PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 33/2007

Implementação do Modelo e da Arquitetura BDI

Ingrid Oliveira de Nunes

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 33/2007 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Dezembro, 2007

Implementação do Modelo e da Arquitetura BDI *

Ingrid Oliveira de Nunes

ioliveira@inf.puc-rio.br

Abstract. Agents with reasoning are been used to model intelligent behavior in multi-agent systems. The architecture proposed by Rao and Geogreff in (Rao & Georgeff 1995), which is based in the BDI model (belief-desire-intention), has been used with success in situations where modeling the human reasoning is necessary. Several implementations have emerged for this architecture. Therefore, this work presents and compare four of these implementations: JACK, Jadex, JAM and Jason.

Keywords: Cognitive Agents, BDI, Multi-agent Systems.

Resumo. Agentes com raciocínio estão sendo utilizados para modelar comportamento inteligente em sistemas multi-agentes. A arquitetura proposta por Rao e Geogreff em (Rao & Georgeff 1995), a qual é baseada no modelo BDI (*belief-desire-intention*), tem sido utilizada com sucesso em situações onde a modelagem do raciocínio humano é necessária. Uma série de implementações surgiram para essa arquitetura. Assim, este trabalho apresenta e compara quatro dessas implementações: JACK, Jadex, JAM e Jason.

Palavras-chave: Agentes Cognitivos, BDI, Sistemas Multi-agentes.

---

* Trabalho patrocinado pelo Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil

Responsável por publicações:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Sumário

1 Introdução

A Engenharia de Software Orientada a Agentes é um paradigma de programação visto como um novo objeto de pesquisa na Engenharia de Software. Muitos trabalhos relacionados com computação baseada em agentes foram publicados (Jennings 1999, Wooldridge & Ciancarini 2000). Sistemas complexos normalmente podem ser decompostos em subsistemas organizados de uma forma hierárquica. Esses subsistemas trabalham em conjunto com o intuito de atingir os objetivos do sistema como um todo. Além disso, eles também possuem seus próprios objetivos. Portanto, a abstração de agentes torna-se uma metáfora natural para sua implementação, na qual um sistema complexo é decomposto em termos de múltiplos componentes autônomos de um modo flexível, a fim de atingir seus objetivos.

Freqüentemente, existe a necessidade da incorporação de inteligência nos agentes. Além de possuírem suas características básicas, tais como autonomia, pró-atividade e habilidade social, eles também podem ter a capacidade de raciocínio e aprendizagem. Estudos da Ciência Cognitiva e técnicas da Inteligência Artificial são utilizados para adicionar essa característica aos agentes. Agentes de software com a capacidade de raciocínio passam a ser denominados agentes cognitivos.

Atualmente, a melhor forma de modelagem conhecida de agentes cognitivos é através do modelo *belief-desire-intention* (BDI) (Georgeff, Pell, Pollack, Tambe & Wooldridge 1999). Os conceitos deste modelo foram inicialmente propostos por Bratman (Bratman 1987). O modelo consiste de crenças, desejos e intenções como atitudes mentais que geram a ação humana. Rao and Georgeff apresentaram em (Rao & Georgeff 1995) a arquitetura BDI. Eles adaptaram o modelo proposto por Bratman, transformando-o em uma teoria formal e um modelo de execução para agentes de software baseados na noção de crenças, objetivos e planos. Assim, começaram a surgir implementações para a arquitetura BDI. A primeira implementação que obteve sucesso foi o Procedural Reasoning Systems (PRS) (Georgeff & Lansky 1986).

Este trabalho tem por objetivo fazer um estudo e comparação entre as principais implementações da arquitetura BDI. Na seção 2, é feita uma visão geral do modelo e da arquitetura BDI. A seção 3 apresenta implementações dessa arquitetura, relacionando quatro delas: JACK, Jadex, JAM e Jason. Finalmente, na seção 4, faz-se uma comparação entre as implementações apresentadas.

2 Modelo e Arquitetura BDI

Existem muitas abordagens que propõem diferentes tipos de atitudes mentais e seus relacionamentos. Entre elas, a que mais se destaca é o modelo BDI originalmente proposto por Bratman (Bratman 1987) como uma teoria filosófica do raciocínio prático, explicando o comportamento humano com as seguintes atitudes: crenças, desejos e intenções. A suposição básica do modelo BDI é que ações são derivadas a partir de um processo chamado raciocínio prático, o qual é constituído de dois passos. No primeiro passo, deliberação (de objetivos), faz-se a seleção de um conjunto de desejos que devem ser alcançados, de acordo com a situação atual das crenças do agente. O segundo passo é responsável pela determinação de como esses desejos concretos produzidos como resultado do passo anterior podem ser atingidos através do uso dos meios disponíveis

ao agente (Wooldridge 2000). As três atitudes mentais que compõem o modelo BDI são melhor detalhadas a seguir:

- ***Crenças (****Beliefs****)*** – Representam as características do ambiente, as quais são atualizadas apropriadamente após a percepção de cada ação. Podem ser vistas como o componente informativo do sistema.
- ***Desejos (****Desires****)*** – Contêm informação sobre os objetivos a serem atingidos, bem como as prioridades e os custos associados com os vários objetivos. Podem ser pensados como a representação do estado motivacional do sistema.
- ***Intenções (****Intentions****)*** – Representam o atual plano de ação escolhido. Capturam o componente deliberativo do sistema.

Rao and Georgeff (Rao & Georgeff 1995) adotaram o modelo BDI para agentes de software apresentando uma teoria formal e um interpretador BDI abstrato que é a base para praticamente todos os sistemas BDI históricos e atuais. O interpretador opera sobre as crenças, objetivos e planos do agente, os quais representam os conceitos das noções mentalísticas ligeiramente modificados. A diferença mais significativa é que os objetivos são um conjunto consistente de desejos concretos que podem ser atingidos todos juntos, assim evitando a necessidade de uma complexa fase de deliberação de objetivos. A principal tarefa do interpretador é a realização do processo meios-fim através da seleção e execução de planos para um dado objetivo ou evento. O primeiro sistema implementado com sucesso baseado neste interpretador foi o Procedural Reasoning Systems (PRS) (Georgeff & Lansky 1986), que teve como sucessor um sistema chamado dMARS (D'Inverno, Kinny, Luck & Wooldridge 1997, D'Inverno, Luck, Georgeff, Kinny & Wooldridge 2004) .

O processo de raciocínio prático em um agente BDI é apresentado na Figura 1 (fonte (Wooldridge 1999)). Como a figura mostra, existem sete componentes principais em um agente BDI:

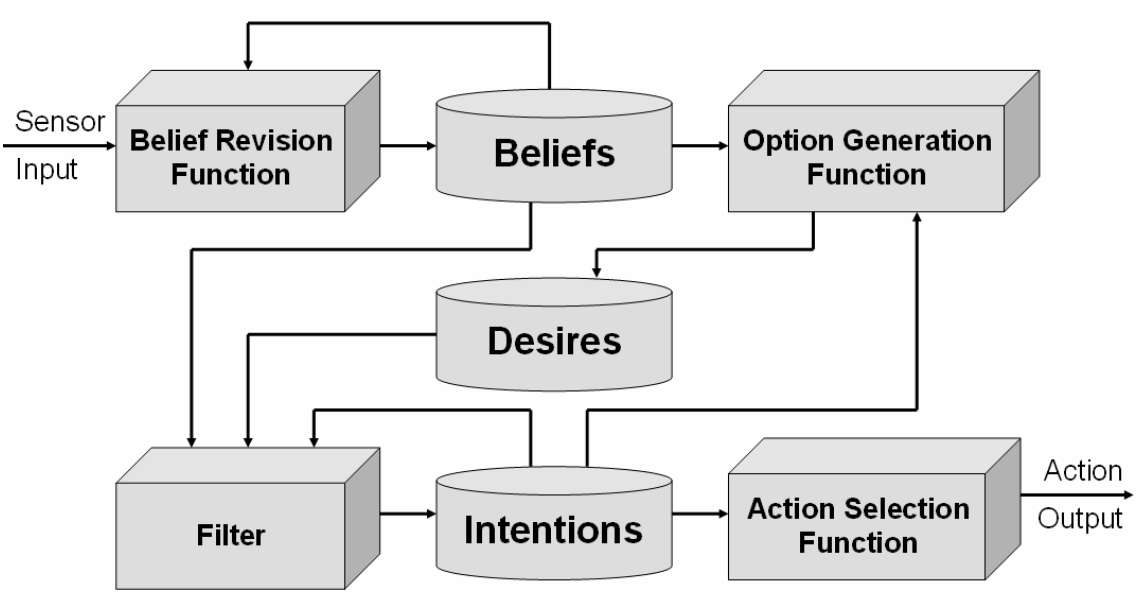

Figura 1: Diagrama de uma arquitetura *belief-desire-intention* genérica.

- um conjunto de crenças (*beliefs*) atual, representando a informação que o agente tem sobre seu ambiente;

- uma função de revisão de crenças (*belief revision function*), a qual determina um novo conjunto de crenças a partir da percepção da entrada e das crenças do agente;
- uma função de geração de opções (*option generation function*), a qual determina as opções disponíveis ao agentes (seus desejos), com base nas suas crenças sobre seu ambiente e nas suas intenções;
- um conjunto de opções (*desires*) corrente que representa os possíveis planos de ações disponíveis ao agente;
- uma função de filtro (*filter*), a qual representa o processo de deliberação do agente, que determina as intenções do agente com base nas suas crenças, desejos e intenções atuais;
- um conjunto de intenções (*intentions*) atual, que representa o foco atual do agente, isto é, aqueles estados que o agente está determinado a alcançar;
- uma função de seleção de ação (*action selection function*), a qual determina uma ação a ser executada com base nas suas intenções atuais.

3 Implementações da Arquitetura BDI

Existe uma série de implementações para a arquitetura BDI. Essas implementações são frameworks que permitem o desenvolvimento dos agentes em uma determinada linguagem e fornecem uma plataforma para a execução deles. Nesta seção, são abordadas quatro implementações: JACK, Jadex, JAM e Jason.

3.1 JACK

JACK Intelligent Agents™ (Howden, R¨onnquista, Hodgson & Lucas 2001) é um framework em Java para o desenvolvimento de sistemas multi-agentes. Ele foi construído pela *Agent Oriented Software Pty. Ltd.* (AOS), sediada em Melbourne, Austrália. A empresa tem por objetivo fornecer uma plataforma para aplicações comerciais, industriais e de pesquisa. Dessa forma, o framework oferece alta performance, uma implementação leve da arquitetura BDI e uma maneira fácil de ser estendida para dar suporte a diferentes modelos de agentes e requisitos específicos das aplicações.

A linguagem utilizada pelo JACK (JACK Agent Language) é construída a partir da linguagem Java. Ela, além de estender as funcionalidades de Java, suporta um novo paradigma de programação, isto é, ela é uma linguagem de programação orientada a agentes. A seguir, seguem extensões feitas à linguagem Java:

- Definição de novas classes, interfaces e métodos;
- Extensões à sintaxe de Java para suportar novas classes, definições e comandos orientados a agentes. JACK possui um compilador que converte as adições sintáticas em classes e comandos escritos em Java puro;
- Extensões semânticas (diferenças na execução) para dar suporte ao modelo de execução requerido por um sistema orientado a agentes. O conjunto de classes responsável por dar suporte à execução de programas feitos na linguagem JACK é chamado *kernel*. Essas classes fazem o gerenciamento automático da concorrência entre as tarefas sendo executadas em paralelo. Além disso, fornecem um comportamento *default* do agente em reação aos eventos, falhas de ações e tare-

fas. Também oferece uma infraestrutura de comunicação nativa leve, de alta performance para aplicações multi-agentes.

A linguagem JACK permite que os programadores desenvolvam componentes que são necessários aos agentes BDI e o seu comportamento. As unidades funcionais presentes na linguagem que permitem isso são (JACK intelligent agents: JACK manual 2005):

- ***Agent*** – Usada para definir o comportamento de um agente de software inteligente. Isso inclui as capacidades que um agente possui, que tipo de mensagens e eventos ele responde e que planos ele irá utilizar para atingir seus objetivos;
- ***Capability*** – Capacidades representam aspectos funcionais de um agente que podem ser plugadas quando necessárias. Uma capacidade pode ser feita através de planos, eventos, conjuntos de crenças e outras capacidades;
- ***BeliefSet*** – Representa as crenças do agente, fazendo uso de um modelo relacional genérico. O conjunto de crenças foi especificamente projetado de forma que possam se fazer consultas sobre ele utilizando-se membros lógicos. Membros lógicos são como membros de dados comuns, exceto pelo fato que eles seguem regras de programação lógica, como ocorre, por exemplo, na linguagem Prolog;
- ***View*** – Permite que consultas de propósito geral possam ser feitas sobre o modelo de dados. Este deve ser implementado usando vários *BeliefSets* ou estruturas de dados da linguagem Java;
- ***Event*** – Descreve uma ocorrência, para a qual o agente deve tomar uma ação como resposta. Eventos são classificados em três tipos: (i) *Internal stimuli* são eventos que o agente envia para ele mesmo, normalmente como resultado da execução de métodos de raciocínio em planos que o agente possui; (ii) *External stimuli* podem ser mensagens vindas de outros agentes ou percepções que o agente recebe do seu ambiente; e (iii) *Motivations* são motivações que o agente deve ter, tais como objetivos que o agente quer atingir;
- ***Plan*** – Os planos de um agente são análogos a funções. Eles são as instruções que o a gente segue para tentar atingir seus objetivos e tratar os eventos designados a ele.

Assim, seguindo o modelo BDI, os agentes JACK são componentes de software autônomos que têm objetivos explícitos para atingir ou eventos para tratar (desejos). O agente pode exibir um comportamento racional sob estímulos pró-ativos (direcionado a objetivos) e reativos (orientado a eventos). Cada agente tem: (1) um conjunto de crenças sobre o mundo; (2) um conjunto de eventos que ele irá responder; (3) um conjunto de objetivos que ele pode desejar atingir, os quais podem ser tanto em resposta a uma requisição de um agente externo, como uma conseqüência à ocorrência de um evento, ou quando uma ou mais de suas crenças mudam; e (4) um conjunto de planos que descrevem como ele pode lidar com os objetivos e eventos que possam surgir.

Quando um agente JACK é criado no sistema, ele normalmente fica inativo até que ele receba um objetivo que deve ser atingido ou um evento ao qual ele deve responder. Uma vez que ele receba tal objetivo ou tal evento, o agente determina quais ações são necessárias para alcançar o objetivo ou responder ao evento.

Primeiramente, o agente irá verificar em suas crenças se o objetivo ou o evento já foi tratado. Caso afirmativo, ele não irá realizar ações relacionadas a ele, caso contrário, ele irá consultar seu conjunto de planos para determinar um plano apropriado para tratar o objetivo ou o evento.

Se o plano for bem sucedido, o objetivo será atingido ou o evento será tratado. Caso ele não tenha sido bem sucedido, o agente irá continuar a tentar planos alternativos até que o objetivo ou o evento tenha sido tratado ou ele tenha tentado todos os seus planos relevantes (neste caso, o agente falha no tratamento do objetivo ou do evento).

Os planos consistem de passos que o agente deve tomar para o tratamento do objetivo ou do evento. Esses passos podem implicar no envio de novos eventos ao próprio agente, ou enviar mensagens para outros agentes para solicitar serviços. Dessa forma, os agentes são capazes de colaborar um com outro.

AOS e seus parceiros desenvolveram uma série de aplicações de software autônomas, nas áreas como UVAs (*Unmanned Aerial Vehicles*), gerenciamento de tráfego aéreo e escalonamento em tempo real.

3.2 Jadex

Jadex (Pokahr, Braubach & Lamersdorf 2003) é um mecanismo de raciocínio orientado a agentes, no qual agentes racionais são escritos em XML e na linguagem de programação Java. Um dos aspectos principais do Jadex é que ele não apresenta uma nova linguagem de programação. Ao invés disso, os agentes Jadex podem ser programados nos ambientes de desenvolvimento integrado orientados a objetos.

Outro conceito importante é a independência do Jadex. Como ele é fracamente acoplado com o middleware sobre o qual ele é executado, Jadex pode ser usado com diferentes plataformas de agentes. Atualmente, dois adaptadores estão disponíveis. O primeiro é a plataforma JADE (Bellifemine, Claire & Greenwood 2007), a qual é bem conhecida e de código livre. O segundo é o adaptador Jadex Standalone, o qual é um ambiente pequeno mas rápido.

Jadex segue o modelo BDI, utilizando crenças, objetivos e planos como objetos de primeira classe, que podem ser criados e manipulados dentro do agente. No Jadex, agentes têm crenças, que são armazenadas em uma base de crenças. Objetivos representam motivações concretas, como por exemplo, estados a serem atingidos, e influenciam no comportamento do agente. Para atingir seus objetivos, o agente executa planos, os quais são roteiros procedurais codificados em Java. A arquitetura abstrata de um agente Jadex é apresentada na Figura 2 (fonte (Pokahr & Braubach 2007)).

Os principais componentes do Jadex são apresentados com mais detalhes a seguir:

- ***Capability*** – Capacidades permitem que crenças, planos e objetivos sejam colocados em um módulo de agente. Este módulo pode ser reusado sempre que necessário. Capacidades podem conter subcapacidades formando um hierarquia arbitrária de módulos.
- ***Beliefs*** – Crenças representam o conhecimento do agente sobre o seu ambiente e sobre si mesmo. Em Jadex, as crenças podem ser qualquer objeto Java. Elas são armazenadas em uma base de crenças e podem ser referenciadas em expressões, bem como ser acessadas e modificadas por planos usando a interface da base de crenças.
- ***Goals*** – Objetivos compõem a postura motivacional do agente, são o que orientam suas ações. Diferentemente dos sistemas BDI tradicionais, os quais tratam objetivos como um tipo especial de evento, objetivos são um conceito central no Jadex, o qual segue a idéia geral que objetivos são desejos concretos e momentâneos de um agente. Para qualquer objetivo que ele tenha, um agente irá engajar-

se em ações apropriadas, até que ele considere o objetivo como atingido, inatingível ou não mais desejado.

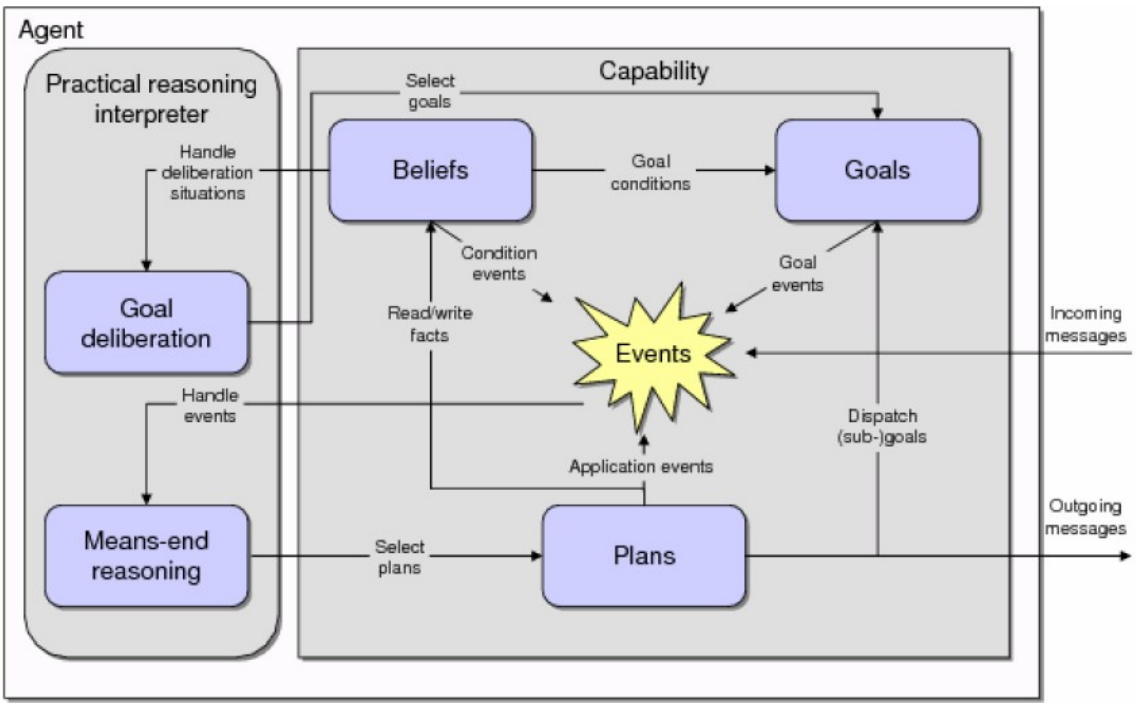

Figura 2: Arquitetura Abstrata Jadex.

- ***Plans*** – Planos representam a forma como o agente atuará em seu ambiente. Entretanto, os planos pré-definidos pelo desenvolvedor compõem uma biblioteca de ações que o agente pode executar. Dependendo da situação corrente, planos são selecionados em resposta à ocorrência de eventos ou de objetivos. A seleção de planos é feita automaticamente pelo sistema e representa um aspecto principal da infraestrutura BDI.
- ***Events*** – Uma importante propriedade dos agentes é a capacidade de reagir a diferentes tipos de eventos. Jadex suporta dois tipos de eventos a nível de aplicação, os quais podem ser definidos pelo desenvolvedor. Eventos internos podem ser usados para denotar uma ocorrência dentro de um agente, enquanto eventos mensagem representam uma comunicação entre dois agente ou mais. Eventos normalmente são tratados por planos.

Raciocínio no Jadex é um processo que consiste de dois componentes entrelaçados. Por um lado, o agente reage a mensagens que chegam, a eventos internos e a objetivos através da seleção e execução de planos. Por outro lado, o agente continuamente delibera seus objetivos correntes, para decidir a respeito de um subconjunto consistente, o qual deve ser perseguido.

O interpretador consiste de uma agenda de componentes armazenando meta-ações a serem executadas. O modo básico de operação é simples: o agente seleciona uma meta-ação de sua agenda e a executa quando as pré-condições da ação são satisfeitas. Caso contrário, a ação é simplesmente descartada. A execução da ação pode produzir ações posteriores, que são adicionadas à agenda seguindo uma estratégia de inserção customizável. Atualmente, a estratégia de inserção principalmente distingue ações relacionadas e não relacionadas, onde ações relacionadas são adicionadas como nodos filho do nodo atual.

Além da criação de novas entradas na agenda, a execução de ações podem ter efeitos colaterais que são de importância para o agente, tal como quando uma crença é alterada ou um objetivo é descartado. Essas ocorrências são capturadas como eventos de sistema e podem causar alterações nele. Elas são computadas por um componente de determinação de alteração correspondente. Para determinar quais efeitos que um evento de sistema tem, o componente avalia as condições impactadas. Se uma nova condição é disparada, novas ações são produzidas e adicionadas à agenda.

De modo contrário a outros sistemas BDI, Jadex intencionalmente não introduz uma nova linguagem de programação de agentes, mas ao invés disso, usa técnicas de engenharia de software estabelecidas, como Java e XML. O mecanismo é especificamente projetado para a construção de sistemas que utilizam princípios e práticas de engenharia de software existentes e ficam como uma camada independente que pode ser flexivelmente implantada em plataformas *middleware*, como por exemplo JADE.

Jadex possui uma série de ferramentas disponíveis. Elas permitem o controle da execução dos agentes, verificando uma série de dados, tais como suas crenças, objetivos e planos, depuração e documentação. Além disso, é fornecido um plugin para a ferramenta Protégé, que serve para o projeto de ontologias.

Três aplicações desenvolvidas com o Jadex são citadas na página do projeto: MedPAge, Dynatech e Bookstore.

3.3 JAM

JAM (Huber 1999) é uma arquitetura de agentes inteligentes que foi feita com base em uma série de teorias sobre agentes e frameworks para agentes inteligentes. Entre as influências sobre a arquitetura, podem ser citadas a teoria BDI (Bratman 1987), as arquiteturas de agentes inteligentes do Procedural Reasoning System (PRS) (Georgeff & Lansky 1986) e sua implementação chamada UMPRS (Lee, Huber, Kenny & Durfee 1994).

Cada agente JAM é composto por cinco componentes primários: um modelo do mundo (*world model*), uma biblioteca de planos (*plan library*), um interpretador (*interpreter*), uma estrutura de intenções (*intention structure*) e um observador (*observer*). A Figura 3 (fonte (Huber 1999)) ilustra a arquitetura.

- ***World Model*** – O *world model* é um banco de dados que representa as crenças do agente. Ele armazena fatos que o agente usa para representar o estado corrente do mundo. Estados das variáveis, sensores de informação, conclusões a partir de deduções e inferências são algumas das informações que são mantidas.
- ***Plan Library*** – A *plan library* é uma coleção dos planos que o agente pode usar para atingir seus objetivos. Um plano define uma especificação procedural para alcançar um objetivo. Sua aplicabilidade é limitada a um objetivo em particular, além disso, pode possuir pré-condições e ser mantido em um contexto.
- ***Interpreter*** – O *interpreter* pode ser considerado o cérebro do agentes. É ele quem raciocina a respeito do que o agente deve fazer e quando. Ele é responsável por selecionar e executar planos baseado nos objetivos e no contexto da situação corrente. Associada ao interpretador, está a estrutura de intenções, uma pilha de execução de objetivos com e sem planos instanciados.
- ***Intention Structure*** – A *intention structure* é um modelo interno dos objetivos e das atividades correntes do agente. É ela que armazena as informações sobre o progresso que o agente fez para atingir seus objetivos de nível mais alto.

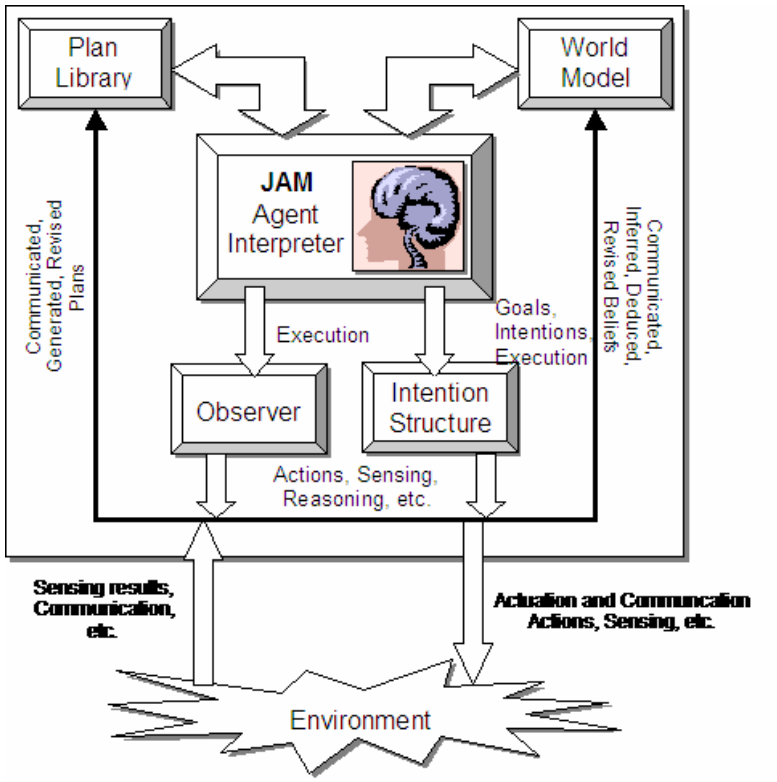

Figura 3: Arquitetura JAM.

- ***Observer*** – O *observer* pode ser considerado um plano leve que é executado entre os passos do plano, a fim de realizar funcionalidades que não estão determinadas no curso normal do plano. Um exemplo é guardar mensagens que estejam sendo enviadas. Assim, essa funcionalidade é tipicamente usada para atualizar o modelo do mundo independentemente do ciclo normal de execução do plano.

O funcionamento da arquitetura JAM é dado como segue. Alterações no modelo do mundo ou a divulgação de novos objetivos disparam uma busca por planos que podem ser aplicados à situação. Assim, é formada uma lista de planos aplicáveis (APL). O interpretador JAM seleciona um plano dessa lista e o coloca como intenção, isto é, o coloca para ser executado. Isso significa que o plano será colocado na estrutura de intenções do agente. O agente pode ou não executá-lo imediatamente, visto que isto depende da utilidade do plano em relação com as outras intenções que já estão na estrutura.

A arquitetura JAM é implementada em Java. Para o desenvolvimento do agente, o desenvolvedor lida explicitamente com o modelo do mundo, a biblioteca de planos, o observador e a especificação dos objetivos iniciais do agente. Existem algumas funcionalidades ainda não implementadas na arquitetura, tais como geração de planos e aprendizado.

3.4 Jason

Jason (Bordini, Wooldridge & Hübner 2007) é uma plataforma de desenvolvimento de sistemas multi-agentes baseada em um interpretador para uma versão estendida da linguagem AgentSpeak(L).

A linguagem AgentSpeak(L) foi apresentada em (Rao 1996). Ela é uma linguagem de programação orientada a agentes baseada na lógica de primeira ordem, com even-

tos e ações. Ela é inspirada na arquitetura BDI (Rao & Georgeff 1995) e na lógica BDI. Além disso, ela é baseada em implementações já existentes de sistemas BDI, tais como o *Procedural Reasoning System* (PRS) (Georgeff & Lansky 1986) e o *Distributed Multi-Agent Reasoning System* (dMARS) (D'Inverno et al. 2004). Na sua definição original, ela era somente uma linguagem abstrata de programação. Dessa forma, Jason foi implementado tendo como base AgentSpeak, mas também oferece uma série de extensões que são necessárias para o desenvolvimento de sistemas multi-agentes.

Em um programa escrito em AgentSpeak, as crenças, os desejos e as intenções do agente não são explicitamente representadas como fórmulas modais. Ao invés disso, o desenvolvedor deve usar as notações da linguagem. O estado corrente do agente, o qual é um modelo de si mesmo, seu ambiente, e outros agentes podem ser vistos como crenças. Estados aos quais o agente deseja atingir baseado em estímulos externos e internos podem ser vistos como desejos. E a escolha de planos que satisfazem um dado estímulo pode ser vista como intenções.

A especificação de um agente em AgentSpeak(L) consiste de:

- ***Beliefs Base*** – É um conjunto de crenças base, as quais são fórmulas atômicas de primeira ordem.
- ***Goal*** – É um estado do sistema, ao qual o agente deseja chegar. Existem dois tipos de objetivos:
  - ***Achievement Goal***: afirma que o agente deseja atingir o estado de mundo onde a fórmula atômica associada é verdadeira.
  - ***Test Goal***: afirma que o agente deseja testar se a fórmula atômica associada é (ou pode ser unida com) uma de suas crenças.
- ***Plan Library*** – É uma biblioteca de planos do agente, na qual cada plano determina um conjunto de ações que deve ser executado a fim de um dado objetivo ser atingido. Cada plano é constituído de:
  - ***Head***: formado por um evento ativador (*triggering event*), o qual define quais eventos podem iniciar a execução de um plano e um conjunto de literais representando um contexto. Um evento pode ser interno, quando gerado pela execução de um plano em que um subobjetivo precisa ser alcançado, ou externo, quando gerado pelas atualizações de crenças que resultam da percepção do ambiente. O contexto deve ser conseqüência lógica do conjunto de crenças do agente no momento em que o evento é selecionado pelo agente para o plano ser considerado aplicável.
  - ***Body***: O corpo do plano inclui ações básicas, ou seja, ações que representam operações atômicas que o agente pode executar a fim de alterar o ambiente.

Detalhes de como funciona um interpretador de AgentSpeak podem ser vistos na Figura 4 (fonte (Machado & Bordini 2002)). Ela mostra os aspectos essenciais do interpretador para a definição original da AgentSpeak, entretanto não inclui as extensões implementadas no Jason. Na figura, conjuntos (de crenças, eventos, planos e intenções) são representados por retângulos. Losangos representam seleção (de um elemento de um conjunto). Círculos representam algum processamento envolvido com a interpretação do programa AgentSpeak.

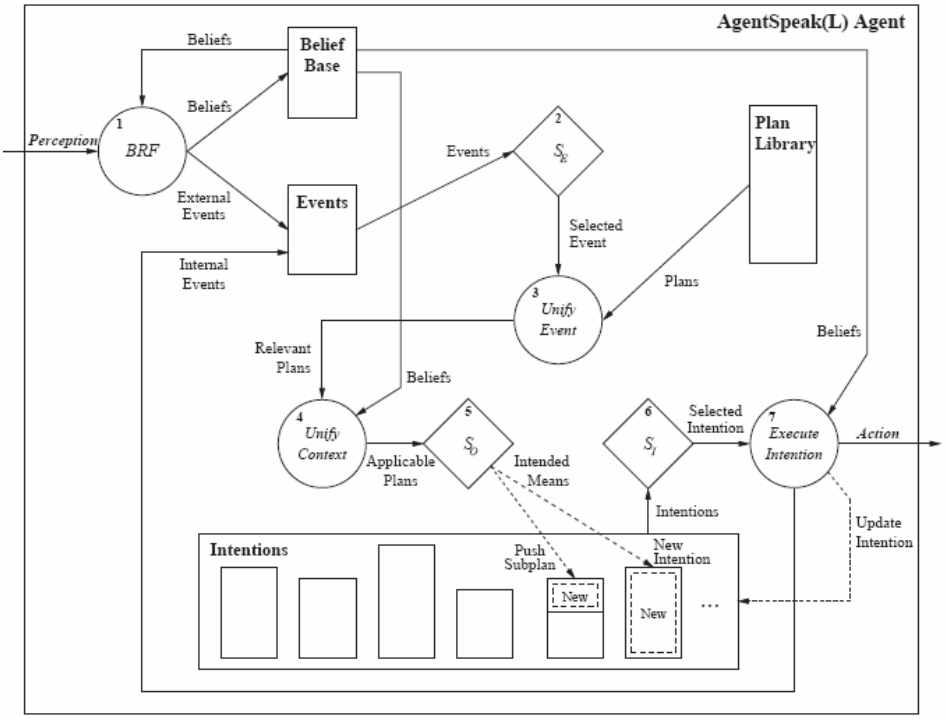

Figura 4: Ciclo de Interpretação de um Programa AgentSpeak.

Jason é implementado em Java e está disponível em código aberto. Entre suas características pode-se citar a configuração do sistema multi-agentes através de um arquivo texto, podendo-se optar pela execução em um ambiente distribuído. Além disso, um ambiente de desenvolvimento integrado (IDE) é disponibilizado, o qual permite a execução de programas também em modo de depuração. Uma vantagem do uso da linguagem AgentSpeak para o desenvolvimento de sistemas multi-agentes é que ela possui semântica formal, o que possibilita a verificação formal de sistemas programados com esta linguagem.

4 Comparação

As tabelas 1 e 2 exibem um resumo dos aspectos relacionados com cada uma das implementações da arquitetura BDI estudadas. Observa-se que eventualmente os desejos (objetivos) não estão explicitamente presentes, mas são vistos como um tipo particular de evento.

Um ambiente de desenvolvimento integrado (IDE) é muito importante para o desenvolvimento de programas. Apenas o destaque de palavras reservadas da linguagem de programação já é uma facilidade oferecida ao desenvolvedor. Por isso, a falta de um IDE para a linguagem utilizada pelo JAM é um desvantagem a ser considerada. Como o Jadex é baseado apenas em Java e XML, ambientes já existentes, tais como o Eclipse, podem ser utilizados.

Além dos aspectos apontados nas tabelas, existem outros pontos particulares das plataformas. O JACK é a única plataforma que não é livre, além disso, ele mostra uma forte preocupação com a indústria. O Jadex diferencia-se por não requerer o uso de uma nova linguagem, também, pelo fato dele normalmente ser executado sobre a plataforma JADE, ele é *FIPA-Compliant*. O Jadex possui um plugin para ser utilizado com a ferramenta Protégé, a qual serve para fazer o projeto de ontologias. Já a linguagem

utilizada pelo Jason possui semântica formal, permitindo a verificação formal dos programas. Não foram observadas características relevantes do JAM.

Visto que todas as plataformas são construídas como extensão da linguagem Java, elas acabam herdando todas as vantagens fornecidas por ela. Entre essas vantagens, pode-se citar a portabilidade e reuso de componentes e bibliotecas existentes.

Plataforma	Linguagem	Principais Componentes
JACK	JACK Agent Language	Capability, BeliefSet, View, Event, Plan
Jadex	Java e XML	Capability, Beliefs, Goals, Plans, Events
JAM	JAM	World Model, Plan Library, Interpreter, Intention Structure, Observer
Jason	AgentSpeak (L)	Beliefs Base, Goal, Plan Library

Tabela 1: Linguagens e Componentes.

Plataforma	Ferramentas	Aplicações Comerciais
JACK	IDE, Debug	UVAs, Gerenciamento de Tráfego Aéreo, Real-time scheduling
Jadex	Ferramentas para execução, debug e documentação	MedPAge, Dynatech, Bookstore
JAM	-	-
Jason	IDE, Mind Inspector	-

Tabela 2: Ferramentas e Aplicações.

5 Conclusão

Sistemas multi-agentes são uma importante nova direção na área de Engenharia de Software. Em certos sistemas, há a necessidade que agentes possuam a capacidade de raciocínio. Assim, surgiu a arquitetura BDI, que modela agentes com crenças, desejos e intenções, viabilizando a implementação de agentes com raciocínio que segue o modelo BDI.

Neste trabalho, foram apresentados quatro frameworks baseados na arquitetura BDI. JACK mostra uma forte preocupação com aplicações industriais. Jadex não introduz linguagem de programação nova e possui uma série de ferramentas que auxiliam no processo de desenvolvimento. JAM não apresentou características relevantes, além de não possuir um ambiente de desenvolvimento. Jason é baseado em uma linguagem que possui semântica formal.

Referências Bibliográficas

Bellifemine, F. L., Claire, G. & Greenwood, D. (2007), **Developing Multi-Agent Systems with JADE**, John Wiley & Sons, Inc., New York, USA.

Bordini, R. H., Wooldridge, M. & Hübner, J. F. (2007), **Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)**, John Wiley & Sons.

Bratman, M. E. (1987), **Intention, Plans, and Practical Reason**, Cambridge, MA.

D'Inverno, M., Kinny, D., Luck, M. & Wooldridge, M. (1997), **A formal specification of dMARS**, in 'Agent Theories, Architectures, and Languages', pp. 155–176.

D'Inverno, M., Luck, M., Georgeff, M. P., Kinny, D. & Wooldridge, M. J. (2004), '**The dMARS architecture: A specification of the distributed multi-agent reasoning system**', Autonomous Agents & Multi-Agent Systems 9, 5–53.

Georgeff, M. & Lansky, A. (1986), **Procedural knowledge**, in 'IEEE Special Issue on Knowledge Representation', Vol. 74, pp. 1383–1398.

Georgeff, M., Pell, B., Pollack, M., Tambe, M. & Wooldridge, M. (1999), **The belief-desire-intention model of agency**, in J. Müller, M. P. Singh & A. S. Rao, eds, 'Proceedings of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL-98)', Vol. 1555, Springer-Verlag: Heidelberg, Germany, pp. 1–10.

Howden, N., Rönnquista, R., Hodgson, A. & Lucas, A. (2001), **Jack intelligent agents™: Summary of an agent infrastructure**, in 'The Fifth International Conference on Autonomous Agents', Montreal, Canada.

Huber, M. J. (1999), **Jam: a bdi-theoretic mobile agent architecture**, in 'AGENTS '99: Proceedings of the third annual conference on Autonomous Agents', ACM, New York, NY, USA, pp. 236–243.

JACK intelligent agents: JACK manual (2005), Technical Report 4.1, Agent Oriented Software Pvt. Ltd, Melbourne, Australia.

Jennings, N. R. (1999), **Agent-Oriented Software Engineering**, in F. J. Garijo & M. Boman, eds, 'Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent System Engineering (MAAMAW-99)', Vol. 1647, Springer-Verlag: Heidelberg, Germany, pp. 1–7.

Lee, J., Huber, M. J., Kenny, P. G. & Durfee, E. H. (1994), **UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications**, in 'Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS)', Houston, Texas, pp. 842–849.

Machado, R. & Bordini, R. H. (2002), **Running AgentSpeak(L) agents on Sim Agent**, in 'ATAL '01: Revised Papers from the 8th International Workshop on Intelligent Agents VIII', Springer-Verlag, London, UK, pp. 158–174.

Pokahr, A. & Braubach, L. (2007), **Jadex user guide**, Technical Report 0.96, University of Hamburg, Hamburg, Alemanha.

Pokahr, A., Braubach, L. & Lamersdorf, W. (2003), '**Jadex: Implementing a BDI-Infrastructure for JADE Agents**', EXP – in search of innovation 3(3), 76–85.

Rao, A. S. (1996), **AgentSpeak(L): BDI agents speak out in a logical computable language**, in 'MAAMAW '96: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world: agents breaking away', Springer-Verlag New York, Inc., Secaucus, NJ, USA, pp. 42–55.

Rao, A. S. & Georgeff, M. P. (1995), **BDI-agents: from theory to practice**, in 'Proceedings of the First Intl. Conference on Multiagent Systems', San Francisco.

Wooldridge, M. (1999), '**Intelligent agents**', pp. 27–77.

Wooldridge, M. & Ciancarini, P. (2000), **Agent-Oriented Software Engineering: The State of the Art**, in P. Ciancarini & M. Wooldridge, eds, 'First Int. Workshop on Agent-Oriented Software Engineering', Vol. 1957, Springer-Verlag, Berlin, pp. 1–28.

Wooldridge, M. J. (2000), **Reasoning about Rational Agents**, MIT Press.