Implementação da Arquitetura de Agentes BDI com o Ambiente de Desenvolvimento SNTool

André Olinto Latansio

UNICAMP – Programa de Pós-Graduação FEEC

***Resumo*. Sistemas Multi-Agentes são úteis para implementar processos operacionais com interações complexas entre os seus elementos constituintes na realização de suas tarefas operacionais. O comportamento inteligente de um Sistema Multi-Agentes pode ser modelado usando-se a Arquitetura de Agentes BDI. Diversas plataformas (*frameworks*) de implementação para a arquitetura BDI foram propostas, e estão disponíveis para uso. Este trabalho propõe uma implementação da arquitetura BDI usando uma outra arquitetura desenvolvida para modelar sistemas multi-agentes: os Sistemas de Agentes Semiônicos. O SNToolkit é um ambiente de desenvolvimento útil na modelagem de um sistema de agentes semiônicos. E será usado para modelar a arquitetura BDI.**

***Palavras-chave:* Agentes BDI, *Frameworks*, SNTool**

I. INTRODUÇÃO

SISTEMAS Complexos podem ser decompostos em subsitemas de menor complexidade e organizados hierárquicamente [1]. Cada subsistema possui seus próprios objetivos, e, em conjunto, estes subsitemas cooperam para cumprir os objetivos essenciais do sistema como um todo. Nesse contexto, os sistemas multi-agentes se encaixam perfeitamente para a implementação desse tipo de sistema, de maior complexidade. Dependendo da complexidade do sistema, há a necessidade de incorporar inteligência aos agentes que o comporão.

Esses agentes *cognitivos* podem ser modelados através do *modelo BDI* (*Belief-Desire-Intention*).

A proposta deste trabalho é implementar o modelo BDI por meio de uma rede de agentes semiônicos [9]. Esses agentes possuem portas de entrada e portas de saída. Objetos são transportados para dentro e para fora do agente semiônico, por meio dessas portas. Além disso, o agente também possui estados internos descritivos. O agente opera por meio de funções de avaliação e de transformação, que implementam a sua lógica de operação. Os agentes que compõem a rede trocam objetos entre si. Sendo que estes são transformados (ou destruídos) dentro do agente, ou ainda, geram novos objetos nas saídas do agente. Essas interações dependem das implementações das funções de avaliação e de transformação.

II. ARQUITETURA BDI

A. Modelo BDI

O Modelo BDI é uma teoria filosófica do raciocínio prático, proposta inicialmente por Bratman [1]. O comportamento humano é modelado considerando-se as seguintes atitudes mentais: *crenças, desejos e intenções.*

Crenças

A crença se constitui numa visão do agente em relação ao seu ambiente [2]. O agente usa as crenças para expressar suas expectativas sobre os possíveis estados futuros do seu ambiente. Essas crenças podem se referir ao "mundo" habitado pelo agente, podem se referir a outros agentes e as suas interações com estes, e ainda, podem se constituir de crenças sobre as suas próprias crenças. Uma crença expressa um conhecimento do agente sobre alguma informação presente no seu ambiente operativo. As crenças podem mesmo serem contraditórias.

Desejos

Enquanto as crenças seriam os estados prováveis do ambiente do agente, os desejos representam os estados desejáveis que o sistema de operação do agente poderia apresentar [2]. Os desejos motivam o agente a realizar as tarefas para as quais ele foi projetado a executar. Um desejo causa uma intenção de agir (ação) de modo a cumprir uma determinada meta ou conjunto de metas. Para um desejo se concretizar, na realização de uma tarefa, são necessários: o conhecimento (crença) que o dispara, e, o contexto favorável à sua realização.

Intenções

São consideradas um subconjunto dos desejos [2]. Quando o agente decide cumprir uma meta (desejo), este torna-se uma intenção. Determinam o processo de raciocínio prático, mediante a execução de determinadas ações. Escolhida uma intenção, a cumprir, ocorrerá um direcionamento do raciocínio prático, futuro, no sentido de realizar tal intenção: o agente deverá considerar as ações que são coerentes à realização da intenção atual.

B. Operação do Agente BDI

O projeto de um Agente BDI envolve a determinação, prévia, dos aspectos comportamentais do agente, ou seja: deve haver algum conhecimento sobre como o agente deverá executar as suas tarefas [2]. No seu projeto são especificados as suas crenças e desejos; no entanto, a escolha das intenções é determinada pelo próprio agente, de acordo com as suas opções (desejos) disponíveis.

O processo de raciocínio prático, base do modelo BDI, determina as ações do agente mediante a execução dos dois passos desse processo: *primeiramente,* seleciona-se um conjunto de desejos (deliberação de objetivos) que devem ser realizados de acordo com as crenças atuais do agente; *na seqüência,* determina-se como esses desejos, selecionados anteriormente, podem ser realizados (intenções) por meio dos recursos disponíveis ao agente, no momento presente, incluindo as crenças atuais do agente [1].

O desenvolvimento do raciocínio prático do agente é um problema, do ponto de vista da escolha das intenções pelo próprio agente [2]. Às vezes, pode ser necessário que o agente abandone a execução de determinadas intenções e retome (ou escolha) outras intenções. Esse balanceamento entre as diversas opções disponíveis e os seus critérios para as suas realizações, em constante mudança, pode envolver um custo computacional relativamente alto – para as aplicações em tempo real (por exemplo, jogos de computador). O custo de reavaliar constantemente sua situação impede os agentes deliberativos (agentes BDI) de serem usados em qualquer tipo de aplicação. Uma possível solução desse problema poderia ser fixar a realização de todas as tarefas referentes a uma intenção, antes de ponderar a realização de outra intenção.

C. Interpretador BDI

O interpretador BDI, desenvolvido por Rao e Georgeff – e que é a base de praticamente todos os sistemas BDI históricos e atuais – opera sobre as crenças, objetivos e planos do agente [1]. Os objetivos do agente constituem-se de desejos concretos, ou seja, realizáveis naquele momento, que podem ser atingidos todos juntos – evitando a necessidade de uma complexa fase de deliberação de objetivos. O interpretador seleciona e executa os planos para realizar um dado objetivo.

O processo de raciocínio prático implementado no interpretador BDI pode ser visualizado na Fig. 1.

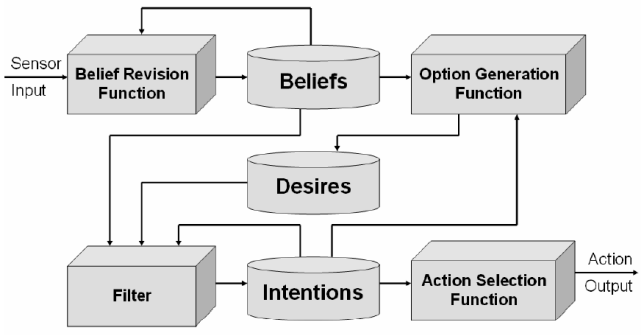

Fig. 1. Diagrama da Arquitetura BDI [1].

O interpretador compreende sete componentes principais que constituem um agente BDI [1]:

- Conjunto de crenças (*Beliefs*) atual, que representa as informações do agente sobre o seu ambiente;
- Função de revisão de crenças (*Belief Revision Function*), determina novas crenças à partir das informações dos sensores do agente (*Sensor Input*) e das crenças atuais;
- Conjunto de opções (*Desires*) corrente, que representa os possíveis planos de ação disponíveis ao agente;
- Função de geração de opções (*Option Generation Function)*, determina as opções disponíveis (desejos) ao agente, com base nas suas crenças e intenções (atuais);
- Conjunto de intenções (*Intentions*) atual, que representa os estados atuais que o agente pretende alcançar;
- Função de Filtro (*Filter*), representa o processo de deliberação do agente, determina as intenções do agente com base nas suas crenças, desejos e intenções atuais;
- Função de seleção de ação (*Action Selection Function*), determina qual ação será executada – enviada aos atuadores (*Action Output*) do agente – com base nas suas intenções atuais.

III. Implementações da Arquitetura BDI

A. Sistemas e Frameworks mais usuais

O primeiro sistema baseado no interpretador BDI – desenvolvido por Rao e Georgeff – a ser implementado com sucesso, foi o *PRS (Procedural Reasoning Systems)* [1]. Seu sucessor foi um sistema chamado *dMARS (distributed Multi-Agent Reasoning System).* Posteriormente, foram desenvolvidos vários *frameworks* de desenvolvimento de agentes, baseados na arquitetura BDI, que utilizam uma determinada linguagem e fornecem uma plataforma de execução desses agentes. Dentre essas plataformas, destacam-se aquelas baseadas em Java (*JACK, Jadex, JAM, Jason*).

B. PRS

É um *framework* para construção de sistemas de raciocínio, em tempo real, que realizam tarefas complexas em ambientes dinâmicos [2]. Sua arquitetura é composta pelos seguintes componentes: *banco de dados*, contém as crenças sobre o ambiente; *conjunto de objetivos*, que esperam ser executados; *conjunto de planos*, descreve as seqüências de ações e testes a serem executados para realizar os objetivos; *intenções*, contendo os planos escolhidos para execução; *interpretador*, manipula os demais componentes, escolhendo os planos mais apropriados.

O PRS é um sistema de planejamento reativo e direcionado a metas [3]. Esse direcionamento a metas permite gerar raciocínio e realizar tarefas complexas. É um sistema ideal para implementar raciocínio de alto nível em robôs.

C. dMARS

É uma ferramenta, baseada em C++, usada em ambientes de desenvolvimento de sistemas orientados a agentes [4]. Voltada para aplicações em domínios dinâmicos de conhecimento incerto e complexo. Aplicada em telecomunicações, tráfego aéreo, gerência de negócios, etc. De configuração rápida e de fácil integração.

Os agentes *dMARS* monitoram o mundo. Percebendo seus estados internos e eventos externos, e, colocando-os numa fila de eventos [2]. Um interpretador gerencia as operações do agente. Executando um ciclo: *atualiza a fila de eventos*, observados do seu ambiente; *gera desejos possíveis*, baseados em planos compatíveis com eventos; *seleciona um evento para execução*, do conjunto de planos compatíveis; *coloca o evento numa pilha de intenção*, nova ou já existente dependendo do evento ser ou não um sub-objetivo; *seleciona uma intenção da pilha*, executa-a se for uma ação ou envia-a para a fila de eventos caso seja um sub-objetivo.

Os agentes *dMARS* não planejam tudo, como no PRS. Os planos devem ser definidos durante a modelagem do agente. O planejamento que o agente exerce consiste em expandir os sub-objetivos de acordo com o contexto da situação.

IV. JACK

A. Linguagem de Programação

JACK Intelligent Agents^{TM} é um *framework* de desenvolvimento de sistemas multi-agentes baseado em Java [1]. Oferece alta performance e uma implementação leve da arquitetura BDI. A sua linguagem (*JACK Agent Language*) estende as funcionalidades de Java, além de ser orientada a agentes. Apresenta unidades funcionais que implementam o agente BDI e seu comportamento:

Agent. Define o comportamento de um agente de software inteligente. Suas capacidades, tipos de mensagens, eventos e os planos para atingir seus objetivos.

Capability. Representam aspectos funcionais de um agente que podem ser plugadas se necessário. Podem ser feitas através de planos, eventos, conjunto de crenças e outras capacidades.

BeliefSet. Representa as crenças do agente. Usa um modelo relacional genérico. As consultas utilizam-se de membros lógicos que seguem regras de programação lógica.

View. Permite consultas de propósito geral sobre o modelo de dados. Implementado usando *BeliefSets* ou estruturas de dados Java.

Event. Ocorrência que espera uma ação, resposta, do agente. Podem ser eventos que o agente envia para si mesmo (*Internal stimuli*). Percepções do seu ambiente, ou mensagens de outros agentes (*External stimuli*). Ou, motivações que o agente deve ter, objetivos a atingir (*Motivations*).

Plan. Planos do agente, análogos a funções. São instruções que o agente segue para atingir seus objetivos e tratar os eventos. Podem implicar no envio de novos eventos ao agente, ou a outros agentes. Assim, os agentes podem colaborar entre si para realizar as suas tarefas.

B. Plataforma de Desenvolvimento

O *JACK Development Enviroment (JDE)* permite a criação de agents, eventos, planos, *beliefsets*, etc. por meio de controles *drag and drop* [5]. O "esqueleto" do código JACK é gerado automaticamente. Também provê um Editor de Planos Gráfico que permite compor o plano por meio de notação gráfica, ao invés de código textual.

Inclui uma ferramenta de design (*Design Tool*) que permite visualizar diagramas no estilo Prometheus. Essa ferramenta pode ser usada para criar as estruturas do sistema, colocando as entidades na tela de desenho e ligando-as juntas. O JDE também provê algumas ferramentas de depuração.

V. JADEX

A. Componentes da Plataforma

Jadex é um mecanismo de raciocínio orientado a agentes, onde os agentes são escritos em Java e XML [1]. Pode ser usado com diferentes plataformas de agentes. A arquitetura abstrata do agente Jadex é ilustrada na Fig. 2. O interpretador consite numa agenda de componentes que armazenam meta-ações a serem executadas. O agente seleciona uma meta-ação da agenda e a executa quando suas pré-condições são satisfeitas. Caso contrário, a ação é descartada. Seus principais componentes são:

Capability. Permitem que crenças, planos e objetivos sejam colocados num módulo de agente. Podem conter sub-capacidades.

Beliefs. Representam o conhecimento do agente sobre seu ambiente. Podem ser qualquer objeto Java, e são armazenadas em uma base de crenças.

Goals. Objetivos motivacionais do agente, orientam suas ações. Jadex trata-os como desejos concretos e momentâneos – e não como eventos especiais, como em outros sistemas BDI.

Plans. Representam a forma como o agente atuará em seu ambiente. Planos são selecionados em resposta à ocorrência de eventos ou objetivos. A seleção de planos é feita automaticamente pelo sistema.

Events. Eventos, podem ser de dois tipos. Eventos internos detonam uma ocorrência dentro do agente. Eventos mensagem representam uma comunicação entre agentes.

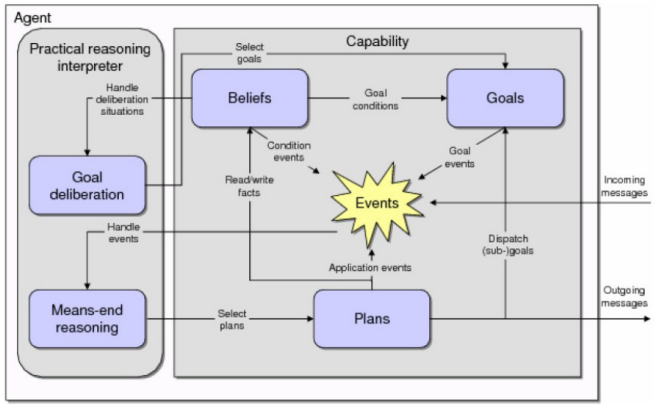

Fig. 2. Arquitetura Abstrata Jadex [1].

B. *Estrutura do Agente Jadex*

O agente Jadex está estruturado em duas partes [6]. Um arquivo XML (*ADF, Agent Description File*) contendo a descrição do agente (Fig. 3). E, os planos procedurais escritos em Java (*Plan*). Os planos são classes Java. O arquivo ADF descreve a estrutura interna do agente (*beliefs, goals, plans, events, capabilities, etc*).

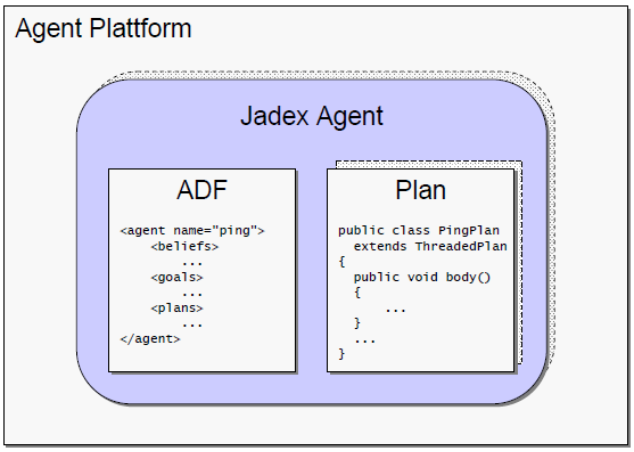

Fig. 3. Agente Jadex [6].

VI. JAM

A. *Componentes Primários*

JAM é uma arquitetura de agentes inteligentes implementada em Java [1]. Cada agente JAM é composto pelos componentes:

World Model. Banco de dados que representa as crenças do agente sobre o estado corrente do mundo. Armazena o estado das variáveis e sensores, as conclusões das deduções e as inferências.

Plan Library. Coleção dos planos do agente para atingir seus objetivos. Define a especificação procedural para alcançar um objetivo.

Interpreter. Pode ser considerado o cérebro do agente. Define o que o agente fará e quando. Seleciona e executa planos, baseado nos objetivos e contexto da situação.

Intention Structure. Modelo interno dos objetivos e atividades correntes do agente. Armazena as informações sobre o progresso do agente para atingir seus objetivos.

Observer. Pode ser considerado um plano leve executado entre os passos do plano, para realizar funcionalidades não descritas no plano.

B. *Arquitetura*

JAM é implementado como uma coleção de programas de aprendizado e classificação distribuídos, ligados entre si por uma rede de *Datasites* [7]. Um *Datasite* consiste de: uma ou mais bases de dados locais; agentes de aprendizado; agentes de meta-aprendizado; um arquivo de configuração local; e, recursos de animação e gráficos (*GUI*). Os *Datasites* colaboram entre si para trocar agentes classificadores que são computados em agentes de aprendizado.

VII. JASON

A. *Componentes da Plataforma*

Jason é uma plataforma de desenvolvimento de sistemas multi-agentes implementada em Java [1]. É baseada num interpretador da linguagem *AgentSpeak(L)*. Um ambiente de desenvolvimento integrado (*IDE*) encontra-se disponível, permitindo a execução e depuração dos programas. A especificação do agente em AgentSpeak(L) consiste de:

Beliefs Base. Conjunto de crenças base, que são fórmulas atômicas de primeira ordem.

Goal. Estado do sistema, que o agente deseja atingir. Pode ser de dois tipos: *Achievement Goal*, o agente deseja atingir o estado de mundo associado à fórmula atômica; *Test Goal*, o agente deseja testar se a fórmula atômica associada pertence às suas crenças.

Plan Library. Biblioteca de planos do agente, onde cada plano consiste num conjunto de ações para atingir um objetivo. O plano é constituído de: *Head*, composto por um evento ativador que define quais eventos iniciam a execução de um plano associado a um contexto; *Body*, inclui ações básicas que representam operações atômicas executadas a fim de alterar o ambiente.

B. *Linguagem AgentSpeak(L)*

A linguagem AgentSpeak(L) implementa agentes BDI na forma de um sistema de planejamento reativo [8]. Esses sistemas, em permanente execução, reagem a eventos que ocorrem no seu ambiente, executando planos parcialmente instanciados.

$$
\begin{array}{lcll}
ag & ::= & bs \quad ps & \\
bs & ::= & b_1 \ldots b_n & (n \geq 0) \\
at & ::= & \mathtt{P}(t_1, \ldots, t_n) & (n \geq 0) \\
ps & ::= & p_1 \ldots p_n & (n \geq 1) \\
p & ::= & te : ct \leftarrow h & \\
te & ::= & +at \mid -at \mid +g \mid -g & \\
ct & ::= & at \mid \neg at \mid ct \wedge ct \mid \mathsf{T} & \\
h & ::= & a \mid g \mid u \mid h; h & \\
a & ::= & \mathtt{A}(t_1, \ldots, t_n) & (n \geq 0) \\
g & ::= & !at \mid ?at & \\
u & ::= & +at \mid -at &
\end{array}
$$

Fig. 4. Sintaxe AgentSpeak(L) [8].

O agente (*ag*) deve ser especificado por um conjunto de crenças (*bs*) e planos (*ps*), de acordo com a gramática exposta na Fig. 4. Uma crença é uma fórmula atômica (*at*) sem variáveis, onde *b* é a meta-variável para crenças. Fórmulas atômicas são predicados compostos por um símbolo predicativo (*P*) e termos padrão da lógica de primeira ordem (t_n). Um plano compreende um evento ativador (*te*), um contexto (*ct*) e uma seqüência (*h*) de ações, objetivos ou atualizações de crenças. Um evento ativador adiciona (+*at*) ou remove (-*at*) crenças, ou, objetivos (+*g* / -*g*). Objetivos podem ser de realização (*!at*) ou de teste (*?at*). As ações são predicados usuais, onde, um símbolo de ação (*A*) é usado no lugar do símbolo predicativo.

VIII. IMPLEMENTAÇÃO COM O SNTOOL

A. *Modelo usando Agentes Semiônicos*

O SNTool (*Semionic Network Toolkit*) [10] é o ambiente de desenvolvimento de redes de agentes semiônicos [9] que será usado para modelar a arquitetura BDI, neste trabalho.

A Fig. 5 mostra a rede de agentes semiônicos utilizada para implementar as funcionalidades da arquitetura BDI. Trata-se de uma "página" (*page*) com uma interface (*interface*) de entrada – ***input***, do tipo ***Crenca***, e uma interface de saída – ***output***, do tipo ***Acao***. Onde ***Crenca*** e ***Acao*** são classes (objetos) que contém as informações do agente BDI, respectivamente, sobre as crenças do seu ambiente, e, as ações que o mesmo deverá executar. O projeto do agente deliberativo (BDI) consistirá de uma segunda página (por exemplo, *Main*) onde outros sêmions (agente semiônico) deverão implementar as funções necessárias para gerar o objeto ***Crenca*** – que será transportado para a *rede BDI* – a partir das informações captadas pelos sensores do agente; e, tratar o objeto ***Acao*** – devolvido pela *rede BDI* – decodificando suas informações em ações executadas nos atuadores do agente. Essa página, do agente, deverá conter um objeto do tipo ***superplace*** – responsável pela interligação da página do agente com a página que implementa o modelo BDI.

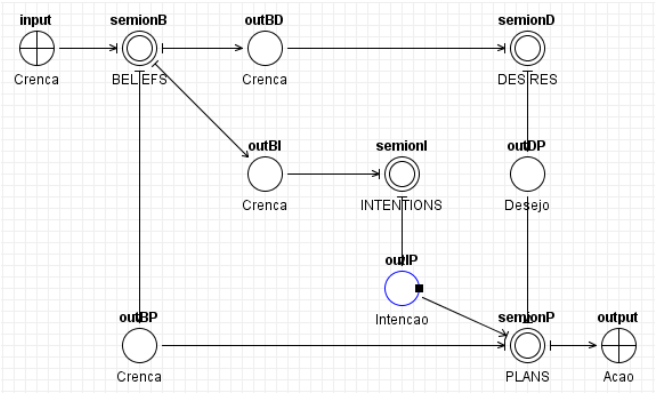

Fig. 5. Modelo BDI usando agentes semiônicos.

A *rede BDI* é constituída pelos objetos *Crenca, Desejo, Intencao e Acao*; e pelos sêmions *BELIEFS, DESIRES, INTENTIONS e PLANS*. O objeto ***Crenca*** é composto pelos estados (*states*) que representam as possíveis crenças do agente – um estado para cada tipo de crença. Os valores dos estados são objetos *String* que representam os valores possíveis para aquela crença. O objeto ***Desejo*** possui os estados que representam os desejos que o agente pode assumir durante sua operação. Seus valores (*strings*) indicam a opção, sim ou não, de realizar um desejo. O objeto ***Intencao*** tem estados representativos das possíveis intenções do agente – semelhantes aos desejos. O objeto ***Acao*** traduz (seus estados) as ações que o agente, de fato, pretende realizar.

BELIEFS

É o sêmion responsável por capturar o objeto ***Crenca*** que entra na rede e repassá-lo para os outros sêmions da rede (*DESIRES, INTENTIONS e PLANS*). Para isso, utiliza três estados internos do tipo ***Crenca***. Uma função, *gerarCrencas()*, armazena o objeto de entrada num dos estados e, após, copia o seu conteúdo para os demais estados. Os estados internos são, então, enviados um para cada uma das três saídas do sêmion.

DESIRES

Define os desejos atuais do agente. Recebe um objeto ***Crenca***, na sua entrada, e armazena-o num estado interno. Mediante os estados desse objeto, define os estados de um objeto ***Desejo***, salvo num estado interno seu, que será posteriormente enviado à saída do sêmion. Uma função, *gerarDesejos()*, implementa as regras de decisão que podem gerar desejos contraditórios.

INTENTIONS

Este sêmion define as intenções atuais do agente. Armazena o objeto ***Crenca***, que recebe, num estado interno e utiliza-o para alterar os estados de um objeto ***Intencao***, internamente armazenado. Este será enviado para sua saída. A função *gerarIntencoes()* define as intenções – às vezes, conflitantes – mediante algumas regras de decisão.

PLANS

É o sêmion responsável pela deliberação do agente. A sua função *gerarAcao()* possui as regras para resolver os conflitos causados pelos desejos e intenções atuais do agente. O objeto ***Acao***, criado, será enviado para sua saída (e, conseqüentemente, para fora da rede). Suas entradas recebem objetos (***Crenca***, ***Desejo*** e ***Intencao***) que são armazenados em estados internos. São os estados desses objetos que, de fato, determinarão as ações do agente.

B. *Estudo de Caso: Robô Coletor*

Considera-se um ambiente composto por paredes (objetos grandes), "bolinhas" (objetos pequenos) e luzes. Um robô dotado de motores (esquerdo e direito) e de um braço, com garra, move-se por esse *mundo* executando a função de coletar bolinhas e depositá-las na fonte de luz.

Inicialmente, modelam-se as regras (*sensores/atuadores*) que determinam as operações do robô. Dessas regras, retiram-se os *desejos* (Fig. 6) e as *intenções* (Fig. 7). Neste caso, as intenções do robô – ou seja, os desejos mais prioritários – são: navegar pelo seu ambiente, coletar as bolinhas e, depois de capturá-las, depositá-las na fonte de luz.

LATERAL_DIR?(SIM) :: GIRAR (DIR)
LATERAL_DIR?(NAO) :: SEGUIR()
LATERAL_ESQ?(SIM):: GIRAR (ESQ)
LATERAL ESQ ?(NAO) :: SEGUIR()
OBJETO_DIM?(PEQ) :: SEGUIR()
OBJETO_DIM?(GRAND) :: AFASTAR()
OBJETO_DIST?(PERTO) :: CAPTURAR ()
OBJETO_DIST?(LONGE) :: SEGUIR()

LUZ_DIR?(SIM) :: GIRAR (DIR)
LUZ_DIR?(NAO) :: SEGUIR()
LUZ_ESQ?(SIM) :: GIRAR (ESQ)
LUZ_ESQ?(NAO) :: SEGUIR()
LUZ_DIST?(PERTO) :: AFASTAR()
LUZ_DIST?(PERTO) :: SOLTAR()
LUZ_DIST?(LONGE) :: SEGUIR()

Fig. 6. Regras que definem os desejos do robô (agente).

PARADO?(SIM) :: AFASTAR()	*PRESENTE?(OBJETO)* :: SOLTAR()
PARADO?(NAO) :: SEGUIR()	*PRESENTE?(NENHUM)* :: CAPTURAR()

Fig. 7. Regras que definem as intenções do robô.

Para realizar essas intenções, é fundamental que o robô nunca fique "parado" (***PARADO?()***). E se algum objeto estiver preso em sua garra (***PRESENTE?()***) o robô deve se empenhar em encontrar uma fonte de luz; caso contrário, o robô se empenhará em encontrar as bolinhas (e se manter afastado da fonte de luz).

Todas as demais regras operativas do sistema constituem-se nos desejos do robô – que o auxiliarão a cumprir as suas intenções, por exemplo, girar para inspecionar as dimensões de um objeto próximo lateralmente. Um outro conjunto de regras (Fig. 8) determina as ações que o robô, de fato, irá executar. Percebe-se que é aqui que são resolvidos os conflitos do sistema. Os desejos e as intenções atuais do agente são escolhidos livremente, sem maiores restrições. Nota-se, segundo as regras do sistema, que uma fonte de luz próxima causa tanto o desejo de se afastar, quanto o de se aproximar da mesma. Com as intenções, a ação de capturar ou soltar uma bolinha causa uma parada momentânea do robô (conflito de intenções).

CRENÇAS:	INTENÇÕES:	DESEJOS:	AÇÕES:
	SEGUIR()	SEGUIR()	SEGUIR()
OBJETO_DIM?(PEQ)	CAPTURAR()	CAPTURAR()	CAPTURAR()
LUZ_DIST?(PERTO)	SOLTAR()	GIRAR (*X*)	GIRAR (*X*)
LUZ_DIST?(LONGE)	CAPTURAR()	GIRAR (*X*)	GIRAR (*X*)
!LUZ_DIST?(PERTO)	SOLTAR()	AFASTAR()	AFASTAR()
	CAPTURAR()	AFASTAR()	AFASTAR()
	AFASTAR()		AFASTAR()
	SOLTAR()	SOLTAR() & !GIRAR (*X*)	SOLTAR()

Fig. 8. Regras que determinam as ações do robô. Onde, *X* significa uma direção (*ESQ / DIR*) e o símbolo "*!*" é uma negação, e "*&*" é o condicional *and*.

Convém ressaltar que a execução de uma ação, ou conjunto de ações, causa modificações nos sensores do robô e, conseqüentemente, alteram as suas crenças – que modificarão as suas ações.

Cabe ao agente – implementado numa segunda página (*page*) – tratar o conjunto de ações (*Acao*), transformando-as em ações coerentes executadas nos seus atuadores. E, resolver conflitos de ações, caso necessário.

IX. Conclusão

A plataforma de desenvolvimento de redes de agentes semiônicos – o SNTool – é um ambiente de trabalho simples e útil na elaboração de sistemas multi-agentes. Quando associados à arquitetura BDI, os agentes semiônicos demonstram grande poder computacional no que tange a implementação de *raciocínio* a um agente.

Dividindo-se as regras operacionais de um agente em *desejos* e *intenções*, baseados em *crenças*, torna mais prático a modelagem do raciocínio de um agente. As regras são mais simples de se ler, e, o comportamento que o agente demonstrará torna-se mais previsível. O número de regras também tende a ser menor.

O agente BDI, composto por uma rede semiônica, é uma opção a se pensar quando da elaboração de um agente deliberativo que opera num ambiente em constante mudança.

Referencias Bibliográficas

[1] NUNES, I. O.. *Implementação do Modelo e da Arquitetura BDI.* Monografia em Ciência da Computação, Departamento de Informática – PUCRJ, 2007.
Disponível em ftp://ftp.inf.puc-rio.br/pub/docs/techreports/07_33_nunes.pdf

[2] CARVALHO, F. G.. *Comportamento em Grupo de Personagens do Tipo Black&White.* Dissertação de Mestrado em Informática, Departamento de Informática do Centro Técnico e Científico – PUCRJ, 2004.
Disponível em http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0210488_04_cap_04.pdf

[3] DASTANI, M.. *Multi-Agent Programming. Agent Speak(L) and its Interpreter Jason.*
Disponível em http://www.cs.uu.nl/docs/vakken/map/jason.pdf

[4] ZAMBERLAM, A. O., GIRAFFA, L. M. M., MÓRA, M. C.. *X-BDI: uma ferramenta para programação de agentes BDI.* Relatório Técnico 009/2001, Faculdade de Informática – PUCRS, 2001.
Disponível em http://www.pucrs.br/inf/pos/mestdout/rel_tec/tr009.pdf

[5] WINIKOFF, M.. *JACK™ Intelligent Agents: an Industrial Strength Platform.* RMIT University, Australia.
Disponível em http://www.cs.rmit.edu.au/agents/www/papers/MAP05-W.pdf

[6] DASTANI, M.. *Multi-Agent Programming. Jadex: A BDI Reasoning Engine.*
Disponível em http://www.cs.uu.nl/docs/vakken/map/slides/jadex.pdf

[7] STOLFO, S., PRODROMIDIS, A. L., TSELEPIS, S., LEE, W., FAN, D. W., CHAN, P. K.. *JAM: Java Agents for Meta-Learning over Distributed Databases.* Department of Computer Science – Columbia University, New York / Computer Science – Florida Institute of Technology, Melbourne.
Disponível em http://www.aaai.org/Papers/KDD/1997/KDD97-012.pdf

[8] HÜBNER, J. F., BORDINI, R. H., VIEIRA, R.. *Introdução ao Desenvolvimento de Sistemas Multiagentes com Jason.* Grupo de Inteligência Artificial, Departamento de Sistemas e Computação – FURB / Department of Computer Science – University of Durham, UK / Centro de Ciências Exatas e Técnológicas, Centro de Ciências da Comunicação – UNISINOS.
Disponível em http://www.das.ufsc.br/~jomi/pubs/2004/Hubner-eriPR2004.pdf

[9] GUERRERO, J. A. S.. *Rede de Agentes: Uma Ferramenta para o Projeto de Sistemas Inteligêntes.* Tese de Mestrado, DAC-FEEC – UNICAMP, 2000.
Disponível em http://www.dca.fee.unicamp.br/~gudwin/ftp/publications/JasgThesis.pdf

[10] *Semionic Network Toolkit – SNTool.* DCA-FEEC – UNICAMP.
Disponível em http://sntool.sourceforge.net/