve enviar uma mensagem com conteúdo "`true`".

Dinâmico: No modo de execução assíncrono (conforme visto na seção 5.6), o ambiente pode sofrer alterações enquanto um agente estiver deliberando, devido à ações de outros agentes presentes no ambiente. Mesmo no modo síncrono, apesar do sistema aguardar pelas ações de todos os agentes, uma ação pode gerar uma mudança no ambiente que bloqueia a ação de outro agente, o que do ponto de vista do agente bloqueado, faria com que o ambiente parecesse dinâmico.

Discreto: como o "espaço físico" do ambiente é representado através de uma grade, o ambiente é discretizado em coordenadas inteiras.

5.8 Implementação

Esta seção trata da implementação do interpretador da linguagem ELMS. Aspectos da implementação de outras tecnologias envolvidas como o AgentSpeak(XL) e o SACI serão mencionados de maneira sucinta, apenas para complementar as informações sobre a implementação do interpretador da linguagem ELMS.

O interpretador da linguagem ELMS foi desenvolvido utilizando a linguagem de programação JAVA. A linguagem JAVA foi escolhida por ser suportada em uma grande variedade de plataformas. Além disto, a interface gráfica da plataforma MASSOC (seção 6) e o SACI são implementados também na linguagem JAVA. O interpretador da linguagem AgentSpeak(XL) (seção 2.4) foi implementado em C++ e sua versão corrente funciona em sistemas baseados em Linux.

A linguagem ELMS possui um sintaxe baseada em XML (eXtensible Markup Language). A escolha da sintaxe XML foi feita com vistas a permitir a futura integração da plataforma MASSOC com ferramentas avançadas para a visualização das simulações (inclusive via *web*). Além disto, o uso de XML facilita a interpretação do código fonte, devido ao fato de que a linguagem JAVA, assim como várias outras, possui vários recursos para a interpretação de XML. Da mesma forma, a geração de código também é facilitada por uma grande variedade de ferramentas que manipulam este formato. É importante lembrar que a sintaxe XML, utilizada no ELMS, é *case-sensitive*, ou seja, letras maiúsculas diferem das minúsculas.

5.8.1 Estrutura do Interpretador

A implementação do interpretador da linguagem ELMS esta estruturada basicamente em duas partes. A primeira parte faz a leitura do código-fonte ELMS, especificado através de uma interface gráfica ou em um editor de texto puro. Esta etapa inicializa as estruturas de dados definidas para a representação de um ambiente multi-agente com as informações fornecidas pelo usuário. A segunda parte realiza a execução do ambiente, manipulando a estrutura de dados que representa o ambiente. Estas duas estruturas foram implementadas de maneira independente, de forma a possibilitar a utilização ou alteração de cada uma delas individualmente. Por exemplo, é possível criar uma nova sintaxe para linguagem apenas substituindo-se a parte que faz a leitura do código fonte e preenche as estruturas de dados, sem contudo precisar alterar o processo de execução de ambientes ELMS. A parte do interpretador que faz a inicialização das estruturas de dados a partir do código fonte ELMS, está agrupada em um pacote JAVA chamado `massoc.elms` contendo cerca de 6000 linhas de código.

No Anexo C encontra-se uma listagem das principais classes desenvolvidas para o interpretador ELMS, com uma breve descrição de cada uma. No Anexo D é possível

verificar-se os detalhes de como é feita a comunicação entre ambiente e agentes.

5.8.2 Integração do Interpretador ELMS e AgentSpeak(XL)

O interpretador AgentSpeak(XL), em sua versão corrente, utiliza *sockets* para recepção da percepção e envio da ação escolhida pelo agente. Como o AgentSpeak(XL) é implementado em C++, enquanto o ELMS e SACI foram implementados em JAVA, foi implementada uma camada de software (*wrapper*) que gera um processo executando o interpretador AgentSpeak(XL) e que implementa um "agente SACI", tornando possível o uso do AgentSpeak(XL) em nossa plataforma[15]. Esta camada de software foi implementada em JAVA, está agrupada no pacote `massoc.elms.utils`, e tem cerca de 600 linhas. Como o objetivo do protótipo do interpretador ELMS seria testar a viabilidade do uso da linguagem ELMS para a simulação de ambientes, optou-se por utilizar comunicação via *sockets* neste protótipo por facilitar a implementação. Em versões futuras pretende-se fazer uso direto do interpretador AgentSpeak(XL), para eliminar a comunicação via *sockets* entre o *wrapper* e o interpretador AgentSpeak(XL).

No uso do ELMS em conjunto com o AgentSpeak(XL), conforme a proposta da abordagem MASSOC, deve-se utilizar strings que iniciem com letras minúsculas para nomear classes de agentes, recursos e ações nas respectivas definições ELMS. Isto deve ser feito para que os agentes AgentSpeak(XL) não interpretem de maneira errada as percepções enviadas pelo ambiente, pois em AgentSpeak(XL), assim como em PROLOG, inicial maiúscula é usada para denotar variáveis lógicas.

5.8.3 Estrutura de Dados

A estrutura de dados que armazena um ambiente é organizada em diversas classes, tendo como classe principal a classe `EnvironmnentData`. Esta classe armazena as seguintes informações:

- o nome dado pelo usuário ao ambiente, nome que será utilizado também como nome da sociedade SACI a ser criada para a simulação;
- informações sobre a grade (opcional) que representa o espaço físico da simulação;
- vetor com os nomes das reações que as células podem realizar;
- vetor com os nomes das classes de recursos que realizam reações;
- vetor com os nomes das classes de agentes que participam desta simulação;
- vetor de objetos `Instance`, que referenciam uma instância de agente ou recurso, para armazenar uma referência aos agentes que estão presentes na simulação porém não estão associados a uma "mente" externa, por exemplo um interpretador AgentSpeak(XL) (no caso do MASSOC);
- um vetor que armazena quais dados devem ser enviados como saída da simulação (observáveis);
- um gerador de números randômicos (um objeto `Random`) para a simulação, cuja semente é inicializada no início da simulação;

[15] O Anexo D mostra as mensagens (KQML) trocadas por este *wrapper* que permite o uso do interpretador AgentSpeak(XL) como um agente SACI.

- uma estrutura para armazenamento de variáveis de controle do ambiente, por exemplo o passo atual da simulação;

- uma tabela *hash* que armazena referências para as definições de classes de agentes, recursos, tipos de percepções, e reações e ações que podem ser realizadas no ambiente da simulação.

As estruturas de dados que armazenam tanto as definições de classes de recursos como a de agentes, são especializações da classe `ElmsClass`, que armazena as seguintes informações:

- o nome dado a classe de recurso ou agente;

- um vetor dinâmico que armazena referências para as instâncias desta classe;

- informações sobre os atributos da classe de recurso ou agente, tais como nome, tipo e valor inicial.

A estrutura de dados que armazena as definições de classes de recursos (classe `ResourceDef`) armazena, além das informações de sua super-classe, um vetor com os nomes das reações que este recurso é capaz de realizar. A estrutura que armazena as definições de classes de agentes (classe `AgentDef`) armazena, além das informações de sua super-classe, dois vetores: um com os nomes das percepções a que o agente tem acesso e outro com os nomes das ações que o agente é capaz de realizar no ambiente.

As estruturas de dados que armazenam as informações sobre as reações (classe `ReactionDef`) possui as seguintes informações:

- o nome da reação;

- uma árvore que armazena as precondições para que a reação seja ativada;

- o vetor de comandos que devem ser executados caso a reação seja ativada.

O armazenamento das definições de ações são feitas pela classe `ActionDef` que estende a classe `ReactionDef`. Esta classe possui, além das propriedades herdadas de sua super-classe, uma matriz de duas colunas onde na primeira coluna é armazenado o nome do parâmetro da ação e na segunda coluna é armazenado um número inteiro que identifica o tipo do parâmetro da ação.

Para o armazenamento das definições de percepções foi definida a classe `PerceptionDef`, que armazena as seguintes informações:

- o nome da percepção;

- uma árvore que armazena as precondições para que a percepção possa ser realizada;

- um vetor com as propriedades que devem ser enviadas ao agente caso as condições da percepção sejam satisfeitas.

5.9 Exemplo

Nesta seção será apresentada a modelagem do ambiente da simulação apresentada no artigo *"Understanding the functions of norms in social groups through simulation"* de Conte e Castelfranchi (1995a), que foi posteriormente estendido em (CASTELFRANCHI; CONTE; PAOLUCCI, 1998). Será apresentada de forma resumida o ambiente da simulação adaptada do artigo original. Esta seção tem como objetivo exemplificar a modelagem de um ambiente através do uso da linguagem ELMS, portanto aspectos relacionados à modelagem do comportamentos dos agentes existentes nesta simulação não serão abordados.

5.9.1 Descrição do Ambiente da Simulação

Na simulação proposta, existem agentes que têm como objetivo comer as comidas existentes no ambiente. Caso o agente não perceba nenhuma comida livre, este poderá em alguns casos, agredir um agente que estiver mais próximo caso este esteja comendo; o objetivo neste caso é roubar a comida do agente, caso o agente que estiver comendo perca a disputa.

Nesta simulação é utilizada uma grade bidimensional de tamanho $[10 \times 10]$. Em cada célula pode haver no máximo um agente, porém comida e um agente podem estar presentes em uma mesma célula.

Os agentes da simulação podem mover-se; verticalmente ou horizontalmente uma célula de cada vez, caso a célula não esteja ocupada por outro agente. Cada agente possui uma "força" inicial de valor 40. Para realizar uma movimentação na grade uma unidade de força é consumida, enquanto que para realizar uma agressão a outro agente são consumidas 4 unidades de força, tanto para o agente que ataca como para o atacado. Cada agente tem o sentido de `visao` que pode perceber comidas e agentes na célula em que está presente e nas 4 células vizinhas[16]. O agente pode "cheirar" uma comida que esteja a uma distância de dois passos de sua célula, porém não pode perceber agentes a esta distância.

Existem no ambiente uma proporção de dois agentes para cada comida. Ao comer a comida, ação que demora 2 ciclos, o agente terá sua força aumentada em 20 unidades. O agente só terá sua quantidade de força aumentada caso ele complete os dois ciclos com a comida. Se o agente for atacado enquanto estiver comendo e perder sua comida, não haverá acréscimo em sua força.

Atacar um agente significa tomar a comida deste. Caso o atacante seja mais forte (o valor de sua força atual é maior) que o atacado, a comida mudará para a posição do atacante, ambos o atacante e o atacado permanecem em suas posições na grade.

Neste ambiente são possíveis quatro tipos de ações:

Mover para: que tem como precondição o agente desta ação estar a um passo de seu destino e o agente deve possuir pelo menos uma unidade de força. Os efeitos desta ação serão a redução de uma unidade na força do agente e a localização do agente mudará em uma posição.

Comer: para realizar a ação de comer, o agente deverá estar em uma célula que tenha comida. Ao realizar a ação por dois ciclos sem ter sido atacado, a força do agente será aumentada em 20 unidades e a comida será removida da célula. Outra comida aparecerá em uma posição aleatória da grade.

[16] As diagonais não são consideradas

Atacar: o agente a ser atacado deve estar a um passo do atacante e deve haver comida na célula do atacado. O agente deve possuir pelo menos 4 unidades de força. A força de ambos os agentes é reduzida em 4 unidades.

Permanecer: para realizar esta ação todas as células ao redor do agente (nas quatro direções) devem estar ocupadas e sem comida.

5.9.2 Modelagem do Ambiente da Simulação

Uma simulação para este cenário foi elaborada seguindo a abordagem MASSOC, com pequenas adaptações, mas seguindo a idéia da simulação original. Nesta seção é apresentada a modelagem resumida do ambiente descrito na seção anterior. A definição completa em ELMS deste ambiente encontra-se no Anexo B.

5.9.2.1 Definição da Grade

A grade que representa o espaço físico do ambiente é definida da seguinte maneira:

```
<DEFGRID SIZEX = "10" SIZEY = "10">
   <BOOLEAN NAME = "ocupado"> "FALSE" </BOOLEAN>
   <BOOLEAN NAME = "comida">  "FALSE" </BOOLEAN>
</DEFGRID>
```

No trecho acima é definida uma grade de tamanho $[10 \times 10]$ onde cada célula possui um atributo `ocupado` e outro `comida`. O atributo `ocupado` indica a existência de um agente na célula, enquanto `comida` indica a existência de comida.

5.9.2.2 Definição dos Recursos

A simulação apresenta como recurso apenas o item `comida` cuja classe está definida no trecho abaixo:

```
<RESOURCE NAME="comida">
   <INTEGER NAME="id_dono">  -1</INTEGER>
   <INTEGER NAME="comendo">  -1</INTEGER>
   <INTEGER NAME="forca_cen"> 0</INTEGER>
   <INTEGER NAME="forca_sup"> 0</INTEGER>
   <INTEGER NAME="forca_inf"> 0</INTEGER>
   <INTEGER NAME="forca_dir"> 0</INTEGER>
   <INTEGER NAME="forca_esq"> 0</INTEGER>
   <BOOLEAN NAME ="alocado">"FALSE"</BOOLEAN>
   <REACTIONS LIST="r_ataqueS r_ataqueI r_ataqueD r_ataqueE
                    reacao_comer aparecer"/>
</RESOURCE>
```

O atributo `comendo` indica a quantidade de ciclos que o agente está "comendo", o atributo `id_dono` indica o índice do agente que está de posse da comida. Os atributos `forca_cen`, `forca_sup`, `forca_inf`, `forca_dir`, `forca_esq` indicam as forças externas ao recurso `comida` que estão sendo aplicadas a ele. O atributo `alocado` indica se a comida está alocada em alguma posição da grade.

As reações `r_ataqueS`, `r_ataqueI`, `r_ataqueD` e `r_ataqueE` estão relacionadas à movimentação do recurso comida ao ser disputado. Enquanto `reacao_comer`, `sorteia` e `aparecer` estão relacionadas ao consumo e a mudança de posição da comida.

5.9.2.3 *Definição dos Agentes*

Assim como os recursos, a simulação apresenta apenas uma classe de agente. No artigo existem três tipos de agentes, porém esta classificação refere-se apenas a grupos de agentes que têm comportamentos distintos, mas do ponto de vista do ambiente[17] todos os agentes são iguais. Na abordagem MASSOC, os comportamentos distintos dos agentes seriam definidos através de diferentes programas AgentSpeak(XL). Abaixo está definido a classe de agente `agente`. Esta possui como atributos a propriedade `forca` que indica a quantidade de força do agente e um atributo `id` que indica o índice da instância do agente.

O agente não possui a ação `permanecer` descrita na seção anterior, por ser equivalente à ausência de ação, que é indicada pelo agente do envio ao ambiente da palavra reservada `true`.

```
<AGENT NAME="agente">
<INTEGER NAME="forca"> 40 </INTEGER>
<INTEGER NAME="id"> "SELF"</INTEGER>
<PERCEPTIONS LIST="visao olfato"/>
<ACTIONS LIST="mover comer atacarS
               atacarD atacarI atacarE"/>
</AGENT>
```

O agente pode realizar a ação de atacar em quatro direções e como estas são tratadas de forma diferenciada pelo ambiente, foram criadas quatro ações básicas distintas.

5.9.2.4 *Definição das Percepções*

Os agentes desta simulação possuem dois tipos de percepções `visao` e `olfato`. Da maneira como estes tipos de percepção estão definidos nesta seção, o agente não percebe duas vezes a mesma comida através de diferentes percepções. Apesar da definição original da simulação não deixar claro se isto ocorre, esta característica não afeta a coerência da simulação.

O trecho abaixo define a percepção `visão` através da qual o agente pode saber quais agentes e comidas estão presentes em sua célula e nas células vizinhas.

```
<PERCEPTION NAME="visao">
   <CELL_ATT ATTRIBUTE = "CONTENTS">
      <X> +0 </X> <Y> +0 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "CONTENTS">
      <X> +0 </X> <Y> -1 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "CONTENTS">
      <X> +1 </X> <Y> +0 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "CONTENTS">
      <X> +0 </X> <Y> +1 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "CONTENTS">
```

[17] Pela forma como são "fisicamente" percebidos no ambiente.

```
            <X> -1 </X> <Y> +0 </Y>
         </CELL_ATT>
      </PERCEPTION>
```

O trecho abaixo (resumido) define a percepção `olfato` através da qual o agente pode saber onde existem comidas a dois passos de sua posição atual. Para a definição completa da percepção veja o Anexo B.

```
      <PERCEPTION NAME="olfato">
         <CELL_ATT ATTRIBUTE = "comida">
            <X> +0 </X> <Y> -2 </Y>
         </CELL_ATT>
         <CELL_ATT ATTRIBUTE = "comida">
            <X> -1 </X> <Y> -1 </Y>
         </CELL_ATT>
      <!--..........RESUMIDO............-->
         <CELL_ATT ATTRIBUTE = "comida">
            <X> +0 </X> <Y> +2 </Y>
         </CELL_ATT>
      </PERCEPTION>
```

5.9.2.5 Definição das Reações

Foram definidas para a classe de recurso `comida` quatro reações para mover a comida em caso de uma disputa, uma reação para aumentar a energia de quem a consuma e uma reação que faz com que a comida apareça em uma célula não ocupada após ser consumida.

O trecho abaixo define a reação (resumida) `r_ataqueS` que é acionada quando há uma disputa pela comida e o mais forte é o agente que está na posição superior à posição atual da comida. As reações `r_ataqueD`, `r_ataqueE` e `r_ataqueI` são similares e estão no Anexo B.

```
<REACTION NAME="r_ataqueS">
   <PRECONDITION>
      <GREATERTHAN>
         <OPERAND>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERAND>
         <OPERAND>
            <ELEMENT_ATT NAME= "comida"  ATTRIBUTE= "forca_cen">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERAND>
      </GREATERTHAN>
<!--.............RESUMIDO.............-->
      </PRECONDITION>
   <MOVE>
        <ELEMENT NAME = "SELFCLASS">
           <INDEX>"SELF"</INDEX>
```

```
        </ELEMENT>
        <FROM>
           <CELL>
             <X>+0</X> <Y>+0</Y>
           </CELL>
        </FROM>
        <TO>
            <CELL>
                <X>+0</X> <Y>-1</Y>
            </CELL>
        </TO>
      </MOVE>
      <ASSIGN>
          <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
              <INDEX>"SELF"</INDEX>
          </ELEMENT_ATT>
          <EXPRESSION> 0 </EXPRESSION>
      </ASSIGN>
<!--.............RESUMIDO.............-->
</REACTION>
```

A reação `reacao_comer` tem como precondição o atributo `comendo` ser igual a 2. Quando a precondição for satisfeita, a comida será removida da grade, a força do agente que a consumiu será aumentada em 20 unidades e o atributo `alocado` da comida receberá valor `FALSE`.

No trecho abaixo está resumida a definição da reação `aparecer` que é ativada quando o agente não está alocado na grade.

```
<REACTION NAME="aparecer">
   <PRECONDITION>
<!--.............RESUMIDO.............-->
   </PRECONDITION>
   <IN_RAND>
        <PRECONDITION>
            <EQUAL>
                <OPERAND>
                     <CELL_ATT ATTRIBUTE = "comida">
                        <X>"X"</X>
                        <Y>"Y"</Y>
                    </CELL_ATT>
                </OPERAND>
                <OPERAND> "FALSE" </OPERAND>
            </EQUAL>
        </PRECONDITION>
        <ELEMENT NAME = "SELFCLASS">
           <INDEX>"SELF"</INDEX>
        </ELEMENT>
   </IN_RAND>
   <ASSIGN>
```

```
        <CELL_ATT ATTRIBUTE = "comida">
           <X>+0</X> <Y>+0</Y>
        </CELL_ATT>
        <EXPRESSION> "TRUE" </EXPRESSION>
   </ASSIGN>
</REACTION>
```

5.9.2.6 Definição das Ações

A ação `mover` é definida no trecho abaixo:

```
<ACTION NAME="mover">
   <PARAMETER NAME="XNOVO" TYPE="INTEGER"/>
   <PARAMETER NAME="YNOVO" TYPE="INTEGER"/>
   <PRECONDITION>
      <EQUAL>
         <OPERAND>
            <CELL_ATT ATTRIBUTE = "ocupado">
               <X> +"XNOVO" </X> <Y> +"YNOVO" </Y>
            </CELL_ATT>
         </OPERAND>
         <OPERAND> "FALSE"</OPERAND>
      </EQUAL>
   </PRECONDITION>
<!--.............RESUMIDO.............-->
        <MOVE>
           <ELEMENT NAME = "SELFCLASS">
              <INDEX>"SELF"</INDEX>
           </ELEMENT>
           <FROM>
              <CELL> <X>+0</X> <Y>+0</Y> </CELL>
           </FROM>
           <TO>
              <CELL>
                <X> +"XNOVO" </X> <Y> +"YNOVO" </Y>
              </CELL>
           </TO>
        </MOVE>
<!--.............RESUMIDO.............-->
</ACTION>
```

A ação `mover` recebe como parâmetros dois valores inteiros que indicam a nova coordenada para onde o agente deve mover-se. Note que o agente deverá enviar para o ambiente coordenadas relativas à sua posição atual.

A ação `comer` recebe como parâmetro um valor inteiro que identifica a comida que irá ser consumida. A ação `comer` está definida no trecho abaixo:

```
<ACTION NAME="comer">
   <PARAMETER NAME="IDcomida" TYPE="INTEGER"/>
   <PRECONDITION>
```

```
      <EQUAL>
          <OPERAND>
             <ELEMENT_ATT NAME="comida" ATTRIBUTE="comendo">
                <INDEX>"IDcomida"</INDEX>
             </ELEMENT_ATT>
          </OPERAND>
          <OPERAND> -1 </OPERAND>
      </EQUAL>
   </PRECONDITION>
   <ASSIGN>
        <ELEMENT_ATT NAME="comida" ATTRIBUTE="comendo">
           <INDEX>"IDcomida"</INDEX>
        </ELEMENT_ATT>
        <EXPRESSION> 1 </EXPRESSION>
   </ASSIGN>
   <ASSIGN>
        <ELEMENT_ATT NAME="comida" ATTRIBUTE="id_dono">
        <INDEX>"IDcomida"</INDEX>
        </ELEMENT_ATT>
        <EXPRESSION> "SELF" </EXPRESSION>
   </ASSIGN>
</ACTION>
```

As ações `atacarS`, `atacarD`, `atacarI` e `atacarE` são utilizadas pelo agente para atacar outros agentes que tenham comida em uma das quatro direções possíveis. Para que a ação seja executada, é necessário que haja um agente e uma comida na direção a ser atacada. Caso a ação seja executada, a força dos agentes será decrementada e será marcado no recurso a força do agente que a "puxou". Através da reação, o recurso `comida`, irá decidir qual o agente que "puxou" mais forte e a comida será alocada na célula do agente com a maior força.

5.9.2.7 Definição dos Observáveis

Os ítens observáveis são o resultado da simulação, conforme visto nas seções anteriores. A definição original do problema (CONTE; CASTELFRANCHI, 1995a; CASTELFRANCHI; CONTE; PAOLUCCI, 1998) não especifica quais os dados de saída, mas os dados mais relevantes seriam o posicionamento dos agentes e comidas e a evolução das forças dos agentes, para se ter conhecimento da ocorrência de agressões. O trecho seguinte define estes ítens como observáveis:

```
   <OBSERVABLE>
       <CELL_ATT ATTRIBUTE = "CONTENTS">
          <X> "ALL"</X> <Y> "ALL"</Y>
       </CELL_ATT>
       <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
           <INDEX> "ALL" </INDEX>
       </ELEMENT_ATT>
   </OBSERVABLE>
```

6 PLATAFORMA MASSOC

O objetivo principal do projeto MASSOC (**M**ulti-**A**gent **S**imulations for the **SOC**ial Sciences) é fornecer uma plataforma para criação de simulações baseadas em agentes que não exija grande experiência em programação dos seus usuários. Mais especificamente, esta plataforma deve permitir o projeto e implementação de agentes cognitivos, com uma interface gráfica para facilitar as especificações de ambientes multiagentes, planos, agentes e simulações completas, e também o gerenciamento de bibliotecas destes componentes. A partir de informações dadas pelos usuários, o sistema gera códigos fontes para os interpretadores das linguagens de programação de agentes cognitivos e para a especificação de ambientes multiagentes.

Em nossa abordagem, o raciocínio dos agentes é especificado com o uso do AgentSpeak(XL) (ver seção 2.4), uma extensão da linguagem de programação orientada a agentes BDI chamada AgentSpeak(L). Os ambientes são especificados em ELMS, a linguagem para a descrição de ambientes multiagentes que é apresentada nesta dissertação (ver capítulo 5).

O componente ELMS e a interface para desenvolvimento de simulações foram desenvolvidos em nosso grupo de pesquisa especificamente para o projeto MASSOC, apesar da linguagem ELMS poder ser utilizada separadamente em outras plataformas. Já o AgentSpeak(XL), apesar de desenvolvido também no II-UFRGS, foi desenvolvido em paralelo ao projeto MASSOC.

Para a comunicação entre os agentes, o ambiente e a interface, está sendo utilizada a ferramenta SACI (ver seção 4.7). Esta suporta comunicação baseada em KQML e fornece uma infraestrutura para controle de agentes distribuídos. O SACI foi escolhido para integrar os componentes da abordagem MASSOC por fornecer meios que eram necessários para a implementação da comunicação em nossa plataforma, como por exemplo era necessário uma ferramenta que permitisse a comunicação de agentes em alto nível. Todos os agentes que participam da simulação, a interface e o controlador de ambiente são registrados em uma sociedade SACI. Através do SACI, cada membro da sociedade pode comunicar com os outros membros da sociedade, simplesmente mandando mensagens endereçadas com o nome do destinatário na sociedade. Deste modo, é possível para qualquer agente que faça uso do SACI interagir com outros componentes da simulação. Por exemplo, pode-se desenvolver uma interface para que agentes humanos tomem parte em uma sociedade simulada, apesar de isto não ser um dos objetivos principais do projeto MASSOC, esta interface para agentes humanos seria muito útil na análise e correção de simulações.

Porém, ainda faltam nesta plataforma algum meio para a especificação explícita de estruturas sociais (grupos e organizações), que é muito importante para simulações sociais. Prover mecanismos para especificar tais estruturas é parte dos objetivos a longo prazo,

que também incluem uma tentativa de conciliar cognição e emergência. Este último objetivo é inspirado na idéia de Castelfranchi (2001) de que apenas simulações sociais com agentes cognitivos irão permitir o estudo das "mentes" dos agentes individualmente e a emergência de ações coletivas, que interagem determinando-se mutuamente. Em outras palavras, planeja-se (a longo prazo) prover as condições básicas para o projeto MASSOC ajudar no estudo de um problema fundamental nas ciências sociais, que é de grande relevância nos SMA também: o problema da relação micro-macro (CONTE; CASTELFRANCHI, 1995b).

6.1 Interface de Gerenciamento de Simulações

Está em desenvolvimento uma interface gráfica para facilitar a criação de simulações seguindo a abordagem MASSOC. Esta interface também permite o controle de execução da simulação, que chamamos MASSOC*Manager*. Além de facilitar a criação de simulações, o MASSOC *Manager* integra as diversas tecnologias usadas em nossa abordagem, tal como o interpretador ELMS e o interpretador AgentSpeak(XL) com o uso do SACI.

A principal função da interface é o suporte à criação de bibliotecas de planos, agentes e ambientes. Além da criação, a interface também suporta o gerenciamento dos arquivos destes diversos componentes de simulação, estimulando o reaproveitamento destes componentes em diferentes simulações. Um arquivo que contenha uma biblioteca de planos consiste de uma série de definições de planos na sintaxe AgentSpeak(XL). Em uma biblioteca de agentes, cada agente[1] é representado por um nome, um conjunto de crenças AgentSpeak(XL) iniciais e uma lista de rótulos para planos contidos nas respectivas bibliotecas. A definição de um ambiente é informada pelo usuário através da interface gráfica, que irá gerar automaticamente o código ELMS em sua sintaxe XML, que é enviada para o interpretador ELMS (descrito no capítulo 5).

A figura 6.1 mostra de maneira geral a interface para o usuário. É provida uma "área de trabalho", onde o usuário pode criar e editar agentes, planos e ambientes, que poderão ser utilizados para definição de simulações multiagente. Planos e agentes são definidos em AgentSpeak(XL), enquanto a definição de ambiente fornece todas as informações necessárias para a especificação de um ambiente ELMS. Outras características da interface são a criação, execução e monitoramento da simulação, que ainda estão em desenvolvimento. Esta parte da plataforma fornece a integração dos vários sistemas que compõem a abordagem MASSOC.

Na janela de definição de simulação, o usuário determina o conjunto de agentes e o ambiente escolhidos para uma dada simulação. Para a definição do ambiente, o MASSOC *Manager* verifica quais tipos de agentes podem participar na simulação, e permite ao usuário escolhê-los. Para cada um dos tipos habilitados, deve ser determinado o número de instâncias que serão criadas. Cada um destes agentes executa um interpretador AgentSpeak(XL) com o código-fonte gerado através do MASSOC *Manager*. Depois do usuário ter informado o ambiente pretendido e as instâncias dos agentes que devem ser instanciados, a simulação pode ser iniciada. É possível abortar a execução de agentes através do MASSOC *Manager*, assim como também é possível criar novas instâncias de agentes e recursos.

Uma janela de execução fornece a informação sobre os componentes de uma

[1] Na verdade, isto se refere a *classes* de agentes, de forma que cada uma dessas definições de agentes pode ter várias instância em uma simulação.

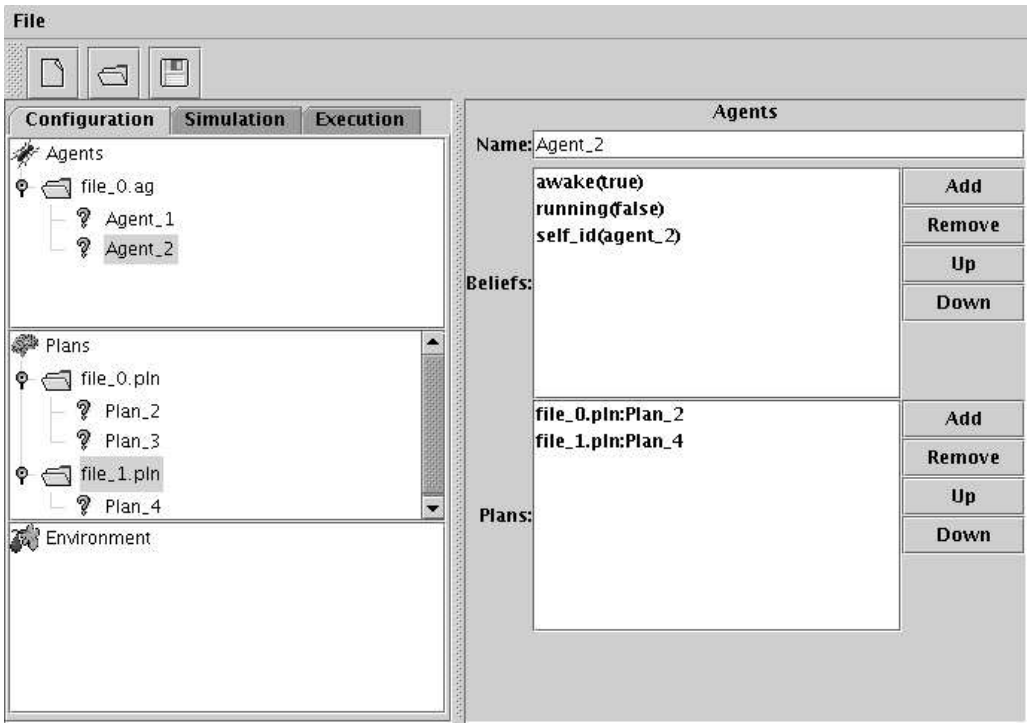

Figura 6.1: A interface do MASSOC *Manager*.

simulação (agentes e recursos) que estão ativos em uma simulação. Nas informações sobre os agentes, é possível obter informações sobre seu estado interno e externo. O estado interno indica as atitudes mentais do agente, como por exemplo suas crenças, objetivos e intenções, durante a execução da simulação. O estado externo está relacionado com as características (propriedades) do agente que são perceptíveis aos outros agentes através do ambiente. Estes dois níveis de informações dos agentes podem ser acessados separadamente na janela de execução. Para os recursos, é possível observar os valores correntes associados com as propriedades dos recursos.

6.2 Interação dos Componentes

A figura 6.2 mostra uma visão geral do funcionamento das várias partes do MASSOC que são controladas pela interface, e mostra como elas se relacionam entre si. Através da interface gráfica, o MASSOC *Manager*, o usuário realiza a modelagem do comportamento dos agentes, e então será gerado o "Código-ASPK" apropriado, que é a definição de um agente na linguagem AgentSpeak(XL). Também através da interface é definido o "Código ELMS" que é a descrição do ambiente na linguagem ELMS. Após definidos o ambiente e os códigos dos agentes, o MASSOC *Manager* inicia o *Controlador de Ambiente*. Após iniciado, o processo controlador do ambiente processa a especificação ELMS do ambiente, gerando as estruturas de dados que representam o ambiente. A seguir, a sociedade SACI é criada, através da qual os agentes e o ambiente se comunicam. Então os agentes são criados, executando instâncias do interpretador AgentSpeak(XL), cada um recebendo um código AgentSpeak(XL) específico para a função que este deverá cumprir na sociedade, tal como definido pelo projetista da simulação. Os agentes se conectam à sociedade SACI, através da qual receberão as percepções determinadas e enviarão as ações que estes venham a escolher realizar no ambiente.

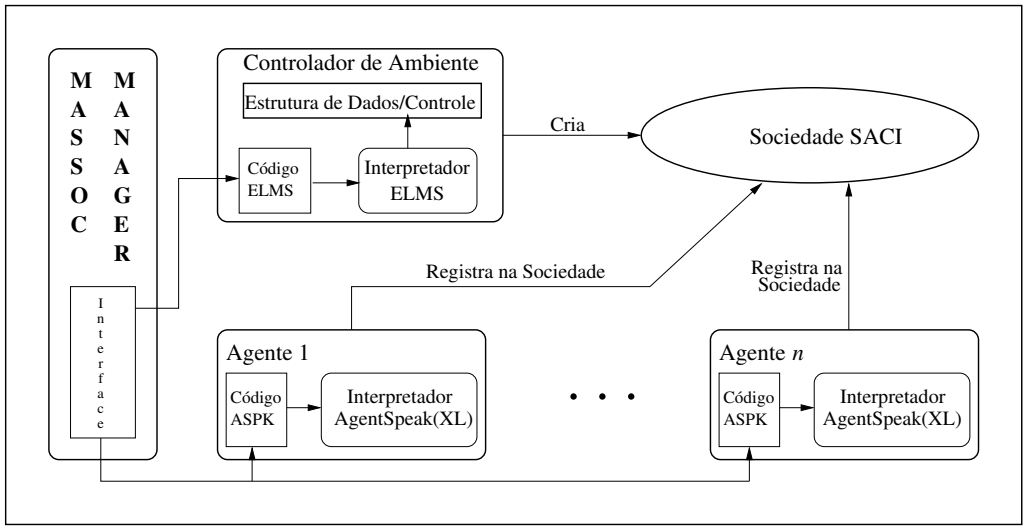

Figura 6.2: Componentes MASSOC.

As seguintes ações são realizadas em uma simulação, algumas destas estão descritas na figura 6.3):

1. o MASSOC *Manager* inicia o interpretador da linguagem ELMS, o que processa a especificação do ambiente dado como entrada;
2. o interpretador gera as estruturas de dados que serão usadas para simular o ambiente especificado;
3. a seção de inicialização do ambiente é executada;
4. o processo controlador de ambiente (gerado pelo interpretador ELMS) cria uma sociedade SACI e se registra como um membro da sociedade;
5. o MASSOC *Manager* se registra como um membro da sociedade SACI;
6. o gerenciador de simulação cria um processo que executa um interpretador AgentSpeak(XL) para cada agente, cada um com seu código apropriado como entrada;
7. cada um dos agentes se registra como membro da sociedade;
8. o gerenciador verifica se todos os componentes estão prontos para o início da simulação;
9. o gerenciador envia um sinal para o controlador de ambiente para este iniciar a simulação;
10. o ambiente envia a para os agentes suas respectivas percepções;
11. os agentes executam o ciclo de raciocínio sobre as percepções recebidas e cada agente informa ao ambiente qual ação escolheu realizar;
12. execução das ações recebidas, de acordo com o modo de execução síncrono ou assíncrono (ver seção 5.6)
13. o ambiente envia os valores das propriedades observáveis para a interface;

Este ciclo é repetido a partir do passo 9, até que a simulação alcance sua condição de término ou o usuário escolha por interrompe-la.

A figura 6.3 mostra a estrutura de uma simulação MASSOC e as principais interações que formam um ciclo de simulação. Os retângulos "Agente" representam interpretadores AgentSpeak(XL) executando códigos de agentes definidos através do MASSOC *Manager*. O "Controlador de Ambiente" representa o processo em execução que controla a estrutura de dados gerada pelo interpretador da linguagem ELMS e as estruturas utilizadas no controle de execução do ambiente. A forma como ocorre a comunicação entre o "Controlador de Ambiente" e os agentes pode ser vista com maiores detalhes no Anexo D. Todos os componentes estão dentro da elipse "Sociedade SACI" representando o fato de que todos os componentes são membros da sociedade SACI e através desta é realizada a interação dos mesmos.

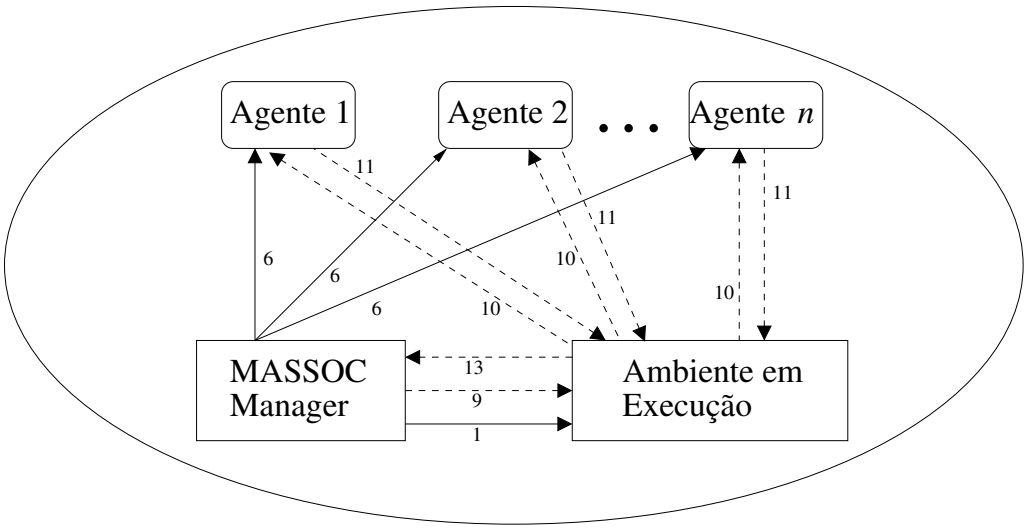

Figura 6.3: Interações entre Componentes MASSOC.

6.3 Estudo de Caso: Simulação de Aspectos Sociais do Crescimento Urbano

Nesta seção é apresentado uma das simulações em desenvolvimento e os primeiros resultados obtidos através desta. Esta simulação de crescimento urbano foi desenvolvida em parceria com o professor Dr. Rômulo Krafta do PROPUR-UFRGS. A seção 6.3.1 e a seção 6.3.2 foram adaptadas do trabalho apresentado por Denise de Oliveira em (OLIVEIRA, 2002).

6.3.1 Descrição do Problema

O problema a ser tratado nesta simulação é avaliar o papel das classes sociais na dinâmica do crescimento urbano, caracterizado pela interação entre agentes consumidores e produtores de espaço urbano. A produção do espaço gera renda para os produtores de acordo com sua aceitação pelos consumidores. Os consumidores decidem a localização e valor dos imóveis que compram, de acordo com seu poder de compra e preferências em relação às classes sociais dos vizinhos. A dinâmica proposta é um processo iterativo no qual Unidades de Forma Construída (UFCs) são alocadas sobre um território dividido em

terrenos do mesmo tamanho, e atividades sociais são alocadas em tais imóveis construídos (UFCs). Consumidores decidem sobre localização de suas atividades, tendo em vista a oferta de imóveis e produtores decidem sobre localização de UFCs a serem produzidas, tendo em vista as oportunidades de geração de renda.

Agentes desenvolvem atitudes e preferências em relação a outros agentes, de forma que qualquer decisão sempre será tomada com base em uma situação urbana (estado atual do sistema espacial) e um contexto social. Qualquer decisão de qualquer agente gera externalidades, ou seja, provoca outros efeitos, além daqueles pretendidos pelo seu protagonista (KRAFTA, 1999). Externalidades afetam o desempenho de outros agentes, positiva ou negativamente, de forma que o sistema está permanentemente submetido a modificações e novas acomodações provocadas pelas reações dos agentes a externalidades.

6.3.1.1 Descrição dos Agentes

Como mencionado anteriormente, há três tipos diferentes de agentes no sistema, sendo que cada um tem um papel bem distinto dentro da sociedade, executando ações diferentes no sistema. Nesta seção cada tipo de agente será descrito detalhadamente.

Produtores Imobiliários

Os produtores imobiliários são os agentes que constroem UFCs, vendem as UFCs e ainda são capazes de alocar ou desalocar agentes de construções ocupadas. As construções são feitas de modo a obter maior lucro e maior poder de construção. Cada vez que uma UFC é comprada, o valor dela vai para o agente que a construiu e o tamanho dela volta para seu poder de construção. Embora haja um número de agentes PI inicial, novos agentes podem ser admitidos ao longo da execução, sendo que todos entram com capacidade construção e capital de construção básicos.

A porcentagem de UFCs, vendidas em um ciclo está diretamente relacionada aos tipos de UFCs construídas e as mais procuradas pelos outros agentes para a compra, assim há uma especialização no que se refere a tipos de UFCs; agentes dariam preferência por tipos que vendam melhor, desta forma, o sucesso de um tipo em uma iteração provoca oferta dilatada daquele tipo na próxima rodada, o que pode refletir em mau desempenho, e retração na seguinte. A competição entre este tipo de agentes garante que a sua iniciativas serão seguidas por outros, causando uma perda de vantagens espaciais e a necessidade de busca por novos lugares (KRAFTA, 1999).

Consumidores Residenciais

Os agentes consumidores comerciais estão divididos em classes de acordo com sua renda, que determina preferências pessoais e poder de compra, e estão distribuídos em uma pirâmide de renda e população.

O objetivo principal dos agentes deste tipo é a sua satisfação em relação a localização de sua moradia em relação aos serviços e as classes sociais a que pertencem os demais agentes, sendo que cada classe de agentes residenciais tem exigências bem definidas, sendo:

Agentes da classe A: tendem a afastar-se dos agentes de classes inferiores (B e C) e todos os agentes comerciais, já que não necessitam de serviços próximos.

Agentes da classe B: tendem a afastar-se dos agentes de classe C, aproximar-se dos agentes da classe A e não têm restrições em relação aos agentes comerciais.

Agentes da classe C: tendem a aproximar-se dos agentes comerciais, não tendo restrições em relação a agentes residenciais.

Consumidores Comerciais

Os consumidores comerciais estão divididos em porte e tipo de serviço oferecido. O porte indica o raio de atuação do CC e o serviço indica o tipo de serviço que o agente oferece para a sua população atendida. O tipo de serviço influencia diretamente na determinação da UFC a ser ocupada já que alguns serviços podem ser concorrentes (por exemplo, dois CC de mesmo porte com o mesmo serviço dentro do mesmo raio de atuação), de modo que o agente escolherá a UFC que lhe oferecer melhor relação custo/benefício. Há uma declaração inicial de tipos de serviço admitidos, com seus respectivos parâmetros de população e riqueza mínimos requeridos; sempre que esses requisitos forem atendidos, um novo agente comercial pode estabelecer-se no sistema. Existir no sistema não significa estar alocado em uma UFC; para isto é necessária a verificação de outras condições: cada tipo de serviço demanda a existência de um número mínimo de consumidores potenciais dentro de um raio de alcance estabelecido.

6.3.1.2 Definição do Ambiente

Assume-se como ambiente uma matriz regular quadrada de $n \times n$ terrenos, onde não há nenhuma característica pré urbana. Cada terreno tem uma capacidade de edificação fixa e igual, deste modo cada terreno contém um número limitado de UFCs.

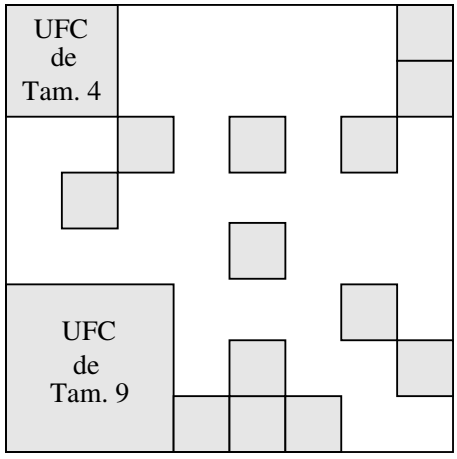

Figura 6.4: Exemplo de Terreno com Algumas UFCs

A forma construída, decorre da ação dos agentes PIs, sendo que cada UFC diferencia-se por tamanho e valor. O tamanho máximo de uma UFC é o tamanho de um terreno, que no caso foi definido como 64 unidades básicas (tamanho mínimo de uma UFC), e uma UFC pode ter o tamanho de quantas unidades básicas forem definidas desde que não saia dos limites máximo e mínimo e que o somatório dos tamanhos das UFCs do terreno não ultrapasse 64. A Figura 6.4 de Oliveira (2002) mostra um exemplo de terreno, onde já estão construídas uma UFC de tamanho 4, uma de tamanho 9 e 13 UFCs de tamanho 1, havendo assim espaço para que mais 38 unidades sejam alocadas.

Os valores das UFCs serão fixados com base no valor da terra, tomando este como sendo 20% do valor da UFC. O valor da terra inicialmente é igual em toda a cidade, mas após a urbanização ele muda. Qualquer valor inicial atribuído a uma UFC será desvalorizado a cada iteração de acordo com a idade (tempo decorrido entre uma iteração e

outra) que se reflete na perda de valor das UFC e se está desocupada ou não. As taxas de desvalorização serão fixas e atribuídas pelo operador.

6.3.2 Implementação do Sistema

O sistema multiagente foi implementado utilizando-se abordagem MASSOC, vista no capítulo anterior, utilizando a linguagem AgentSpeak(XL) para a implementação dos agentes e a linguagem ELMS para a descrição do ambiente. Nas seções que seguem serão apresentadas, de maneira resumida, a forma como foi implementado o primeiro protótipo da simulação, que ainda está em desenvolvimento, em paralelo com o desenvolvimento da abordagem MASSOC. Descrições mais completas, códigos de agentes e do ambiente podem ser vistos em (OLIVEIRA, 2002).

6.3.2.1 Produtores Imobiliários

As crenças iniciais dos produtores imobiliários são:

- um número único que identifica o agente na simulação (`id`);
- um número limite de UFCs que o agente pode construir;
- a quantidade inicial de dinheiro que o agente possui para gastar em construções;
- crenças que indicam o tamanho mínimo e máximo que cada padrão (A, B, C, P, M ou G) de UFC deve ter;
- crenças que indicam o tipo e o padrão das UFCs que devem ser construídas, sendo que essa escolha tem probabilidades diferentes para cada tipo e padrão.

Todas essas crenças já estão presentes antes que o agente receba percepções do ambiente.

Quando o ambiente enviar, pela primeira vez, uma percepção que unifique com `pi(instance(Index),id(Id))`, esta percepção fará com que o interpretador AgentSpeak(XL) gere um evento que inicializa o ciclo do agente. A partir disso, o agente se inicia no ambiente enviando o seu capital atual e após essa ação, ele escolhe de maneira aleatória o primeiro tipo, padrão e local da UFC a ser construída e aciona diretamente o plano de construir (`constroi`).

Percebendo que há um local (terreno) com o número de vagas e o valor da terra unitária compatíveis com o que ele busca, ele vai tentar construir neste terreno, executando a ação `alocar` no ambiente da simulação. Caso a ação `alocar` seja executada com sucesso, o agente, através de um outro plano, vai escolher outro tipo de UFC para construir. Caso a ação `alocar` não tenha obtido sucesso, isto gerará um evento (`-!constroi`) que indica que o plano `constroi` falhou, o agente irá retornar a sua capacidade anterior e irá em busca de um novo tipo de construção.

O plano de escolha, verifica se a capacidade de construção atual do agente é maior ou igual a uma unidade, escolhe de modo randômico, mas com probabilidades definidas, o tipo, tamanho e padrão da UFC a ser construída e começará a busca de um terreno para alocá-la.

Existem três planos de procura, sendo que o primeiro escolhe o terreno onde haja maior número de agentes residenciais para a construção de UFCs comerciais, o segundo escolhe o terreno onde houver maior concentração de agentes comerciais para a construção de UFCs residenciais de classe baixa e o último escolhe o terreno onde há

menor concentração de agentes comerciais para a construção de UFCs das classes residenciais mais altas. Após ter escolhido o terreno, o agente vai tentar construir neste terreno ativando o plano `constroi`, passando as informações sobre local e tipo de UFC desejada.

6.3.2.2 Consumidores Residenciais

No caso dos agentes residenciais a diferença de classe não precisa ser explicitada através de diferentes classes de agentes ELMS, visto que o comportamento dos agentes residenciais não difere, do ponto de vista do ambiente. Durante o tempo de execução o ambiente pode mandar para o agente qual a nova classe a que ele pertence.

O agente têm apenas duas crenças iniciais, `classe(Classe)` que indica a qual classe o agente pertence, onde `Classe` deve estar instanciada como *a*, *b* ou *c*. A crença `ocupa(nenhum)` significa que ele ainda não ocupa nenhuma UFC; o agente possui também a crença `dinheiro(Dinheiro)`, na qual `Dinheiro` indica quanto dinheiro o agente possui para a aquisição de uma UFC.

Quando o ambiente enviar, pela primeira vez, uma percepção que unifique com `cr(instance(Index),id(Id))`, esta percepção fará com que o interpretador AgentSpeak(XL) gere um evento que inicializa o ciclo do agente.

A partir disso, o agente se inicializa no ambiente, enviando o seu capital inicial e após essa ação, ele procura uma UFC para ocupar. O plano de procura, verifica se há alguma UFC disponível que seja da classe do agente, escolhe randomicamente uma entre as disponíveis para a sua classe e ativa o plano de verificação da vizinhança desta UFC.

Os planos para verificação de vizinhos são caracterizados por terem como condições as preferências de cada classe. A classe menos exigente (a que tem menos precondições) são os agentes de classe C, que apenas verifica se há pelo menos um agente comercial no mesmo bloco que encontra-se a UFC candidata a compra, mas como isso é apenas uma preferência (não está implementada no ambiente como precondição), se esta condição não for verdadeira, ele irá comprar a UFC do mesmo modo. O plano de verificação com maior número de precondições é o do agente de classe A, que tem restrições em relação às duas outras classes de agentes.

O plano AgentSpeak(XL) para a compra de UFC é igual para qualquer uma das três classes. Os planos AgentSpeak(XL) para a escolha de uma ação são diferenciados para cada classe de agente residencial. Porém para cada classe de UFC (A, B ou C) existe uma ação de compra diferente, apesar de todas serem ações de compra, estas ações têm conseqüencias diferentes no ambiente.

Os planos de compra da UFC, enviam a ação de compra para o ambiente; caso ele falhe, o agente busca uma nova UFC para compra. Os agentes de classe A e B verificam a localização de novas UFCs assim que compram a primeira para que estes agentes possam mudar-se caso haja uma UFC com uma vizinhança melhor do que a de sua atual moradia.

6.3.2.3 Consumidores Comerciais

Os agentes consumidores comerciais são os que têm comportamento mais simples, já que não mudam-se e as restrições de lugar são as mesmas mudando apenas o número de clientes e riqueza mínima para cada porte.

Suas crenças iniciais indicam qual é o seu porte, serviço e quanto é o seu capital inicial. Além disto, o agente também conhece o número mínimo de clientes e concorrentes que o ambiente deve possuir para que ele possa estabelecer-se em um determinado local.

De modo similar aos agentes residenciais, quando o agente receber pela primeira vez

uma percepção que unifique com `cc(instance(Index),id(Id))`, o interpretador irá gerar um evento que inicializa o ciclo do agente. A partir disso, o agente se inicializa no ambiente enviando o seu capital inicial e após essa ação, ele procura uma UFC para ocupar.

O plano de procura de UFC, verifica se existe alguma UFC disponível para o seu porte, verifica se os níveis de riqueza e comércio do sistema estão compatíveis com o seu porte, seleciona uma UFC entre as disponíveis para o seu porte e executa a ação de comprar.

6.3.2.4 Ambiente

Definiu-se uma cidade composta de 4×4 terrenos (células), indicando a existência de 16 terrenos disponíveis. A seguir, serão descritos os principais componentes deste ambiente de forma detalhada, a descrição completa do ambiente na linguagem ELMS pode ser encontrada em (OLIVEIRA, 2002).

Recursos e Agentes

Os recursos do ambiente são compostos de terrenos e UFCs. Cada terreno, ou célula, tem como propriedades:

total_a, total_b e total_c número de agentes residenciais pertencentes as classes A, B e C que estão em alguma UFC neste terreno;

total_comercial número de agentes comerciais de todos os portes (P, M e G) que estão neste terreno;

urbanizado indica se já há alguma UFC dentro deste terreno;

valor o valor unitário da terra, ou seja, o valor de um espaço unitário dentro do terreno.

vagas número de vagas disponíveis neste terreno.

Cada terreno pode conter de 1 a 64 UFCs, recursos alocados pelos agentes produtores, cujas propriedades são:

imobiliário número de identificação do seu agente produtor;

tipo indica o tipo de agente que pode habitá-la (comercial ou residencial);

padrão o padrão a que ela pertence (A, B ou C se for residencial; ou P, M ou G caso seja do tipo comercial);

preço preço de compra da UFC;

tamanho tamanho da UFC, sendo considerado o número de unidades básicas que ela ocupa no terreno;

disponível se está disponível para compra (*true*) ou se já está ocupada.

Além destes dois recursos principais, foi também definido um recurso chamado `totais`, para realizar o cálculo de alguns valores que servem de percepção e saída de resultados, contendo os números totais de agentes alocados no sistema. Possui como propriedades: `residencial` e `comercial`, que armazenam o total de agentes

residenciais e comerciais, respectivamente; e as propriedades `a`, `b` e `c` que armazenam os números totais dos agentes de cada classe na cidade.

Definiu-se três classes de agentes que o ambiente admite: produtores imobiliários (pi), consumidores residenciais (cr) e consumidores comerciais (cc). Cada classe de agentes tem percepções, propriedades e ações diferentes no sistema, porém as propriedades `dinheiro` (que indica quanto dinheiro o agente possui) e `id` (indica qual é a sua identificação no sistema), estão presentes nos três.

pi tem como percepções `cidade` e `conteúdo` e suas ações possíveis são `alocar` e `inicializa`.

cr além das propriedades básicas (`dinheiro` e `id`), também possui o id do lugar que ele ocupa no ambiente (`ocupa`); tem como percepções `r_compráveis`, `conteúdo`, `id` e ocupa. Possui quatro ações possíveis: `comprar_a`, `comprar_b`, `comprar_c` e `inicializa`.

cc possui as mesmas propriedades que os agentes residenciais, tem como percepções `c_compráveis`, `conteúdo`, `id` e `ocupa`. Possui duas ações possíveis: `comprar` e `inicializa`.

Percepções

As percepções definidas no ambiente são aquelas mencionadas anteriormente, na definição dos agentes. São elas:

cidade esta percepção é uma das recebidas pelos agentes produtores, e é composta por todos os atributos de todas as células mais todos os atributos do recurso `totais`;

conteúdo recebendo esta percepção, o agente visualiza todos os recursos e agentes que estão alocados em todas as células que compõem o ambiente;

id essa percepção retorna para o agente qual é a sua identificação no sistema;

ocupa retorna para o agente, comercial ou residencial, o número da UFC que ele ocupa no momento atual;

r_compráveis lista todas as UFCs residenciais disponíveis para compra dentro da cidade;

c_compráveis similar à percepção `r_compráveis`, lista as UFCs comerciais disponíveis.

Ações e Reações

As ações que os agentes podem executar e as reações que os recursos podem ter estão definidas no ambiente do seguinte modo:

alocar é a ação que o agente produtor executa no ambiente para a criação de uma UFC com as características escolhidas por ele, para isso o agente deve enviar as informações de `Padrão`, local do terreno (`X` e `Y`), `Tamanho` e `Preço` da UFC a ser alocada. O pré-requisito único para esta ação é que o número de vagas que a célula dispõe seja maior ou igual ao tamanho da UFC que o produtor deseja alocar;

comprar ação que os agentes consumidores executam quando querem comprar uma UFC, informando o índice da UFC desejada e a posição na cidade. Há três tipos de ações que podem ser executadas e dependem do tipo e padrão de UFC a ser comprada. Existem as ações: `comprar` similar, para a compra de UFC do tipo comercial; `comprar_a`, que executa a compra de uma UFC residencial da classe A, `compra_b` e `comprar_c`, mesma ação que apenas compra UFCs de classes diferentes. Em todas essas ações a única precondição é de que a UFC desejada esteja disponível.

inicializa faz com que o valor do parâmetro `dinheiro` no agente que a executa seja inicializado de acordo com seu tipo, porte ou classe.

A única reação que foi definida neste ambiente é a do recurso `totais`, que serve para manter atualizados os contadores do número total de agentes que estão ocupando UFCs no sistema.

Observáveis

Os observáveis, são o conjunto de dados que irão para o arquivo resultado da simulação. A cada passo, são inseridos no arquivo de saída os valores dos atributos e elementos do sistema que forem indicados nesta lista. Nesta simulação, os observáveis são:

totais de cada célula os valores das propriedades `total_a`, `total_b`, `total_c` e `total_comercial` de cada célula do ambiente podem ser observadas como saída.

conteúdos composto pela lista de conteúdos de todas as célula, por exemplo, se na célula de posição (0,0) existe uma UFC alocada, então seu conteúdo seria visível no arquivo de saída como "`cell(position(0,0),ufc(1))`".

atributos das UFCs todos os atributos das UFCs, exceto `tipo` e `disponível`, são visíveis na saída.

totais gerais são todos os atributos (`a`, `b`, `c`, `residenciais` e `comerciais`) do recurso `total`, citado na seção 6.3.2.4, e indicam o número total de agentes que estão alocados em qualquer parte do ambiente, exceto os agentes produtores, que nunca ficam alocados em nenhuma célula.

6.3.3 Resultados das Simulações

As simulações foram realizadas utilizando-se 55 agentes, distribuídos do seguinte modo:

- 10 Produtores Imobiliários;
- 28 Consumidores Residenciais (16 classe C, 8 classe B e 4 da classe A)
- 17 Consumidores Comerciais (10 porte P, 5 porte M e 2 de porte G)

Foram utilizados 7 computadores para que cada simulação completa fosse executada. Cada agente tem um fluxo de dados por passo de aproximadamente 10 Kbytes, referente à percepção enviada para cada agente pelo ambiente. O tempo de duração de uma simulação

com estas características, sendo executados 400 passos, o que levou cerca de 1 hora. Diversas análises de simulações preliminares foram feitas antes de atingir bons resultados finais.

A primeira simulação completa, teve como resultado somente 13 agentes consumidores ocupando UFCs, tendo sido construídas um total 26 UFCs. A Figura 6.5 de Oliveira (2002), foi construída utilizando-se os dados de saída do sistema e mostra a distribuição final dos agentes após 400 passos de simulação. Nesse primeiro resultado, podemos analisar claramente a separação das classes e a disposição dos serviços em torno delas. Com

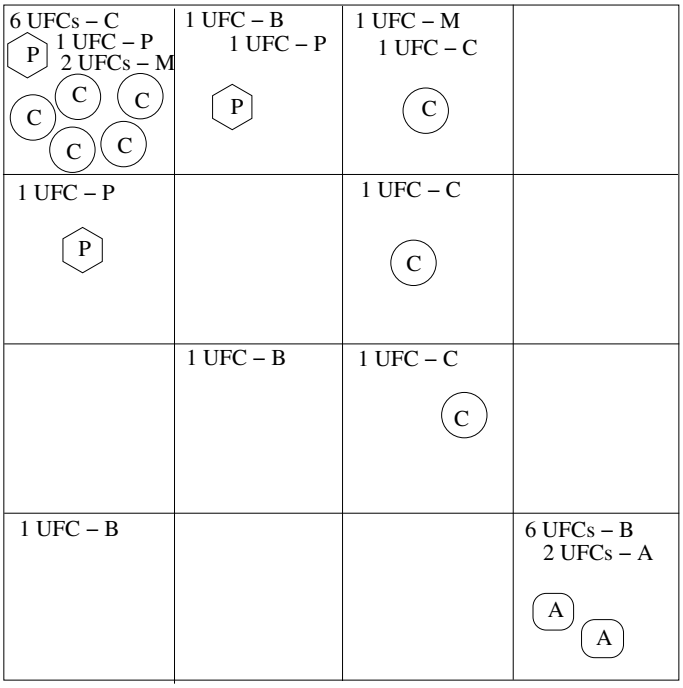

Figura 6.5: Disposição Final dos Agentes e das UFCs na Cidade

esta simulação, observou-se que os agentes produtores deveriam ter construído mais, ou que mais agentes produtores estivessem presentes, pois havia uma demanda por UFCs que não foi atendida.

6.3.4 Considerações

Esta simulação serviu como primeira ferramenta de validação e avaliação da abordagem MASSOC, ajudando de forma significativa no desenvolvimento de tecnologias como a linguagens ELMS, o interpretador AgentSpeak(XL) e a ferramenta de comunicação SACI.

A simulação de crescimento urbano foi utilizada para demonstrar que simulações complexas são feitas nesta abordagem de modo mais simplificado, sem necessidade de programação a nível de rede, sincronização de mensagens e sem utilizar linguagens de programação convencionais, somente utilizando linguagens de mais alto nível, próprias para sistemas multiagentes cognitivos e ambientes multiagentes.

Os agentes da simulação de crescimento urbano têm algumas características reativas, isto porque sua inspiração foram modelos baseados em autômatos celulares (o método mais amplamente utilizado na área de crescimento urbano). Como trabalhos futuros, pretende-se fazer agentes mais cognitivos e mais complexos, apenas reestruturando os códigos AgentSpeak(XL) de modo a refletirem novas características comportamentais.

Agentes mais complexos fariam com que os resultados fossem mais precisos com tempo de simulação maior do que o atual, para uma escala temporal de mesma grandeza. Também pretende-se deixar o ambiente mais completo, adicionando-se características pré-urbanas.

Além destes resultados obtidos como resultado da simulação, talvez os resultados mais significativos foram os obtidos no desenvolvimento das tecnologias envolvidas. Durante o desenvolvimento deste primeiro protótipo da simulação, foram verificados e corrigidas falhas na implementação do interpretador da linguagem. Foram detectadas necessidades de mudanças na estrutura da linguagem, sendo a maioria alterada para a realização da simulação, como por exemplo a inclusão de *palavras-chaves* para facilitar a programação das ações, reações e percepções do ambiente. Apesar de não serem utilizados na versão corrente da simulação, percebeu-se no desenvolvimento da simulação que seria interessante a existência de reações para as células, característica que foi adicionada ao ELMS após o estudo de caso. Percebeu-se também que seria interessante e útil a possibilidade de inclusão de "construtores" para os agentes e recursos, algo que estará incluído em próximas versões do interpretador.

No caso do AgentSpeak(XL), nesta simulação específica foi encontrada a necessidade de haver mais funções matemáticas para a programação do comportamento do agente, devido as origens desta simulação nos autômatos celulares. Esta necessidade de funções matemáticas foi suprida através da inclusão de algumas *ações internas* (ver seção 2.4), porém versões futuras do AgentSpeak(XL) deverão ter algumas destas implementadas em bibliotecas de ações básicas pré-definidas.

O SACI, apesar de não ser desenvolvido em nosso grupo de pesquisa, também obteve melhorias através deste trabalho. Foi observado nesta simulação que o ambiente enviava como percepção para os agentes uma grande quantidade de texto onde haviam várias linhas bastante semelhantes que, se comprimidas, iriam reduzir bastante a quantidade de dados enviadas via rede. Esta observação foi comunicada aos desenvolvedores do SACI, o que resultou na implementação de meios para compressão de mensagens na versão atual do SACI. Com esta compressão, o envio de dados de percepção com cerca de 10 kbytes são reduzidas para cerca de 1 kbyte.

6.4 Trabalhos Relacionados

Nesta seção, são brevemente comparados nossa abordagem com outras plataformas e metodologias. Em particular, uma comparação com o MadKit, uma das mais conhecidas plataformas para construção de sistemas multiagentes. É mencionada a metodologia GAIA, também bastante conhecida para projeto de sistemas multiagentes.

Como mencionado nas seções anteriores, no estágio atual de nosso projeto ainda faltam mecanismos para especificação explícita de estruturas sociais em nossa sociedade de agentes cognitivos. Porém, desconhecemos uma plataforma implementada para desenvolvimento de agentes cognitivos que ofereça tais mecanismos. Vários trabalhos recentes têm proposto diversas plataformas para organizações multiagente. Mas nenhum destes propõem claramente um mecanismo para a implementação de agentes cognitivos, da maneira como apresentada neste trabalho. Em (HÜBNER; SICHMAN; BOISSIER, 2002) são apresentadas interessantes idéias sobre organizações em sociedades de agentes.

O MadKit (GUTKNECHT; FERBER, 2000; GUTKNECHT; FERBER; MICHEL, 2001) é uma plataforma para construção de sistemas multiagentes baseado em uma abordagem organizacional. É embasado em um modelo "agente-grupo-papel". Um agente

pode pertencer a diversos grupos; para cada grupo a qual este pertença, o agente recebe um papel, que pode ser diferente em cada grupo. No Madkit, de acordo com os autores, *"um agente é apenas especificado como uma entidade comunicante que representa papéis dentro de grupos"*, grupos são definidos como conjuntos que agregam agentes, e *"o papel é uma representação abstrata da função, serviço ou identificação de um agente em um grupo"* (GUTKNECHT; FERBER; MICHEL, 2001). O MadKit oferece uma infraestrutura (um conjunto de ferramentas) para construção de sistemas multiagente seguindo este modelo. Um exemplo de tais ferramentas é o *agent micro-kernel*, que controla a comunicação e grupos de agente, uma tarefa realizada pelo SACI em nossa abordagem. O MadKit é uma plataforma para construção de sociedades de agentes, independentemente da arquitetura de agentes usada para a implementação destes. Porém, não existem modelos fixos para a criação dos agentes, assume-se que o usuário irá programá-los utilizando as bibliotecas de suporte, oferecendo apenas modelos para criação de agentes reativos.

De acordo com (DEMAZEAU, 1995) os componentes básicos de um sistema multiagente são agentes, interações, ambientes e organizações. As principais diferenças entre nossa abordagem e o MadKit são os componentes básicos de um sistema multiagente em que a abordagem foca. A plataforma MadKit têm seu foco principalmente nas organizações e no modelo de interações. Em nossa abordagem, os agentes usa a linguagem AgentSpeak(XL), seguindo o modelo BDI (apesar da estrutura geral permita o uso de qualquer tipo de agente, desde que este use a ferramenta SACI e reconheça o formato simples das percepções e ações). Se o nível organizacional é importante para simulações sociais, o uso de agentes cognitivos é também muito importante, de acordo com Castelfranchi (2001).

Continuando a comparação em termos de blocos básicos de construção de sistemas multiagentes, em nossa abordagem, o ambiente é explicitamente definido usando a linguagem ELMS, enquanto na abordagem MadKit, tal como em outras plataformas multiagentes, não há suporte para a especificação de ambientes. O ambiente pode ser o "mundo real" ou é assumido como dado. Em resumo, enquanto a abordagem MadKit tem foco no componente organizacional e estruturas de interações, a abordagem MASSOC tem foco nos agentes, ambiente e nas interações agente-agente e agente-ambiente. Em particular permite a fácil implementação de agentes cognitivos, que não são comuns nas plataformas usadas para simulação social.

Em trabalhos futuros, planejamos criar meios para a definição explícita de organizações em nossa abordagem. Atualmente, as estruturas de interação são definidas implicitamente em cada agente, não existes estruturas rígidas de interações tal como o conceito de papéis. Apesar do nosso estágio de desenvolvimento se encontrar distante disso, nosso objetivo é que no futuro nossa plataforma permitirá a condução de simulações onde as organizações possam emergir das interações entre agentes, considerando fenômenos como "imergência" (*immergence*) e "emergência de segunda ordem" (*second-order emergence*) (CASTELFRANCHI, 1998; CONTE; NIGEL; SICHMAN, 1998); tendo especial interesse nas idéias de Castelfranchi que conciliam cognição e emergência.

Outra fonte de idéias para o nível organizacional de sistemas multiagente é a metodologia Gaia (WOOLDRIDGE; JENNINGS; KINNY, 2000). Assim como o MadKit, o Gaia é também baseado em modelos organizacionais que objetivam o projeto de sistemas multiagentes que podem ser compostos de modelos e teorias de agentes heterogêneos. Todavia, o Gaia é uma metodologia para o projeto e análise de sistemas orientados à agente; não é uma plataforma para o desenvolvimento de sistemas como é o caso do Madkit.

A metodologia Gaia consiste da construção de cinco modelos a partir da "declaração de requisitos" (*requirement statement*). Estes modelos definem respectivamente, os papéis, interações, agentes, serviços e "conhecidos" (*acquaintances*). O primeiro modelo define os possíveis papéis no sistema, onde um papel é definido por um conjunto de responsabilidades, permissões, atividades e protocolos. O modelo de interação define as dependências e relacionamentos entre os vários papéis na organização. O modelo de "conhecidos" define as possíveis elos de comunicação entre os agentes. Sendo esta uma metodologia genérica, esta poderia ser aplicada para a construção de um esquema organizacional completo de uma companhia. Sendo uma metodologia e não uma plataforma, esta não aborda aspectos importantes das simulações, tais como a modelagem de ambientes, como é abordada no projeto MASSOC.

7 CONCLUSÃO

Nesta dissertação foi apresentada a linguagem ELMS para a descrição e geração de ambientes para simulações multiagentes. Esta linguagem está inserida no contexto do projeto MASSOC, que tem como maior contribuição o suporte a criação de agentes BDI e simulações envolvendo este tipo de agentes. A linguagem está em aperfeiçoamento, com base nos trabalhos em desenvolvimento como o estudode caso da seção 6.3.

O projeto MASSOC apresenta uma combinação de técnicas distintas de SMA que consideramos as mais adequadas para a construção de simulações baseadas em SMA. A plataforma possui uma interface gráfica, em desenvolvimento, para auxiliar no gerenciamento de bibliotecas de planos, agentes, ambientes e simulações multiagentes, além de realizar o controle de execução das simulações. Apesar de ser um trabalho em andamento, a plataforma foi testada em uma aplicação (conforme já visto, na área de crescimento urbano, na seção 6.3), que poderá vir a produzir resultados interessantes. Assim, a abordagem MASSOC é uma proposta promissora para simulação social com agentes cognitivos, enquanto que outras plataformas disponíveis são em geral para agentes reativos (ver seção 6.4). Isto permite que no futuro possa pensar-se em questões tais como conciliar emergência e cognição, como proposto por Castelfranchi (1998; 2001).

7.1 Principais Contribuições

As contribuições realizadas neste trabalho estão inseridas no contexto da modelagem e desenvolvimento de sistemas multiagentes. O uso do AgentSpeak(L) e sua extensão o AgentSpeak(XL), têm obtido grande destaque por ser uma linguagem de alto nível com grande embasamento na teoria BDI. Trabalhos como (MOREIRA; BORDINI, 2002; BORDINI; MOREIRA, 2002) relacionam a linguagem AgentSpeak(L) com a lógica BDI.

O projeto MASSOC, em sua fase inicial, baseava-se no uso da linguagem AgentSpeak(L) para a definição do comportamento de agentes BDI e no desenvolvimento de uma interface gráfica que ajudasse pessoas sem conhecimentos aprofundados em programação, tal como cientistas sociais, pudessem criar simulações sociais com agentes BDI. Era necessário um meio que facilitasse a criação de ambientes para estas simulações, sendo a criação de um modelo para a especificação de ambientes uma das contribuições desta dissertação. Desta forma, esta dissertação contribui para o projeto MASSOC, em um aspecto fundamental que é a definição dos ambientes.

A partir da definição da linguagem ELMS, como meio para definição de ambientes, foi desenvolvido um protótipo de interpretador que simulasse ambientes definidos na linguagem ELMS. Com este protótipo e o estudo de caso realizado foi possível verificar algumas deficiências desta abordagem, corrigindo-se algumas delas; outras estão mencionadas na seção 6.3.4.

Desta forma, acreditamos que esta dissertação, juntamente com o projeto MASSOC, apresenta uma contribuição para a área de SMA, permitindo a implementação de simulações baseadas em SMA de forma simples. Além disto, as simulações geradas com este enfoque podem ser bastante mais sofisticadas do que as normalmente geradas com outras plataformas disponíveis, já que permite o uso de agentes cognitivos seguindo a arquitetura BDI, a principal arquitetura para agentes cognitivos na área de sistemas multiagente.

7.2 Trabalhos Futuros

Como trabalhos futuros, existem várias possibilidades para a melhoria desta abordagem. Em particular, pretendo-se no futuro considerar um aspecto da simulação baseada em agentes que é a especificação de estruturas sociais (por exemplos grupos e organizações). Como discutido em seções anteriores, este é um aspecto importante para simulações sociais e para o qual o MASSOC ainda não prove nenhum mecanismo.

Seria interessante incluir na plataforma MASSOC, formas para a visualização da simulação, a partir das descrições instantâneas da simulação geradas pelo interpretador no formato da linguagem ELMS, que é baseado em XML. Isto permite, por exemplo, a criação de um visualizador para simulações via *web*, ou um visualizador baseado em técnicas de realidade virtual. Também seria interessante a integração com pacotes para análise estatística, facilitando a análise dos resultados de simulação. Trabalhos futuros também incluem o progresso com a simulação social de aspectos do crescimento urbano. Para um maior aperfeiçoamento da plataforma, seria muito importante o desenvolvimento de novas simulações, além da continuidade das simulações em desenvolvimento.

Como o principal objetivo do protótipo implementado era verificar a viabilidade da abordagem, não preocupou-se com a performance do sistema. Seria importante, para uma plataforma que se propõe a criar simulações com grandes quantidades de agentes cognitivos, a otimização do código do interpretador, assim como dos outros módulos envolvidos.

Existem várias extensões ou melhorias específicas na implementação que poderiam melhorar a simulação de ambientes, tais como:

- Seria interessante implementar na linguagem ELMS a possibilidade de definir construtores (conjuntos de comandos para inicialização) para recursos e agentes.
- Permitir o uso de intervalos nos comandos. Por exemplo, no comando de somatório, permitir que se defina como operandos propriedades de células que estejam em determinado intervalo. Atualmente só é possível realizar a soma de propriedades selecionando os eixos a serem percorridos.
- Implementar funções que dêem ao projetista a possibilidade de definir a forma que os comandos de uma ação são executados. Atualmente os comandos são executados apenas seqüencialmente e as precondições só podem ser associadas à ação e não a cada comando individualmente.
- Implementar maneiras para que o projetista possa acrescentar "ruído" nas percepções dos agentes e nos parâmetros enviados pelos agentes ao ambiente. Aumentando dessa forma, a complexidade e o realismo dos ambientes simulados.
- Implementar funções para que o simulador de ambientes responda, através do SACI, a consultas KQML, como por exemplo consultas do tipo `ask-if`.

- Implementar uma camada de software que faça uso direto do interpretador AgentSpeak(XL), eliminado a comunicação via *sockets*, o que aumentaria a performance da simulação.

- A implementação de uma interface através da qual agentes humanos possam participar da simulações.

Outras tarefas importantes a serem realizadas seria a implementação de melhorias na interface gráfica para a definição de ambientes, como por exemplo implementar formas para automatizar a execução das simulações.

Os algoritmos definidos na Secao 5.6 para um ciclo de simulação não são absolutos, existem outras maneiras para a implementação de ciclos de execução, tanto síncronos quanto assíncronos. A realização de um estudo sobre outras formas para execução do ambiente, permitindo ao projetista definir, de acordo com a necessidade de sua simulação, a forma como deseja a execução do ambiente, complementando os modos síncrono e assíncrono como apresentados neste trabalho.

Em um trabalho a longo termo, seria interessante a investigação dos mecanismos para reconciliar cognição e emergência seguindo as idéias de Castelfranchi (2001), e incorporando estes mecanismos na plataforma MASSOC, permitindo seu uso para o estudo do problema do elo micro-macro.

APÊNDICE A DTD ELMS

```
<!-- XML ELMS DTD
    DTD for the ELMS Language
    AUTHOR Fabio Y. Okuyama (okuyama@inf.ufrgs.br)
    VERSION: 0.1F (12/02/03) -->

<!-- Internal Entities -->
<!ENTITY % ATTdef "(BOOLEAN| INTEGER| FLOAT| STRING)+">

<!ENTITY % OPER "ADD| SUBTRACT| MULTIPLY| DIVIDE| MOD| SUM| PROD">

<!ENTITY % EVAL "EQUAL|UNEQUAL|GREATERTHAN|LESSTHAN|AND|OR">

<!ELEMENT ENVIRONMENT (DEFGRID?, RESOURCE*, AGENT+,
                       PERCEPTION+, ACTION+, REACTION*, OBSERVABLE*,
                       ENV*, INITIALIZATION?, (INSTANCE|GRID)*)>
        <!ATTLIST ENVIRONMENT NAME ID #REQUIRED>

<!ELEMENT RESOURCE (%ATTdef;, REACTIONS?)>
        <!ATTLIST RESOURCE NAME ID #REQUIRED>

<!ELEMENT AGENT (%ATTdef;, PERCEPTIONS, ACTIONS)>
        <!ATTLIST AGENT NAME ID #REQUIRED>

<!ELEMENT PERCEPTION (PRECONDITION?,(ELEMENT_ATT|CELL_ATT|CTRL)+)>
        <!ATTLIST PERCEPTION NAME ID #REQUIRED>

<!ELEMENT ACTION (PARAMETER*, PRECONDITION?, ASSIGN*,
                            (IN|OUT|MOVE|NEW|DELETE)*)>
        <!ATTLIST ACTION NAME ID #REQUIRED>

<!ELEMENT REACTION (PRECONDITION?, ASSIGN*,(IN|OUT|NEW|DELETE)*)>
        <!ATTLIST REACTION NAME ID #REQUIRED>

<!ELEMENT OBSERVABLE (ELEMENT_ATT|CELL_ATT|CTRL)+>

<!ELEMENT ENV EMPTY>
    <!ATTLIST ENV PROPERTY      NMTOKEN #REQUIRED
```

```
                    VALUE         NMTOKEN #REQUIRED>

<!ELEMENT CTRL EMPTY>
    <!ATTLIST CTRL PROPERTY NMTOKEN #REQUIRED>

<!ELEMENT INITIALIZATION (NEW*, IN*)>

<!ELEMENT INSTANCE (PROPERTY+)>
        <!ATTLIST INSTANCE  NAME  IDREF   #REQUIRED
                            INDEX NMTOKEN "0">

<!ELEMENT PROPERTY EMPTY>
        <!ATTLIST PROPERTY  NAME  IDREF   #REQUIRED
                            VALUE NMTOKEN #REQUIRED>

<!ELEMENT GRID (PROPERTY*, CONTENT)>
        <!ATTLIST GRID  X NMTOKEN       #REQUIRED
                        Y NMTOKEN       #REQUIRED
                        Z NMTOKEN       "0">

<!ELEMENT CONTENT EMPTY>
    <!ATTLIST CONTENT   NAME  IDREF     #REQUIRED
                        INDEX NMTOKEN   "0">

<!ELEMENT DEFGRID (%ATTdef;)>
    <!ATTLIST DEFGRID SIZEX NMTOKEN #REQUIRED
                      SIZEY NMTOKEN #REQUIRED
                      SIZEZ NMTOKEN "1">

<!ELEMENT BOOLEAN (#PCDATA|%EVAL;)*>
    <!ATTLIST BOOLEAN NAME ID #REQUIRED>

<!ELEMENT INTEGER (#PCDATA|%OPER;)*>
    <!ATTLIST INTEGER NAME ID #REQUIRED>

<!ELEMENT FLOAT (#PCDATA|%OPER;)*>
    <!ATTLIST FLOAT NAME ID #REQUIRED>

<!ELEMENT STRING EMPTY>
    <!ATTLIST STRING NAME ID #REQUIRED
                     VALUE CDATA "">

<!ELEMENT PERCEPTIONS EMPTY>
        <!ATTLIST PERCEPTIONS LIST IDREFS #REQUIRED>

<!ELEMENT ACTIONS EMPTY>
    <!ATTLIST ACTIONS LIST IDREFS #REQUIRED>
```

```
<!ELEMENT REACTIONS EMPTY>
    <!ATTLIST REACTIONS LIST IDREFS #REQUIRED>

<!ELEMENT PRECONDITION ((EQUAL|UNEQUAL|GREATERTHAN|LESSTHAN)+)>

<!ELEMENT ELEMENT (INDEX?)>
    <!ATTLIST ELEMENT NAME IDREF #REQUIRED>

<!ELEMENT ELEMENT_ATT (INDEX?)>
    <!ATTLIST ELEMENT_ATT NAME IDREF #REQUIRED
                      ATTRIBUTE IDREF #REQUIRED>

<!ELEMENT INDEX (#PCDATA|%OPER;| CELL_ATT|ELEMENT_ATT|RANDOM)*>

<!ELEMENT CELL_ATT (X, Y, Z?,(ATTRIBUTE|ELEMENT_ATT))>
    <!ATTLIST CELL_ATT NAME IDREF #REQUIRED
                   ATTRIBUTE IDREF #REQUIRED>

<!ELEMENT CELL (X, Y, Z?,(ATTRIBUTE))>

<!ELEMENT X (#PCDATA|%OPER;|CELL_ATT|ELEMENT_ATT|RANDOM)*>

<!ELEMENT Y (#PCDATA|%OPER;|CELL_ATT|ELEMENT_ATT|RANDOM)*>

<!ELEMENT Z (#PCDATA|%OPER;|CELL_ATT|ELEMENT_ATT|RANDOM)*>

<!ELEMENT ATTRIBUTE EMPTY>
    <!ATTLIST ELEMENT NAME IDREF #REQUIRED>

<!ELEMENT EQUAL (OPERAND, OPERAND)>

<!ELEMENT AND (OPERAND, OPERAND)>

<!ELEMENT OR (OPERAND, OPERAND)>

<ELEMENT  NOT (%EVAL;)>

<!ELEMENT UNEQUAL (OPERAND, OPERAND)>

<!ELEMENT GREATERTHAN (OPERAND, OPERAND)>

<!ELEMENT LESSTHAN (OPERAND, OPERAND)>

<!ELEMENT OPERAND (#PCDATA|%OPER;|%EVAL;|CELL_ATT|ELEMENT_ATT|RANDOM)*>

<!ELEMENT PARAMETER EMPTY>
    <!ATTLIST PARAMETER NAME IDREF #REQUIRED
                        TYPE (BOOLEAN|INTEGER|FLOAT|STRING) #REQUIRED>
```

```
<!ELEMENT ASSIGN ((ELEMENT_ATT|CELL_ATT), EXPRESSION)>
<!ELEMENT EXPRESSION (#PCDATA|%OPER;|%EVAL;|CELL_ATT|ELEMENT_ATT|RANDOM)*>
<!ELEMENT IN (ELEMENT, CELL)>
<!ELEMENT IN_RAND (ELEMENT, PRECONDITION?)>

<!ELEMENT OUT (ELEMENT, CELL)>

<!ELEMENT MOVE (ELEMENT, FROM, TO)>

<!ELEMENT FROM (CELL)>

<!ELEMENT TO (CELL)>

<!ELEMENT NEW (INDEX, N?, CELL?)>
    <!ATTLIST NEW NAME IDREF #REQUIRED
                  REFERENCE ID "new">

<!ELEMENT N (#PCDATA|%OPER;|CELL_ATT|ELEMENT_ATT|RANDOM)*>

<!ELEMENT DELETE (INDEX)>
    <!ATTLIST DELETE NAME IDREF #REQUIRED>

<!ELEMENT ADD (OPERAND, OPERAND)>

<!ELEMENT SUBTRACT (OPERAND, OPERAND)>

<!ELEMENT MULTIPLY (OPERAND, OPERAND)>

<!ELEMENT DIVIDE (OPERAND, OPERAND)>

<!ELEMENT MOD (OPERAND, OPERAND)>

<!ELEMENT SUM (CELL_ATT|ELEMENT_ATT)>

<!ELEMENT PROD (CELL_ATT|ELEMENT_ATT)>

<!ELEMENT RANDOM (MIN, MAX)>

<!ELEMENT RAND EMPTY>

<!ELEMENT MIN (#PCDATA)>

<!ELEMENT MAX (#PCDATA)>
```

APÊNDICE B EXEMPLO DE CÓDIGO ELMS

```
<ENVIRONMENT NAME = "AMBIENTE">
<DEFGRID SIZEX="10" SIZEY = "10">
   <BOOLEAN NAME = "ocupado"> "FALSE" </BOOLEAN>
   <BOOLEAN NAME = "comida"> "FALSE" </BOOLEAN>
</DEFGRID>

<RESOURCE NAME="comida">
   <INTEGER NAME="id_dono">  -1</INTEGER>
   <INTEGER NAME="comendo">  -1</INTEGER>
   <INTEGER NAME="forca_cen"> 0</INTEGER>
   <INTEGER NAME="forca_sup"> 0</INTEGER>
   <INTEGER NAME="forca_inf"> 0</INTEGER>
   <INTEGER NAME="forca_dir"> 0</INTEGER>
   <INTEGER NAME="forca_esq"> 0</INTEGER>
   <BOOLEAN NAME ="alocado">"FALSE"</BOOLEAN>
   <REACTIONS LIST="r_ataqueS r_ataqueI r_ataqueD r_ataqueE
                    reacao_comer aparecer"/>
</RESOURCE>

<AGENT NAME="agente">
   <INTEGER NAME="forca">  40</INTEGER>
   <INTEGER NAME="id"> "SELF"</INTEGER>
   <PERCEPTIONS LIST="visao olfato"/>
   <ACTIONS LIST="mover comer atacarS
                  atacarD atacarI atacarE"/>
</AGENT>

<PERCEPTION NAME="visao">
   <CELL_ATT ATTRIBUTE = "CONTENTS">
      <X> +0 </X> <Y> +0 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "CONTENTS">
      <X> +0 </X> <Y> -1 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "CONTENTS">
      <X> +1 </X> <Y> +0 </Y>
   </CELL_ATT>
```

```
  <CELL_ATT ATTRIBUTE = "CONTENTS">
      <X> +0 </X> <Y> +1 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "CONTENTS">
      <X> -1 </X> <Y> +0 </Y>
   </CELL_ATT>
</PERCEPTION>

<PERCEPTION NAME="olfato">
   <CELL_ATT ATTRIBUTE = "comida">
      <X> +0 </X> <Y> -2 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "comida">
      <X> -1 </X> <Y> -1 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "comida">
      <X> +1 </X> <Y> -1 </Y>
   </CELL_ATT>
  <CELL_ATT ATTRIBUTE = "comida">
      <X> -2 </X> <Y> +0 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "comida">
      <X> +2 </X> <Y> +0 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "comida">
      <X> -1 </X> <Y> +1 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "comida">
      <X> +1 </X> <Y> +1 </Y>
   </CELL_ATT>
   <CELL_ATT ATTRIBUTE = "comida">
      <X> +0 </X> <Y> +2 </Y>
   </CELL_ATT>
</PERCEPTION>

<REACTION NAME="r_ataqueS">
   <PRECONDITION>
      <GREATERTHAN>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
         <OPERATOR>
            <ELEMENT_ATT NAME= "comida"  ATTRIBUTE= "forca_cen">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
```

```
   </GREATERTHAN>
   <GREATERTHAN>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_dir">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
   </GREATERTHAN>
   <GREATERTHAN>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_inf">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
   </GREATERTHAN>
     <GREATERTHAN>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_esq">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
   </GREATERTHAN>
</PRECONDITION>
<MOVE>
     <ELEMENT NAME = "SELFCLASS">
        <INDEX>"SELF"</INDEX>
     </ELEMENT>
     <FROM>
        <CELL>
          <X>+0</X> <Y>+0</Y>
        </CELL>
     </FROM>
     <TO>
```

```
            <CELL>
                <X>+0</X> <Y>-1</Y>
            </CELL>
          </TO>
        </MOVE>
        <ASSIGN>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
                <INDEX>"SELF"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION> 0 </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_cen">
                <INDEX>"SELF"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION> 0 </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_inf">
                <INDEX>"SELF"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION> 0 </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_dir">
                <INDEX>"SELF"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION> 0 </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_esq">
                <INDEX>"SELF"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION> 0 </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="comendo">
                <INDEX>"SELF"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION> -1 </EXPRESSION>
        </ASSIGN>
</REACTION>

<REACTION NAME="r_ataqueD">
   <PRECONDITION>
      <GREATERTHAN>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_dir">
```

```
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_cen">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
   </GREATERTHAN>
   <GREATERTHAN>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_dir">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
   </GREATERTHAN>
   <GREATERTHAN>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_dir">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_inf">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
   </GREATERTHAN>
   <GREATERTHAN>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_dir">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_esq">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
   </GREATERTHAN>
</PRECONDITION>
<MOVE>
     <ELEMENT NAME = "SELFCLASS">
```

```
      <INDEX>"SELF"</INDEX>
  </ELEMENT>
  <FROM>
      <CELL>
          <X>+0</X> <Y>+0</Y>
      </CELL>
      </FROM>
  <TO>
      <CELL>
          <X>+1</X> <Y>+0</Y>
      </CELL>
  </TO>
</MOVE>
<ASSIGN>
    <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
        <INDEX>"SELF"</INDEX>
    </ELEMENT_ATT>
    <EXPRESSION> 0 </EXPRESSION>
</ASSIGN>
<ASSIGN>
    <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_cen">
        <INDEX>"SELF"</INDEX>
    </ELEMENT_ATT>
    <EXPRESSION> 0 </EXPRESSION>
</ASSIGN>
<ASSIGN>
    <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_inf">
        <INDEX>"SELF"</INDEX>
    </ELEMENT_ATT>
    <EXPRESSION> 0 </EXPRESSION>
</ASSIGN>
<ASSIGN>
    <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_dir">
        <INDEX>"SELF"</INDEX>
    </ELEMENT_ATT>
    <EXPRESSION> 0 </EXPRESSION>
</ASSIGN>
<ASSIGN>
    <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_esq">
        <INDEX>"SELF"</INDEX>
    </ELEMENT_ATT>
    <EXPRESSION> 0 </EXPRESSION>
</ASSIGN>
<ASSIGN>
    <ELEMENT_ATT NAME="comida" ATTRIBUTE="comendo">
        <INDEX>"SELF"</INDEX>
    </ELEMENT_ATT>
    <EXPRESSION> -1 </EXPRESSION>
```

```
      </ASSIGN>
</REACTION>

<REACTION NAME="r_ataqueI">
   <PRECONDITION>
      <GREATERTHAN>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_inf">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_cen">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
      </GREATERTHAN>
      <GREATERTHAN>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_inf">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_dir">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
      </GREATERTHAN>
        <GREATERTHAN>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_inf">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
        </GREATERTHAN>
        <GREATERTHAN>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_inf">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
         <OPERATOR>
```

```
          <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_esq">
             <INDEX> "SELF" </INDEX>
          </ELEMENT_ATT>
       </OPERATOR>
      </GREATERTHAN>
</PRECONDITION>
<MOVE>
     <ELEMENT NAME = "SELFCLASS">
        <INDEX>"SELF"</INDEX>
     </ELEMENT>
     <FROM>
         <CELL>
             <X>+0</X> <Y>+0</Y>
         </CELL>
         </FROM>
     <TO>
         <CELL>
             <X>+0</X> <Y>+1</Y>
         </CELL>
     </TO>
   </MOVE>
   <ASSIGN>
       <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
           <INDEX>"SELF"</INDEX>
       </ELEMENT_ATT>
       <EXPRESSION> 0 </EXPRESSION>
   </ASSIGN>
   <ASSIGN>
       <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_cen">
           <INDEX>"SELF"</INDEX>
       </ELEMENT_ATT>
       <EXPRESSION> 0 </EXPRESSION>
   </ASSIGN>
   <ASSIGN>
       <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_inf">
           <INDEX>"SELF"</INDEX>
       </ELEMENT_ATT>
       <EXPRESSION> 0 </EXPRESSION>
   </ASSIGN>
   <ASSIGN>
       <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_dir">
           <INDEX>"SELF"</INDEX>
       </ELEMENT_ATT>
       <EXPRESSION> 0 </EXPRESSION>
   </ASSIGN>
   <ASSIGN>
       <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_esq">
           <INDEX>"SELF"</INDEX>
```

```
            </ELEMENT_ATT>
            <EXPRESSION> 0 </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="comendo">
                <INDEX>"SELF"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION> -1 </EXPRESSION>
        </ASSIGN>
</REACTION>

<REACTION NAME="r_ataqueE">
   <PRECONDITION>
      <GREATERTHAN>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_esq">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_cen">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
      </GREATERTHAN>
      <GREATERTHAN>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_esq">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
         <OPERATOR>
                <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_dir">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
      </GREATERTHAN>
      <GREATERTHAN>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_esq">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_inf">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
```

```
   </GREATERTHAN>
   <GREATERTHAN>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_esq">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
      <OPERATOR>
         <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
            <INDEX> "SELF" </INDEX>
         </ELEMENT_ATT>
      </OPERATOR>
   </GREATERTHAN>
</PRECONDITION>
<MOVE>
     <ELEMENT NAME = "SELFCLASS">
        <INDEX>"SELF"</INDEX>
     </ELEMENT>
     <FROM>
         <CELL>
             <X>+0</X> <Y>+0</Y>
         </CELL>
         </FROM>
      <TO>
         <CELL>
             <X>-1</X> <Y>+0</Y>
         </CELL>
     </TO>
   </MOVE>
   <ASSIGN>
       <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_sup">
           <INDEX>"SELF"</INDEX>
       </ELEMENT_ATT>
       <EXPRESSION> 0 </EXPRESSION>
   </ASSIGN>
   <ASSIGN>
       <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_cen">
           <INDEX>"SELF"</INDEX>
       </ELEMENT_ATT>
       <EXPRESSION> 0 </EXPRESSION>
   </ASSIGN>
   <ASSIGN>
       <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_inf">
           <INDEX>"SELF"</INDEX>
       </ELEMENT_ATT>
       <EXPRESSION> 0 </EXPRESSION>
   </ASSIGN>
   <ASSIGN>
```

```
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_dir">
                <INDEX>"SELF"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION> 0 </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="forca_esq">
                <INDEX>"SELF"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION> 0 </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="comendo">
                <INDEX>"SELF"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION> -1 </EXPRESSION>
        </ASSIGN>
</REACTION>

<REACTION NAME="reacao_comer">
   <PRECONDITION>
      <EQUAL>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="comendo">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
         <OPERATOR> 2 </OPERATOR>
      </EQUAL>
   </PRECONDITION>
      <ASSIGN>
          <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
              <INDEX>
                <ELEMENT_ATT NAME="comida" ATTRIBUTE="id_dono">
                   <INDEX> "SELF" </INDEX>
                </ELEMENT_ATT>
              </INDEX>
          </ELEMENT_ATT>
          <EXPRESSION>
             <ADD>
               <OPERATOR>
                 <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
                    <INDEX>
                       <ELEMENT_ATT NAME = "comida"
                                ATTRIBUTE = "id_dono">
                           <INDEX> "SELF" </INDEX>
                       </ELEMENT_ATT>
                    </INDEX>
```

```
                </ELEMENT_ATT>
              </OPERATOR>
             <OPERATOR> 20 </OPERATOR>
            </ADD>
          </EXPRESSION>
      </ASSIGN>
      <OUT>
        <ELEMENT NAME = "SELFCLASS">
           <INDEX>"SELF"</INDEX>
        </ELEMENT>
        <CELL> <X>+0</X> <Y>+0</Y> </CELL>
      </OUT>
      <ASSIGN>
          <ELEMENT_ATT NAME="comida" ATTRIBUTE="alocado">
              <INDEX> "SELF" </INDEX>
          </ELEMENT_ATT>
          <EXPRESSION> "FALSE" </EXPRESSION>
      </ASSIGN>
</REACTION>

<REACTION NAME="aparecer">
   <PRECONDITION>
      <EQUAL>
         <OPERATOR>
            <ELEMENT_ATT NAME="comida" ATTRIBUTE="alocado">
               <INDEX> "SELF" </INDEX>
            </ELEMENT_ATT>
         </OPERATOR>
         <OPERATOR> "FALSE" </OPERATOR>
      </EQUAL>
   </PRECONDITION>
   <IN_RAND>
        <PRECONDITION>
            <EQUAL>
                <OPERAND>
                     <CELL_ATT ATTRIBUTE = "comida">
                        <X>"X"</X>
                        <Y>"Y"</Y>
                    </CELL_ATT>
                </OPERAND>
                <OPERAND> "FALSE" </OPERAND>
            </EQUAL>
        </PRECONDITION>
        <ELEMENT NAME = "SELFCLASS">
           <INDEX>"SELF"</INDEX>
        </ELEMENT>
   </IN_RAND>
   <ASSIGN>
```

```
        <CELL_ATT ATTRIBUTE = "comida">
           <X>+0</X> <Y>+0</Y>
        </CELL_ATT>
        <EXPRESSION> "TRUE" </EXPRESSION>
   </ASSIGN>
</REACTION>

<ACTION NAME="mover">
   <PARAMETER NAME="XNOVO" TYPE="INTEGER"/>
   <PARAMETER NAME="YNOVO" TYPE="INTEGER"/>
   <PRECONDITION>
      <EQUAL>
         <OPERATOR>
            <CELL_ATT ATTRIBUTE = "ocupado">
               <X> +"XNOVO" </X> <Y> +"YNOVO" </Y>
            </CELL_ATT>
         </OPERATOR>
         <OPERATOR> "FALSE"</OPERATOR>
      </EQUAL>
   </PRECONDITION>
        <ASSIGN>
          <CELL_ATT ATTRIBUTE = "ocupado">
             <X>+0</X> <Y>+0</Y>
          </CELL_ATT>
          <EXPRESSION> "TRUE" </EXPRESSION>
        </ASSIGN>
        <MOVE>
           <ELEMENT NAME = "SELFCLASS">
              <INDEX>"SELF"</INDEX>
           </ELEMENT>
           <FROM>
              <CELL> <X>+0</X> <Y>+0</Y> </CELL>
           </FROM>
           <TO>
              <CELL>
                <X> +"XNOVO" </X> <Y> +"YNOVO" </Y>
              </CELL>
           </TO>
        </MOVE>
        <ASSIGN>
          <CELL_ATT ATTRIBUTE = "ocupado">
               <X>+0</X> <Y>+0</Y>
          </CELL_ATT>
          <EXPRESSION> "TRUE" </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
          <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
               <INDEX>"SELF"</INDEX>
```

```
          </ELEMENT_ATT>
          <EXPRESSION>
             <SUBTRACT>
                <OPERATOR>
                  <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                   <INDEX>"SELF"</INDEX>
                  </ELEMENT_ATT>
                </OPERATOR>
                 <OPERATOR> 1 </OPERATOR>
             </SUBTRACT>
          </EXPRESSION>
        </ASSIGN>
</ACTION>

<ACTION NAME="comer">
   <PARAMETER NAME="ID" TYPE="INTEGER"/>
   <PARAMETER NAME="IDcomida" TYPE="INTEGER"/>
   <PRECONDITION>
      <EQUAL>
          <OPERATOR>
             <ELEMENT_ATT NAME="comida" ATTRIBUTE="comendo">
                <INDEX>"SELF"</INDEX>
             </ELEMENT_ATT>
          </OPERATOR>
          <OPERATOR> -1</OPERATOR>
      </EQUAL>
   </PRECONDITION>
   <ASSIGN>
        <ELEMENT_ATT NAME="comida" ATTRIBUTE="comendo">
           <INDEX>"IDcomida"</INDEX>
        </ELEMENT_ATT>
        <EXPRESSION> 1 </EXPRESSION>
   </ASSIGN>
   <ASSIGN>
        <ELEMENT_ATT NAME="comida" ATTRIBUTE="id_dono">
        <INDEX>"IDcomida"</INDEX>
        </ELEMENT_ATT>
        <EXPRESSION> "ID" </EXPRESSION>
   </ASSIGN>
</ACTION>

<ACTION NAME="atacarS">
   <PARAMETER NAME="ATACADO" TYPE="INTEGER"/>
   <PARAMETER NAME="idComida" TYPE="INTEGER"/>
   <PRECONDITION>
        <EQUAL>
            <OPERATOR>
                <CELL_ATT ATTRIBUTE = "ocupado">
```

```
                <X>+0</X> <Y>-1</Y>
              </CELL_ATT>
          </OPERATOR>
          <OPERATOR>"TRUE"</OPERATOR>
      </EQUAL>
      <EQUAL>
          <OPERATOR>
              <CELL_ATT ATTRIBUTE = "ocupado">
                <X>+0</X> <Y>-1</Y>
              </CELL_ATT>
          </OPERATOR>
          <OPERATOR>"TRUE"</OPERATOR>
      </EQUAL>
</PRECONDITION >
      <ASSIGN>
         <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
            <INDEX>"SELF"</INDEX>
         </ELEMENT_ATT>
         <EXPRESSION>
         <SUBTRACT>
            <OPERATOR>
              <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                 <INDEX>"SELF"</INDEX>
              </ELEMENT_ATT>
            </OPERATOR>
            <OPERATOR> 4 </OPERATOR>
         </SUBTRACT>
         </EXPRESSION>
      </ASSIGN>
      <ASSIGN>
         <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
            <INDEX>"ATACADO"</INDEX>
         </ELEMENT_ATT>
         <EXPRESSION>
         <SUBTRACT>
            <OPERATOR>
              <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                 <INDEX>"ATACADO"</INDEX>
              </ELEMENT_ATT>
            </OPERATOR>
            <OPERATOR> 4 </OPERATOR>
         </SUBTRACT>
         </EXPRESSION>
      </ASSIGN>
      <ASSIGN>
         <ELEMENT_ATT NAME = "comida" ATTRIBUTE = "forca_cen">
            <INDEX>"idComida"</INDEX>
         </ELEMENT_ATT>
```

```
            <EXPRESSION>
                 <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                    <INDEX>"ATACADO"</INDEX>
                 </ELEMENT_ATT>
            </EXPRESSION>
         </ASSIGN>
         <ASSIGN>
            <ELEMENT_ATT NAME = "comida" ATTRIBUTE = "forca_inf">
               <INDEX>"idComida"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION>
                 <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                    <INDEX>"SELF"</INDEX>
                 </ELEMENT_ATT>
            </EXPRESSION>
         </ASSIGN>
</ACTION>

<ACTION NAME="atacarD">
   <PARAMETER NAME="ATACADO" TYPE="INTEGER"/>
   <PARAMETER NAME="idComida" TYPE="INTEGER"/>
   <PRECONDITION>
        <EQUAL>
            <OPERATOR>
                <CELL_ATT ATTRIBUTE = "ocupado">
                  <X>+1</X> <Y>+0</Y>
                </CELL_ATT>
            </OPERATOR>
            <OPERATOR>"TRUE"</OPERATOR>
        </EQUAL>
        <EQUAL>
            <OPERATOR>
                <CELL_ATT ATTRIBUTE = "ocupado">
                  <X>+1</X> <Y>+0</Y>
                </CELL_ATT>
            </OPERATOR>
            <OPERATOR>"TRUE"</OPERATOR>
        </EQUAL>
   </PRECONDITION>
        <ASSIGN>
           <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
              <INDEX>"SELF"</INDEX>
           </ELEMENT_ATT>
           <EXPRESSION>
           <SUBTRACT>
              <OPERATOR>
               <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                   <INDEX>"SELF"</INDEX>
```

```
                </ELEMENT_ATT>
              </OPERATOR>
              <OPERATOR> 4 </OPERATOR>
           </SUBTRACT>
           </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
           <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
              <INDEX>"ATACADO"</INDEX>
           </ELEMENT_ATT>
           <EXPRESSION>
           <SUBTRACT>
              <OPERATOR>
                <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                   <INDEX>"ATACADO"</INDEX>
                </ELEMENT_ATT>
              </OPERATOR>
              <OPERATOR> 4 </OPERATOR>
           </SUBTRACT>
           </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
           <ELEMENT_ATT NAME = "comida" ATTRIBUTE = "forca_cen">
              <INDEX>"idComida"</INDEX>
           </ELEMENT_ATT>
           <EXPRESSION>
                <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                   <INDEX>"ATACADO"</INDEX>
                </ELEMENT_ATT>
           </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
           <ELEMENT_ATT NAME = "comida" ATTRIBUTE = "forca_esq">
              <INDEX>"idComida"</INDEX>
           </ELEMENT_ATT>
           <EXPRESSION>
                <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                   <INDEX>"SELF"</INDEX>
                </ELEMENT_ATT>
           </EXPRESSION>
        </ASSIGN>
</ACTION>

<ACTION NAME="atacarI">
   <PARAMETER NAME="ATACADO" TYPE="INTEGER"/>
   <PARAMETER NAME="idComida" TYPE="INTEGER"/>
   <PRECONDITION>
        <EQUAL>
```

```
            <OPERATOR>
                <CELL_ATT ATTRIBUTE = "ocupado">
                  <X>+0</X> <Y>+1</Y>
                </CELL_ATT>
            </OPERATOR>
            <OPERATOR>"TRUE"</OPERATOR>
        </EQUAL>
        <EQUAL>
            <OPERATOR>
                <CELL_ATT ATTRIBUTE = "ocupado">
                  <X>+0</X> <Y>+1</Y>
                </CELL_ATT>
            </OPERATOR>
            <OPERATOR>"TRUE"</OPERATOR>
        </EQUAL>
    </PRECONDITION >
        <ASSIGN>
           <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
              <INDEX>"SELF"</INDEX>
           </ELEMENT_ATT>
           <EXPRESSION>
           <SUBTRACT>
              <OPERATOR>
                <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                   <INDEX>"SELF"</INDEX>
                </ELEMENT_ATT>
              </OPERATOR>
              <OPERATOR> 4 </OPERATOR>
           </SUBTRACT>
           </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
           <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
              <INDEX>"ATACADO"</INDEX>
           </ELEMENT_ATT>
           <EXPRESSION>
           <SUBTRACT>
              <OPERATOR>
                <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                   <INDEX>"ATACADO"</INDEX>
                </ELEMENT_ATT>
              </OPERATOR>
              <OPERATOR> 4 </OPERATOR>
           </SUBTRACT>
           </EXPRESSION>
        </ASSIGN>
        <ASSIGN>
           <ELEMENT_ATT NAME = "comida" ATTRIBUTE = "forca_cen">
```

```
                <INDEX>"idComida"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION>
                 <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                    <INDEX>"ATACADO"</INDEX>
                 </ELEMENT_ATT>
            </EXPRESSION>
         </ASSIGN>
         <ASSIGN>
            <ELEMENT_ATT NAME = "comida" ATTRIBUTE = "forca_sup">
               <INDEX>"idComida"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION>
                 <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                    <INDEX>"SELF"</INDEX>
                 </ELEMENT_ATT>
            </EXPRESSION>
         </ASSIGN>
</ACTION>

<ACTION NAME="atacarE">
   <PARAMETER NAME="ATACADO" TYPE="INTEGER"/>
   <PARAMETER NAME="idComida" TYPE="INTEGER"/>
   <PRECONDITION>
         <EQUAL>
             <OPERATOR>
                 <CELL_ATT ATTRIBUTE = "ocupado">
                   <X>-1</X> <Y>+0</Y>
                 </CELL_ATT>
             </OPERATOR>
             <OPERATOR>"TRUE"</OPERATOR>
         </EQUAL>
         <EQUAL>
             <OPERATOR>
                 <CELL_ATT ATTRIBUTE = "ocupado">
                   <X>-1</X> <Y>+0</Y>
                 </CELL_ATT>
             </OPERATOR>
             <OPERATOR>"TRUE"</OPERATOR>
         </EQUAL>
   </PRECONDITION >
         <ASSIGN>
            <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
               <INDEX>"SELF"</INDEX>
            </ELEMENT_ATT>
            <EXPRESSION>
            <SUBTRACT>
               <OPERATOR>
```

```
                <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                   <INDEX>"SELF"</INDEX>
                </ELEMENT_ATT>
             </OPERATOR>
             <OPERATOR> 4 </OPERATOR>
          </SUBTRACT>
          </EXPRESSION>
       </ASSIGN>
       <ASSIGN>
          <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
             <INDEX>"ATACADO"</INDEX>
          </ELEMENT_ATT>
          <EXPRESSION>
          <SUBTRACT>
             <OPERATOR>
               <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                  <INDEX>"ATACADO"</INDEX>
               </ELEMENT_ATT>
             </OPERATOR>
             <OPERATOR> 4 </OPERATOR>
          </SUBTRACT>
          </EXPRESSION>
       </ASSIGN>
       <ASSIGN>
          <ELEMENT_ATT NAME = "comida" ATTRIBUTE = "forca_cen">
             <INDEX>"idComida"</INDEX>
          </ELEMENT_ATT>
          <EXPRESSION>
               <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                  <INDEX>"ATACADO"</INDEX>
               </ELEMENT_ATT>
          </EXPRESSION>
       </ASSIGN>
       <ASSIGN>
          <ELEMENT_ATT NAME = "comida" ATTRIBUTE = "forca_dir">
             <INDEX>"idComida"</INDEX>
          </ELEMENT_ATT>
          <EXPRESSION>
               <ELEMENT_ATT NAME="agente" ATTRIBUTE="forca">
                  <INDEX>"SELF"</INDEX>
               </ELEMENT_ATT>
          </EXPRESSION>
       </ASSIGN>
</ACTION>

<OBSERVABLE>
   <CELL_ATT ATTRIBUTE = "CONTENTS">
      <X> "ALL"</X> <Y> "ALL"</Y>
```

```
   </CELL_ATT>
   <ELEMENT_ATT NAME = "agente" ATTRIBUTE = "forca">
      <INDEX>"ALL"</INDEX>
   </ELEMENT_ATT>
</OBSERVABLE>
</ENVIRONMENT>
```

APÊNDICE C PRINCIPAIS CLASSES ELMS

Classe	Extends	Descrição
ActionCall	Object	Armazena uma Ação escolhida e seus parâmetros.
ActionDef	ReactionDef	Armazena definição de Ação.
AgentDef	ElmsClass	Armazena a definição de uma classe de agente.
AgentInst	ElmsClassIntance	Armazena dados sobre uma instância de agente.
Assign	Command	Armazena um comando de atribuição.
Cell	Object	Armazena informações de uma célula.
Command	Object	Classe abstrata da qual se derivam os comandos.
Condition	ExpressionNode	Armazena uma condição.
Coordinates	Object	Armazena uma coordenada da grade.
ElmsClass	Object	Classe abstrata de classes ELMS.
ElmsInterpreter	Object	Classe que realiza o parsing do arquivo-fonte.
ElmsAsynExecutor	Object	Classe que executa ambientes em modo assíncrono.
ElmsSynExectuor	Object	Classe que executa ambientes em modo síncrono.
EnvironmentData	Object	Classe que armazena dados de um ambiente.
ExpressionNode	Object	Armazena nodo de uma árvore de expressões.
GridCommand	Command	Armazena comandos relacionados a grade.
Instance	Object	Armazena uma referência a uma instância.
IntExpression	Object	Armazena uma expressão inteira.
ObservablesPrinter	Object	Controla a saída de dados observáveis.
OperationNode	ExpressionNode	Armazena uma operação matemática.
Param	Object	Armazena um parâmetro de uma ação.
PerceptionDef	Object	Armazena a definição de uma percepção.
PreconditionTree	Object	Nó raiz de uma árvore de condições.
Property	Object	Uma referência a uma propriedade de um elemento.
Propertyarray	Object	Estrutura de dados que armazena propriedades.
ReactionDef	Object	Estrutura que armazena a definição de uma reação.
ResourceDef	Object	Estrutura que armazena a definição de um recurso.
ResourceInst	ElmsClassInstance	Armazena dados sobre uma instância de recurso.
TermNode	ExpressionNode	Armazena um nodo terminal de uma expressão.
Value	Object	Armazena um valor de tipo variável.

APÊNDICE D MENSAGENS TROCADAS ENTRE AGENTES E SIMULADOR

Este anexo apresenta as comunicações realizadas, através do SACI, entre a interface e os agentes e também do processo que controla o ambiente com os agentes da simulação. Este anexo tem como objetivo ser uma "referência rápida" para quem desejar implementar agentes que utilizem ambientes ELMS.

Assim que a camada de software que conecta com o interpretador AgentSpeak(XL) é iniciada, ela dispara um processe executando o interpretador e se conecta à sociedade SACI, cujo nome da sociedade e máquina onde se encontra foram previamente especificados pelo usuário. Campos como `sender`, `reply-with` e `in-reply-to` são automaticamente preenchidos pelo SACI.

- Após registrar-se com um nome provisório na sociedade SACI, o agente, irá enviar uma mensagem endereçada para "MATRIX"[1] com a performativa `ask-name` e no campo `content` o agente irá preencher o nome da classe a qual pertence. Por exemplo:

  (ask-name
  :receiver MATRIX
  :content agente
  :sender noname00
  :reply-with id1)

- Este agente irá receber como resposta do controlador de ambientes:

  (tell
  :receiver noname00
  :sender MATRIX
  :in-reply-to id1
  :content agente0)

  A mensagem acima, com a performativa `tell` está informando ao agente que o nome do agente na sociedade é `agente0`. Após receber essa mensagem o agente deverá se registrar na sociedade SACI com o nome informado pelo controlador de ambientes.

[1] Nome padrão para o processo que controla o ambiente.

- As percepções são enviadas em mensagens com a seguinte forma:

  (perceived
  :receiver agente0
  :sender MATRIX
  :content *lista de percepções*)

  A lista de percepções é um objeto *Vector* que contém objetos `String` representando as percepções.

- As ações são enviadas com a seguinte formato:

  (action
  :receiver MATRIX
  :sender agente0
  :agentType agente
  :agentIndex 0
  :step 1
  :content *lista de parâmetros*)

  O campo `step` indica a que passo da simulação a ação se refere (no caso da simulação síncrona). A lista de parâmetros é um objeto *Vector* que contém objetos `String` representando os parâmetros.

- Caso a ação selecionada pelo agente falhe, o agente receberá a seguinte mensagem:

  (tell
  :receiver agente0
  :sender MATRIX
  :content FAIL)

- O agente é finalizado após receber a seguinte mensagem da interface:

  (control
  :receiver agente0
  :sender INTERFACE
  :content END)

A camada de software conectando o interpretador AgentSpeak(XL) com a sociedade SACI, ao receber estas mensagem, envia pelo socket mensagens no formato aceito pelo interpretador AgentSpeak(XL). Da mesma forma, as mensagens emitidas pelo interpretador AgentSpeak(XL) são inseridas em mensagens KQML para transmissão via SACI.

REFERÊNCIAS

ÁLVARES, L. O. C.; SICHMAN, J. S. Introdução aos Sistemas Multiagentes. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, JAI, 16.,1997, Brasília. **Anais...** Brasília : Unb, 1997. [S.l.: s.n.], 1997.

BAILEY, J.; GEORGEFF, M.; KEMP, D.; KINNY, D. Active databases and agent systems - a comparison. In: INTERNACIONAL WORKSHOP ON RULES IN DATABASE SYSTEMS, 2., 1995, Athens, Greece. **Rules in Database**: proceedings. Berlim: Springer-Verlag, 1995. p. 342-356 (Lecture Notes in Computer Science, 985).

BORDINI, R. H.; BAZZAN, A. L. C.; JANNONE, R. O.; BASSO, D. M.; VICARI, R. M.; LESSER, V. R. AgentSpeak(XL): efficient intention selection in BDI agents via decision-theoretic task scheduling. In: INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTI-AGENT SYSTEMS, AAMAS, 1., 2002, Bologna, Italy. **Bringing People and Agents Together**: proceedings. New York: ACM, 2002. p.1294-1302.

BORDINI, R. H.; MOREIRA, Á. F. Proving the Asymmetry Thesis Principles for a BDI Agent-Oriented Programming Language. **Electronic Notes in Theoretical Computer Science**, Amsterdam, v.70, n.5, 2002. Disponível em: <http://www.elsevier.nl/locate/entcs/volume70.html>. Acesso em: jan. 2003.

BORDINI, R. H.; VIEIRA, R.; MOREIRA, Á. F. Fundamentos de Sistemas Multiagentes. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, JAI, 20.,2001, Fortaleza. **Anais...** Fortaleza: Sociedade Brasileira de Computação, 2001. v.2 p.3–41.

BRATMAN, M. E.; ISRAEL, D. J.; POLLACK, M. E. Plans and Resource-Bounded Practical Reasoning. **Computational Intelligence**, [S.l.], v.4, p.349–355, 1988.

CARLEY, K. M.; GASSER, L. Computer Oranization Theory. In: WEISS, G. (Ed.). **Multiagent Systems:** a modern approach to distributed artificial intelligence. Cambridge, MA: MIT Press, 1999. p.299–330.

CASTELFRANCHI, C. Simulating with Cognitive Agents: the importance of cognitive emergence. In: INTERNATIONAL WORKSHOP ON MULTI-AGENT SYSTEMS AN AGENT BASED SIMULATION, MABS,1., 1998, Paris. **Multi-Agent Systems and Agent-Based Simulation**: proceedings. Berlin: Springer-Verlag, 1998. p. 26-44. (Lecture Notes in Artificial Intelligence, 1534).

CASTELFRANCHI, C. The Theory of Social Functions: challenges for computational social science and multi-agent learning. **Cognitive Systems Research**, Amsterdan, v.2, n.1, p.5–38, Apr. 2001.

CASTELFRANCHI, C.; CONTE, R.; PAOLUCCI, M. Normative reputation and the costs of compliance. **JASSS**, Guildford, UK, v.1, n.3, 1998. Disponível em: <http://www.soc.surrey.ac.uk/JASSS/1/3/3.html>. Acesso em: mar. 2001.

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

CONTE, R.; CASTELFRANCHI, C. **Cognitive and Social Action**. London: UCL Press, 1995.

CONTE, R.; NIGEL, G.; SICHMAN, J. MAS and Social Simulation: a suitable commitment. In: INTERNATIONAL WORKSHOP ON MULTI-AGENT SYSTEMS AND AGENT-BASED SIMULATION,MABS, 1., 1998, Paris. **Multi-Agent Systems and Agent-Based Simulation**: proceedings. Berlin: Springer-Verlag, 1998. p1–9.(Lecture Notes in Artificial Intelligence, 1534).

CORTEN, E. et al. **Soccerserver Manual**: ver. 5 rev. 00 beta. 1999. Disponível em: <http://citeseer.nj.nec.com/corten99soccerserver.html>. Acesso em: ago. 2001.

DEMAZEAU, Y. From Cognitive Interactions to Collective Behaviour in Agent-Based Systems. In: EUROPEAN CONFERENCE ON COGNITIVE SCIENCE, 1995, Saint-Malo. **Proceedings...** [S.l.: s.n.], 1995.

D'INVERNO, M.; LUCK, M. Engineering AgentSpeak(L): A Formal Computational Model. **Journal of Logic and Computation**, Oxford, v.8, n.3, p.1–27, 1998.

DROGOUL, A.; FERBER, J. Multi-Agent Simulation as a Tool for Modeling Societies: application to social differentiation in ant colonies. In: EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD, MAAMAW, 4., 1992, San Martino al Cimino, Italy. **Artificial Social Systems**. Berlin: Springer-Verlag, 1994. p.3–23. (Lecture Notes in Computer Science, v.830).

FERBER, J. **Multi-Agent Systems**: an introduction to distributed artificial intelligence. London: Addison-Wesley, 1999.

FININ, T.; FRITZSON, R.; MCKAY, D. A Language and Protocol to Support Intelligent Agent Interoperability. In: CE & CALS WASHINGTON CONFERENCE, 1992. **Proceedings...** [S.l.: s.n.], 1992. Disponível em: <http://www.cs.umbc.edu/kqml/papers/>. Acesso em: mar. 2001.

FININ, T. et al. KQML as an Agent Communication Language. In: INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, CIKM, 3., 1994. **Proceedings...** New York:ACM Press, 1994a. Disponível em: <http://www.cs.umbc.edu/kqml/papers/>. Acesso em: mar. 2001.

FININ, T. et al. **Specification of the KQML Agent-Communication Language (plus example agent policies and architectures)**. [S.l.]: The DARPA Knowledge Sharing Initiative—External Interfaces Working Group, 1994b. Disponível em: <http://logic.stanford.edu/sharing/papers/>. Acesso em: mar. 2001.

GASSER, L. Distribution and Coordination of Tasks among Intelligent Agents. In: SCANDINAVIAN CONFERENCE ON ARTIFICIAL INTELLIGENCE, 1., 1987, Tromsø, Norway. **Proceedings...** [S.l.: s.n.], 1987.

GILBERT, N.; TROITZSCH, K. G. **Simulation for the Social Scientist**. [S.l.]: Open University Press, 1999.

GUTKNECHT, O.; FERBER, J. The MADKIT Agent Platform Architecture. In: AGENTS WORKSHOP ON INFRASTRUCTURE FOR MULTI-AGENT SYSTEMS, 2000. **Proceedings...** [S.l.: s.n.], 2000. p.48–55.

GUTKNECHT, O.; FERBER, J.; MICHEL, F. Integrating Tools and Infrastructures for Generic Multi-Agent Systems. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 5., 2001, Montreal, Canada. **Proceedings...** New York: ACM Press, 2001. p.441–448.

HÜBNER, J. F.; SICHMAN, J. S. SACI: Uma Ferramenta para Implementação e Monitoração da Comunicação entre Agentes. In: IBERO-AMERICAN CONFERENCE ON AI, IBERAMIA, 7., 2000, São Carlos. **Proceedings...** São Paulo: USP, 2000. p.47–56. Disponível em: <http://lti.pcs.usp.br/saci/>. Acesso em: abr. 2001.

HÜBNER, J. F.; SICHMAN, J. S. **SACI Programming Guide**. [S.l.]: Universidade de São Paulo - Laborátorio de Técnicas Inteligentes, 2001. Disponível em: <lti.pcs.usp.br/saci/>. Acesso em: nov. 2001.

HÜBNER, J. F.; SICHMAN, J. S.; BOISSIER, O. $\mathcal{M}$OISE$^+$: Towards a structural, functional, and deontic model for MAS organization. In: INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTI-AGENT SYSTEMS, AAMAS, 1., 2002, Bologna, Italy. **Proceedings...** [S.l.: s.n.], 2002.

HUHNS, M.; LARRY, M.; STEPHENS, L. Multiagent Systems and Societies of Agents. WEISS, G. (Ed.). **Multiagent Systems**: a modern approach to distributed artificial intelligence. Cambridge, MA: The MIT Press, 1999. p.79–120

HUHNS, M.; SINGH, M. Agents and Multi-agent Systems: themes approaches, and challenges. In: HUHNS, M.N.;SINGH, M.P. (Ed.). **Readings in Agents**. San Francisco, CA: Morgan Kaufmann, 1998. p.1–23.

JENNINGS, N. R.; SYCARA, K. P.; WOOLDRIDGE, M. J. A Roadmap of Agent Research and Development. **Autonomous Agents and Multi-Agent Systems**, Hingham, MA, v.1, n.1, p.7–38, 1998.

KINNY, D.; GEORGEFF, M. Modelling and Design of Multi-Agent System. In: INTERNATIONAL WORKSHOP ON AGENT THEORIES, ARCHITECTURES, AND LANGUAGES, ATAL, 3., Budapest, Hungary, 1996. **Intelligent Agents III:** proceedings... Berlin: Springer-Verlag, 1996.p. 1–20. (Lecture Notes in Artificial Intelligence, 1193).

KRAFTA, R. Spatial Self-Organization and the prodution of the city. **Revista URBANA**, Caracas, Venezuela, n.24, p.49–62, 1999.

LUCK, M.; D'INVERNO, M. A Formal Framework for Agency and Autonomy. In: INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS, ICMAS, 1., San Francisco, CA, 1995. **Proceedings...** Menlo Park, CA: AAAI Press, 1995. p.254–260.

MACHADO, R.; BORDINI, R. H. Running AgentSpeak(L) Agents on SIM_AGENT. In: INTERNATIONAL WORKSHOP ON AGENT THEORIES, ARCHITECTURES, AND LANGUAGES, ATAL, 8., 2001, Seattle, WA, 2001. **Proceedings...** [S.l.: s.n.], 2001.

MAYFIELD, J.; LABROU, Y.; FININ, T. Desiderata for Agent Communication Languages. In: AAAI SPRING SYMPOSIUM ON INFORMATION GATHERING, 1995. **Proceedings...** [S.l.: s.n.], 1995.

MAYFIELD, J.; LABROU, Y.; FININ, T. Evaluation of KQML as an Agent Communication Language. In: INTERNATIONAL WORKSHOP ON AGENT THEORIES, ARCHITECTURES, AND LANGUAGES, ATAL, 1995, Montreal, Canada. **Intelligent Agents II:** proceedings... Berlin: Springer-Verlag, 1996. p.347–360. (Lecture Notes In Artificial Intelligence, 1034).

MOREIRA, Á. F.; BORDINI, R. H. An Operational Semantics for a BDI Agent-Oriented Programming Language. In: WORKSHOP ON LOGICS FOR AGENT-BASED SYSTEMS, LABS; INTERNATIONAL CONFERENCE ON PRINCIPLES OF KNOWLEDGE REPRESENTATION AND REASONING,KR, 8., 2002, Toulose, France, 2002. **Proceedings...** [S.l.: s.n.], 2002. p.45–59.

NODA, I. et al. Soccer Server: a tool for research on multi-agent systems.**Applied Articial Intelligence**, London, v.12, n.2-3, 1998.

OLIVEIRA, D. de. **Simulação de Aspectos Sociais do Crescimento Urbano com Sistemas Multiagentes Cognitivos**.2002. 64f. Projeto de Diplomação (Bacharelado em Ciência da Computação) - Instituto de Informática,UFRGS, Porto Alegre.

PRIETULA, M.; CARLEY, K.; GASSER, L. (Ed.). **Simulating Organizations**: computational models of institutions and groups. Menlo Park, CA: AAAI Press, 1998.

RAO, A. S. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In: WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD, MAAMAW, 7., 1996, London. **Proceedings...** Berlin: Springer-Verlag, 1996. p.42–55. (Lecture Notes in Artificial Intelligence, 1038).

RAO, A. S.; GEORGEFF, M. P. BDI Agents: from theory to practice. In: INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS, ICMAS, 1995, San Francisco. **Proceedings...** Menlo Park, CA: AAAI Press, 1995. p.312–319.

RUSSEL, S.; NORVIG, P. **Artificial Intelligence:** a modern approach. Englewood Cliffs: Prentice-Hall, 1995.

SEARLE, J. R. **Speech Acts**: an essay in the philosophy of language. Cambridge: Cambridge University Press, 1969.

SHOHAM, Y. Agent-Oriented Programming. **Artificial Intelligence**, Amsterdan, v.60, p.51–92, 1993.

SICHMAN, J.; DEMAZEAU, Y.; BOISSIER, O. When Can Knowledge-Based Systems be Called Agents. In: BRAZILIAN SYMPOSIUM ON ARTIFICIAL INTELLIGENCE, SBIA, 9., 1992, Rio de Janeiro. **Proceedings...** [S.l.:s.n.], 1992.

SINGH, M. P.; RAO, A. S.; GEORGEFF, M. P. Formal Methods in DAI: logic-based representation and reasoning. In: WEISS, G. (Ed.). **Multiagent Systems:** a modern approach to distributed artificial intelligence. Cambridge, MA: MIT Press, 1999. p.331–376.

WEISS, G. (Ed.). **Multiagent Systems:** a modern approach to distributed artificial intelligence. Cambridge, MA: MIT Press, 1999.

WOOLDRIDGE, M. Intelligent Agents. In: WEISS, G. (Ed.). **Multiagent Systems:** a modern approach to distributed artificial intelligence. Cambridge, MA: MIT Press, 1999. p.27–77.

WOOLDRIDGE, M. J.; JENNINGS, N. R.; KINNY, D. The Gaia Methodology for Agent-Oriented Analysis and Design. **Autonomous Agents and Multi-Agent Systems**, Norwell, MA, v.3, n.3, p.285–312, 2000.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent Agents: theory and practice. **The Knowledge Engineering Review**, Cambridge, v.10, n.2, p.115–152, 1995. Disponível em: <http://www.elec.qmw.ac.uk/dai/pubs>. Acesso em: mar. 2001.