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

GUSTAVO LUIZ KIELING

Inserção de Conhecimento Probabilístico para Construção de Agentes BDI Modelados em Redes Bayesianas

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

Profa. Dra. Rosa Maria Vicari
Orientador

Porto Alegre, março de 2011

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Kieling, Gustavo Luiz

Inserção de Conhecimento Probabilístico para Construção de Agentes BDI Modelados em Redes Bayesianas / Gustavo Luiz Kieling. – Porto Alegre: PPGC da UFRGS, 2011.

80 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, RS–BR, 2011. Orientador: Rosa Maria Vicari.

1. Agentes. 2. BDI. 3. Conhecimento probabilístico. 4. Sistemas multiagentes. 5. Redes bayesianas. I. Vicari, Rosa Maria. II. Título.

AGRADECIMENTOS

Agradeço a todos os funcionários e professores da Universidade Federal do Rio Grande do Sul, em especial do Programa de Pós-Graduação em Computação, que fazem desta Pós-Graduação uma das mais conceituadas no Brasil. Em particular, agradeço a Professora Rosa Vicari, que foi fundamental na realização deste trabalho.

Aos Professores Rafael Bordini e Jomi Hübner pela disposição em ajudar e esclarecer dúvidas.

Aos meu pais e a minha irmã por todo o amor incondicional que SEMPRE me deram, especialmente nas horas mais difíceis. Ao meu avô, tios, tias, primos, primas, cunhado, cachorros, gatos e papagaios que sempre estiveram ao meu lado. Agradeço especialmente a minha avó, que muito rezou por mim. Também agradeço ao meu tio que foi meu primeiro professor de informática.

Aos meus colegas e amigos de Mestrado e do Grupo de Inteligência Artificial, sempre prestativos e que proporcionaram inúmeros momentos de lazer em meio ao trabalho.

Aos amigos que por muito tempo tiveram que se contentar com a minha ausência por eu estar fazendo este trabalho.

A todos os funcionários da Dinamize, que têm sido muito importantes para meu crescimento pessoal e profissional e por fazerem meus dias (e noites) mais divertidos.

Muito obrigado a todos!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

BDI	Beliefs, Desires, Intentions
COPA	COnhecimento Probabilístico em Agentes BDI
GUI	Graphic User Interface
IP	Internet Protocol
MDP	Markov Decision Process
PL	Probabilistic Language
POMDP	Partially Observable Markov Decision Process
PRS	Procedural Reasoning System
TPC	Tabela de Probabilidades Condicionais
XL	eXtended Language

LISTA DE FIGURAS

LISTA DE TABELAS

Insertion of Probabilistic Knowledge into BDI Agents Construction Modelled in Bayesian Networks

ABSTRACT

Achieving faithful representation of knowledge is a historic and still unreached goal in the area of Artificial Intelligence. Problems are solved and decisions are made taking into consideration different kinds of knowledge, from which many are biased, inaccurate, ambiguous or still incomplete.

Computational systems that store knowledge in many different ways have been built in order to emulate the capacity of human knowledge representation, taking into consideration the several inherent difficulties to it.

Within this context, this paper proposes an experiment that utilizes two distinct ways of representing knowledge: symbolic, BDI in this case, and probabilistic, Bayesian Networks in this case.

In order to develop a proof of concept of this propose of knowledge representation, examples that will be built through agent oriented programming technology will be used. For that, implementation of a MultiAgent System was developed, extending the *Jason* framework through the implementation of a plugin called *COPA*. For the representation of probabilistic knowledge, a Bayesian Network building tool, also adapted to this system, was used.

The case studies showed improvement in the management of uncertain knowledge in relation to the building approaches of classic BDI agents, i.e., that do not use probabilistic knowledge.

Keywords: agents, bayesian networks, BDI, multiagent systems, probabilistic knowledge.

RESUMO

A representação do conhecimento de maneira mais fiel possível à realidade é uma meta histórica e não resolvida até o momento na área da Inteligência Artificial. Problemas são resolvidos e decisões são tomadas levando-se em conta diversos tipos de conhecimentos, os quais muitos são tendenciosos, inexatos, ambíguos ou ainda incompletos.

A fim de tentar emular a capacidade de representação do conhecimento humano, levando-se em conta as diversas dificuldades inerentes, tem-se construído sistemas computacionais que armazenam o conhecimento das mais diversas formas.

Dentro deste contexto, este trabalho propõe um experimento que utiliza duas formas distintas de representação do conhecimento: a simbólica, neste caso BDI, e a probabilística, neste caso Redes Bayesianas.

Para desenvolvermos uma prova de conceito desta proposta de representação do conhecimento estamos utilizando exemplos que serão construídos através da tecnologia de programação voltada para agentes. Para tal, foi desenvolvida uma implementação de um Sistema MultiAgente, estendendo o *framework Jason* através da implementação de um *plugin* chamado *COPA*. Para a representação do conhecimento probabilístico, utilizamos uma ferramenta de construção de Redes Bayesianas, também adaptada a este sistema.

Os estudos de caso mostraram melhorias no gerenciamento do conhecimento incerto em relação às abordagens de construções de agentes BDI clássicos, ou seja, que não utilizam conhecimento probabilístico.

Palavras-chave: Agentes, BDI, conhecimento probabilístico, sistemas multiagentes, redes bayesianas, bayesian networks, BDI, agents, Jason, probabilistic knowledge.

1 INTRODUÇÃO

A representação de conhecimento continua sendo um grande desafio para a comunidade de Inteligência Artificial (IA). No entanto, segundo PATEL-SCHNEIDER (1985), a natureza exata do papel desta área é indefinida. Sistemas de representação do conhecimento podem variar de pacotes para manipular estruturas de dados até sistemas de IA completos que planejam ou fazem gestão de recursos.

A principal dificuldade na representação do conhecimento fiel à realidade ocorre pelo fato de o conhecimento ser uma propriedade humana muito difícil de ser representada de maneira formal, a fim de ser interpretado por sistemas computacionais em geral. O enorme número de variáveis existentes num ambiente e a alta complexidade de possíveis relações entre estas dificultam a formalização do conhecimento.

Frequentemente temos de lidar com conhecimento probabilístico, em que nos baseamos em informações incompletas, inexatas, imparciais ou até mesmo ambíguas para raciocinar e tomar decisões. Dadas informações técnicas e também sobre experiências passadas, podemos, por exemplo, fazer a seguinte afirmação: “Hoje, a probabilidade de chuva é de 80%”. Devido a essa incerteza, é improvável que tenhamos a capacidade de afirmar categoricamente algumas informações, acoplando uma probabilidade a elas.

Nesta dissertação de Mestrado, apresentamos estudos de caso preliminares que buscam integrar conhecimento simbólico e probabilístico na seleção de planos para os agentes artificiais. O *framework* escolhido para a construção destes agentes foi o *Jason*. Sua arquitetura de sistema é baseada em BDI (Belief-Desire-Intetion), tecnologia para a representação do conhecimento, e é voltado para a programação de Sistemas MultiAgentes.

Para facilitar a inclusão do conhecimento no sistema pelo usuário especialista, utilizamos uma rede probabilística, mais especificamente, uma Rede Bayesiana (RB), que é um modelo gráfico probabilístico que representa um conjunto de variáveis discretas e suas dependências condicionais através de um grafo acíclico direcionado. Através deste modelo de representação, o conhecimento passa a ter um número probabilístico, ou seja, não será mais admitido somente como falso ou verdadeiro, mas sim com um grau de certeza. A ferramenta de construção de RBs utilizada aqui foi o *JAmplia*.

A implementação deste trabalho foi realizada com o desenvolvimento de um *plugin* chamado *COPA* (**CO**nhecimento **P**robabilístico em **A**gentes BDI). Nele, estendemos o *Jason* através da reimplementação do algoritmo de seleção de planos e da adição de novas funcionalidades no *JAmplia* para a comunicação destas duas ferramentas.

1.1 Objetivos

O principal objetivo deste trabalho é acrescentar conhecimento probabilístico às crenças do sistema BDI, utilizando Redes Bayesianas para modelar as informações do am-

biente. Esta alteração possibilita que um sistema de agentes, por exemplo, desenvolvido através de BDI utilize revisão de crenças probabilísticas em vez de heurísticas para a escolha dos seus planos. Desta maneira, espera-se facilitar a construção de agentes que, de alguma forma, trabalhem com informações incertas.

Outro objetivo é a união de duas áreas de pesquisa da Inteligência Artificial. Redes Bayesianas e sistemas BDI passam a contribuir para a construção de um sistema que gerencia tanto o conhecimento incerto (oriundo da RB) quanto o conhecimento simbólico (originado do sistema BDI), resultando em um melhor gerenciamento do conhecimento em geral.

Para verificarmos as possibilidades desta proposta, construímos uma prova de conceito desenvolvida com a tecnologia de Sistemas MultiAgentes. Estudos de caso também foram desenvolvidos para exemplificar a utilização da proposta deste trabalho.

1.2 Trabalhos Relacionados

A representação de conhecimento probabilístico integra duas áreas bem estabelecidas: agentes BDI e Redes Bayesianas. Muitos trabalhos significativos a respeito destas áreas têm sido desenvolvidos desde os anos oitenta. Nesta seção, serão apresentados cinco trabalhos que de alguma forma estão relacionados ao aqui proposto. Estes foram escolhidos por serem os trabalhos de relevância nacional e internacional que mais se assemelham ao *COPA*.

1.2.1 Agentes Graduados

Um modelo para agentes BDI graduados foi descrito em CASALI; GODO; SIERRA (2005) o qual permite a representação explícita da incerteza em crenças, desejos e intenções. A arquitetura proposta é baseada em sistemas multicontextos para modelar crenças (*BC - Beliefs Context*), desejos (*DC - Desires Context*), intenções (*IC - Intentions Context*), planos (*PC - Plan Context*) e comunicações (*CC - Communication Context*) graduados, isto é, onde a graduação corresponde à probabilidade de o agente acreditar na crença. A especificação do sistema multicontexto de um agente contém três componentes básicos: contextos, lógicas e regras de ligação, que possibilitam a propagação das conseqüências entre as teorias.

O *PC* computa o custo associado a cada plano e procura planos para alterar o ambiente atual em outro onde algum objetivo é satisfeito. O *CC* é responsável pela comunicação do agente com o mundo, recebendo e enviando mensagens. O modelo de agente BDI neste trabalho pode ser definido como: $A_g = (\{BC, DC, IC, PC, CC\}, \Delta_{br})$, onde Δ_{br} são as regras de inferência com premissas e conclusões em diferentes contextos. As crenças são atreladas às probabilidades da forma $B\varphi$, sendo interpretada como "φ é provável", onde φ indica a probabilidade desta crença ser verdadeira.

CASALI; GODO; SIERRA (2005) distinguem o que é positivamente desejável do que não é rejeitado através de dois operadores: $D^+\varphi$ (φ é positivamente desejável) e $D^-\varphi$ (φ é negativamente desejável). As preferências do agente são expressas por uma teoria contendo expressões quantitativas com preferências positivas e negativas, como $(D^+\varphi, \alpha)$ ou $(D^-\varphi, \alpha)$, e também expressões qualitativas como $D^+\psi \rightarrow_L D^+\varphi$ (ou $D^-\psi \rightarrow_L D^-\varphi$), expressando que φ é menos preferível (ou rejeitado, respectivamente) à ψ.

Assim como nos estados mentais abordados anteriormente, intenções também têm associação com graduações. Pode-se construir uma teoria T contendo a fórmula $I\psi \rightarrow_L$

$I\varphi$, indicando que o agente deve tentar executar φ antes de ψ e não deve tentar executar ϕ se $(I\phi, \delta)$ é uma fórmula em T e $\delta < Threshold$. Isto pode significar que o lucro em atingir ϕ é baixo ou seu custo é alto.

1.2.2 Raciocínio Prático em Incerteza

Em AMGOUD; PRADE (2007), é apresentado um *framework* formal para o raciocínio prático (BRATMAN; ISRAEL; POLLACK, 1988) baseado em uma máquina abstrata argumentativa, dividido em três passos:

1. É computado um conjunto de desejos atingíveis, dado o estado atual;

2. É computado um conjunto de planos (extensões) compatíveis entre si, isto é, que são atingíveis juntos;

3. O resultado dos dois passos anteriores é combinado para encontrar a melhor extensão, podendo levar em consideração o número e/ou a importância de atingir tal desejo por extensão e o número de planos por desejo na extensão.

Segundo AMGOUD; PRADE (2007), há uma grande dificuldade na área em definir exatamente a natureza do raciocínio prático. Enquanto alguns pesquisadores a consideram um problema de inferência, outros a definem como um problema de tomada de decisão. Assim, este foi o primeiro trabalho a envolver ambas abordagens, dividindo em dois passos de inferência e um de tomada de decisão.

Um dos pontos fracos desta abordagem é a incapacidade de, caso o agente acredite em $\neg c$, desejar c. Por exemplo, se o agente acreditar na crença $\neg temCarro$, será incapaz de desejar atingir o estado do ambiente onde a crença $temCarro$ seja verdadeira, pois o fechamento dedutivo consistente não distingue literais de desejos e literais de crenças.

1.2.3 Bayes Jason

Desenvolvida por CALCIN (2007), esta ferramenta estende a gramática de AgentSpeak(L), criando uma linguagem adaptada para a modelagem de Redes Bayesianas juntamente com as crenças de um agente.

Bayes Jason não implementa interface gráfica do usuário (*Graphic User Interface, GUI*), inclusive sendo citado como trabalho futuro. Como a RB é inteiramente modelada dentro do agente, exige do usuário conhecimento prévio de RBs, podendo dificultar a programação do mesmo, visto que as probabilidades são determinadas pela posição em uma lista para cada variável da rede.

Para ilustrar como um agente poderia ser modelado com essa ferramenta, utilizaremos um exemplo apresentado no próprio trabalho citado anteriormente. O alarme de uma residência é acionado quando nota algum movimento suspeito. Entretanto, esse movimento pode ter sido causado por um ladrão ou por um terremoto. No primeiro caso, a ação recomendada seria chamar a polícia, enquanto que no segundo, seria simplesmente parar de soar o alarme. A RB correspondente a este exemplo é apresentada na figura 1.1.

O agente poderá escolher qual plano executar conforme o valor de certeza da variável *assalto*. O plano 2 será ativado se o valor da crença *assalto*(*verdadeiro*) for maior ou igual a 0.8, chamando a polícia. Caso este valor estiver dentro do intervalo [0.5, 0.8), podemos acreditar que não existem evidências suficientes para que uma medida drástica seja tomada, porém seria adequado ligar para o vizinho (plano 3). Se a probabilidade for

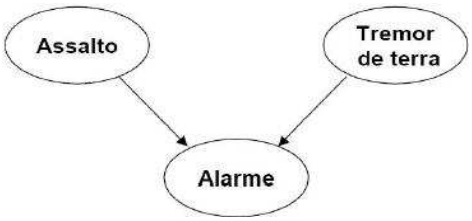

Figura 1.1: RB para o exemplo de um alarme doméstico.

menor que 0.5, acreditamos que o melhor a fazer é não adotar qualquer ação, ignorando as evidências (plano 4).

Segundo CALCIN (2007), essa ferramenta melhorou consideravelmente o tratamento de incertezas que o *Jason* utiliza atualmente. Por exemplo, deparando-se com a evidência *alarme(ativado)*, o agente certamente adotaria o segundo plano, pois não há forma alguma de avaliar o grau de certeza da crença *assalto(verdadeiro)*.

O código abaixo apresenta o código-fonte do agente, contendo suas crenças, juntamente com a RB e seus planos. A rede bayesiana é modelada na base de crenças do agente, da linha 2 até a linha 9. O objetivo $realizarAcoes$ é adicionado na linha 16, obrigando o agente escolher um plano a ser executado entre os três relevantes (linhas 17, 19 e 21). O plano a ser executado será aquele que tiver seu contexto analisado como verdadeiro. Caso mais de um plano estiver nesta condição, será dada preferência para aquele que estiver em primeiro lugar na lista de planos, isto é, o primeiro definido no código-fonte do agente. Analisaremos os tipos de planos e como é realizada a escolha do plano para execução na seção 2.2.3.

```
1  // Definição dos nós e estados da rede
2  assalto(verdadeiro)<assalto: verdadeiro,falso>.              (crença 1)
3  tremorTerra(verdadeiro)<tremorTerra: verdadeiro,falso>.      (crença 2)
4  alarme(ativado)<alarme: ativo,desativado>.                   (crença 3)
5
6  // Definição das tabelas de probabilidades
7  assalto:<0.001,0.999>.                                       (tabela 1)
8  tremorTerra:<0.002,0.998>.                                   (tabela 2)
9  alarme:<0.950,0.950,0.290,0.001,0.05,0.05,0.71,0.999>.       (tabela 3)
10
11 // Planos do agente
12 +inicio(agenteA) : true                                      (plano 1)
13    <- .print("Iniciando agente");
14       .setEvidence(alarme,ativado);
15       .updateBelief(assalto);
16       !realizarAcoes(agenteA).
17 +!realizarAcoes(agente) : .compValueBelief(assalto(verdadeiro),gt,0.8)
18    <- .print("Ligar para a policia").                         (plano 2)
19 +!realizarAcoes(agente) : .compValueBelief(assalto(verdadeiro),gt,0.5)
20    <- .print("Ligar para o vizinho e perguntar").            (plano 3)
21 +!realizarAcoes(agente) : .compValueBelief(assalto(verdadeiro),gt,0)
22    <- .print("Nada aconteceu").                              (plano 4)
```

Para a comparação de valores de probabilidade no contexto dos planos (linhas 17, 19 e 21) utilizou-se o operador *gt* (*greater than*, ou *maior que* em tradução livre), implementado juntamente com a função de comparação de valores $compValueBelief$.

Até o presente momento, o Bayes Jason ainda não teve sua implementação concluída, não tendo previsão para término. Também não foi possível localizar seu código-fonte.

1.2.4 Agentes Baseados em Personalidade

O trabalho de PEREIRA et al. (2008) descreve uma abordagem para a autorregulagem de processos de trocas sociais baseados em personalidade em Sistemas MultiAgentes. Visto que os agentes não necessariamente têm acesso aos estados internos dos outros agentes, o processo de decisão deve ser feito em um modo *parcialmente observado*. Portanto, a decisão sobre a melhor troca que um agente deve propor a outro para atingir o equilíbrio social é modelado como um Processo de Decisão de Markov Parcialmente Observável (*Partially Observable Markov Decision Process, POMDP*) global para cada traço de personalidade que o agente possa assumir:

Egoísta: o agente está interessado em trocas que o beneficiem;

Altruísta: o agente busca o benefício do outro;

Fanático pelo equilíbrio: o agente tem uma alta probabilidade de aceitar trocas que conduzam o sistema para um estado de equilíbrio;

Tolerante: a probabilidade que o agente aceite qualquer tipo de troca é alta.

O mecanismo de controle social é realizado por um supervisor de equilíbrio centralizado. Baseado nas personalidades citadas, este supervisor decide quais ações deve recomendar aos agentes para que o sistema atinja o equilíbrio através de um intervalo qualitativo totalmente observável MDP (*Qualitative Interval Markov Decision Process, QI-MDP*).

O processo de decisão da melhor troca que o agente α deve propor ao agente β é modelado como POMDP$_{\alpha\beta}$. Para cada par do modelo de personalidades, o agente α tem a lista completa de todas as possibilidades de valores dos ganhos de α (E$_\alpha$, conhecido por α) e β (E$_\beta$, não-observável por α). Assim, para cada modelo de personalidade que β possa assumir, o POMDP$_{\alpha\beta}$ é decomposto em três sub-POMDPs, um para cada estado interno atual de α, quando α está sendo favorecido (POMDP$^{+}_{\alpha\beta}$), em equilíbrio (POMDP$^{0}_{\alpha\beta}$) e desfavorecido (POMDP$^{-}_{\alpha\beta}$).

Baseado nas políticas de controle social e no trabalho desenvolvido em (SIMARI; PARSONS, 2006 apud PEREIRA et al., 2008) foi desenvolvido o algoritmo *policyToBDIplans* para extrair planos BDI dos grafos de políticas POMDP$^{*}_{\alpha\beta}$. As simulações realizadas, em AgentSpeak utilizando *Jason*, apresentaram que a abordagem proposta é viável e pode ser uma boa solução em aplicações baseadas na teoria de trocas sociais.

1.2.5 *Jason* com Lógica Fuzzy

Em FARIAS (2009), foi proposta uma implementação na plataforma *Jason* de um modelo de agente BDI-Fuzzy, possibilitando um agente ter um grau de certeza em suas crenças, desejos e planos. Assim, o agente dessa arquitetura deverá ser dotado de mecanismos que levem em consideração os graus fuzzy em crenças, desejos e planos. Atualmente este trabalho ainda encontra-se em fase de implementação. Sua previsão de término é para 2011.

1.2.6 Considerações

Apesar de todos os trabalhos apresentados gerenciarem informações probabilísticas de alguma forma, somente CALCIN (2007) prevê a utilização de Redes Bayesianas. BayesJason faz o caminho oposto do *COPA*. Isto é, as crenças probabilísticas são construídas

pelo programador na base de crenças do agente e podem ser transformadas em uma rede bayesiana, enquanto que no *COPA* as crenças dos agentes são montadas na rede bayesiana e transportadas para a base de crenças dos agentes.

No COPA, o programador tem a possibilidade de modelar o conhecimento do especialista de forma intuitiva, aproveitando todas as vantagens da utilização de uma ferramenta para construção de redes bayesianas. Além disso, o desenvolvedor do sistema pode implementar os agentes artificiais de modo a levarem em consideração as probabilidades das crenças definidas na RB, em vez de utilizá-las de forma estática e definidas antes do início da execução do sistema.

A utilização de redes bayesianas dentro da base de crenças do agente apresenta diversas dificuldades, tais como:

- Dificuldade no posicionamento das probabilidades para cada estado de cada variável;
- Ausência de relação entre as variáveis, impossibilitando o cálculo de inferência;
- Manutenção extremamente custosa;
- Necessidade de alterar a linguagem AgentSpeak(XL);
- Utilização de limiares no contexto dos planos;
- Necessidade da definição dos nós, estados e suas respectivas tabelas de probabilidade no código-fonte do agente, resultando em um código consideravelmente grande.

Embora integre redes bayesianas e o modelo BDI, BayesJason utiliza RBs no contexto de seleção de intenções, ou seja, nos planos da linguagem AgentSpeak(L), mais especificamente no disparador (*trigger*) dos planos. Assim, o programador ainda deve definir limiares para as probabilidades das crenças no contexto dos planos, tornando-os grandes, dificultando suas construções e comprometendo a exploração de comportamentos diferenciados. O *COPA* faz o trabalho de análise de probabilidades para o programador. BayesJason não utiliza a representação do conhecimento para verificar a compatibilidade entre estados mentais, nem recuperar estados do ambiente em que desejos possam ser atingidos.

CASALI; GODO; SIERRA (2005) utilizam conhecimento incerto em agentes BDI de forma mais geral do que no *COPA*. Diferentemente do trabalho aqui apresentado, probabilidades (graduações) estão presentes em todas as partes do sistema, desde crenças, desejos e intenções até planos e comunicações.

Os trabalhos de AMGOUD; PRADE (2007), PEREIRA et al. (2007) e FARIAS (2007) não desenvolveram novas técnicas para representação de conhecimento incerto. O primeiro estuda o raciocínio prático, levando em consideração o fator de incerteza ao deliberar ações a serem executadas pelo agente. O segundo desenvolveu uma técnica para trocas sociais de agentes baseados em personalidades. Visto que o estado dos outros agentes é não-observável pelo que está propondo uma troca, este deve levar em consideração os diversos estados possíveis dos outros agentes, gerando a incerteza no seu processo de raciocínio. O terceiro estudo analisado difere do *COPA* por estar baseado em outra lógica (Fuzzy) para seus agentes. O modo de inserção do conhecimento não foi alterado, mas sim como o *Jason* manipula-o, passando a utilizar graus fuzzy em suas crenças, desejos e planos. Entretanto, como ainda está em fase de implementação, deverá ser estudado mais a fundo quando concluído, para que seja possível tirar conclusões mais definitivas.

1.3 Estrutura do Texto

Além da introdução, onde foram apresentados os trabalhos relacionados, este trabalho é divido em mais três partes. No capítulo 2, as principais ferramentas utilizadas são apresentadas em detalhe, afim de que o leitor familiarize-se com tais tecnologias e tenha o conhecimento necessário para entender o funcionamento da ferramenta aqui desenvolvida.

A principal contribuição deste trabalho é apresentada no capítulo 3, onde também são detalhadas todas as alterações realizadas nas ferramentas utilizadas, analisados dois estudos de caso e discutidas as contribuições deste trabalho, além das vantagens e desvantagens em relação à abordagem clássica.

Por fim, no capítulo 4 são feitas as considerações finais e levantamento de trabalhos futuros.

2 REFERENCIAL BIBLIOGRÁFICO

Neste capítulo serão apresentados os principais conceitos e tecnologias utilizadas neste trabalho. Primeiro é explicado o conceito de Sistemas MultiAgentes, seus diversos tipos e arquiteturas, além das características de agentes em geral. Tal estudo é importante pois fundamenta as escolhas da arquitetura e linguagens adotadas para implementar as extensões aqui propostas. Em seguida é apresentado o *framework Jason*, detalhando sua arquitetura, funcionamento e sua atual capacidade no tratamento do conhecimento incerto.

Na seção 2.3 são estudados os principais conceitos sobre redes probabilísticas, conhecimento incerto e redes bayesianas, além de serem apresentadas alternativas para representação da incerteza. É dado enfoque especial à tecnologia de redes bayesianas, pois é deste modelo de representação do conhecimento que obtemos as informações de probabilidades utilizadas na implementação deste trabalho.

No final deste capítulo são apresentadas as considerações a respeito dos estudos realizados.

2.1 Sistemas MultiAgentes

Os Sistemas MultiAgentes (SMAs) compõe uma área de pesquisa contida na Inteligência Artificial Distribuída, que compreende questões relativas à computação distribuída em sistemas de inteligência artificial. Seu estudo baseia-se em uma sociedade como um todo, não em um único indivíduo. Assim, o maior foco desta área é a comunicação e interação entre as entidades do ambiente. Segundo BORDINI; VIEIRA; MOREIRA (2001), outro grande desafio é a especificação interna de um agente, em que tipicamente deseja-se uma representação simbólica daquilo que o agente sabe sobre o ambiente (e sobre os outros agentes naquele ambiente), bem como daquilo que o agente pretende atingir.

Para LESSER (1999), um SMA é um sistema computacional em que dois ou mais agentes interagem ou trabalham em conjunto de forma a desempenhar determinadas tarefas ou satisfazer um conjunto de objetivos. A investigação científica e a implementação prática de SMA estão focadas na construção de padrões, princípios e modelos que permitam a criação de pequenas e grandes sociedades de agentes semi-autônomos, capazes de interagir convenientemente de forma a atingirem seus objetivos.

O estudo de SMA envolve diversas áreas do conhecimento, como psicologia, ciência cognitiva, sociologia, entomologia, economia, teoria das organizações, entre outras. Talvez a área que mais tenha contribuído para o avanço de sistemas de inteligência artificial tradicionais (monolíticos) para Sistemas MultiAgentes seja a sociologia. Este fato é explicado pela presença da corrente filosófica denominada pragmatismo filosófico, que está em todo o trabalho contemporâneo das ciências sociais, e que concebe toda atividade

humana, inclusive a cognição, como uma prática social, o que somente ocorre em um processo de interação entre indivíduos (LAVE, 1988 apud BORDINI; VIEIRA; MOREIRA, 2001).

2.1.1 Agentes

O termo *agente* possui diversas definições na literatura computacional. Entre as mais aceitas estão as de WOOLDRIDGE; JENNINGS (1995) e RUSSELL; NORVIG (2009). Para WOOLDRIDGE; JENNINGS (1995), agentes são sistemas que apresentam um comportamento determinado por um processo de raciocínio baseado na representação de suas atitudes, tais como crenças, comprometimentos e desejos. Eles afirmam que um sistema pode ser visto como um agente se possuir as seguintes propriedades:

Autonomia: capacidade de o agente poder funcionar sem intervenção humana, baseando suas ações em seu conhecimento armazenado sobre o ambiente;

Habilidade Social: capacidade de interação com outros agentes através de uma linguagem comum;

Reatividade: habilidade de notar mudanças em seu ambiente e atuar de acordo com estas mudanças;

Pró-Atividade: o agente não deve atuar apenas por percepção, mas sim procurar alcançar uma meta, apresentando iniciativa.

Ainda segundo WOOLDRIDGE; JENNINGS (1995), o fato de pesquisadores da área não chegarem a um consenso não representa um problema, pois se muitas pesquisas da área de SMA são bem sucedidas, fica evidente a falta de necessidade na definição da terminologia. Entretanto, abre a possibilidade do uso inadequado do termo. Para resolver este impasse, os autores propuseram duas noções de agentes: uma fraca e uma forte.

A noção fraca é relativamente livre de problemas, sendo a forma mais comum na qual o termo é utilizado. É utilizada para caracterizar um sistema de hardware ou software dotado das propriedades de autonomia, habilidade social, reatividade e pró-atividade. Já a noção forte caracteriza entidades que além das propriedades acima, possuem características ou conceitos aplicados usualmente em seres humanos, tais como crenças, conhecimento, intenção, emoção e uma interface que representa visualmente o estado do agente.

A figura 2.1 apresenta uma visão geral do modelo de agente proposto em WOOLDRIDGE; JENNINGS (1995), apresentando de forma simplificada todos os aspectos de agentes. Nela, podemos ver de uma forma geral a execução interna de um agente. Através de suas percepções, o próximo estado do ambiente observado pelo agente é atualizado e, logo após, uma ação é escolhida para execução. Em seguida, este ciclo é reiniciado.

O termo *agente* tornou-se muito popular na área de Ciência da Computação após a segunda metade da década de 90. Nesta perspectiva, o termo *agentes de software* (GENESERETH; KETCHPEL, 1994 apud BORDINI; VIEIRA; MOREIRA, 2001) foi criado, definindo praticamente qualquer processo comunicante como um agente. Em RUSSELL; NORVIG (2009), o termo *agente* é definido como algo com capacidade de atuar através de sua percepção em um dado ambiente. Para RUSSELL; NORVIG (2009), a inteligência está altamente ligada a ações racionais. Agir de forma racional significa agir para alcançar as metas definidas por alguém, dadas as crenças deste.

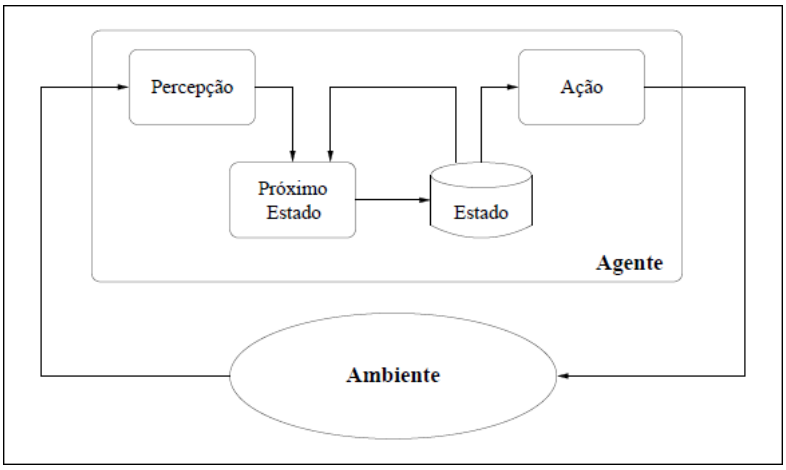

Figura 2.1: Modelo geral de agente (traduzido).

Já para MAES (1995), agentes são sistemas computacionais residentes em ambientes dinâmicos complexos, os quais percebem e atuam autonomamente, realizando um conjunto de objetivos ou tarefas para os quais foram designados.

DEMAZEAU (1995) denomina agente como cada uma das entidades ativas em um dado ambiente, formando uma sociedade. Um agente recebe informações e raciocina sobre o ambiente, sobre outros agentes e decide quais ações deve realizar e quais objetivos deve seguir. Um agente também é capaz de controlar suas ações, diferentemente de noções estáticas como módulos, conjunto de regras e bases de conhecimentos.

2.1.2 Tipos de SMA

Existem dois principais tipos de Sistemas MultiAgentes: os reativos (FERBER, 1999 apud BORDINI; VIEIRA; MOREIRA, 2001) e os cognitivos. Em um SMA reativo, a ideia de inteligência não está no agente, mas na interação entre um grande número de agentes. Os integrantes deste SMA são muito simples e não possuem nenhuma representação do estado do ambiente onde estão atuando, nem dos outros agentes e tampouco de ações passadas. Seu comportamento pode ser descrito como um autômato finito simples, baseando-se em um esquema estímulo-resposta, como em uma colônia de insetos, por exemplo.

Um SMA cognitivo possui relativamente poucos agentes, os quais são sistemas complexos, podendo possuir algumas das seguintes características:

Percepção: o agente tem de ser capaz de perceber mudanças no ambiente, muitas vezes geradas por ações de outros agentes;

Ação: um agente age constantemente com o intuito de modificar o estado atual do ambiente em outro estado onde acredite que seu objetivo é satisfeito;

Comunicação: uma das possíveis ações dos agentes cognitivos é comunicar-se com outros presentes no mesmo ambiente, o que pode ser essencial para atingir a coordenação entre os mesmos;

Representação: o agente possui uma visão do que acredita ser verdade sobre o ambiente e sobre os outros agentes que compartilham o mesmo ambiente;

Motivação: para o agente agir por iniciativa própria, é necessário existir, além da representação do conhecimento do agente, uma representação de aspectos motivacionais, como desejos ou objetivos, ou seja, uma representação de estados do ambiente que o agente deseja atingir;

Deliberação: o agente deve ser capaz de decidir (deliberar) quais estados do ambiente em que este se encontra serão os objetivos a serem seguidos, dentre aqueles possíveis de ocorrerem no futuro, dada uma motivação e uma representação do estado atual;

Raciocínio e Aprendizagem: pode-se aumentar o desempenho e complexidade dos agentes através da extensão de técnicas de inteligência artificial clássica para, por exemplo, raciocínio e aprendizagem para múltiplos agentes.

2.1.3 Arquitetura BDI

Diversas arquiteturas de agentes deliberativos são baseadas na utilização de três principais estados mentais: crenças, desejos e intenções, abreviados por BDI, do inglês *beliefs*, *desires* e *intentions*, respectivamente. Crenças são visões que o agente possui sobre o ambiente em que ele situa-se. É o conhecimento do ambiente de forma explícita, podendo ser incompleto ou até mesmo incorreto (BRATMAN; ISRAEL; POLLACK, 1988). Em GIRAFFA; VICARI (1998), o termo *desejo* é definido como uma noção abstrata, que especifica as preferências do agente em relação aos estados futuros do mundo ou o curso das ações que o agente, possivelmente, quer que se verifiquem. Objetivo é um subconjunto dos desejos que são compatíveis entre si. Para MÓRA et al. (1999), uma intenção é como um compromisso que o agente assume para atingir um determinado objetivo, pois ao contrário dos desejos, não se pode ter intenções conflitantes, devendo buscar suporte nas crenças do agente, isto é, o agente não pode intencionar o que não acredita.

A arquitetura BDI genérica pode ser apresentada de forma esquemática como na figura 2.2. Nela, podemos ver que o agente executa quatro principais ações, descritas a seguir.

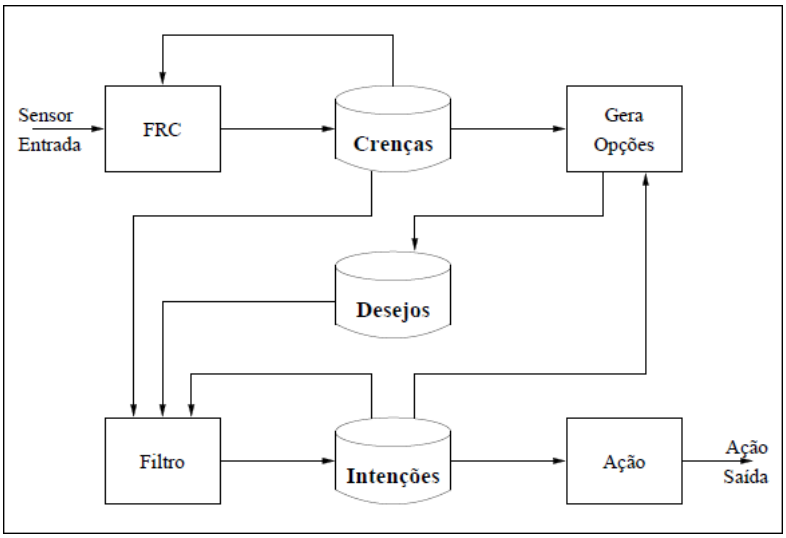

Figura 2.2: Modelo geral de arquitetura BDI (traduzido). Fonte: WOOLDRIDGE (1999)

1. Função de Revisão de Crenças (FRC): percebe alterações no ambiente e atualiza sua base de crenças para que reflitam o novo estado agente; assim, novas opções de estados a serem atingidos podem vir a ser disponíveis;

2. Gera Opções: dadas as intenções com as quais o agente já está comprometido, verifica quais as novas alternativas de opções a serem realizadas, deliberando com qual entre estas novas opções o agente se comprometerá e atualizando seus desejos. Agora é preciso definir qual curso de ações será seguido para atingir os objetivos atuais do agente, levando-se em consideração os outros cursos de ação com que o agente já se comprometeu, a fim de evitar incoerências;

3. Filtro: atualiza o conjunto de intenções do agente, com base nas crenças e desejos já atualizados e nas intenções existentes;

4. Ação: o agente escolhe qual ação específica deve ser executada.

O modelo BDI é especialmente interessante, pois combina três componentes distintos (WOOLDRIDGE, 2000):

Componente Filosófico: a fundamentação filosófica para o modelo BDI vem do trabalho de DENNETT (1987) sobre sistemas intencionais e de BRATMAN (1987) sobre raciocínio prático;

Componente de Arquitetura de Software: o modelo BDI não exige uma implementação específica, podendo ser estendido de diversas maneiras diferentes;

Componente Lógico: um grupo de lógicas captura os aspectos chaves do modelo BDI como um conjunto de axiomas lógicos.

2.1.4 Linguagens para desenvolvimento de agentes

Entende-se que linguagens para desenvolvimento de agentes são ferramentas que permitem a alguém programar um sistema de hardware ou software em termos de algumas concepções desenvolvidas por uma teoria de agentes. Neste trabalho, abordaremos duas das principais linguagens utilizadas por grande parte das ferramentas atualmente desenvolvidas: AgentSpeak(L) e 3APL.

2.1.4.1 AgentSpeak(L)

Esta linguagem estende Prolog para suportar a programação baseada em agentes. Introduzida em RAO (1996), AgentSpeak(L) disponibiliza vários complementos utilizados na construção de agentes, tais como satisfação de predicados, diferentes tipos de desejos; distinção entre objetivos normais e especiais relevantes para o modelo BDI e altera o mecanismo de resolução, formando e abandonando planos em função das reações internas do agente chamados eventos geradores.

BORDINI et al. (2002) ainda estenderam esta linguagem, adicionando ainda mais poder para a programação de agentes. A nova linguagem criada com esta extensão foi batizada de *AgentSpeak(XL) (eXtended Language)*. Mais detalhes sobre AgentSpeak(XL) constam na seção 2.2.4.

2.1.4.2 3APL

3APL (*An Abstract Agent Programming Language*) é uma linguagem de programação para implementar agentes cognitivos que oferece meios para construção de crenças, objetivos, capacidades básicas (tais como atualização de crenças, ações externas, ou ações de comunicação) de agentes e um conjunto de regras de raciocínio prático através do qual

os objetivos do agente podem ser atualizados ou revistos. Rodando em uma plataforma própria, um programa em 3APL é executado por meio de um interpretador que delibera sobre as atitudes cognitivas dos agentes (DASTANI; RIEMSDIJK; MEYER, 2005).

Segundo os autores deste trabalho, 3APL possibilita *backtracking* e facilita a aplicação dos diversos aspectos cognitivos de agentes. Um ambiente compartilhado pode ser implementado na linguagem de programação Java, construído como uma classe. Seus métodos corresponderão às ações que um agente pode executar no ambiente. Além de interagir com o ambiente, os agentes podem interagir entre si, através das especificações propostas pela FIPA.

Ainda segundo DASTANI; RIEMSDIJK; MEYER (2005), a plataforma 3APL tem uma interface visual para o monitoramento e depuração dos agentes que estão sendo executados e um editor de sintaxe para a coloração de edição de código-fonte. Lançado como um software baseado em Java, contém algumas interfaces que podem ser usadas para desenvolver *plugins* e bibliotecas. Uma plataforma 3APL também pode conectar-se como cliente ou servidor para outras plataformas 3APL em uma rede, permitindo a comunicação entre os agentes 3APL em cada plataforma.

2.1.5 Comunicação em Sistemas MultiAgentes

A comunicação entre agentes em uma sociedade é muito importante, em geral. Quando os indivíduos devem realizar ações de coordenação, cooperação ou competição (negociação), a comunicação ganha extrema importância. Quanto mais eficiente o processo de comunicação, mais rapidamente os agentes poderão atingir seus objetivos.

2.1.5.1 Atos de Fala

Os estudos da comunicação em SMA são baseados em trabalhos da filosofia da linguagem, especialmente na teoria dos atos de fala, desenvolvida em contraposição aos trabalhos de semântica lógica, os quais se preocupam apenas com questões relacionadas aos valores e condições de verdade da linguagem.

A teoria dos atos de fala afirma que nem todas as sentenças podem ser analisadas em termos de condição de verdade. Não faz sentido algum atribuir um valor de verdade à sentença “Abra a porta”, por exemplo. Poderíamos afirmar que a sentença é válida e que foi pronunciada de maneira apropriada. Uma pessoa com o desejo de a porta ser aberta está realizando uma ação imperativa. Caso não exista porta alguma, a sentença perde o sentido. Esta ideia também vale para outras sentenças, como promessas, perguntas, asserções, requisições, sugestões e respostas. Assim, estas sentenças que não podem ser interpretadas simplesmente como verdadeiras ou falsas são denominadas *performativas*.

Quando um agente expressa um conteúdo semântico através da preposição *abrir(porta)*, as seguintes sentenças podem ser pretendidas:

- Asserção: a porta foi aberta;
- Pergunta: a porta foi aberta?;
- Ordem: abra a porta;
- Expressão de um desejo: eu quero que a porta seja aberta;
- Expressão hipotética de intenção: Se a porta for aberta, eu realizarei minha missão.

A diferença no uso do *conteúdo proposicional* (no exemplo, *abrir(porta)*), é determinada pela *força ilocucionária* em cada sentença (SEARLE, 1969 apud BORDINI; VIEIRA; MOREIRA, 2001). A teoria dos atos de fala ainda propõe algumas outras propriedades. *Locução* é o enunciado emitido pelo falante através de algum meio físico, por exemplo, fala ou escrita. *Ilocução* é a intenção associada ao enunciado do falante. *Perlocução* é a ação resultante ou efeito da locução.

Na comunicação entre agentes, a força ilocucionária faz-se presente de forma mais explícita do que na comunicação humana, definindo claramente a intenção do agente emissor. Assim, o receptor da mensagem não tem dúvidas de como tratá-la. Esta clareza simplifica o processo de construção de agentes comunicantes, sendo utilizada como base para implementação de protocolos de comunicação entre agentes.

2.1.6 Linguagens de Comunicação entre Agentes

As Linguagens de Comunicação de Agentes (*Agent Communication Language - ACL*) foram concebidas para padronizar a interação entre agentes, permitindo a interoperabilidade entre sistemas de vários agentes. Existem três principais esforços visando a padronização de sistemas baseados em agentes: MASIF (não abordado neste trabalho pois é menos utilizado), KQML e FIPA.

KQML (*Knowledge Query Manipulation Language*) (FININ et al., 1994) foi uma das primeiras linguagens baseadas na teoria dos atos de fala a especificar um suporte à interação social entre agentes em um SMA. Mesmo sendo uma das ACL mais importantes, não é considerada um padrão *de facto*, uma vez que não existe consenso na comunidade em uma especificação única. Assim, surgiram diversas variações da KQML, comprometendo a interoperabilidade entre SMAs que utilizem diferentes dialetos. A linguagem é formada por um conjunto de *performativas* que determinam o objetivo do agente remetente da mensagem, referindo às *forças ilocucionárias*. Por exemplo, uma afirmação tem a performativa $tell$ e objetiva alterar as *crenças* do agente destinatário.

O formato básico de uma mensagem KQML é conforme segue abaixo:

```
(performativa    :sender    <word>
                 :receiver  <word>
                 :language  <word>
                 :ontology  <word>
                 :content   <expression>)
```

A linguagem de comunicação entre agentes proposta pela FIPA (*Foundation for Intelligent Physical Agents*) (FIPA, 2003) é baseada em KQML para permitir a migração de sistemas que já utilizavam esta ACL. A FIPA é uma organização sem fins lucrativos estabelecida na cidade de Geneva, Suíça, em 1996. Seu grande mérito é ter sido a primeira linguagem de fato implementada e utilizada em diversos sistemas, tendo uma abordagem própria para comunicação entre agentes, baseada na teoria dos atos de fala. Além disso, as especificações da FIPA vêm sendo constantemente atualizadas e melhoradas.

2.1.7 JADE

JADE (*Java Agent Development framework*), descrito em BELLIFEMINE; CAIRE; GREENWOOD (2007), é um ambiente *open source* para desenvolvimento de aplicações baseada em agentes conforme as especificações da FIPA (*Foundation for Intelligent Physical Agents*) para interoperabilidade entre Sistemas MultiAgentes, totalmente implemen-

tado em Java. Desenvolvido e suportado pelo CSELT (*Centro Studi E Laboratori Telecomunicazioni*, ou Centro de Estudo e Laboratórios de Telecomunicações em tradução livre para o português) da Universidade de Parma, seu principal objetivo é simplificar e facilitar o desenvolvimento de SMAs, garantindo um padrão de interoperabilidade através de um abrangente conjunto de agentes de serviços de sistema, os quais tanto facilitam como possibilitam a comunicação entre agentes, de acordo com as especificações da FIPA: serviço de nomes (*naming service*) e páginas amarelas (*yellow-page service*), transporte de mensagens, serviços de codificação e decodificação de mensagens e uma biblioteca de protocolos de interação (padrão FIPA).

A complexidade do ciclo de vida dos agentes, além do transporte, codificação e interpretação de mensagens são gerenciados pelo JADE, possibilitando que o programador se preocupar somente com os aspectos do sistema em si. Assim, pode ser considerado um *middleware* de agentes que implementa um *framework* de desenvolvimento e uma plataforma de agentes.

De acordo com BELLIFEMINE et al. (2005), JADE foi escrito em Java devido a características particulares da linguagem particularmente pela programação orientada a objeto em ambientes distribuídos heterogêneos. Foram desenvolvidos tanto pacotes Java com funcionalidades prontas pra uso quanto interfaces abstratas para se adaptar de acordo com a funcionalidade da aplicação de agentes.

JADE também permite o encapsulamento de diversos agentes em estruturas chamadas de pacotes (*containers*), ilustrado na figura 2.3, sendo que um ou mais pacotes podem estar dentro de uma plataforma (*plataform*). Esta organização pode ser importante neste projeto, pois podemos estruturar agentes com diferentes objetivos em um pacote ou plataforma.

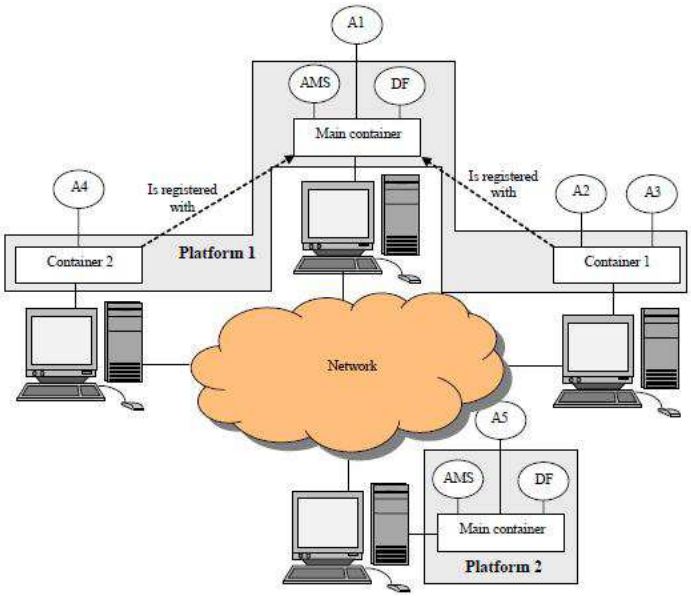

Figura 2.3: Exemplo de uma arquitetura no JADE.

Ontologias podem ser utilizadas para identificar uma determinada mensagem. Assim, somente os agentes aptos a tratarem mensagens com este tipo de ontologia irão efetivamente recebê-la. A ontologia, neste caso, é do tipo $String$, não tendo qualquer padrão nem obedecendo a regras formais.

Os agentes AMS (*Agent Management System*) e DF (*Directory Facilitator*) são responsáveis por diversos serviços oferecidos pelo JADE. Somente pode existir um agente

de cada tipo no sistema. O AMS provê serviços de registro de agentes e ciclo de vida, mantendo um diretório de identificadores de agentes e estado do agentes. O DF fornece o serviço de páginas amarelas.

As ações a serem executadas pelo agente dentro do SMA definem seu comportamento, sendo fundamental para o cumprimento de seus objetivos. No JADE, o programador deve obrigatoriamente herdar a classe $jade.core.Agent$, implementar métodos específicos e adicionar um ou mais comportamentos (classe $Behaviour$ ou suas subclasses).

Para iniciar um agente, o seguinte comando deve ser executado:

```
java jade.Boot [options] [agentList]
  [options]
    -container
    -host hostName
    -port portNumber
    -gui
    ...
  [agentList]
    agenteName:className
```

onde $container$ é o pacote em que o agente deve ser inserido, $host$ se refere ao endereço IP (ou nome) da máquina onde o agente deve hospedar-se (o padrão é máquina local), $port$ é a porta a ser utilizada (a padrão é 1099) e gui (*Graphic User Interface*) indica se a interface gráfica do JADE deve ser carregada ou não. $agentList$ é a lista de agentes que farão parte deste pacote, separados por espaço. Após a execução da linha de comando acima, o agente deve cumprir uma série de etapas. O início da execução do agente é compreendido de três principais passos:

1. O construtor do agente é executado;
2. Um identificador é atribuído ao agente, que é registrado no AMS, seguindo o seguinte padrão: $agentName$ + "@" + $hostName$ + ":" + $portNumber$ + "/JADE";
3. O método $setup$ é executado. Nele, devem constar o registro do agente em um ou mais domínios e a adição de um ou mais comportamentos.

2.1.8 Ferramentas

Entre as diversas ferramentas para o desenvolvimento de agentes cognitivos BDI, neste trabalho serão abordadas quatro: *dMARS*, *Jason*, *Jack* e *LORA*. Estas ferramentas foram selecionadas após uma revisão bibliográfica e aqui descritas por permitirem o desenvolvimento de agentes e fornecerem uma plataforma de execução distribuída ou centralizada. Ainda existem muitas outras ferramentas nesta mesma área, como JADEX e ZEUS, não abordadas aqui.

2.1.8.1 dMARS

dMARS (*Distributed Multi-Agent Reasoning System*) (D'INVERNO et al., 1998) é um ambiente de implementação e desenvolvimento orientado a agente para construção de sistemas complexos, distribuídos e de alto desempenho. Projetado para configuração rápida e facilidade de integração, facilita a concepção, manutenção e re-engenharia do sistema. Sua construção está baseada no Sistema de Raciocínio Procedural, ou em inglês,

Procedural Reasoning System (PRS), desenvolvido pela SRI International, no qual foi baseado na primeira implementação de um sistema BDI, o IRMA (*Intelligent Resource-bounded Machine Architecture*) (BRATMAN; ISRAEL; POLLACK, 1988).

O PRS é um exemplo clássico de uma arquitetura de agente, na qual o mesmo é constituído por planos (programa) dentro de uma arquitetura. O programador desenvolve o agente conforme ele quer que o agente se comporte, entretanto muitas ações não são definidas pelos seus planos, e sim pela arquitetura. Um exemplo é a base de crenças. Embora o desenvolvedor possa customizá-la, não é necessário preocupar-se com sua programação.

O modelo BDI é estruturado em agentes dMARS através de planos. Cada agente tem uma biblioteca de planos especificando o curso de ação que pode ser tomado pelo agente para realizar suas intenções. Uma biblioteca de planos de um agente representa seu conhecimento procedural ou seu conhecimento sobre como fazer. Segundo D'INVERNO et al. (1998), cada plano contém vários componentes:

Trigger: rótulo (nome) do plano. Por exemplo, o plano "apresentar trabalho" pode ser disparado pelo evento "apresentação";

Contexto ou pré-condições: especifica as circunstâncias sob as quais a execução de um plano pode iniciar. Por exemplo, o plano "apresentar trabalho" pode ter a pré-condição "ter trabalho pronto";

Condições de manutenção: um plano também pode ter condições de manutenção que caracterizam as circunstâncias que devem permanecer verdadeiras enquanto o plano é executado;

Corpo: Define um conjunto ordenado de ações a serem realizadas, para atingir seus objetivos (ou sub-objetivos). O plano "trabalho" pode ter o corpo "definir área"; "definir orientador"; "propor tema", etc. Nesse sentido, "definir orientador" é um sub-objetivo (passo prévio que deve ser realizado quando o processamento do plano alcança este ponto).

Agentes dMARS modelam suas especialidades como um conjunto de planos sensíveis ao contexto. Esses planos podem mudar o ambiente e seguir os objetivos do agente, permitindo que permaneçam direcionados a objetivos sem comprometer suas habilidades de reação a novas situações. Os agentes podem modificar seus comportamentos para conservar o alinhamento das mudanças ocorridas no ambiente em tempo real. Concorrência e cooperação são suportadas mesmo quando os agentes envolvidos são distribuídos pela rede. Além de raciocinar sobre seu ambiente, agentes dMARS são capazes de raciocinar sobre seu próprio estado mental, isto é, podem refletir sobre suas próprias crenças, metas e intenções.

2.1.8.2 JACK

A plataforma JACK (BUSETTA et al., 2000) é um *framework* desenvolvido em Java para desenvolvimento de Sistemas MultiAgentes. Comercializada pela empresa australiana *Agent Oriented Software Pty. Ltd.* (AOS), foi baseada nas experiências anteriores de PRS e dMARS (seção 2.1.8.1), utiliza o modelo BDI e provê uma linguagem e interface gráfica de planejamento próprias.

A linguagem fornecida por JACK permite a utilização de código Java dentro dos componentes, o que facilita a programação dos agentes, utilizando variáveis, cursores e estruturas de dados. Os cursores são particularmente úteis quando realizadas consultas à base

de crenças do agente. A semântica para estas consultas assemelha-se às linguagens de programação de lógica e SQL.

O núcleo da plataforma é um compilador multiagente em tempo de execução extensível. Quando agentes, planos, eventos e recursos são especificados, o próprio JACK faz o gerenciamento da execução do sistema, incluindo a passagem de mensagens, raciocínio e meta-raciocínio. O objeto de comunicação utilizado pelo JACK é o JACOB (JACK Object Modeller), onde objetos Java são serializados em formato ASCII, similar aos formatos YAML e XML.

2.1.8.3 LORA

LORA (*Logic Of Rational Agents*) (WOOLDRIDGE, 2000) é uma abordagem formal para a especificação de sistemas baseados em agentes, que inclui um componente BDI para a especificação da arquitetura de agentes, um componente temporal para especificar o comportamento do sistema e uma ação para representar as ações dos agentes. Este *framework* não é um modelo executável, nem existe uma maneira simples de automatizá-lo. Assim, não é possível utilizá-lo como um formalismo de representação de conhecimento ou uma linguagem de programação.

LORA também pode ser utilizado para definir o modelo de resolução de problemas de forma cooperativa, uma adaptação da teoria de cooperação introduzida em WOOLDRIDGE; JENNINGS (1995). Este modelo trata a resolução de problemas cooperativamente, dividindo em um processo de quatro etapas. Primeiramente, um agente identifica a capacidade para cooperação em uma de suas ações; é feita uma tentativa de solicitação de auxilio a um grupo de agentes, o qual tenta chegar a um acordo para a execução de um plano em conjunto para atingir este objetivo, que é então executado por todo o grupo.

2.1.8.4 Jason

A ferramenta de desenvolvimento de agentes *Jason* será amplamente apresentada e discutida na sessão 2.2. Sua escolha para o desenvolvimento dos agentes deste trabalho deu-se, principalmente pelo fato de ter código aberto, possibilitando sua extensão na forma de um *plugin*.

Outros fatores que contribuíram para esta decisão foram ter sido desenvolvido na mesma linguagem das outras ferramentas utilizadas neste trabalho, ter ampla documentação disponível na internet e já ter sido testado e utilizado por diversos programadores ao redor do mundo. Além disso, *Jason* permite comunicação entre agentes, possibilita o desenvolvimento e execução de uma plataforma de SMA, atendendo todas as necessidades do trabalho aqui desenvolvido.

2.2 O Ambiente de Desenvolvimento Jason

Neste capítulo, apresentaremos o ambiente de desenvolvimento de agentes Jason (BORDINI; HÜBNER; WOOLDRIDGE, 2007), abordando seus pontos fortes e fracos, alternativas de manuseio do conhecimento incerto, sua arquitetura e principais aplicações. Também discutiremos como a inserção do conhecimento probabilístico pode melhorar o processo de escolha de planos, sempre executando aquele com as crenças de maior probabilidade. A apresentação e discussão da implementação realizada serão amplamente discutidas no capítulo 3.

Jason é um software livre e aberto, o qual pode ser redistribuído ou modificado sob os termos da licença GNU Lesser General Public License. É distribuído na forma de um

plugin sob a forma do IDE (*Integrated Development Environment*) Jedit[1], sendo assim um ambiente para a configuração, codificação e execução de um SMA.

O Jason[2] (sigla do inglês *A Java-based AgentSpeak Interpreter Used with Saci For Multi-Agent Distribution Over the Net*) é um *framework* desenvolvido na linguagem Java para desenvolvimento de agentes escritos na linguagem AgentSpeak(XL). Um de seus principais usos é no desenvolvimento de SMAs, podendo ser executado em diversos computadores simultaneamente, utilizando arquiteturas como SACI, JADE ou Moise+ (HÜBNER; SICHMAN; BOISSIER, 2004).

Existem três principais construtores a serem utilizados: crenças, objetivos e planos. Existem vários outros construtores, portanto apresentaremos em detalhes somente os principais, mostrando os outros em exemplos ao longo deste capítulo.

2.2.1 Crenças

Para armazenar todas as crenças do agente, o Jason possui uma estrutura chamada "base de crenças" (*belief base*), que nada mais é que uma coleção de literais. Um literal é um predicado sobre um estado do ambiente ou sua negação. Como em linguagens de programação lógicas, informações são representadas na forma simbólica por *predicatos*, como em

```
ceu(azul).
```

representando que o agente acredita que o objeto ceu tem a propriedade de ser $azul$. Também podemos definir uma relação entre dois ou mais objetos, como por exemplo

```
gosta(maria, flores)
```

indicando que o agente acredita que o objeto $maria$ relaciona-se com o objeto $flores$ através da relação $gosta$.

As crenças de cada agente referem-se ao que o agente acredita que seja verdade dentro do ambiente. Como no exemplo mostrado acima, pode ser que a Maria não goste de flores, apesar de o agente acreditar o contrário.

Para mais detalhes sobre expressões lógicas e suas utilizações no Jason, é aconselhado pesquisar na literatura já existente, como em BORDINI; HÜBNER; WOOLDRIDGE (2007), CLOCKSIN; MELLISH (1987), BRATKO (1986) e STERLING; SHAPIRO (1994).

2.2.1.1 Anotações

O Jason possibilita a utilização de um termo complexo que está fortemente associado às crenças chamado de *anotações*. Ele pode ser utilizado para armazenar qualquer conhecimento adicional correspondente àquela crença. Neste trabalho, somente adicionaremos o conhecimento de probabilidade referente à crença, identificado pela anotação p. Essa anotação sempre estará associada a um número entre 0 e 1, no qual corresponde a probabilidade em que o agente acredita que a variável encontra-se no estado atual. Seu uso é notado através de colchetes logo após a crença, como em $chuva(sim)[p(0.9)]$, representando que o agente acredita que a probabilidade de chuva é de 90%.

No exemplo acima, a anotação p não tem significado para o interpretador do Jason. Entretanto, existem algumas que apresentam significado importante, como *informação de*

[1] disponível em `http://www.jedit.org`
[2] disponível em `http://jason.sourceforge.net`

percepção (quando o agente adquire o conhecimento sentindo o ambiente, simbolizado por $percept$), *comunicação* (ao comunicar-se com outros agentes, este precisa saber qual agente forneceu tal informação) e *notas mentais* (utilizado para o agente lembrar-se de ações ou informações úteis ocorridas no passado que serão utilizadas posteriormente).

A utilização de anotações não aumenta o poder de expressão da linguagem. Poderíamos representar esta mesma informação na forma $chuva(sim, 0.9)$. Entretanto, a utilização de anotações apresenta duas principais vantagens:

1. *Legibilidade do código*: todos os detalhes referentes àquela crença são organizados de tal forma que torna a base de crenças mais legível;

2. *Gerenciamento da base de crenças*: possivelmente reduz o número de crenças necessárias para representar informações sobre um mesmo objeto, além de explicitamente conectar os detalhes de um objeto com sua respectiva crença.

As anotações são armazenadas juntamente com as crenças. Portanto, ao se recuperar uma crença, pode-se também recuperar todas as anotações associadas a ela. Caso se queira substituir a probabilidade desta crença, pode-se fazê-lo atribuindo um novo valor a ela, assim como feito da primeira vez. Caso se queira adicionar, ao mesmo tempo, outra crença além da probabilística, a adicionamos dentro dos mesmos colchetes, separando as duas anotações por vírgula.

2.2.1.2 Negação Forte

Existem duas abordagens para manipular negação em programação lógica: *suposição de mundo fechado* e *negação por falha*. A primeira afirma que tudo aquilo que nem é conhecido como verdade nem derivável de fatos conhecidos, é assumido como falso. Esta suposição é muito simplista e forte para muitas aplicações, pois nem tudo que é desconhecido pelo agente é necessariamente falso. A segunda abordagem acontece quando o interpretador falha ao derivar a fórmula utilizando os fatos e regras no programa. O único operador que pode ser utilizado neste contexto é o "not", retornando verdadeiro se o interpretador falhar na derivação da fórmula.

Ainda existe outro tipo de negação, através da utilização do operador "$\sim$", chamado de *negação forte*. É utilizado quando o agente crê algo ser falso, como por exemplo, em $tempo(ensolarado)$ significa que ele acredita o tempo estar ensolarado, ao contrário de $\sim tempo(chuvoso)$, que indica o agente crer o tempo *não* estar chuvoso.

2.2.1.3 Regras

A noção de *regras* em Prolog é muito parecida em AgentSpeak, onde podemos chegar a conclusões a partir do conhecimento já existente. Utilizando-as na base de crenças, tornamos o contexto de planos (apresentado na seção 2.2.3) mais sucinto. Por exemplo:

```
chuva(sim)[p(X)] :- umidade(alta)[p(Y)]
                  & pressao(alta)[p(Z)]
                  & X = Y*0.4 + Z*0.6 .
```

Neste exemplo, a probabilidade é definida em tempo real em função da probabilidade das outras variáveis, acreditando que a probabilidade da variável $chuva$ atingir o estado sim varia conforme a probabilidade das crenças $umidade$ e $pressao$, ambas no estado $alta$. À esquerda do operador "$:-$" pode haver somente um literal, que é a conclusão feita caso a condição à direita for satisfeita, de acordo com a base de crenças atual.

2.2.2 Objetivos

O conjunto de objetivos de um agente é uma das partes mais importantes em sua construção, pois determina o que o agente fará. Enquanto as crenças expressam o que o agente acredita ser verdade no momento, objetivos indicam estados do ambiente que ele deseja atingir. Dado um objetivo g, o agente compromete-se alterar o estado do ambiente até atingir o estado em que acredita que g é verdadeiro. Este uso de objetivos é chamado de *objetivo declarativo*.

Em AgentSpeak, existem dois tipos de objetivo: *objetivo de teste* e *objetivo de realização*. Existe uma pequena diferença entre um objetivo em Prolog e em AgentSpeak. Em Prolog, um objetivo é basicamente uma "pergunta" que o agente faz à base de dados a fim de recuperar alguma informação. Como o uso deste tipo de objetivo também é importante em AgentSpeak, este tipo de objetivo foi chamado de objetivo de teste, sendo utilizado através do operador "$?$". Caso a crença $contaBancaria(500)$ estivesse presente na base de crenças do agente e tivéssemos o objetivo $?contaBancaria(X)$ a variável X seria unificada com o valor 500.

Objetivos de realização são denotados pelo uso do operador "!". Ao utilizá-lo, o agente se compromete a atingir um estado onde ele acredita que aquele objetivo seja verdadeiro. Para que isso aconteça, o agente executa planos que ele acredita ser necessários para tal objetivo ser alcançado.

2.2.3 Planos

Como o agente está constantemente observando seu ambiente, a cada ciclo de raciocínio será escolhido um novo plano a ser executado, a fim de atingir seus objetivos, podendo alterar seu ambiente. Os planos do agente estão armazenados em uma *biblioteca de planos*, inicialmente formada pelos planos que o próprio programador escreveu em sintaxe AgentSpeak.

Um plano tem basicamente três partes: evento de disparo ($triggering\_event$), contexto ($context$) e seu corpo ($body$). As duas primeiras partes juntas são chamadas de cabeçalho ($head$). Os marcadores "$:$" e "$<-$" são utilizados para identificá-los, como mostra o exemplo abaixo:

```
triggering_event : context <- body
```

2.2.3.1 Eventos de Disparo (triggering_event)

Os eventos de disparo são utilizados pelo agente para pesquisar em sua base de planos quais planos podem ser disparados dado certo evento. Detalharemos como esta pesquisa é feita na seção 2.2.5. O plano que coincidir com um evento em particular é dito *relevante* para este. Em um ciclo, um ou mais planos podem ser relevantes. Caso não haja qualquer plano relevante, um erro é gerado e a execução do agente interrompida.

Um agente pode ter dois tipos de atitudes mentais: crenças e objetivos, no qual se divide em objetivos de teste e de realização. Eventos podem fazer modificações tanto de remoção quanto de adição. Eventos de adição e remoção de crenças ocorrem quando um agente atualiza sua base de crenças, por exemplo. Eventos de adição de objetivos acontecem, na maior parte das vezes em consequência da execução de outros planos, ou pela comunicação com outro agente.

2.2.3.2 Contexto (context)

O contexto do plano é utilizado para avaliar a situação atual do ambiente, a fim de analisar se a execução do plano é possível no momento ou não. Caso a análise do contexto indique $true$, o plano é dito *aplicável*, sendo um candidato para execução.

Um contexto é uma combinação de *literais padrão* e expressões relacionais, as quais são muito parecidas com a sintaxe de Prolog. Um literal padrão é um literal que pode utilizar o operador not para negação, conhecido como *negação padrão*. Portando, existem quatro tipos de literais possíveis em um contexto: "$\&$", "$|$", "not" e "$\sim$". O contexto pode ser formado por expressões lógicas combinando estes literais através dos operadores *and* (conjunção) denotado por "$\&$", *or* (disjunção), por "$|$" ou ainda a negação como operador de falha not. Esta mesma sintaxe pode ser utilizada nas regras presentes na base de crenças.

É importante ressaltar que não acreditar que l é falso (isto é, $not \sim l$) é diferente de acreditar que l é verdadeiro, pois o agente pode simplesmente ser ignorante a respeito de l, ou seja, ele não tem qualquer informação sobre l. O exemplo abaixo ressalta esta diferença.

```
+!jogar(Futebol)
    :  (not ~tempo(ensolarado)) & jogadores(J) & (J >= 11)
    <- !marcarJogo.
```

Quando o agente tiver o objetivo de jogar futebol, um novo evento será criado, pois existe um novo objetivo a ser atingido. Caso o agente acredite que tempo está ensolarado (ou não tenha qualquer informação a respeito) e o número de jogadores é maior ou igual a onze, este plano poderá ser executado para manipular tal evento, adicionando o objetivo $marcarJogo$ ao seu conjunto de objetivos. Note que o valor da crença $jogadores$ na base de crenças foi associado à variável J, assim foi possível realizar o teste para verificar se existia o número de jogadores.

2.2.3.3 Corpo (body)

O corpo do plano é uma sequência de fórmulas, separadas por ponto-e-vírgula ("$;$"), que determinam as ações a serem executadas para que o agente atinja o objetivo proposto pelo plano em que este corpo está. Existem seis tipos de fórmulas que podem ser utilizadas aqui: *ações* (funções nativas do Jason), *objetivos de realização*, *objetivos de teste*, *notas mentais*, *ações internas* (definidas pelo usuário) e *expressões* ($X \geq Y + 1$, por exemplo). Outros objetivos podem aparecer dentro do corpo, sendo chamados de *subobjetivos*.

Ações internas são funções que o usuário pode construir e utilizar no código do agente. Obrigatoriamente devem implementar a interface $InternalAction$, porém é aconselhado utilizar a classe $DefaultInternalAction$, que facilita sua criação. Ações internas devem sobrescrever o método $execute$, a ser executado pelo interpretador do Jason quando esta ação é invocada. O primeiro parâmetro deste método contém todas informações sobre o estado atual do agente. O segundo é do tipo $Unifier$, que pode ser utilizado para unificar o valor de retorno deste método com uma variável passada como parâmetro. O terceiro parâmetro é uma lista de termos que contém os argumentos passados pelo programador desta ação interna.

2.2.4 AgentSpeak(XL)

Como já descrito na seção 2.1.4.1, a linguagem AgentSpeak(L) foi introduzida em RAO (1996), com o objetivo de possibilitar a construção de agentes na arquitetura BDI através da especificação de um conjunto de crenças e um conjunto de planos. Segundo HÜBNER; BORDINI; VIEIRA (2004), as informações dos desejos, assim como as alternativas disponíveis do agente para ativar as intenções estão implicitamente representadas nos planos.

AgentSpeak(XL), definida em BORDINI et al. (2002), estende AgentSpeak(L) através do acoplamento de novas funcionalidades, como mecanismos de seleção de intenções e gerador de planos em tempo real.

2.2.5 O Interpretador do Jason

Até agora foram detalhadas todas as características e propriedades dos agentes desenvolvidos em AgentSpeak. Nesta seção, é explicado como o Jason coloca os agentes em execução. Um agente opera segundo um *ciclo de raciocínio*, o qual pode ser dividido em dez etapas. A figura 2.4 apresenta uma visão geral da arquitetura de um agente programado em Jason.

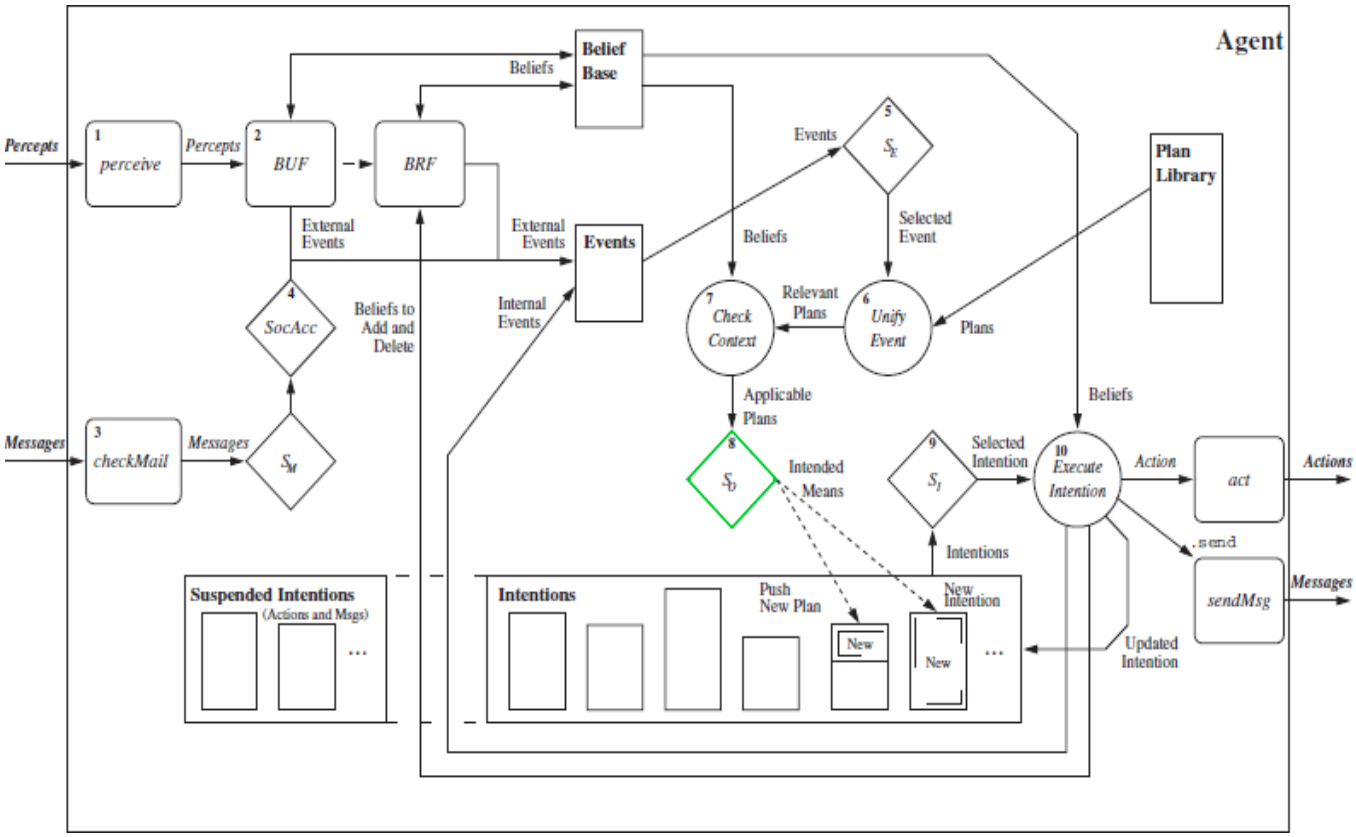

Figura 2.4: Ciclo de raciocínio do Jason.

Nesta figura, as caixas arredondadas, os losangos e os círculos identificam funções do ciclo de raciocínio. Os dois primeiros componentes podem ser customizados pelo programador, diferentemente dos círculos, por serem partes essenciais do interpretador. Os retângulos são os principais componentes da arquitetura, pois identificam o estado atual do agente (base de crenças, conjunto de eventos, biblioteca de planos e conjunto de intenções).

O ciclo de raciocínio do Jason é bem definido através de dez passos, identificados nas figuras em números presentes em alguns elementos. No passo 1 (**Perceber do ambiente**) o interpretador verifica as modificações ocorridas no estado do ambiente através de sensores em sua arquitetura. As alterações são armazenadas de forma simbólica em uma lista de literais, onde cada item é uma *percepção* (*percept*), representando simbolicamente uma propriedade do estado atual do ambiente.

O segundo passo (**Atualizar crenças**) já é mais complexo. Dado p o conjunto das atuais percepções feitas no passo 1 e b o conjunto dos literais na base de crenças do agente, o interpretador atualiza a base de crenças adicionando todo literal *l* ausente em *b* mas presente em *p* e removendo todo literal *l* presente em *b* mas ausente em *p*.

Toda alteração na base de crenças realizada neste passo gera um *evento externo*, representado como um par. O primeiro componente é a alteração realizada e o segundo é a intenção causadora da geração deste evento. Visto que eventos externos não são gerados por uma intenção, eles são acoplados a uma *intenção vazia*.

Por exemplo, se o agente perceber como está o tempo pela primeira vez, irá adquirir a crença $tempo(ensolarado)[source(percept)]$, possibilitando que um plano relevante com o evento de disparo $tempo(ensolarado)$ possa ser executado.

A recepção de mensagens enviadas de outros agentes é realizada no passo 3 (**Receber mensagens de outros agentes**). O interpretador verifica se novas mensagens foram entregues ao agente verificando sua "caixa de correio", que nada mais é que uma lista das mensagens endereçada ao agente. Como somente uma mensagem pode ser processada a cada ciclo, o interpretador do Jason escolhe sempre a primeira desta lista.

O passo 4 (**Selecionar mensagens "socialmente aceitáveis"**) consiste na verificação se a mensagem escolhida para processamento no passo 3 é aceitável ou não pelo agente. Atualmente, todas as mensagens de todos os agentes são aceitas.

Uma das características dos agentes BDI práticos é de constantemente estarem manipulando eventos, que podem representar alterações de percepções no ambiente ou alterações nos objetivos do agente. O quinto passo (**Seleção de um evento**) seleciona um evento a partir de uma lista de eventos pendentes. Assim como a lista de mensagens do passo 2, esta lista é uma fila, onde o primeiro elemento inserido é o primeiro a ser executado. Caso esta lista esteja vazia, o interpretador segue diretamente para o passo 9.

Agora que a base de crenças já foi atualizada e um evento escolhido para ser executado, devemos escolher um plano que manipule este evento. No passo 6 (**Recuperar todos os planos relevantes**), é feita uma pesquisa na biblioteca de planos (*Plan Library* na figura 2.4) por todos os planos que tiverem o evento de disparo que pode ser unificado com o do evento em questão, assim formando uma lista de planos relevantes.

O sétimo passo (**Determinar os planos aplicáveis**) consiste na verificação de quais podem ser ditos *aplicáveis* no momento. Para cada elemento desta lista, seu contexto é recuperado e analisado, verificando quais planos são relevantes, isto é, dada a base de crenças do agente, têm chance de sucesso. Os planos que tiveram seu contexto creditado como verdadeiro, são adicionados em uma lista de planos aplicáveis.

Caso o agente escolha qualquer um desses planos, espera-se que a execução deste seja realizada com sucesso e seu objetivo atingido, mesmo não tendo nenhuma informação sobre a possibilidade de falha ou probabilidade de sucesso.

A seleção do plano a ser executado é realizada no passo 8 (**Selecionar um plano aplicável**), através de uma função customizável pelo programador. O losango que representa este passo, no centro da figura 2.4, está destacado em verde, pois é o algoritmo deste passo que será alterado. Conforme a implementação padrão, a escolha do plano é realizada com

base na ordem em que os planos aparecem na biblioteca de planos, que é determinada pela ordem em que foram escritas no código-fonte do agente. Outra forma de adicionarmos um plano à biblioteca é o recebermos de outro agente através de uma mensagem, sendo adicionado ao final da lista de planos

Atualmente, o Jason escolhe o primeiro plano desta lista. A alteração mais importante deste trabalho no Jason será neste ponto, pois o agente não mais escolherá o primeiro plano da lista, e sim o plano com maior probabilidade de tornar-se realidade. Discutiremos como essa alteração será feita mais adiante, no capítulo 3.

Comprometer-se a executar um plano significa que o agente tem a intenção de seguir o curso de ações presentes no plano, certamente sendo adicionada mais tarde no Conjunto de Intenções (*Set of Intentions*, na figura 2.4). O plano escolhido é chamado de *meios intencionados*, pois o corpo do plano será o meio que o agente intenciona.

Antes de realizar qualquer ação, o agente deve escolher uma intenção entre aquelas prontas para execução. Esta escolha é feita no passo 9 (**Selecionar uma intenção para posterior execução**), fazendo esta escolha através do algoritmo *round-robin*, no qual cada elemento da lista é selecionado conforme sua posição na lista. Portanto, todas as intenções são escolhidas um mesmo número de vezes.

No passo 10 (**Executar um passo da intenção**), agora que já sabemos qual intenção executar, o agente iniciará o curso de sua execução. Existem seis tipos de ações possíveis de ser executadas neste ponto, podendo ou não alterar o ambiente: ação de ambiente, objetivo de realização, objetivo de teste, notas mentais, ações internas e expressões.

Antes de iniciar um novo ciclo de raciocínio, é necessário atualizar as variáveis utilizadas. Intenções suspensas, que estavam esperando algum tipo de resposta, podem agora trocar de status caso a resposta esteja disponível. As intenções que foram executadas por completo são retiradas do conjunto de intenções. Os planos que terminaram a execução também são removidos do conjunto de planos. Agora o interpretador está pronto para iniciar um novo ciclo.

2.2.6 Um Exemplo de Agente: Robôs Coletores de Lixo em Marte

Para exemplificar a programação de um agente no Jason, utilizaremos um exemplo apresentado em CAMPBELL; DINVERNO (1990) apud HÜBNER; BORDINI; VIEIRA (2004). Também serão apresentadas dificuldades que os agentes deste sistema possam encontrar no tratamento de incertezas e suas possíveis soluções.

O cenário utilizado aqui trata de dois robôs coletores de lixo no planeta Marte. O robô $r1$ procura por lixo e ao encontrar, o pega, se dirige até o robô $r2$, deixa o lixo em uma posição próxima dele e retorna para sua última posição. Enquanto isso, o robô $r2$ situa-se próximo ao incinerador, onde todo lixo encontrado deve ser queimado. Ao detectar que o robô $r1$ deixou lixo para ser incinerado, este agente pega o lixo e coloca-o no incinerador. O ambiente foi divido em uma matriz de posições com sete linhas e sete colunas. Os autores deste exemplo modelaram o ambiente utilizando a ferramenta Prometheus (WINIKOFF; PADGHAM, 2004), apresentado na figura 2.5.

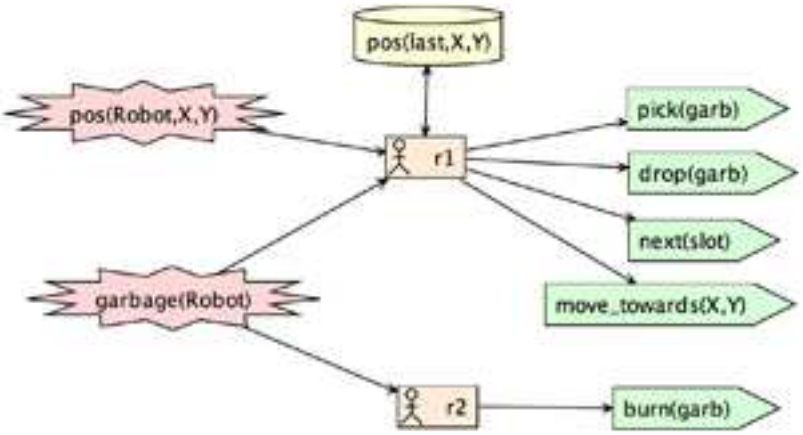

Figura 2.5: Modelagem do sistema para o exemplo de robôs coletores de lixo em Marte

Conforme HÜBNER; BORDINI; VIEIRA (2004), um Sistema MultiAgente deve ser configurado em um arquivo texto com extensão $.mas2j$ para ser executado no Jason, onde se pode definir todas as configurações do ambiente e os agentes que farão parte do sistema. Cada agente é definido por seu nome simbólico que, por padrão, será o mesmo nome do arquivo, de extensão $.asl$, com o código AgentSpeak deste agente.

A estrutura do sistema é descrita no arquivo $mars.mas2j$, apresentado abaixo. Nele são definidos os parâmetros de infra-estrutura ($Centralised$, linha 2), indicando que todos os agentes serão cadastrados na mesma plataforma; ambiente ($MarsEnv$, linha 3), discutido a seguir; agentes ($r1$ e $r2$, linha 4).

```
1 MAS mars {
2   infrastructure: Centralised
3   environment: MarsEnv
4   agents: r1; r2;
5 }
```

A classe que implementa o ambiente, codificada em Java, contém métodos auxiliares utilizados pelos agentes e algumas regras que o ambiente deve obedecer. Este exemplo ainda contém uma interface gráfica, apresentada nas figuras 2.6 e 2.7, onde podemos visualizar a posição de cada agente no ciclo atual, as posições com lixo e o caminho percorrido por $r1$.

Na figura 2.7, podemos observar que o lixo da posição anterior, presente na figura 2.6, já foi recolhido. Também podemos notar que o agente $r1$ continuou sua busca por mais unidades de lixo a partir da posição seguinte, encontrando mais uma unidade e já carregando-a até o agente $r2$.

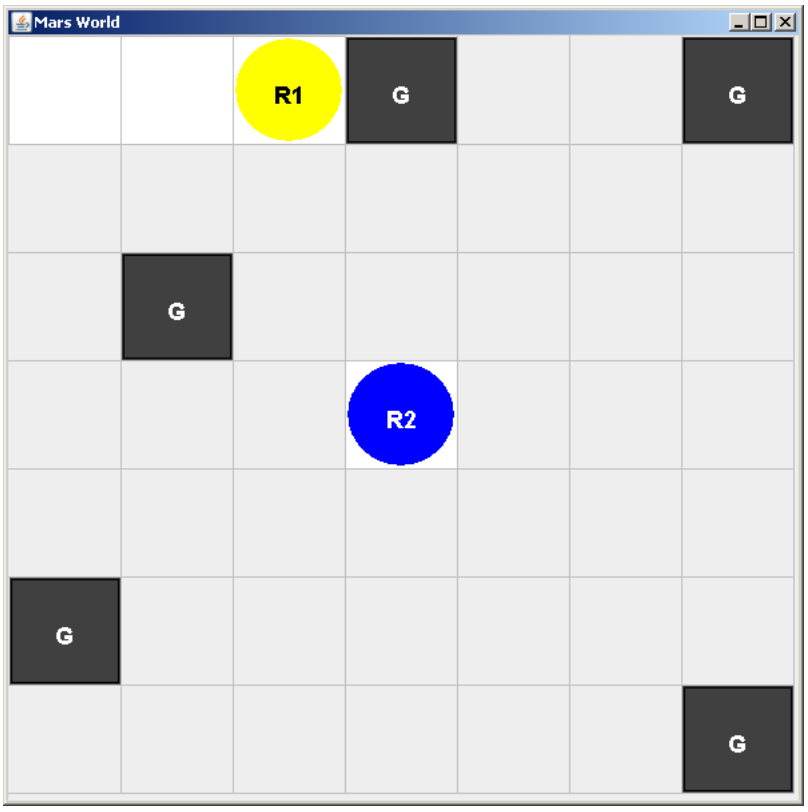

Figura 2.6: Interface gráfica do exemplo Robôs Coletores de Lixo em Marte no momento em que o agente $r1$ está prestes a encontrar uma unidade de lixo

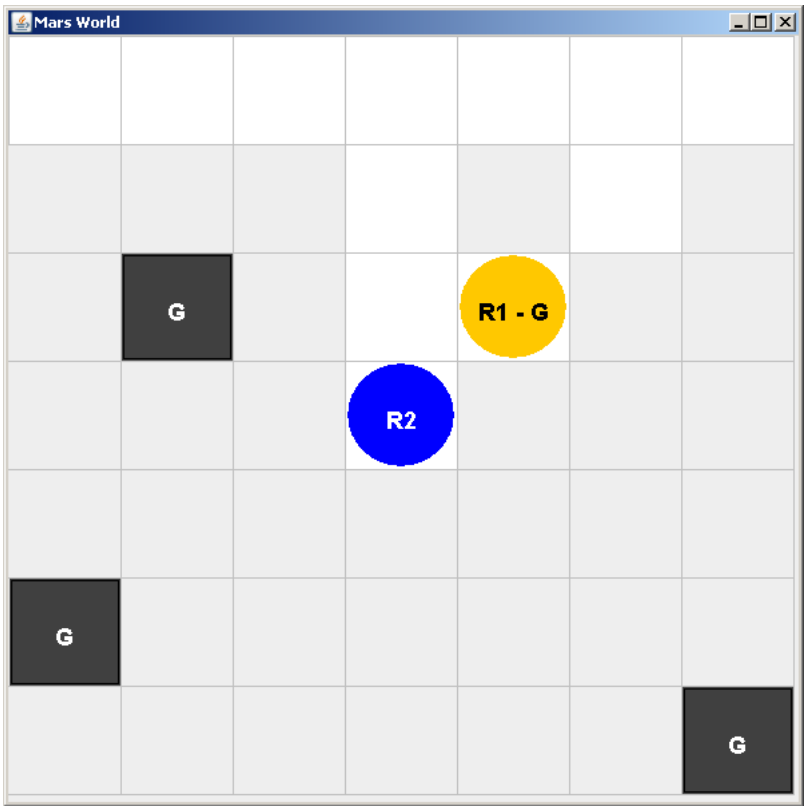

Figura 2.7: Momento em que o agente $r1$ carrega outra unidade de lixo, após já ter recolhido uma em posição anterior

O código-fonte do agente $r1$ que segue abaixo foi fortemente baseado no exemplo contido na própria pasta de instalação do Jason. Nas linhas 6 e 17, em vez de utilizar o operador "!" para a adição de um novo objetivo, foi utilizado "!!". Este operador permite o agente prosseguir sua execução antes mesmo que este objetivo tenha sido atingido. Esta alteração embora sutil, é fundamental para o bom funcionamento do agente.

```
1  at(P) :- pos(P,X,Y) & pos(r1,X,Y).
2
3  !check(slots).
4  +!check(slots) : not garbage(r1)                    (p1)
5    <- next(slot);
6       !!check(slots).
7  +!check(slots).                                     (p2)
8
9  +garbage(r1) : not .desire(carry_to(r2))            (p3)
10   <- !carry_to(r2).
11
12 +!carry_to(R)                                       (p4)
13   <- ?pos(r1,X,Y);
14      -+pos(last,X,Y);
15      !take(garb,R);
16      !at(last);
17      !!check(slots).
18
19 +!take(S,L) : true                                  (p5)
20   <- !ensure_pick(S);
21      !at(L);
22      drop(S).
```

```
23
24 +!ensure_pick(S) : garbage(r1)                    (p6)
25   <- pick(garb);
26     !ensure_pick(S).
27 +!ensure_pick(_).                                 (p7)
28
29 +!at(L) : at(L).                                  (p8)
30 +!at(L)                                           (p9)
31   <- ?pos(L,X,Y);
32     move_towards(X,Y);
33     !at(L).
```

A base de crenças do agente $r1$ contém somente a regra da linha 1, indicando suas coordenadas X e Y no mapa. O objetivo (desejo) inicial deste agente é $!check(slots)$, o qual o faz analisar sua posição atual em busca de lixo. Assim, teremos dois planos relevantes ($p1$ e $p2$). Durante sua procura, caso o agente $r1$ encontre lixo na superfície marciana, será informado pelo ambiente através da percepção $garbage(r1)$, sendo adicionada à base de crenças do agente. Enquanto esta percepção não ocorrer, somente $p1$ será executado, pois apesar de ambos serem aplicáveis, este é o primeiro da lista de planos aplicáveis. Ao passar por uma posição com lixo, $p1$ deixa a lista de planos relevantes, que agora contém os planos $p2$ e $p3$.

A ação interna nativa do Jason $desire$ indica se o agente possui o desejo passado como parâmetro, ou seja, neste caso, se o agente já está carregando lixo até o agente $r2$. Em caso negativo, este desejo é adicionado como um objetivo, fazendo com que $p4$ torne-se um plano relevante e, por consequência, também aplicável, pois seu contexto é vazio. Este plano contém cinco subobjetivos necessários para o agente $r1$ executar a tarefa de limpar aquele espaço: (i) recuperar sua posição no mapa; (ii) atualizá-la; (iii) pegar o lixo presente nesta posição, levando-o até o agente $r2$; (iv) voltar para a posição onde inicialmente encontrou o lixo; (v) continuar a procura por novas posições com lixo na superfície marciana. O item (iii) é realizado através de três passos (plano $p5$). O primeiro ($p6$) garante que o agente pegará o lixo S desta posição. Já o segundo ($p8$ e $p9$), redireciona $r1$ até $r2$, posicionado em L. Finalmente, o terceiro passo ($drop(S)$, uma ação do ambiente) larga S nesta posição.

O objetivo $ensure\_pick(S)$ (linha 19) realiza a ação de pegar o lixo até que este seja efetivamente pego. Uma função implementada na classe que estende o ambiente diz, de forma randômica, se nas primeiras duas tentativas o agente teve sucesso em pegar o lixo ou não. Na terceira tentativa, o agente sempre tem sucesso.

Este cenário somente dispõe de informações completas, não exigindo dos agentes envolvidos um processo de raciocínio muito complexo nem maiores dificuldades. Entretanto, pensemos nas seguintes possibilidades:

1. Se o sistema de detecção de lixo do agente $r1$ não fosse totalmente confiável? O agente pode acreditar que em uma posição há lixo, tentando pegá-lo, onde na verdade não há lixo algum. O agente pode ainda acreditar que possui lixo, quando na realidade está descarregado, executando o processo de ir até o incinerador, desperdiçando tempo e recursos;

2. Se o sistema de detecção de lixo dos agentes também apresentasse defeito? Ao acreditar, equivocadamente, na existência de lixo a ser incinerado, os agentes novamente desperdiçarão tempo e recursos, ou ainda caso acredite, também equivocadamente,

que uma posição esteja limpa, o ambiente continuará sujo sem o conhecimento dos agentes;

3. Se o sistema de localização de ambos os agentes não fosse totalmente confiável? O sistema que informa as coordenadas atuais dos agentes apresentando defeito, diversos problemas de deslocamento dos agentes podem ocorrer, como o acúmulo de lixo em setores do ambiente, já que o agente $r1$ pode largar o lixo em locais não apropriados ou $r2$ não perceber a presença de lixo em um local;

4. Se existisse um limitante de tempo para o agente $r1$ limpar a área? Caso a área tivesse de ser limpa em um determinado período de tempo, talvez o agente $r1$ obtivesse um melhor desempenho não procurando lixo perto de posições em que já tivesse sido encontrado, deixando para investigá-las quando acreditasse que toda a área já estivesse limpa;

5. Se a existência de lixo fosse mais provável em um determinado local? Se o agente $r1$ acreditar, por exemplo, que é mais provável encontrar lixo longe do incinerador, pode investigar primeiro estas áreas, deixando locais próximos ao incinerador por último;

6. Se fosse mais importante coletar lixo em um determinado local? Assim como no exemplo acima, $r1$ pode dar preferência a estes locais, deixando a investigação das outras áreas por último;

7. Se o agente $r1$, podendo carregar mais de uma unidade de lixo (e estes tenham diferentes tamanhos), deve seguir procurando lixo ou levar o que já possui até o incinerador? Caso $r1$ tenha uma mochila ou carrinho, onde possa armazenar lixo de diferentes tamanhos, carregando-o enquanto continua sua procura, este pode decidir entre descarregá-lo no incinerador ou procurar por mais unidades de lixo. Ao encontrar, pode também decidir largar uma unidade de lixo para coletar outra, dependendo do seu tamanho e prioridade.

2.2.7 Como o Jason Trata Incertezas

Como veremos em mais detalhes na sessão 2.3, o conhecimento incerto está presente em todos ambientes, inclusive nos BDI. Muitas vezes o agente adquire suas crenças, ou faz percepções, de fontes não totalmente confiáveis, ou por vezes adquirindo informações parciais, incompletas ou ambíguas. Para poder representar este conhecimento, o agente vê-se obrigando a fazer suposições ou a trabalhar com graus de verdade, assumindo que um dado conhecimento é verdadeiro ou falso, sem meio termo.

Para mostrarmos como o Jason lida com o conhecimento incerto, utilizaremos um exemplo baseados em FAGUNDES; VICARI; COELHO (2007). Este exemplo simples modela as crenças sobre a escolha que o agente deve fazer: ir ao Cinema (*Movies*) ou jogar Futebol (*Soccer*). A escolha é baseada em dois aspectos: Clima (*Weather*) e Jogadores (*Players*). Caso o clima esteja chuvoso, o agente acredita que jogar futebol não é um bom programa, dando preferência ao cinema. O agente também acredita que dias ensolarados são perfeitos para jogar futebol, sendo um desperdício ir ao cinema. Por fim, nosso agente acredita que uma partida de futebol requer jogadores suficientes para formar dois times. A figura 2.8 representa a rede bayesiana construída para modelar estas crenças. As tabelas associadas a cada nodo representam suas respectivas Tabelas de Probabilidades Condicionais (TPCs).

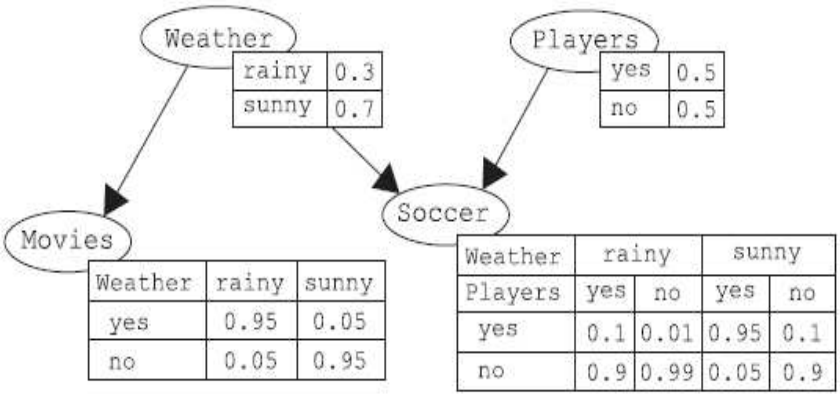

Figura 2.8: Rede bayesiana para o exemplo Cinema ou Futebol.

Tentando modelar este exemplo no Jason, a primeira dificuldade encontrada é, obviamente, como modelar o conhecimento probabilístico, ou seja, como abstrair as probabilidades, fazendo o agente trabalhar somente com certezas. Analisando a CPT do nodo *Soccer*, vemos que a probabilidade de o agente jogar futebol é alta somente quando o clima está ensolarado e há jogadores suficientes. Visto que não temos qualquer informação sobre o estado das variáveis *futebol* e *cinema*, devemos fazer a decisão somente com base nas crenças *clima* e *jogadores*. Portanto, podemos modelar o agente da seguinte forma:

```
1  // Objetivos Iniciais
2  !atividade.
3
4  // Planos
5  +!atividade[source(self)]
6    : (clima(ensolarado)[p(C1)] & (C1>0.7) & jogadores(sim)[p(J1)] & (J1
       >0.5))
7      | (clima(chuvoso)[p(C2)] & (C2<0.2) & jogadores(sim)[p(J2)] & (J2
         >0.99))
8      | (clima(ensolarado)[p(C3)] & (C3<0.9) & jogadores(nao)[p(J3)] & (
         J3>0.4))
9    <-   .print("Futebol").
10 +!atividade[source(self)]
11   : true
12   <-   .print("Cinema").
```

Como este exemplo é meramente didático, o corpo dos planos do agente se resume simplesmente a um comando de escrita na tela (*print*), nas linhas 9 e 12. Caso o agente acredite que a probabilidade do clima estar ensolarado é alta (maior que 70%) e provavelmente existem jogadores suficientes (probabilidade maior que 50%), ele achará que são condições ideais para jogar futebol (plano da linha 5). O agente ainda pode escolher jogar futebol se não está certo sobre o clima chuvoso (menor que 20%) e está certo que há jogadores (maior que 99%) ou não tem informação precisa sobre o tempo ensolarado (menor que 90%) e vê a possibilidade de haver jogadores suficiente (maior que 40%). Caso contrário, escolherá ir ao cinema (plano da linha 10), não importando as condições do clima. Este modelo é bastante simplista e abstrai grande parte da realidade, pois a única chance de o agente jogar futebol é quando ele acreditar que o clima está ensolarado e que provavelmente existem jogadores suficientes. Não há a possibilidade, por exemplo, de o agente escolher jogar futebol mesmo com o clima chuvoso ou com jogadores

insuficientes, como proposto na rede bayesiana da figura 2.8.

Esta implementação apresenta deficiências, como o fato de o programador ter de definir estaticamente o limiar da probabilidade considerada. No exemplo apresentado, utilizou-se 0.7, 0.2 e 0.99 como limiares para a crença *clima* e 0.5, 0.99 e 0.3 para *jogadores*. Além disso, devemos utilizar uma expressão lógica extremamente grande (linhas 6 a 8) para verificar em quais condições o agente enquadrará um plano como aplicável ou não. No contexto do plano foram inseridas somente as condições mais favoráveis para o agente jogar futebol (ver tabela de probabilidades na figura 2.8).

A dificuldade do Jason em ambientes onde a incerteza está presente fica evidente neste exemplo. Mesmo sendo possível simular um raciocínio probabilístico, outros obstáculos foram notados, como o uso de limiares e construção de contextos muito longos.

2.3 Redes Probabilísticas

A representação do conhecimento probabilístico em redes probabilísticas permite manipular a incerteza com base em princípios matemáticos fundamentados, modelando o conhecimento do especialista do domínio de uma forma intuitiva. Outra grande vantagem frente a outras estruturas de representação de conhecimento é a redução do cálculo de inferência a vários cálculos locais, utilizando somente variáveis obtidas de um nó e seus vizinhos em uma estrutura de grafo. Assim, não é necessário realizar o cálculo da distribuição de probabilidade conjunta global. Segundo JENSEN; OLSEN; ANDERSEN (1988), sua representação gráfica explicita relações de dependência entre as variáveis do modelo, sendo uma poderosa ferramenta para modelagem do sistema e aquisição de conhecimento.

Um modelo probabilístico contém dois tipos de informações: a qualitativa, que são as relações de dependências, e a quantitativa, na qual representa os estados das variáveis e suas distribuições de probabilidade. As principais redes probabilísticas avaliadas neste trabalho são Redes Bayesianas e Diagrama de Influências, detalhados ao longo deste capítulo.

2.3.1 Conhecimento Incerto

Construir um agente autônomo com o objetivo de tomar decisões em um ambiente real, onde as informações são imprecisas, parciais, incompletas e até mesmo ambíguas é um grande desafio. O conhecimento deve ser modelado de uma forma computacionalmente compreensível, sendo muitas vezes necessário omitir parte da informação fazendo simplificações (ignorância teórica), não informando (ignorância prática) ou resumindo de forma superficial (devido ao esforço exigido para detalhá-los). Por exemplo, um médico pode diagnosticar certa doença com base nas informações disponíveis no momento. No entanto, este diagnóstico é uma hipótese, podendo ou não estar correta. As simplificações adotadas podem ter origem em conhecimento médico incompleto sobre a patologia em questão (ignorância teórica), sintomas determinantes da patologia não detectados, devido a fase de evolução da doença (ignorância prática) ou não realização de exames complementares (dados sintomáticos grosseiramente resumidos) (LADEIRA; VICARI; COELHO, 1999). Assim, o médico raciocina utilizando conhecimento incerto, atribuindo graus de certeza às crenças obtidas para construir hipóteses.

Uma das áreas do conhecimento que mais incentivou o aparecimento de modelos para tratamento da incerteza foi a médica. Isto talvez possa ser explicado pelo fato de neste campo todos os tipos de incerteza estarem presentes. Alguns exemplos são descritos

abaixo:

- Informação incompleta: muitas vezes, não há registro de maiores informações sobre um paciente ou ele é incapaz de passá-las ao especialista, seja por esquecimento ou falta de interesse;

- Informação errada: mesmo que o paciente passe as informações ao médico, pode descrevê-los de forma equivocada, intencionalmente ou não. Além disso, os exames podem ter sido realizados de forma inadequada, apresentando um resultado errado. Assim, é importante o médico não confiar cegamente nas informações disponíveis;

- Informação imprecisa: Muitos sintomas são difíceis de quantificar, como febre, dor, falta de apetite ou náusea;

- Mundo real não-determinístico: nenhum ser humano é igual ao outro. Assim, dois pacientes podem apresentar os mesmo sintomas, mas reagir de formas completamente distintas ao tratamento de certa doença;

- Modelo incompleto; além de muitos fenômenos médicos serem atualmente desconhecidos, é frequente a falta de consenso entre especialistas frente a um determinado assunto. Além disso, o ser humano é tão complexo, que existem infinitas informações podendo influenciar no diagnóstico de uma doença;

- Modelo inexato: para ser possível quantificar a incerteza de forma ótima, deveríamos inserir um grande número de parâmetros, porém muitas vezes estas informações estão indisponíveis. Assim, é desejável que o sistema leve em consideração a inexatidão do modelo.

O conhecimento incerto também é uma área de estudo da Linguagem Natural, onde existem diversos trabalhos utilizando técnicas para sua representação, como em ZADEH (2006), ZADEH (1996) e ZADEH (2004).

Em ZADEH (2004), é descrito um sistema chamado *Precisiated Natural Language* (PNL, ou *Linguagem Natural Precisa* em tradução livre), que é um conceito que fornece base para computação e raciocínio com informação baseada em percepção, utilizando lógica fuzzy. Segundo o autor, muitas das informações que temos são baseadas em percepções. Caso tenha de estimar a idade de Rose, por exemplo, e a informação que tenho é: "(a) Rose é uns 10 anos mais velha que Beti; e (b) Beti tem dois filhos: um de 20 e poucos e um de 30 e poucos". Dada a questão "Quanto anos Rose tem?", adicionaria "uns 10 anos" à idade estimada de Beti. Mas como poderia estimar a idade de Beti? Humanos têm a habilidade de processar informações considerando outras informações que lógicas baseadas em bivalência não têm. Outra importante questão é a relevância dos itens (a) e (b) para chegar à idade de Rose.

A proposta do autor para resolver este problema é traduzir a linguagem natural em uma Linguagem de Restrição Generalizada (Generalized Constraint Language). Utilizando um dicionário, é transformada em Protoform Language e finalmente traduzir para linguagem humana novamente, utilizando uma Base de Dados de Conhecimento do Mundo (World Knowledge Database), no qual utiliza uma Base de Dados de Dedução (Deduction Database) multiagente modular.

ZADEH (2004) defende o abandono da bivalência e adoção de novas ferramentas, como PNL, para tratar este tipo de informação. Uma das muitas possíveis aplicações

do PNL é seu uso em sistemas de busca. Os buscadores atuais têm muitas capacidades apreciáveis, porém não têm a capacidade de dedução, isto é, a capacidade de responder uma questão sintetizando diversas informações presentes na base de conhecimento.

2.3.1.1 "A Priori" x "A Posteriori"

A expressão "a priori" vem do latim, partindo daquilo que vem antes, designando um tipo de conhecimento que independe de experiência, por exemplo "Toda mulher é do sexo feminino". O conhecimento "a posteriori", do latim, partindo daquilo que vem depois, depende de experiência ou evidências, como em "O inverno gaúcho é chuvoso".

Quando repetimos certo evento n vezes, sendo n um número tendendo ao infinito, podemos observar certa regularidade estatística que nos permite definir as probabilidades de cada estado deste evento. Por exemplo, ao jogarmos um dado não-viciado numerado de 1 a 6 por n vezes, sendo n um número muito grande, o número que cada face do dado apareceu para cima será muito próximo de $\frac{1}{6}$ cada um. Esta regularidade é evidenciada quando a frequência relativa f_A estabiliza-se em torno de um valor dentro do intervalo [0, 1] para toda $A \subseteq S$ associado ao evento, a medida que n tende ao infinito, dado que A simboliza o evento em questão e S o espaço amostral. Segundo este ponto de vista objetivista, as probabilidades são aspectos reais do universo (tendência dos atores comportarem-se de certa maneira), e não descrições do grau de certeza do especialista. Neste caso, os cálculos frequentistas são tentativas de observar o valor real de um estado do evento.

No modelo frequentista para a teoria da probabilidade, considera-se que uma distribuição de probabilidades para os eventos A do espaço amostral S pode ser obtida tomando-se $P(A)$ como o limite das frequências relativas f_A, isto é,

$$P(A) = \lim_{n \to +\infty} f_A = \lim_{n \to +\infty} \frac{n_A}{n} \tag{2.1}$$

dado que n_A representa a quantidade de vezes que o evento A acontece em meio a todos os possíveis (n).

Formalmente, no contexto frequentista, só se pode falar de probabilidade para eventos associados a experimentos possíveis de serem repetidos. Do ponto de vista matemático, essa definição de probabilidade apresenta fraquezas, pois um número limite real pode não existir. Assim, a formalização da definição não obedece rigorosamente à teoria matemática de limite. Isto traz como consequência a dificuldade em demonstrar os teoremas de probabilidade, muito embora esta definição seja bastante intuitiva. A denominação "a posteriori" resulta do fato de se ter de repetir a experiência várias vezes para que o cálculo da probabilidade seja ao menos confiável.

Dado um conhecimento aleatório uniforme definido em um espaço de amostragem S, a probabilidade de ocorrer um dado evento A (contido em S) onde este conhecimento seja verdadeiro é calculado dividindo-se o número de elementos do evento A, n_A, pelo número de elementos do espaço amostral S, n_S, isto é,

$$P(A) = \frac{n_A}{n_S} \tag{2.2}$$

Este cálculo de probabilidade não depende da experiência, originando o nome "a priori". Apesar de muito simples, esta definição apresenta sérias limitações:

- Muitas vezes não é possível identificar e enumerar todos os casos possíveis, por exemplo em situações onde os resultados tenham caráter qualitativo e/ou não se pode efetuar contagens;

- Não faz sentido falar em espaço amostral (n_S) quando o ambiente possui infinitas situações;
- Inexiste um critério para casos igualmente possíveis de ocorrer.

Ao contrário da frequentista, esta definição estabelece um modelo matemático adequado à interpretação de um certo tipo de fenômeno (não todos), e considera-se o seu comportamento ideal em condições ideais.

2.3.2 Representação da Incerteza

A representação do conhecimento em geral é um tema que vem sendo tratado, com grande importância, há muitos anos pela comunidade de Inteligência Artificial (IA). No entanto, segundo PATEL-SCHNEIDER (1985), a natureza exata do papel desta área é indefinida. Sistemas de representação do conhecimento podem variar de pacotes para manipular estruturas de dados até sistemas de IA completos que planejam ou fazem gestão de recursos.

Para representarmos o conhecimento incerto, devemos adotar uma entre três técnicas: *simbólica*, *numérica* ou *fuzzy*. A abordagem simbólica é eficiente para o tratamento de informações incompletas, onde algumas respostas a importantes questões são desconhecidas, porém inadequada para informações imprecisas, onde as respostas são conhecidas, porém aproximadas face à baixa confiabilidade ou inerente imprecisão da linguagem de representação do conhecimento utilizada, pois é impossível quantificar essa incerteza.

A abordagem numérica discretiza o intervalo de valores possíveis a serem utilizados através de valores *fuzzy*. A variável *febre*, por exemplo, poderia ter quatro estados: ausente ((-∞, 37.5]), baixa ((37.5, 38]), moderada ((38, 39]) e forte ((39, ∞)). Assim, é possível definir um cálculo para combinar e propagar as incertezas durante o processo de raciocínio. As técnicas fuzzy, que utilizam tanto a técnica simbólica quanto a numérica, são adequadas para representar conceitos vagos inerentes a termos lingüísticos, tais como:

- Imprecisão linguística de predicados como “alto”, “maior”, “próximo”, “jovem”, etc;
- Imprecisão do tipo “a caixa pesa entre 10 e 16 quilos”;
- Quantificadores imprecisos do tipo “muitos”, “alguns”, “a maioria”, etc. (FLORES, 2003).

2.3.3 Teorema de Bayes

Dada diversas informações probabilísticas, como podemos calcular a probabilidade de certa crença ser verdadeira ou falsa? O Teorema de Bayes constitui uma das primeiras tentativas para se lidar com essa incerteza numérica, calculando a probabilidade condicional de um evento a partir das evidências disponíveis no ambiente.

O cálculo básico da estatística Bayesiana é a probabilidade condicional $P(H|E)$ da hipótese H, dada a evidência E observada. Também é necessário levar em conta a probabilidade *a priori* de H, sem qualquer evidência. Portando, temos:

- $P(H_i|E)$: probabilidade da i-ésima hipótese ser verdadeira, dado a evidência E;
- $P(E|H_i)$: probabilidade da evidência E ser observada, sendo a i-ésima hipótese verdadeira;

- $P(H_i)$: probabilidade “a priori” da i-ésima hipótese ser verdadeira, na ausência de qualquer evidência específica;
- m: número de hipóteses possíveis.

Logo, o Teorema de Bayes afirma

$$P(H_i|E) = \frac{P(E|H_i).P(H_i)}{\Sigma_{k=1}^{m} P(E|H_k).P(H_k)} \tag{2.3}$$

Redes Bayesianas utilizam esta fórmula para a atualização de crenças, face a ocorrência de E, ou seja, ela estabelece que a crença que atribuímos à hipótese H_k, após obtermos a evidência E, pode ser calculada multiplicando nossa crença prévia $P(H_k)$, pela verossimilhança $P(E|H_k)$ que E irá ocorrer se a hipótese H_k for verdadeira.

Caso exista mais de uma evidência, é preciso levá-las em conta no cálculo de atualização das crenças, ficando:

$$P(H_i|E_1, ..., E_n) = P(E_1, ..., E_n|H_i).P(H_i) = \frac{P(E_1, ..., E_n|H_i).P(H_i)}{\Sigma_{k=1}^{m} P(E_1, ..., E_n|H_k).P(H_k)} \tag{2.4}$$

O Teorema de Bayes fornece uma fórmula matemática para o cálculo da distribuição de probabilidades conjuntas dado o relacionamento probabilístico e condicional presente nas redes bayesianas e diagramas de influência. Assim, formaliza as questões qualitativa e quantitativa das redes probabilísticas (FLORES, 2002).

2.3.4 Redes Bayesianas

Uma Rede Bayesiana (RB), também chamada de rede de crenças, rede de opinião ou rede causal, é um modelo gráfico probabilístico que representa um conjunto de variáveis randômicas e suas dependências condicionais através de um grafo acíclico direcionado (PEARL, 1986). Varáveis podem ser quantidades observáveis, parâmetros desconhecidos, hipóteses ou variáveis latentes, enquanto que arestas representam dependências condicionais entre as variáveis.

Em RUSSELL; NORVIG (2009), RBs são definidas formalmente como uma tripla (N, E, P), onde $N = X_1, ..., X_n$ é o conjunto de nodos (variáveis com os estados possíveis de cada X_i representado pelo conjunto mutuamente exclusivo X_i), E é o conjunto de arcos orientados tal que $D = (N, E)$ é um grafo orientado acíclico e P é a distribuição de probabilidades

$$P(x_i, ..., x_n) = \prod_i P(x_i|pa(X_i)) \tag{2.5}$$

onde $pa(X_i)$ é o conjunto de nodos que tem um arco incidindo em X_i (os pais de X_i). Os arcos representam uma dependência direta entre as variáveis, mensurada por tabelas de distribuição de probabilidades condicionais $p(X_i|pa(X_i))$. Se $pa(X_i)$ não existir, a tabela de X_i é reduzida à distribuição de probabilidades dos estados x_i de X_i.

A figura 2.9 ilustra uma RB que descreve a relação causal entre as estações do ano (S), a ocorrência de chuvas durante a estação (C), a utilização do irrigador durante a estação (I), a umidade do pavimento (U), e o fato de o pavimento estar escorregadio (E). Neste exemplo são apresentadas somente variáveis naturais.

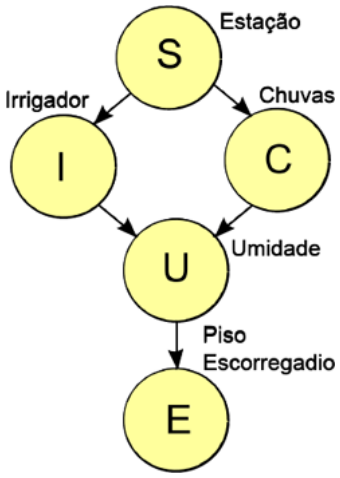

Figura 2.9: Uma RB representando influências causais entre cinco variáveis.

Em Redes Bayesianas, uma evidência em uma variável é uma preposição sobre a certeza de seus estados, os quais afetam as probabilidades de outras variáveis. Pode ser *específica* caso tenhamos certeza sobre o estado exato, ou *virtual* se mais de um estado pode ser verdade. Dada algumas evidências, é possível inferir sentenças com certo grau de certeza.

Existem diversos *frameworks* para o desenvolvimento de Redes Bayesianas. Entretanto, nem todos são adequados para a integração desejada neste trabalho. O *framework* procurado deve ter código aberto ou possuir uma API que possibilite sua extensão para a rede bayesiana poder comunicar-se com os agentes do Jason. Outra funcionalidade necessária é a entrada de novas evidências para simularmos o comportamento dos agentes frente a informações com probabilidade máxima. Vários *frameworks* foram testados, sendo seus prós e contras detalhados a seguir.

2.3.4.1 BNJ

O primeiro a ser testado foi o BNJ (Bayesian Network tools in Java). Descrito em PERRY; STILSON (2002) é um software livre que gera um arquivo do tipo XML, no qual representa a Rede Bayesiana, facilitando a interpretação por um algoritmo. Porém, não correspondeu às expectativas pela incapacidade de manipular novas evidências em tempo real.

2.3.4.2 Hugin

O *framework* Hugin (ANDERSEN et al., 1989) também gera um arquivo do tipo XML e é capaz de gerenciar novas evidências. Contudo, as novas probabilidades geradas pela chegada da evidência não são disponibilizas de algum modo que possibilite sua interpretação por um outro software. A única alternativa possível seria alterar o código-fonte do programa, sendo impossível pelo fato deste não ter código aberto. Este é um dos principais *frameworks* para construção de RB utilizados atualmente, sendo utilizado em universidades e inclusive em projetos comerciais. A figura 2.10 apresenta uma tela com uma RB de exemplo.

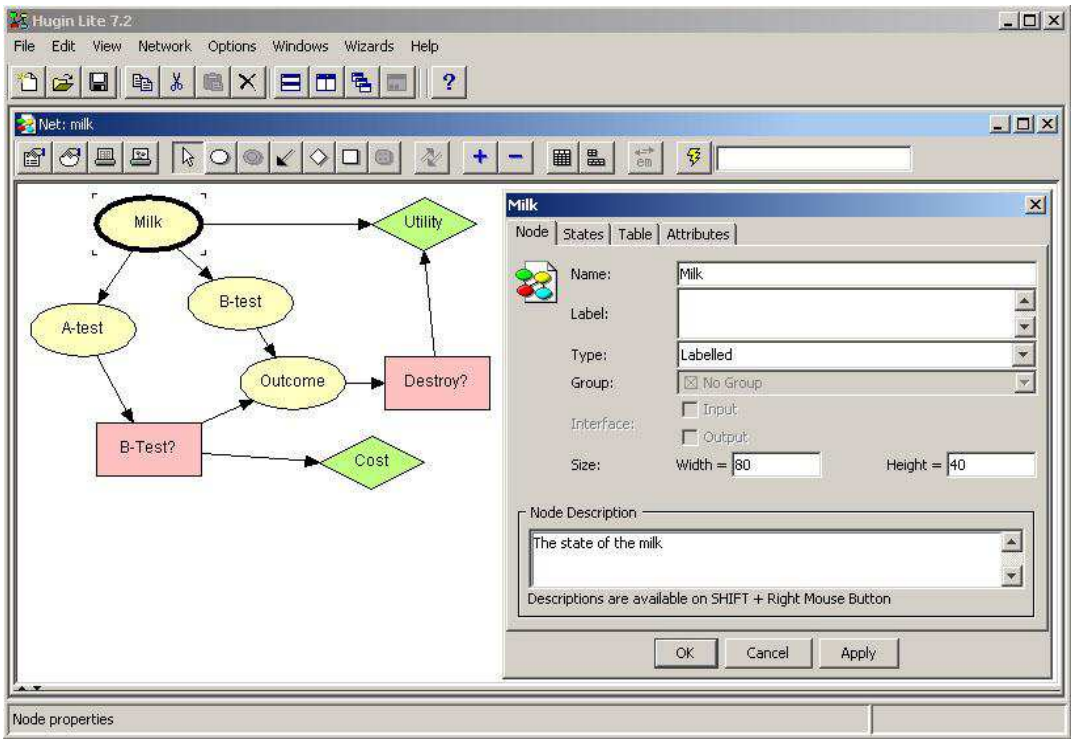

Figura 2.10: Uma RB construída no *framework* Hugin.

2.3.4.3 Java Bayes

Outro software testado foi o JavaBayes (COZMAN, 2000), em que se implementou um novo algoritmo de eliminação de variáveis para redes bayesianas, também apresentado no trabalho citado. A eliminação de variáveis computa a probabilidade marginal de um determinado conjunto de variáveis em uma rede.

2.3.4.4 UnBBayes

Também se cogitou utilizar o construtor de Redes Bayesianas UnBBayes (UNBBAYES: MODELING UNCERTAINTY FOR PLAUSIBLE REASONING IN THE SEMANTIC WEB, 2010), pois é um *framework* com diversas funcionalidades e de código aberto. Caso este fosse escolhido para dar andamento ao projeto, não seria recomendável a alteração de seu código, mas sim a utilização de um paradigma de programação diferente do utilizado para a construção deste e de todos os *frameworks* analisados aqui. O paradigma chamado de Programação Orientada a Aspectos (POA) possibilita a inserção de novas linhas de código sem alterá-lo. Essa necessidade deve-se ao fato de este *framework* ser muito complexo, dificultando a implementação da nova funcionalidade pretendida neste trabalho.

A Programação Orientada a Aspectos (POA) baseia-se na ideia de construir um programa de forma clara, envolvendo tais aspectos, incluindo propriedades como isolamento, composição e reuso de código (KICZALES et al., 1997). Assim, possibilita ao programador a separação clara de componentes e aspectos. Este paradigma provê mecanismos de linguagem que capturam estruturas presentes em diversos pontos do código, possibilitando programar interesses transversais de forma modular, atingindo benefícios usuais de melhorias de modularidade, tais como código mais simples, fácil de desenvolver e manter, possuindo grande potencial de reuso.

A utilização desta ferramenta foi descartada, pois a programação do trabalho pretendido seria de alta complexidade, tanto pela mudança na forma de programação (orientada a aspectos) quanto pela complexidade já existente do próprio *framework* desenvolvido até o momento. Também foi cogitada a utilização de uma API do UnBBayes, que poderia facilitar a implementação deste trabalho. Entretanto, essa possibilidade foi pensada somente após a implementação deste já ter sido concluída.

2.3.4.5 JAmplia

Este *framework* de RB está dentro do projeto AMPLIA (FLORES; VICARI, 2005). Este projeto teve a colaboração de diversos alunos para sua construção, inclusive uma primeira versão de um *framework* de RB, utilizando a linguagem de programação *Delphi*. Em MACHADO (2006), este *framework* foi reimplementado, utilizando a linguagem Java, criando, então, o JAmplia.

Este construtor de RB é menos complexo que o UnBBayes, possibilitando sua edição de forma usual, isto é, utilizando o mesmo paradigma (Orientado a Objetos) utilizado em sua construção. Além disso, o AMPLIA é um projeto do mesmo grupo de pesquisa. Estes foram os principais fatos que contribuíram para a escolha deste *framework* para ser utilizado neste trabalho. Maiores detalhes de como será feita a implementação serão discutidos no capítulo 3. A figura 2.11 mostra uma tela rodando uma RB simples como exemplo.

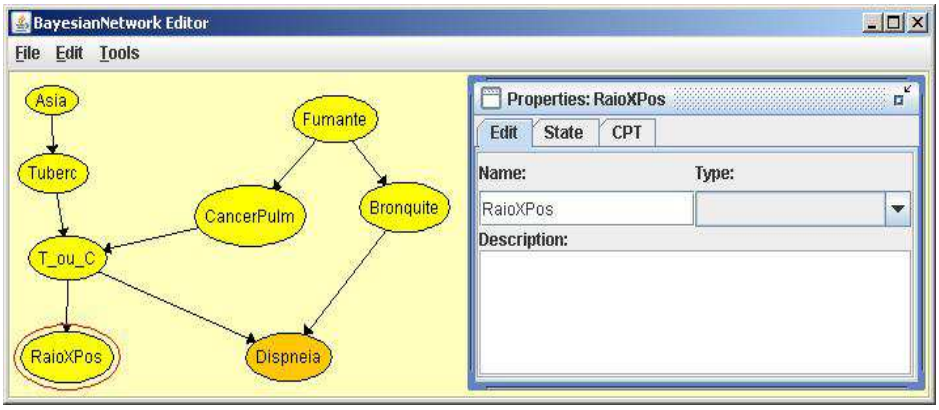

Figura 2.11: Ásia: clínica do torax, no *framework* JAmplia.

Proposto por LAURITZEN; SPIEGELHALTER (1990), este exemplo é conhecido como *Ásia*, formulado da seguinte forma: "Dificuldade de respirar (dispnéia) pode ser devida a tuberculose, câncer de pulmão, bronquite, mais de uma dessas doenças ou nenhuma delas. Uma visita recente a Ásia aumenta o risco de tuberculose enquanto fumar é conhecido ser um fator de risco para câncer de pulmão e bronquite. O resultado de um único raio X não permite distinguir entre câncer de pulmão e tuberculose, assim como também não permitir distinguir entre ambos, a presença ou ausência de dispnéia".

À direita da imagem, o retângulo branco apresenta as informações do nodo selecionado no momento. Ali, podemos editar todas suas informações, como nome, possíveis estados e a tabela de probabilidades condicionais.

2.3.5 Diagramas de Influência

A utilização de conjuntos de variáveis aleatórias, arcos e probabilidades pode não ser a forma mais adequada para representar um agente probabilístico. Seria difícil representar,

por exemplo, um valor inteiro a uma determinada decisão tomada pelo agente. Assim, diagramas de influência são utilizados para este tipo de representação, pois implementam estas funcionalidades estendendo as redes bayesianas, tornando-se muito importantes num processo de tomada de decisão com incerteza.

Introduzido em SHACHTER (1986), um diagrama de influência consiste de um grafo orientado acíclico $G = (N, E)$, onde N é o conjuntos dos nodos, formado por $P \bigcup D \bigcup \Psi$ e E o conjuntos dos arcos, sendo:

- P: conjunto dos nodos de probabilidade (elipses). Cada nodo tem uma tabela de probabilidades condicionais associada, dada por

$$\phi_A = P(A|pa(A)) \tag{2.6}$$

  onde $pa(A)$ é o conjunto dos nodos pais de A, podendo estes serem outros nodos de probabilidade ou de decisão. Se este conjunto for vazio (não possuir pai), a TPC é substituída pelas probabilidades *a priori* da variável em questão, sendo esta considerada uma variável de evidência;

- D: conjunto dos nodos de decisão (retângulos). Indica qual decisão deve ser tomada pelo agente, dados os estados das variáveis pais. Seus pais podem ser outros nodos de decisão ou nodos de probabilidade;

- Ψ: conjunto dos nodos de utilidade (losangos). O nodo deste tipo possui uma tabela contendo a descrição da utilidade como funções das variáveis associadas aos seus pais. Podem ter como pais nodos de decisão ou nodos de probabilidade.

Segundo FLORES (2005), as arestas condicionais incidem em nodos probabilísticos ou de utilidade e representam dependência probabilística. Caso o estado de uma variável não dependa do estado de outra variável, estas são ditas independentes entre si. Um nodo de decisão é pai de um nodo de probabilidade se o seu valor influencia a distribuição da variável aleatória. As arestas incidindo em nodos de decisão são arestas de informação, representando a precedência temporal e a disponibilidade da informação na hora de tomar a decisão. Um nodo de probabilidade é pai de um nodo de decisão se o valor da variável aleatória for conhecido na hora da decisão e puder influenciá-la. Um nodo de decisão é pai de outro nodo de decisão se a primeira decisão for realizada antes da segunda e a influenciar. Um nodo de utilidade representa o valor esperado da utilidade, dados os valores de seus pais. Seus nodos pais são todas as variáveis descrevendo as decisões e variáveis aleatórias que afetam diretamente o cálculo da utilidade.

2.3.6 Por que Utilizar Redes Bayesianas?

Existem diversas vantagens na utilização de redes bayesianas, ressaltando muitas de suas principais características:

- Permitem expressar assertivas de independência de forma visual e fácil de perceber;

- Representam e armazenam uma distribuição conjunta de forma econômica, explorando a esparcidade do relacionamento entre as variáveis;

- Tornam o processo de inferência eficiente computacionalmente;

- Permitem analisar grandes quantidades de dados, para, por exemplo, extrairmos conhecimentos úteis em tomada de decisões;

- Podem ser utilizadas em vários domínios: saúde (diagnóstico de doenças), indústria (controle de autômatos ou de robôs), computação e redes de comunicação (agentes inteligentes), marketing (mineração de dados, gestão da relação com os clientes), bancos e finanças (análise financeira, bolsa de valores); gestão (tomada de decisões, gestão de conhecimento e risco), etc.

Além destas vantagens, o uso de redes bayesianas é preferido frente a outros modelos pois apresenta algumas facilidades, como:

Aquisição de conhecimentos: é possível agrupar conhecimentos de diversas naturezas num mesmo modelo: dados históricos ou empíricos, observações e experiências (expressas na forma de regras lógicas, equações, estatísticas ou probabilidades subjetivas);

Representação de conhecimento: sua representação é explícita, intuitiva e compreensível para qualquer pessoa envolvida com o sistema, facilitando sua validação, eventuais modificações e utilização, pois a pessoa que a utilizará confia mais no modelo que ela tem o domínio do que o sistema tipo "caixa preta".

Multifuncionalidade: podemos utilizar o mesmo modelo para avaliar, prever, diagnosticar, ou otimizar as decisões, contribuindo para o reaproveitamento do sistema em outras funções;

Qualidade da oferta com relação aos programas: atualmente a pesquisa em redes bayesianas está em grande expansão. Diversas ferramentas estão presentes no mercado com funcionalidades relativamente evoluídas: aprendizagem de probabilidades, aprendizagem da estrutura da rede, possibilidade de integrar variáveis contínuas, etc.

2.4 Considerações

Neste capítulo foram estudadas as principais tecnologias utilizadas neste trabalho. Dentre os ambientes para programação de Sistemas MultiAgentes, optou-se pela utilização do Jason em conjunto da ferramenta de comunicação de agente Jade. Entre as ferramentas de desenvolvimento de redes bayesianas estudadas, JAmplia foi a escolhida.

As ferramentas utilizadas aqui foram escolhidas por serem facilmente extensíveis, facilitando a programação da proposta deste trabalho e o desenvolvimento de estudos de caso.

3 DESENVOLVENDO O *PLUGIN* COPA

A extensão proposta neste trabalho está disponível sob a forma de um *plugin*, denominado *COPA* (**CO**nhecimento **P**robabilístico em **A**gentes BDI). Todo o código-fonte e um guia de uso estão disponíveis à comunidade científica em `http://www.inf.ufrgs.br/~glkieling/copa.zip`. Este *plugin* contém a classe que os agentes do Jason devem utilizar e também a extensão da ferramenta de construção de Redes Bayesianas JAmplia.

Como descrito em capítulos anteriores, para alterar o modo que o Jason seleciona um plano para execução, teremos de cumprir três passos principais:

1. Implementar uma nova funcionalidade no JAmplia para o envio de mensagens via *JADE* a todos os agentes cadastrados na plataforma, para que estes recebam os valores de probabilidades das crenças da rede bayesiana;

2. Criar uma função na forma de ação interna no Jason para que o agente possa adicionar as novas crenças recebidas pelo JAmplia;

3. Construir um agente que estenda a classe de agente padrão utilizada no Jason para sobrescrever o método de seleção de planos.

Neste capítulo serão descritas como estas três modificações serão realizadas, explicando detalhadamente cada passo, as ferramentas utilizadas e limitações ainda presentes. Ao final deste capítulo, serão apresentados dois estudos de caso para exemplificar a utilização das novas funcionalidades.

3.1 JADE

A seção 2.1.7 descreve o *framework* JADE, apresentando as principais funcionalidades utilizadas neste trabalho.

JADE foi escolhido para a comunicação entre os agentes do COPA pois já é uma ferramenta consolidada na área de IA, sendo inclusive utilizada em diversos projetos em que o JAmplia está enquadrado. Além disso, apresenta importantes funcionalidades para o desenvolvimento deste trabalho, como interface gráfica, biblioteca de protocolos FIPA, transporte de mensagens e possibilidade de integração com diversos sistemas.

3.2 A Ferramenta JAmplia

Como dito na seção 2.3.4.5, o JAmplia é um ambiente onde podemos criar, editar e compilar redes bayesianas. Ao criar uma nova RB, o usuário pode executar diversas funções próprias de redes bayesianas, como moralização, triangularização, identificação de

cliques, criação de árvore de junção, entrada de evidências, entre outras. Neste trabalho, estas funções não serão abordadas, exceto a entrada de evidências, que passa executar uma nova ação: enviar uma mensagem a todos os agentes cadastrados na plataforma JADE contendo o valor de probabilidade das crenças da rede bayesiana.

A entrada de novas evidências pode ser feita selecionando a opção do menu *Tools -> Distribute Evidence*, opção também utilizada para manter a consistência da árvore de junção. Uma janela intitulada *Beliefs* aparecerá, apresentando todas as variáveis com seus respectivos estados. Clicando no estado em que se deseja acrescentar a evidência e clicando no botão *Inform*, esta evidência será criada, zerando as probabilidades dos outros estados. Para que as TPCs de todas as outras variáveis da rede sejam recalculadas, devemos clicar no botão *Compile*. Assim, a RB é recompilada e as novas probabilidades são apresentadas na tela.

Na figura 3.1 foi utilizado o exemplo apresentado na seção 2.3.4.5. Foi inserida a evidência que o paciente esteve na Ásia, indicado pelo estado $True$ na variável $Asia$ com o valor de 100%. Assim, todas as probabilidades das outras variáveis que tenham algum tipo de relação com esta são recalculadas.

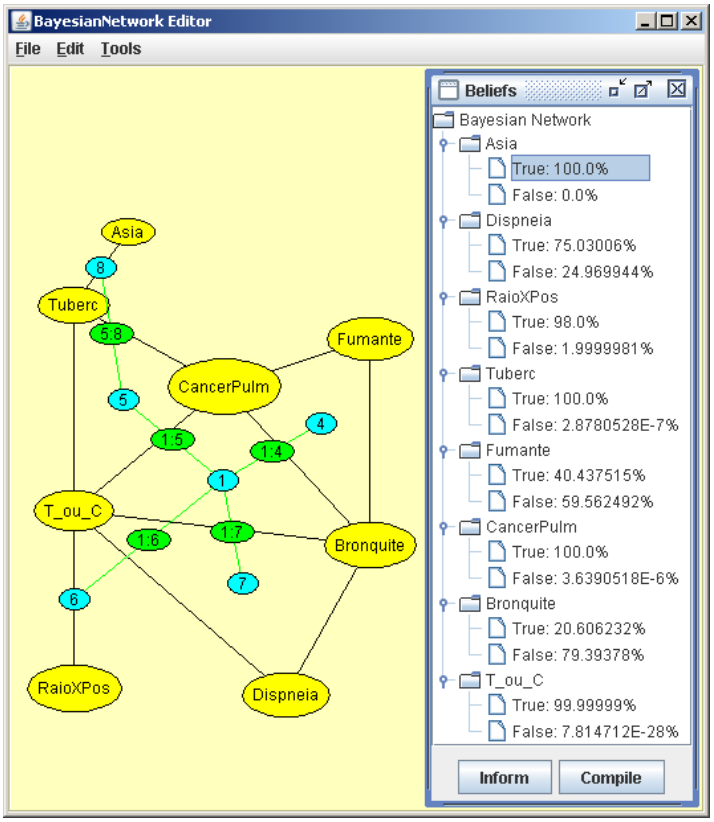

Figura 3.1: Gerando novas evidências no JAmplia

3.2.1 Modificando o JAmplia

Ao recompilar a RB, o método $distributeEvidence$, localizado na classe $JunctionTree$, executa todos os passos necessários para a recompilação da rede. Portanto, a função $sendBeliefs$ foi invocada neste ponto, recebendo como parâmetro a lista de variáveis da RB. Este método contém basicamente quatro passos:

1. Verifica se os nomes do nodo e seus estados são compostos por caracteres válidos.

Somente são aceitas letras de “a” até “Z” (minúsculas e maiúsculas), dígitos de 0 a 9 e sublinhado (_). O tamanho mínimo é 1 e o máximo 50 caracteres;

2. Recupera a probabilidade associada a esta variável;

3. Agrupa o nome da variável, seu estado e sua probabilidade no formato de uma crença na linguagem AgentSpeak;

4. Chama o método $broadcastBelief$, responsável pelo envio das mensagens.

O JAmplia não mais se comportará como um programa isolado, mas sim como um agente do tipo JADE, pois assim poderá enviar mensagens contendo a probabilidade de cada crença aos agentes construídos no Jason. A classe $BeliefSenderAgent$, detalhada adiante, foi construída para abrigar toda a lógica de comunicação do JAmplia no JADE.

Mesmo com todas estas modificações, o usuário do JAmplia ainda pode utilizá-lo como antes, executando a classe $graph.bayes.editor2.BNEditor$. Para rodar o JAmplia como um agente, a classe $jade.Boot$ deve ser executada. Entretanto, como o único meio de iniciar um agente é executando esta classe, o agente deve ser iniciado primeiro, invocando o construtor do JAmplia ao final do método $setup$. Os parâmetros definirão a localização do agente, a classe que o implementa, entre outras configurações. Os parâmetros a serem passados para que o sistema seja executado corretamente mudam conforme a ordem de execução dos agentes. Se o JAmplia é executado primeiro, os parâmetros devem ser diferentes de se for executado após os agentes desenvolvidos no Jason, pois o primeiro agente executado deve sempre iniciar o *container* principal, no qual os outros agentes serão registrados. Abaixo, seguem os parâmetros que devem ser utilizados caso o JAmplia seja o segundo a ser executado.

- $container$: indica que o JAmplia será inserido no pacote principal do JADE ($main$), onde estão cadastrados os agentes *ams*, *df*, *RMA* e *j_environment* (padrões do JADE), além daqueles construídos no Jason;

- $bn : graph.bayes.BeliefSenderAgent$: o nome do agente será bn e a classe a ser executada que o implementa é $BeliefSenderAgent$.

Caso o JAmplia seja o primeiro a ser registrado na plataforma do JADE e o ambiente do Jason o segundo, o primeiro item deve ser excluído. Além disso, este mesmo item deve ser adicionado como parâmetro do JADE na execução do Jason.

Além do método $setup$, a classe $BeliefSenderAgent$ ainda contém os métodos $takeDown$ e $broadcastBelief$. Este é invocado ao final do ciclo de vida do agente, desregistrando-o da plataforma; aquele adiciona ao agente um comportamento ($behaviour$) do tipo $OneShotBehaviour$, sendo executado somente uma vez. Todo comportamento deste tipo deve conter o método $action$, o qual contém as ações a serem executadas por este comportamento.

Ainda foi implementado outro comportamento, que é executado paralelamente aos outros. Este comportamento estende a classe $TickerBehaviour$ e é executado a cada 15 segundos. Seu objetivo é pesquisar quais agentes estão cadastrados na plataforma com a mesma ontologia do agente JAmplia no momento. Como agentes não são adicionados ou removidos muitas vezes durante a execução do sistema, acredita-se que este tempo seja adequado para pesquisa de novos agentes. Os agentes que corresponderem à pesquisa são adicionados em uma lista ($knownAgents$) para ser utilizada no envio das mensagens.

Quando o usuário seleciona a opção de distribuição de evidências, o método $broadcastBelief$ é chamado, recebendo como parâmetro a mensagem a ser enviada. O conteúdo desta mensagem será uma lista com todas as crenças, separadas por vírgula, anexadas as suas respectivas anotações de probabilidade, já na sintaxe AgentSpeak. Todos os agentes da lista $knownAgents$ são adicionados como destinatários. O formato do conteúdo da mensagem é como apresentado abaixo:

```
crenca_1(estado_1)[p(x)],...,crenca_1(estado_j)[p(x)],
...,crenca_n(estado_m)[p(z)]
```

No exemplo da figura 3.1, a seguinte mensagem foi enviada aos agentes cadastrados na plataforma Jade: “ asia(true)[p(1.0)],asia(false)[p(0.0)],dispneia(true)[p(0.7503006)], dispneia(false)[p(0.24969944)],raioxpos(true)[p(0.98)],raioxpos(false)[p(0.01999998)], tuberc(true)[p(1.0)], tuberc(false)[p(2.8780527E-9)],fumante(true)[p(0.40437517)], fumante(false)[p(0.5956249)],cancerpulm(true)[p(1.0)], cancerpulm(false)[p(3.6390517E-8)],bronquite(true)[p(0.20606232)], bronquite(false)[p(0.7939378)],t_ou_c(true)[p(0.99999994)], t_ou_c(false)[p(7.814712E-30)]”

3.3 Implementações no Agente do Jason

Para utilizar as novas funcionalidades propostas neste trabalho, é necessário criar uma classe de agente que estenda a classe $jason.asSemantics.Agent$, a ser utilizado pelo agente implementado no Jason. Este agente abrigará a função $selectOption$, detalhada a seguir. Exemplos de como estes agentes podem ser criados serão vistos na seção 3.5.

3.3.1 Modificando a Seleção de Planos

Como já visto na seção 2.2.5 e na figura 2.4, o passo 8 é responsável pela seleção do plano a ser executado pelo agente. Portanto, o ciclo de raciocínio do Jason não sofrerá alteração, mas sim o algoritmo do passo 8. Mais especificamente o método $selectOption$ da classe $jason.asSemantics.Agent$. Este método recebe como parâmetro a lista de planos aplicáveis, ou seja, a lista de todos os planos que o agente acredita serem atingíveis, retornando o plano a ser executado em um ciclo de execução do Jason. Assim como diversos outros, este método foi inicialmente implementado de maneira que o usuário possa sobrescrevê-lo, inserindo trechos de código a serem executados em detrimento da implementação original. Abaixo, é descrita sua implementação:

```
1  Option selectOption(Lista options) {
2    Se options tem somente um elemento Então
3      retorna o primeiro;
4    Para cada crença na BB
5      Se o literal tiver uma anotação "p" Então
6        Associa este literal com p na lista bbBeliefs;
7    Para cada plano aplicável
8      Se contexto vazio Então
9        probTotal := Threshold;
10     Para cada crença do plano
11       Para cada crença na BB
12         Se crença do plano = crença da BB Então
13           probTotal := probTotal * probBelief;
14     Se probTotal > probGreater Então
15       probGreater := probTotal;
```

```
16          bestOption := i;
17      retorna options[bestOption];
```

Apesar de ter sido construído em Java, este algoritmo foi descrito em português estrutural para melhor entendimento do leitor. A implementação real é bem mais complexa e abrange diversos detalhes aqui suprimidos. Caso a lista recebida como parâmetro tenha somente um elemento, não há outra alternativa além de escolhê-lo (linhas 2 e 3). A seguir, a lista $bbBeliefs$ é preenchida com todas os literais da base de crenças do agente que tenham a anotação p, a qual indica o grau de incerteza que o agente tem sobre esta (linhas 4 a 6). A partir da linha 7 até a linha 16, cada plano é analisado. Caso seu contexto for vazio (aplicável em qualquer situação), a probabilidade dada a ele será o valor de $Threshold$ (limiar). Este valor representa que, se nenhum plano tiver um valor maior que o limiar, este plano será o escolhido. O valor desta variável não obedece a qualquer heurística, sendo atribuído atualmente a 0.25, pois se acredita ser um valor médio para as probabilidades totais dos planos a serem analisados. Nas linhas 10 a 13 é recuperado o valor de probabilidade para cada crença do contexto do plano em questão, multiplicando tais valores e seu valor final armazenado em $probTotal$. Em seguida (linhas 14 a 16), é analisado se o plano do momento detém a maior probabilidade total. Ao final do algoritmo, é retornado o plano com maior probabilidade total (linha 17).

Note que o símbolo de comparação para troca do plano com maior probabilidade no momento (linha 14) é $>$ (maior). Portanto, caso dois planos tenham a mesma probabilidade ao final, o plano a ser escolhido é aquele mais próximo do início da lista, ou seja, aquele que aparecer primeiro no código-fonte do agente.

Também é importante ressaltar que atualmente é assumido que o contexto do plano será composto somente por conectores "$\&$" (conjunção), utilizando, por consequência, a multiplicação entre suas variáveis (linha 13). Obviamente, este fato foge da realidade, pois seguidamente são utilizados outros conectores, como "|" (disjunção) e "$\sim$" (negação). Assim, deveriam ser utilizados os seguinte cálculos: $p(A|B) = p(A) + p(B) - p(A) * p(B)$ para calcular-se a probabilidade de ocorrência do evento A, dada a evidência B, e $\sim P(A) = 1 - P(A)$ para calcular-se a probabilidade da não ocorrência do evento A, respectivamente. Entretanto, a complexidade do algoritmo aumentaria bastante, pois existiria mais de um tipo de operador, gerando uma árvore de conectores, assim dificultando a análise do contexto dos planos por um algoritmo. Portanto, optou-se por implementar somente a lógica do operador de conjunção, deixando a implementação dos outros operadores como trabalho futuro.

3.3.1.1 Compatibilidade com Código Legado

Caso nenhum plano tenha qualquer crença probabilística, o *COPA* retorna o mesmo resultado do algoritmo original do Jason. Como consequência, é mantida a compatibilidade com agentes que não foram inicialmente programados com o *plugin COPA*.

Essa é uma característica muito importante, pois não é em todos os casos em que o uso deste *plugin* é necessário ou até mesmo aconselhável, visto que nem sempre é possível modelar o ambiente com crenças probabilísticas. Até mesmo nos estudos de caso (apresentados na seção 3.5), vários planos foram implementados sem crenças probabilísticas em seus contextos. Caso a compatibilidade não tivesse sido mantida, certamente haveria consequências desastrosas ou até mesmo não-determinísticas na manipulação de planos sem crenças probabilísticas em agentes que foram construídos utilizando a abordagem original de seleção de planos.

3.3.1.2 Complexidade

A complexidade do algoritmo de seleção de planos atual é O(1), pois sempre retorna o primeiro plano da lista. A complexidade do algoritmo do COPA é calculada pela seguinte fórmula:

$$p \times b \times (\Sigma_{c=1}^{p}\ c) \tag{3.1}$$

onde p é o número de planos aplicáveis analisados, b é a quantidade de crenças presentes na base de crenças do agente, e c é a contagem de crenças presentes no contexto de cada plano. Portanto, caso tenhamos um grande número de planos aplicáveis, muitas crenças na base e planos com vários literais em seu contexto, este algoritmo pode vir a demorar um tempo considerável para responder. Esta demora talvez seja crucial na utilização deste *plugin*, pois espera-se que o agente tome decisões muito rapidamente. Como os agentes são simulações de agentes humanos, os agentes devem ter uma velocidade muito parecida com a de nosso raciocínio, não lhes dando o direito de demorar muito tempo para agir, podendo ser a diferença entre o sucesso ou falha do objetivo principal.

3.3.2 Criação de Ações Internas

Felizmente, algumas das ações internas necessárias já tinham sido implementadas pelos desenvolvedores do Jason em exemplos de sistemas que são baixados juntamente com o programa do próprio Jason. As ações são implementadas como classes, como visto na seção 2.2.3.3.

As ações internas criadas foram $register$ e $addBelief$. A primeira foi baseada no exemplo $book - trading$[1], que apresenta como dois ou mais agentes (vendedores e compradores) podem negociar a compra de livros via JADE. Esta ação cadastra o agente na plataforma, associando-o a um determinado serviço (classe $ServiceDescription$). Os dois parâmetros são o nome e tipo de serviço que este agente quer associar-se na plataforma. Este serviço deve ser do mesmo tipo utilizado pelo agente do JAmplia em sua pesquisa, descrita na seção 3.2.1. A segunda ação interna foi baseada no exemplo $tell-rule$[2], onde dois agentes trocam mensagens via JADE, contendo regras presentes em suas bases de crenças. Em vez de regras, a ação $addBelief$ adiciona crenças, as quais são recebidas através do plano $kqml\_received$, como visto na seção 2.1.7.

3.4 Visão Geral

Como já mencionado no começo deste capítulo, e como podemos agora ver na figura 3.2, a implementação do *COPA* foi dividida em três módulos: (a) modificação do JAmplia, (b) criação de ações internas no Jason e (c) alteração do algoritmo de seleção de planos do Jason.

Nesta figura, observamos que as classes $register$, $addBelief$ e $getBelief$, presentes no pacote $copa.ia$, estendem a classe $DefaultInternalAction$, a fim de implementar novas funcionalidades para os agentes do Jason (a). A classe a ser utilizada pelos agentes desenvolvidos é $COPAAgent$, que estende a classe $Agent$ do Jason. A classe $ProbabilisticLiteral$ é utilizada pela $COPAAgent$ para armazenar as crenças probabilísticas do sistema (b). Finalmente, a classe $BeliefSenderAgent$, que estende a classe

[1] diponível em `http://jason.sourceforge.net/mini-tutorial/jason-jade`
[2] disponível em jason/demos/tell-rule

Agent do JADE, possibilita o JAmplia enviar mensagens aos agentes do Jason através da plataforma de comunicação de agentes JADE (c).

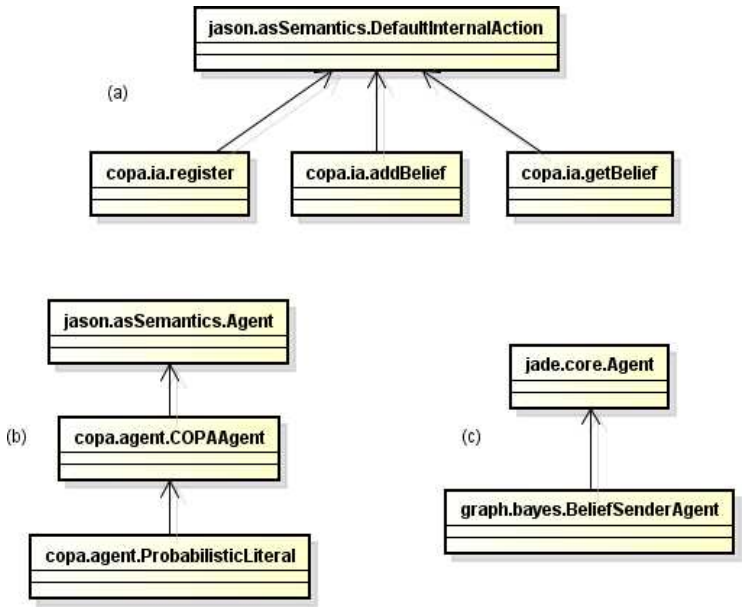

Figura 3.2: Diagrama das classes do *COPA*

3.5 Estudos de Caso

Nesta seção, mostraremos dois exemplos práticos onde a incerteza está presente e os agentes devem trabalhar com ela. Primeiramente, apresentaremos o problema, produziremos uma possível solução e finalmente discutiremos suas vantagens e desvantagens.

3.5.1 Futebol ou Cinema

Este exemplo é o mesmo apresentado na seção 2.2.7, agora adaptado para trabalhar com crenças probabilísticas. O agente deve escolher qual a melhor opção para o momento: ficar em casa, jogar futebol ou ir ao cinema. O código-fonte do agente segue abaixo.

```
1  !registerDF.
2  !atividade.
3
4  +!registerDF
5    <- ms.ia.register("JAmplia-BN", "JADE-Agent").
6  +!kqml_received(Sender, Performative, Belief, MsgId)
7    <- ms.ia.addBelief(Belief);
8       !atividade.
9  +!atividade
10   <- .print("Em casa.").
11 +!atividade[source(self)]
12   : clima(ensolarado) & jogadores(sim)
13   <- .print("Futebol").
14 +!atividade[source(self)]
15   : clima(chuvoso) & jogadores(nao)
16   <- .print("Cinema").
```

O primeiro objetivo do agente (linha 1) é executar o plano $registerDF$, que registra o agente na plataforma do JADE através da ação interna $register$, detalhada anteriormente

na seção 3.3.2. Na linha 2, o segundo objetivo ($atividade$) é adicionado. Este objetivo tem três planos relevantes (linhas 9, 11 e 14). Como inicialmente a base de crenças está vazia, o agente não tem qualquer informação a respeito das crenças $clima$ e $tempo$. Portanto, somente o plano da linha 9 é aplicável. Quando o agente recebe uma mensagem via JADE em sua caixa de correio, o objetivo $kqml\_received$ é adicionado, fazendo com que o plano da linha 6 seja executado.

Neste exemplo, podemos observar o agente escolhendo o plano que tem a maior probabilidade entre suas crenças do seu contexto. É interessante notar que não há necessidade de colocarmos probabilidade alguma no código-fonte do agente. O *COPA* fica responsável por recuperar as probabilidades de cada crença dos planos. Após a primeira mensagem que o JAmplia mandar para os agentes do Jason, estes já terão os três planos presentes na lista de planos aplicáveis. Como o plano da linha 9 simplesmente não tem contexto, é atribuído a ele o valor de $Threshold$. Ele só não será escolhido para execução caso a multiplicação entre as probabilidades das crenças $clima(chuvoso)$ e $jogadores(nao)$ ou $clima(ensolarado)$ e $jogadores(sim)$ sejam menores do que o $Threshold$.

3.5.2 Agente Investidor na Bolsa de Valores

A área da economia está repleta de informações parciais, incompletas e ambíguas. Existem diversos jornais, revistas, artigos, entre outras fontes de informação que, muitas vezes, discorrem sobre uma mesma notícia, porém com pontos de vista conflitantes. Além disso, nem sempre é possível confirmar a veracidade da informação apresentada. Com tantas informações incertas a avaliar, e pelo fato de muito dinheiro estar "em jogo", por vezes os investidores acabam tomando decisões baseando-se não somente na razão, mas também com a emoção. Quando levamos em consideração os sentimentos, nem sempre conseguimos analisar com clareza este tipo de situação, a qual requer raciocínio prático, lógico e calculista. Assim, caso conseguíssemos construir um agente artificial contendo todo nosso conhecimento sobre a área, ele certamente faria as melhores decisões, pois não consideraria qualquer emoção, maximizando os ganhos.

A área de computação afetiva (PICARD, 1997) estuda a importância das emoções na tomada de decisões de agentes, fator extremamente relevante na área de economia. Diferentes perfis de agentes podem levar a diferentes decisões, alterando seus ganhos. Entretanto, não se estudou como esta linha de pesquisa poderia auxiliar neste estudo de caso, pois seu objetivo é apenas exemplificar a utilização do *COPA*.

Este estudo de caso é baseado no modelo apresentado em GLUZ (2008) e KIELING (2008). Originalmente foi desenvolvido para exemplificar como o conhecimento probabilístico poderia ser inserido na linguagem AgentSpeak(XL), definindo novos operadores na linguagem Prolog, dando origem à linguagem AgentSpeak(PL). Neste estudo de caso, não definiremos novos operadores, tampouco criaremos uma nova linguagem, mas estenderemos este modelo de exemplo através da inserção de variáveis na rede bayesiana, representando algumas características do agente investidor, e não somente características do ambiente econômico. Embora simples, este exemplo pode ser utilizado por qualquer pessoa interessada em investir na bolsa de valores.

3.5.2.1 Definição do Estudo de Caso

Suponhamos que um agente *investidor* possua uma quantia em dinheiro previamente definida disponível no banco, ele deve decidir se vale mais a pena investir parte do seu capital na bolsa de valores, retirar o dinheiro que já havia sido investido ou ainda não realizar qualquer transação. O agente *corretor* recebe as informações sobre o estado atual

da economia, calcula a cotação das ações negociadas da bolsa no dia e repassa esta informação ao investidor. Com base em todas estas variáveis, este agente decide qual o plano mais adequado para o momento: investir, resgatar ou fazer nada.

O investidor ($investor$) somente toma uma decisão quando recebe a informação de nova variação da bolsa de valores do corretor ($broker$). Por sua vez, o corretor somente calcula a variação do dia quando recebe novas informações da rede bayesiana. Estas informações são enviadas quando a RB é compilada, ação executada exclusivamente pelo usuário.

A única possibilidade de investimento do agente é aquele regido pelo índice de uma bolsa de valores fictícia. Neste estudo de caso, o lucro do agente é diretamente proporcional à variação deste índice, ou seja, quanto maior este for, mais lucro o agente terá.

O modelo de investimento proposto é simplificado, objetivando servir de exemplo básico para a demonstração de uma possibilidade de representação de modelos probabilísticos e possibilidade de escolha entre diversos planos possíveis pelo *plugin COPA*. Para facilitar o entendimento deste estudo de caso, somente um agente investidor foi adicionado no sistema. Entretanto, n agentes deste tipo podem coexistir, tomando decisões independentemente uns dos outros.

3.5.2.2 Modelagem do Estudo de Caso

Este estudo de caso apresenta um modelo para aplicação de investimento em bolsa de valores, sendo basicamente constituído de duas partes, a serem detalhadas adiante:

1. Rede bayesiana: um agente investidor foi modelado utilizando a ferramenta *JAmplia*;

2. Agentes: investidor e corretor, construídos no Jason.

Para a construção da rede bayesiana, foram levados em consideração tanto características do mercado de ações quanto de personalidade dos investidores. A rede bayesiana desenvolvida é apresentada na figura 3.3, construída no *framework* JAmplia.

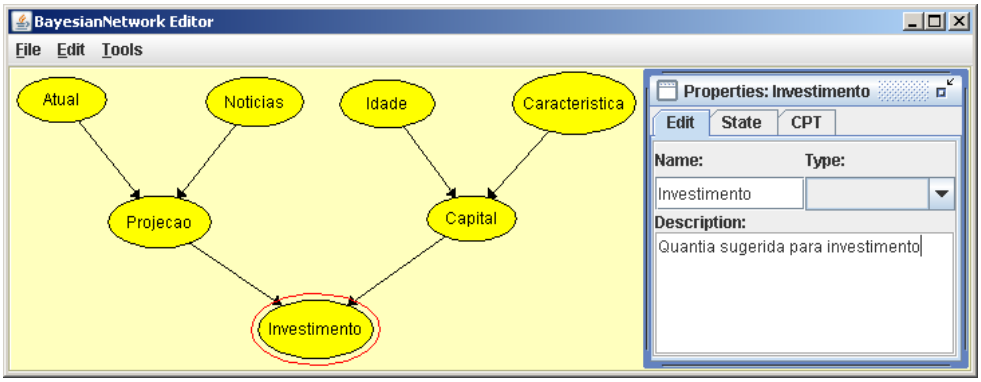

Figura 3.3: RB para o estudo de caso do Agente Investidor na Bolsa de Valores

Probabilidades são definidas para a tendência futura do índice de crescimento de uma bolsa de valores hipotética. Embora a realidade apresente infinitos estados, este índice é classificado em apenas dois estados possíveis: *bom* ou *ruim*. A tendência atual deste índice é representada pela variável de evidência *Atual*. A variável *Noticias* representa a presença ou não de novas notícias importantes (quase sempre consideradas más notícias

Tabela 3.1: Modelo quantitativo da variável *Investimento*

Capital	***Projecao***	***Investimento***	***P(Investimento)***
muito	*bom*	*alto*	*0,9*
muito	*bom*	*medio*	*0,1*
muito	*bom*	*baixo*	*0*
muito	*ruim*	*alto*	*0,2*
muito	*ruim*	*medio*	*0,4*
muito	*ruim*	*baixo*	*0,4*
pouco	*bom*	*alto*	*0,4*
pouco	*bom*	*medio*	*0,5*
pouco	*bom*	*baixo*	*0,1*
pouco	*ruim*	*alto*	*0,1*
pouco	*ruim*	*medio*	*0,25*
pouco	*ruim*	*baixo*	*0,65*

no contexto de investimentos) que possam alterar o índice. A variável *Projecao* estima a nova tendência de crescimento do índice da bolsa, com base em *Atual* e *Noticias*.

Já para as características dos agentes investidores, os nomes dos nodos também são muito intuitivos em seu significado. A variável *Idade* indica a faixa etária do investidor. Como é inviável representar todas as idades possíveis (ou pelo menos um grande número de intervalos), somente dois estados foram abordados: jovem e adulto. O nodo *Caracteristica* representa como o investidor normalmente encara a possibilidade de novos investimentos. Existem dois possíveis perfis: *arrojado* e *conservador*. Baseado nos valores destas duas variáveis, temos em *Capital* uma indicação de quanto o agente está disposto a investir: *muito* ou *pouco*, com relação ao total do dinheiro disponível em caixa. Caso o estado seja *pouco*, pode indicar que o agente esta disposto a retirar parte do dinheiro, em vez de investir ainda mais.

Baseando-se em *Projecao* e *Capital*, a variável *Investimento* estima a tendência geral de investimento dos agentes econômicos na bolsa de valores. Existem apenas três classificações de como a tendência de porte de investimento dos agentes é classificada: *alto*, *medio* e *baixo*.

A RB apresentada na Figura 3.3 forma seu modelo qualitativo, onde são descritas apenas as relações condicionais entre as diversas variáveis probabilísticas. As informações de probabilidades condicionais formam o modelo quantitativo da rede e definem a "força" (probabilidade) com que os nodos pais influenciam os nodos filhos dentro da rede (GLUZ, 2008). A definição do modelo quantitativo para a variável *Investimento* é apresentada na tabela 3.1. Os modelos quantitativos para o restante das variáveis são apresentados no anexo A. Os valores das probabilidades são totalmente fictícios e não foram baseados em nenhum estudo específico, tampouco no conhecimento de um especialista na área, mas sim no conhecimento do autor deste trabalho.

A definição do ambiente no Jason foi feita no arquivo *stockExchange.mas2j*, como pode ser visto na figura 3.4. A comunicação entre todos os agente do ambiente é feita através da ferramenta $Jade$, definido na linha 2. O ambiente do sistema ($environment$, linha 3), é onde foram implementados alguns métodos de apoio. Os agentes investidor ($investor$) e corretor ($broker$) são definidos nas linhas subsequentes, definidos com a classe $copa.agent.COPAAgent$. Os códigos-fonte destes agentes estão descritos em sua totalidade no anexo B.

```
MAS stockExchange {
    infrastructure: Jade
    environment: stock.env.SEEnv
    agents: investor agentClass copa.agent.COPAAgent;
            broker   agentClass copa.agent.COPAAgent;
}
```

Figura 3.4: Arquivo de configuração do estudo de caso do Agente Investidor na Bolsa de Valores

Começaremos apresentando o agente corretor. A única função deste agente é calcular a cotação da bolsa de valores em um dado momento, com base nas crenças enviadas pela rede bayesiana. Além dos planos de apoio, como o de registro no ambiente do JADE e o de recebimento de mensagem, este agente tem mais cinco planos, todos batizados como $calcula$, responsáveis pelo cálculo da cotação da bolsa. Cada plano representa um estado da economia. Multiplicando-se o valor probabilístico de cada crença para cada plano, temos a probabilidade total do plano. O plano que tiver a maior probabilidade total será o escolhido a ser executado. Os cinco estados modelados foram:

1. **Melhor situação**: $Atual$ (estado $ruim$), $Projecao$ (estado bom) e $Noticias$ (estado $otimas$), pois se espera que quando a economia está em baixa, o valor das ações também esteja baixo. Como as novas notícias são muito boas e a projeção é boa, a tendência é o valor das ações subirem bastante, maximizando o lucro do agente;

2. **Pior situação**: $Atual$ (estado bom), $Projecao$ (estado $ruim$) e $Noticias$ (estado $pessimas$), pois se espera que quando a economia está em alta, o valor das ações também esteja alto. Como as novas notícias não são boas e a projeção é ruim, a tendência é o valor das ações caírem bastante, dando prejuízo ao agente;

3. **Situação boa**: $Atual$ (estado bom), $Projecao$ (estado bom) e $Noticias$ (estado $otimas$), pois se espera que a economia estando em alta, com boa projeção e boas notícias, a cotação tende a apresentar leve alta;

4. **Situação ruim**: $Atual$ (estado $ruim$), $Projecao$ (estado $ruim$) e $Noticias$ (estado $pessimas$), pois a economia estando em baixa, com projeção ruim e más notícias, é esperado que a cotação apresente leve queda;

5. **Situação indeterminada**: este plano somente será escolhido para execução quando não houver informações suficientes para determinar o estado da economia, ou seja, quando alguma das variáveis dos planos anteriores não estiver presente na base de crenças do agente, ou ainda quando o valor total de probabilidade de todos os planos anteriores for menor do que o valor do limiar;

Dados os cinco planos do agente corretor, o plano a ser escolhido sempre será aquele que tiver as maiores probabilidades no geral, conforme explicado na seção 3.3.1. Na melhor situação, por exemplo, as variáveis com probabilidade mais alta são *Atual* (com estado *ruim*), *Projecao* (com estado *bom*) e *Noticias* (com estado *otimas*), pois assumimos que o agente terá ganhos mais altos quando o valor das ações está baixo, porém a tendência é de forte crescimento. Assim, um número aleatório entre 0 e 1 é calculado e multiplicado por 2, a fim de ser um valor mais expressivo. O resultado deste cálculo será a variação da bolsa de valores. No pior caso modelado, onde a situação atual é boa, a

projeção é ruim e as notícias são péssimas, o mesmo cálculo é feito, porém o resultado será negativo. A figura 3.5 mostra a implementação do plano na melhor situação.

```
25 // Melhor situacao
26 +!calcula
27     :   atual(ruim) & projecao(boa) & noticias(otimas)
28     <-  .random(R);
29         -+variacao(R * 2);
30         copa.ia.getBelief(variacao(_), V);
31         .print("Hoje teve variacao POSITIVA ", V);
32         .send(investor, tell, V).
```

Figura 3.5: Implementação do agente corretor na melhor situação possível para investimento

Já o agente investidor começa sua execução com algumas crenças iniciais, como o valor presente em caixa e o valor investido no momento. Este agente também recebe todas as atualizações das crenças da rede bayesiana, porém somente toma alguma decisão sobre investimento ao receber a mensagem proveniente do corretor, contendo a cotação atualizada da bolsa de valores. Ao finalmente receber esta informação, o investidor decidirá qual o plano mais adequado para o momento. Existem cinco possibilidades de ação, a saber:

1. **Investir bastante**: caso o agente acredite ser uma excelente oportunidade para ter altos ganhos, investindo 25% do total presente em caixa;

2. **Retirar bastante**: caso o agente acredite na queda da cotação da bolsa de valores, retirando 25% do total presente no investimento;

3. **Investir um pouco**: o investidor pode acreditar que existe uma boa perspectiva de crescimento, mas não estar certo sobre isso. Portanto, investirá somente 10% do total presente em caixa;

4. **Retirar um pouco**: o investidor pode acreditar ser mais provável que a cotação da bolsa tenha uma queda, mas não estar certo sobre isso. Portanto, retirará somente 10% do total presente no investimento;

5. **Fazer nada**: o investidor também pode acreditar que a decisão mais sábia é não fazer qualquer uma das ações acima.

3.5.2.3 Executando o Estudo de caso

Agora vamos colocar os agentes para funcionar e analisar suas reações e decisões. Primeiramente, executamos o projeto desenvolvido no Jason, clicando no botão *Run MAS* da interface. Duas janelas serão criadas: o console do sistema (figura 3.6), onde são impressas todas as informações geradas pelos agentes; e a interface de controle do JADE, na qual podemos gerenciar e visualizar todos os agentes cadastrados na plataforma. Na figura 3.7, podemos ver que além dos próprios agentes do JADE (*RMA*, *ams*, *df* e *j_environment*), os agentes deste estudo de caso também estão cadastrados: *investor* e *broker*, prontos para receber as crenças probabilísticas a serem futuramente enviadas pelo agente da rede bayesiana.

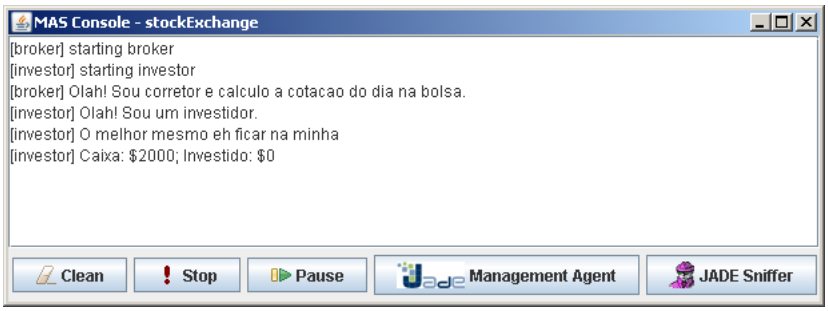

Figura 3.6: Console do sistema ao executar o estudo de caso

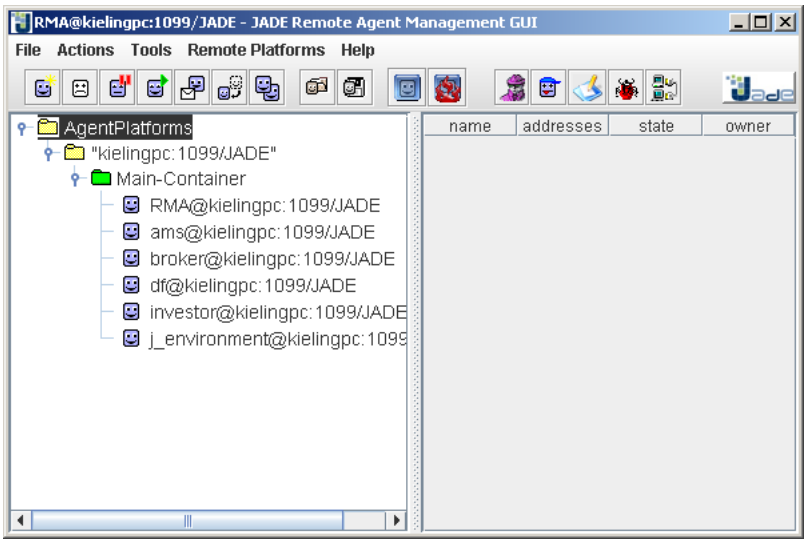

Figura 3.7: Interface de gerenciamento do JADE para este estudo de caso

Para iniciar o *JAmplia* e cadastrar um agente na plataforma JADE, a seguinte linha de comando deve ser executada, conforme visto na seção 2.1.7:

$$java\ jade.Boot\ -container\ server : graph.bayes.BeliefSenderAgent$$

Assim, o agente intitulado $server$ é registrado na plataforma JADE e uma nova janela é criada: o editor de redes bayesianas *JAmplia*. Agora, podemos abrir a RB modelada na figura 3.3 para dar sequência ao estudo de caso. Para começar a enviar as crenças probabilísticas aos agentes cadastrados na plataforma JADE, devemos selecionar a opção *Tools* do menu e em seguida *Distribute Evidence*. Este comando executa as seguintes ações sobre a rede bayesiana: moralização, triangularização, identificação de cliques e construção da árvore de junção, resultando a árvore à esquerda da figura 3.8. À direita, podemos visualizar o cálculo probabilístico para cada variável para o estado atual do ambiente.

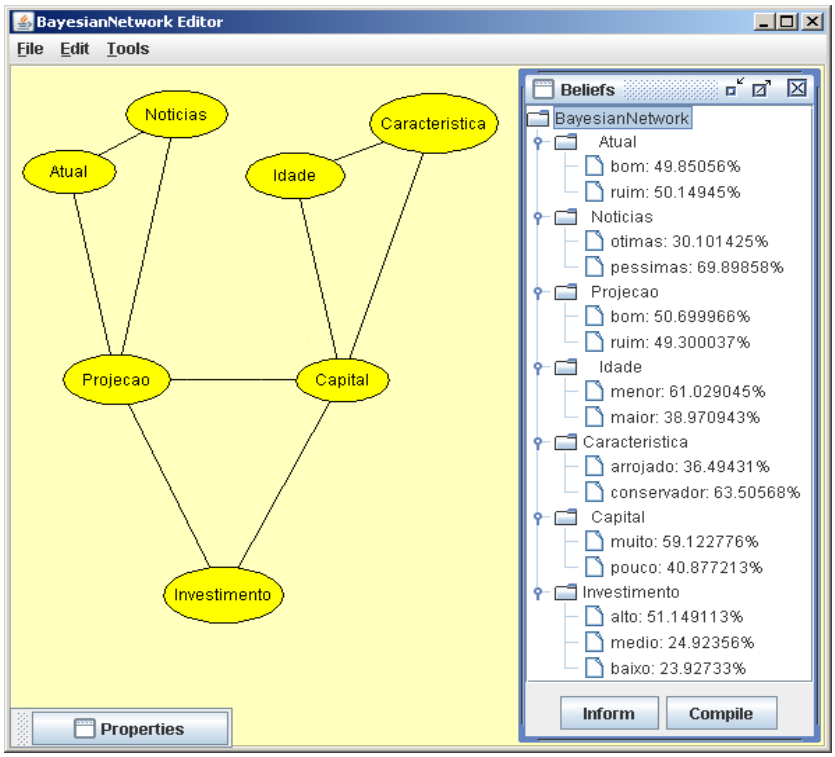

Figura 3.8: Rede bayesiana em execução no JAmplia

Cada vez que o botão *Compile* da janela *Beliefs* é clicado, o agente $server$ agrupa todas as crenças da rede em uma lista e as envia para todos os agentes da plataforma, no caso somente os agentes $investor$ e $broker$. Ao receber esta lista, o agente $investor$ adiciona todas as crenças à sua base, enquanto que o agente $broker$ calcula a cotação da bolsa de valores neste estado do ambiente. Ao finalizar este cálculo, o corretor envia ao(s) investidor(es) o valor da cotação. De posse desta nova informação recebida, agora um agente $investor$ pode atualizar seus ganhos (caso tenha algum valor investido) e decidir por realizar alguma ação sobre seu investimento. Fazendo uma analogia com o mundo real, é como se o investidor pudesse realizar transações somente no começo ou ao final do dia, pois o agente só decide qual ação realizar quando recebe a cotação da bolsa de valores do dia.

Um exemplo de mensagem trocada entre os agentes $server$ e $investor$ é apresentada a seguir.

(**INFORM**
:**sender** (agent-identifier :name server@kielingpc:1099/JADE
:addresses (sequence http://kielingpc:7778/acc))
:**receiver** (set (agent-identifier
:name broker@kielingpc:1099/JADE
:addresses (sequence http://kielingpc:7778/acc)) (agent-identifier
:name investor@kielingpc:1099/JADE
:addresses (sequence http://kielingpc:7778/acc)))
:**content** "atual(bom)[p(0.4985056)],atual(ruim)[p(0.50149447)],
noticias(otimas)[p(0.30101424)],noticias(pessimas)[p(0.6989858)],
projecao(bom)[p(0.5069997)],projecao(ruim)[p(0.4930004)],
idade(menor)[p(0.61029047)],idade(maior)[p(0.38970944)],
caracteristica(arrojado)[p(0.3649431)],caracteristica(conservador)[p(0.6350568)],

capital(muito)[p(0.59122777)],capital(pouco)[p(0.4087721)],
investimento(alto)[p(0.5114911)],investimento(medio)[p(0.2492356)],
investimento(baixo)[p(0.2392733)]"
)

As palavras em negrito são os principais marcadores da mensagem: $INFORM$ indica que a mensagem é do tipo informativa; $sender$ identifica o agente que está enviando a mensagem; $receiver$ contém os dados necessários para o JADE identificar na plataforma o conjunto de agentes que receberão a mensagem; $content$ armazena a informação transmitida na mensagem.

Caso não haja nenhuma evidência sobre o estado da economia, o plano que representa uma pequena variação **negativa** será o escolhido para calcular a cotação da bolsa de valores. Conforme a rede é recompilada, ou surgem novas evidências, as probabilidades são recalculadas. Como estes valores são muitas vezes alterados, outro plano pode vir a ser o escolhido. Neste caso, após algumas compilações, o plano que representa cotação **positiva** passa a ser o escolhido.

Como nas primeiras compilações o agente investidor não vê um ambiente propício para novos investimentos e o total de investimentos está zerado, o plano escolhido é aquele no qual o agente não realiza qualquer transação. Caso surjam novas evidências ou as probabilidades sejam levemente alteradas, é possível, ou até provável que outro plano venha a ser escolhido para execução.

Realizar testes com todos os valores reais é possível, porém muito trabalhoso, pois teríamos de inserir informações completas do estado da economia de todos os dias em um determinado período de tempo. É importante ressaltar que a utilização do *COPA* em ambientes reais é possível. Entretanto, um sistema de coleta de informações teria de ser implementado para simular e validar o comportamentos dos agentes neste estudo de caso. Este sistema não foi implementado neste trabalho, portanto é sugerido como trabalho futuro.

3.6 Comparativo

Antes de perguntarmo-nos como seria a implementação dos estudos de caso sem a utilização de crenças probabilísticas, cabe questionar sua viabilidade. Teríamos de utilizar probabilidades no contexto de seleção de intenções, associando-as ao trigger dos planos, assim como em CALCIN (2007). Isto obriga o programador a definir de forma antecipada em quais situações cada plano será executado, comprometendo a busca de comportamentos diferenciados.

Para tentar emular a mesma capacidade do *COPA* no Jason com seu algoritmo atual de seleção de planos, reimplementamos o estudo de caso do Agente Investidor na Bolsa de Valores. Anotações foram utilizadas para definir um valor mínimo que a probabilidade associada a cada crença deve ter para tornar-se um plano aplicável. A figura 3.9 apresenta como o plano de melhor situação possível para o agente $investor$ pode ser implementado.

```
54 +!acao[source(self)]
55     :   capital(muito)[p(PC)] & (PC > 0.7) & projecao(bom)[p(PP)] & (PP > 0.7)
56         & caixa(C) & (C > 50)
57     <-  ?investido(I);
58         -+caixa(C * 3/4);
59         -+investido(I + (C * 1/4));
60         !imprimiValores.
```

Figura 3.9: Plano que representa a melhor situação para o investidor sem crenças probabilísticas

Note que este plano somente será aplicável se a probabilidade associada aos estados $muito$ e bom das variáveis $capital$ e $projecao$, respectivamente, for acima de 70%. Este fato restringe bastante a possibilidade deste plano ser executado, visto que as probabilidades iniciais raramente são acima desta percentagem. Caso diminuíssemos este limiar, provavelmente este plano seria aplicável em um número muito grande de situações, não dando chance de execução aos outros planos (caso esteja nas primeiras posições), mesmo sendo aplicáveis.

4 CONSIDERAÇÕES FINAIS

A representação de conhecimento é um dos desafios da área de Inteligência Artificial, pois esta é uma propriedade humana muito difícil de ser representada de maneira formal, a fim de ser interpretado por máquinas. A utilização do conhecimento incerto para raciocinar e tomar decisões por vezes inviabiliza a afirmação categórica de algumas sentenças. Muitas vezes, acoplamos um valor probabilístico a elas para melhor explicitar o grau de verdade que temos sobre estas. Sendo assim, é importante a construção de modelos de representação do conhecimento que incorporem formas que indiquem o grau de certeza das informações obtidas do ambiente.

Por outro lado, também são necessárias ferramentas para o desenvolvimento de agentes artificiais capazes de tratarem estas informações simbólicas e não simbólicas, por exemplo, no processo de tomada de decisão autônoma.

Dentro deste escopo apresentamos neste trabalho o *plugin* COPA, que visa apresentar um exercício de representação do conhecimento probabilístico para crenças do modelo BDI. Os estudos de caso desenvolvidos utilizaram agentes BDI modelados em Redes Bayesianas. Para tanto, o algoritmo de seleção de planos dos agentes desenvolvidos no Jason foi alterado, passando a utilizar revisão de crenças probabilísticas ao invés de heurísticas para a escolha dos planos a serem executados pelos agentes.

Neste trabalho foram estudados alguns conceitos fundamentais em IA, como Sistemas MultiAgentes, representação do conhecimento incerto, redes probabilísticas, além de diversas ferramentas de construção de agentes. Também foram estudados trabalhos relacionados, que trataram de ideias similares às propostas no *COPA*. Considerações foram feitas sobre suas diferenças, vantagens e desvantagens com relação ao trabalho aqui desenvolvido.

Através dos estudos de casos feitos, notou-se maior facilidade na implementação de agentes, em particular no desenvolvimento de seus planos, pois ao invés de o agente escolher sempre o primeiro plano aplicável de sua lista, passará a analisar todas as possibilidades, retornando aquele com maior probabilidade de atingir seu objetivo mais rapidamente. Testes com informações reais ainda devem ser realizados para compararmos os resultados obtidos do *COPA* frente à abordagem sem crenças probabilísticas.

Sistemas MultiAgentes e Redes Bayesianas já desenvolvidos podem ser unidos através deste *plugin*, necessitando apenas alterar a classe do agente a ser utilizado para a desenvolvida no *COPA* e utilizar a BN no JAmplia. Esta funcionalidade, já desenvolvida em trabalhos como em CALCIN (2007), é importante na área de Inteligência Artificial, pois une duas áreas de pesquisa, possibilitando o desenvolvimento de novos softwares nestas áreas.

O processo de inclusão de valores probabilísticos, obtidos da Rede Bayesiana, nas crenças BDI e a alteração no algoritmo de seleção de planos não agrega poder de repre-

sentação do conhecimento incerto em Sistemas MultiAgentes, mas é útil pois facilita o desenvolvimento de agentes probabilísticos através da capacidade desenvolvida de manipular tal tipo de conhecimento.

Uma desvantagem desta nova abordagem de seleção de planos é a possível falta de controle que o programador do agente terá sobre o comportamento deste. Por vezes é difícil saber em que momento o agente escolherá um determinado plano, dificultando sua construção.

Na seção 2.2.7 ficou clara a dificuldade encontrada por um programador do Jason ao tratar informações incertas. Nos estudos de caso apresentados, foram mostradas duas situações em que o uso do *plugin* COPA melhorou significativamente a programação dos agentes. Embora os estudos de caso sejam suficientes para comprovar esta melhoria, o *plugin* desenvolvido ainda deve ser exaustivamente testado a procura de possíveis melhorias e erros de implementação.

Como trabalhos futuros, podem ser propostas algumas melhorias ao trabalho até aqui desenvolvido. Quando o usuário recompila a Rede Bayesiana, a probabilidade de todas as crenças é enviada aos agentes cadastrados. Provavelmente uma alteração neste ponto será feita para a rede bayesiana enviar somente a probabilidade das crenças que foram modificadas, diminuindo o número de dados trafegados na rede.

Como já mencionado na seção 3.3.1, o cálculo de probabilidade total dos planos deverá ser aprimorado, para que os operadores "|" e "$\sim$" também possam ser utilizados e as probabilidades das crenças envolvidas sejam adequadamente calculadas.

O algoritmo implementado no método $selectOption$ também deverá ser refinado, pois atualmente sua complexidade está relativamente alta, conforme apresentado na seção 3.3.1.2. É possível que a verificação de todas as crenças da base do agente não seja obrigatória, diminuindo o número de comparações a serem realizadas.

Além de atrelar probabilidade às crenças, poderemos também adicionar probabilidades aos desejos do agente. Caso um desejo seja escolhido muitas vezes não tendo resultado satisfatório, o agente poderia diminuir sua probabilidade, dando chance para que os outros desejos também sejam executados.

Outra nova funcionalidade interessante seria possibilitar que o próprio agente altere as probabilidades de suas crenças, enviando este novo valor à rede bayesiana. Assim, o usuário da RB pode ter visão em tempo real das condições do ambiente onde o agente situa-se.

REFERÊNCIAS

AMGOUD, L.; PRADE, H. Formalizing Practical Reasoning Under Uncertainty: an argumentation-based approach. In: INTERNATIONAL CONFERENCE ON INTELLIGENT AGENT TECHNOLOGY (IAT'07), Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2007. p.189–195.

ANDERSEN, S. K. et al. HUGIN - a Shell for Building Bayesian Belief Universes for Expert Systems. In: IJCAI. **Anais...** Citeseer, 1989. v.2, p.1080–1085.

BELLIFEMINE, F. et al. JADE - A Java Agent Development Framework. In: BORDINI, R. H. et al. (Ed.). **Multi-Agent Programming**. Berlin: Springer, 2005. p.125–147. (Multiagent Systems, Artificial Societies, and Simulated Organizations, v.15).

BELLIFEMINE, F. L.; CAIRE, G.; GREENWOOD, D. **Developing Multi-Agent Systems with JADE**. Chichester, UK: John Wiley & Sons, 2007. (Wiley Series in Agent Technology).

BORDINI, R. H. et al. AgentSpeak(XL): efficient intention selection in BDI agents via decision-theoretic task scheduling. In: FIRST INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTI-AGENT SYSTEMS (AAMAS-2002), Bologna, Italy. **Proceedings...** ACM Press, 2002. v.3, p.1294–1302.

BORDINI, R. H. et al. (Ed.). **Multi-Agent Programming**: languages, platforms and applications. Berlin: Springer, 2005. (Multiagent Systems, Artificial Societies, and Simulated Organizations, v.15).

BORDINI, R. H.; HÜBNER, J. F.; WOOLDRIDGE, M. **Programming multi-agent systems in AgentSpeak using Jason**. [S.l.]: John Wiley & Sons, 2007. 273p. (Wiley Series in Agent Technology, v.1).

BORDINI, R. H.; VIEIRA, R.; MOREIRA, A. F. Fundamentos de sistemas multiagentes. In: XXI CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAçãO (SBC2001), Fortaleza-CE, Brasil. **Anais...** [S.l.: s.n.], 2001. v.2, p.3–41.

BRATKO, I. **Prolog Programming for Artificial Intelligence**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1986.

BRATMAN, M. E. **Intentions, Plans and Practical Reason**. Cambridge, MA: Harvard University Press, 1987.

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.

BUSETTA, P. et al. Structuring BDI Agents in Functional Clusters. In: SIXTH INTERNATIONAL WORKSHOP ON INTELLIGENT AGENTS VI – AGENT THEORIES, ARCHITECTURES, AND LANGUAGES (ATAL'99), Orlando, Florida, USA. **Proceedings...** Springer, 2000. p.277–289. (Lecture Notes in Computer Science, v.1757).

CALCIN, O. G. P. **Bayes Jason**. 2007. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul.

CASALI, A.; GODO, L.; SIERRA, C. Graded BDI Models for Agent Architectures. In: LEITE, J.; TORRONI, P. (Ed.). **Computational Logic in Multi-Agent Systems**. Berlin: Springer, 2005. p.148–148. (Lecture Notes in Computer Science, v.3487).

CLOCKSIN, W. F.; MELLISH, C. **Programming in Prolog**. 3.ed. Berlin: Springer, 1987.

COZMAN, F. G. Generalizing variable elimination in Bayesian networks. In: IBERAMIA/SBIA 2000 WORKSHOPS (WORKSHOP ON PROBABILISTIC REASONING IN ARTIFICIAL INTELLIGENCE), São Paulo, Brazil. **Proceedings...** Tec Art Editora, 2000. p.27–32.

DASTANI, M.; RIEMSDIJK, M. B. van; MEYER, J.-J. C. Programming Multi-Agent Systems in 3APL. In: BORDINI, R. H. et al. (Ed.). **Multi-Agent Programming**. Berlin: Springer, 2005. p.39–67. (Multiagent Systems, Artificial Societies, and Simulated Organizations, v.15).

DEMAZEAU, Y. From cognitive interactions to collective behaviour in agent-based systems. In: FIRST EUROPEAN CONFERENCE ON COGNITIVE SCIENCE, Saint-Malo, France. **Proceedings...** [S.l.: s.n.], 1995. p.117–132.

DENNETT, D. C. **The Intentional Stance**. Cambridge, MA: MIT Press, 1987.

D'INVERNO, M. et al. A Formal Specification of dMARS. In: FOURTH INTERNATIONAL WORKSHOP ON INTELLIGENT AGENTS IV, AGENT THEORIES, ARCHITECTURES AND LANGUAGES (ATAL-97), London, UK. **Proceedings...** Springer-Verlag, 1998. n.1365, p.155–176. (Lecture Notes in Artificial Intelligence).

FAGUNDES, M. S.; VICARI, R. M.; COELHO, H. Deliberation Process in a BDI Model with Bayesian Networks. In: TENTH PACIFIC RIM INTERNATIONAL WORKSHOP ON MULTI-AGENTS (PRIMA 2007), Bangkok, Thailand. **Proceedings...** Springer, 2007. p.207–218. (Lecture Notes in Artificial Intelligence).

FARIAS, G. P. **Implementando Agentes Híbridos BDI-Fuzzy na Plataforma de Agentes Jason**. 2009. Dissertação (Mestrado em Ciência da Computação) — Universidade Católica de Pelotas.

FININ, T. W. et al. KQML As An Agent Communication Language. In: THIRD INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT (CIKM'94), Gaithersburg, Maryland. **Proceedings...** ACM, 1994.

FIPA. **FIPA contract net interaction protocol specification**. [S.l.]: Foundation for Intelligent Physical Agents, 2003. (00029).

FLORES, C. D. **Coordenação de Agentes em Ambientes com Tratamento de Incerteza**. 2002. Exame de Qualificação – Universidade Federal do Rio Grande do Sul.

BARONE, D. (Ed.). **Sociedades artificiais**: a nova fronteira da inteligência nas máquinas. 1.ed. [S.l.]: Bookman, 2003. v.1, p.127–154.

FLORES, C. D. **Negociação Pedagógica Aplicada a um Ambiente Multiagente de Aprendizagem Colaborativa**. 2005. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul.

FLORES, C.; VICARI, R. M. AMPLIA Learning Environment Architecture. In: PROCEEDING OF THE 2005 CONFERENCE ON TOWARDS SUSTAINABLE AND SCALABLE EDUCATIONAL INNOVATIONS INFORMED BY THE LEARNING SCIENCES, Amsterdam, The Netherlands, The Netherlands. **Anais...** IOS Press, 2005. p.662–665.

GIRAFFA, L.; VICARI, R. The Use of Agents Techniques on Intelligent Tutoring Systems. In: XVIII INTERNATIONAL CONFERENCE OF THE CHILEAN COMPUTER SCIENCE SOCIETY (SCCC'98), Washington, DC, USA. **Proceedings...** IEEE Computer Society, 1998. p.76.

GLUZ, J. C. **Tutorial de AgentSpeak(PL)**: representação de modelos probabilísticos e programação de agentes bayesianos. 2008. Estudo complementar.

HÜBNER, J.; BORDINI, R.; VIEIRA, R. Introdução ao desenvolvimento de sistemas multiagentes com Jason. In: XII ESCOLA DE INFORMÁTICA DA SBC, Guarapuava. **Anais...** UNICENTRO, 2004. v.2, p.51–89.

HÜBNER, J. F.; SICHMAN, J. S.; BOISSIER, O. Using the Moise+ for a cooperative framework of MAS reorganisation. In: SEVENTEENTH BRAZILIAN SYMPOSIUM ON ARTIFICIAL INTELLIGENCE, São Luis, Maranhão, Brazil. **Proceedings...** Springer, 2004. p.506–515.

JENSEN, F. V.; OLSEN, K. G.; ANDERSEN, S. K. **An Algebra of Bayesian Belief Universes for Knowledge Based Systems**. Aalborg, Denmark: Institute of Electronic Systems, Aalborg University, 1988.

KICZALES, G. et al. Aspect-oriented programming. In: AKsIT, M.; MATSUOKA, S. (Ed.). **ECOOP'97 – Object-Oriented Programming**. Berlin/Heidelberg: Springer-Verlag, 1997. p.220–242. (Lecture Notes in Computer Science, v.1241).

KIELING, G. L. **AgentSpeak(PL) - Especificação da Extensão de uma Linguagem para Representação de Conhecimentos Probabilísticos e Lógicos**. 2008. Estudo complementar.

LADEIRA, M.; VICARI, R.; COELHO, H. **Redes Bayesianas Multiagentes**. Curso de curta duração ministrado.

LAURITZEN, S. L.; SPIEGELHALTER, D. J. Local computations with probabilities on graphical structures and their application to expert systems. **Journal of the Royal Statistical Society: Series B (Statistical Methodology)**, [S.l.], v.50, n.2, p.157–224, 1990.

LESSER, V. R. Cooperative Multiagent Systems: a personal view of the state of the art. **IEEE Trans. on Knowl. and Data Eng.**, Piscataway, NJ, USA, v.11, n.1, p.133–142, 1999.

MACHADO, G. M. **JAmplia - uma proposta de ambiente multiagente probabilístico inteligente em java voltado para a web**. Trabalho de Conclusão de Curso (Bacharelado) Universidade Feredal do Rio Grande do Sul, 2006.

MAES, P. Artificial life meets entertainment: lifelike autonomous agents. **Commun. ACM**, New York, NY, USA, v.38, n.11, p.108–114, 1995.

MÓRA, M. D. C. et al. BDI Models and Systems: bridging the gap. In: FIFTH INTERNATIONAL WORKSHOP ON INTELLIGENT AGENTS V, AGENT THEORIES, ARCHITECTURES, AND LANGUAGES (ATAL'98), Paris, France. **Proceedings...** Springer-Verlag, 1999. p.11–27.

PATEL-SCHNEIDER, P. F. A decidable first-order logic for knowledge representation. In: NINTH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE (IJCAI'85), San Francisco, CA, USA. **Proceedings...** Morgan Kaufmann Publishers Inc., 1985. p.455–458.

PEARL, J. Fusion, Propagation, and Structuring in Belief Networks. **Artificial Intelligence**, [S.l.], v.29, n.3, p.241–288, 1986.

PEREIRA, D. R. et al. Towards the Self-regulation of Personality-Based Social Exchange Processes in Multiagent Systems. In: XIX BRAZILIAN SYMPOSIUM ON ARTIFICIAL INTELLIGENCE (SBIA '08), Berlin, Heidelberg. **Proceedings...** Springer-Verlag, 2008. v.5249, p.113–123.

PERRY, B. B.; STILSON, J. A. BN-Tools: a software toolkit for experimentation in bbns. In: EIGHTEENTH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, Menlo Park, CA, USA. **Proceedings...** American Association for Artificial Intelligence, 2002. p.963–964.

PICARD, R. W. **Affective Computing**. Cambridge, Massachusetts: The MIT Press, 1997.

RAO, A. S. AgentSpeak(L): BDI agents speak out in a logical computable language. In: SEVENTH EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD (MAAMAW'96), Eindhoven, The Netherlands. **Proceedings...** Springer-Verlag, 1996. n.1038, p.42–55.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence**: a modern approach. 3.ed. Upper Saddle River, NJ, USA: Pearson Education, 2009.

SHACHTER, R. D. Evaluating Influence Diagrams. **Operations Research**, [S.l.], v.34, n.6, p.871–882, 1986.

STERLING, L.; SHAPIRO, E. **The Art of Prolog - Advanced Programming Techniques**. 2.ed. Cambridge, MA, USA: MIT Press, 1994.

WU, G. (Ed.). **UnBBayes**: modeling uncertainty for plausible reasoning in the semantic web. Croatia: IntechWeb, 2010. p.1–28.

WINIKOFF, M.; PADGHAM, L. (Ed.). **Developing Intelligent Agent Systems**: a practical guide. New York, NY, USA: Halsted Press, 2004.

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. **Reasoning about Rational Agents**. Cambridge, Massachusetts: MIT Press, 2000.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent Agents: theory and practice. **Knowledge Engineering Review**, [S.l.], v.10, p.115–152, 1995.

ZADEH, L. A. Fuzzy Logic = Computing with Words. **Fuzzy Systems**, [S.l.], v.4, n.2, p.103–111, 1996. IEEE Trans.

ZADEH, L. A. A note on web intelligence, world knowledge and fuzzy logic. **Data & Knowledge Engineering**, Amsterdam, The Netherlands, v.50, p.291–304, September 2004.

ZADEH, L. A. Generalized theory of uncertainty (GTU) - principal concepts and ideas. **Comput. Stat. Data Anal.**, [S.l.], v.51, n.1, p.15–46, 2006.

APÊNDICE A MODELO QUANTITATIVO PARA AS VARIÁVEIS DO ESTUDO DE CASO DO AGENTE INVESTIDOR

Tabela A.1: Modelo quantitativo da variável *Atual*

Atual	***P(Atual)***
bom	*0,4*
ruim	*0,6*

Tabela A.2: Modelo quantitativo da variável *Noticias*

Noticias	***P(Noticias)***
otimas	*0,2*
pessimas	*0,8*

Tabela A.3: Modelo quantitativo da variável *Projecao*

Atual	***Noticias***	***Projecao***	***P(Projecao)***
bom	*otimas*	*bom*	*0,65*
bom	*otimas*	*ruim*	*0,35*
bom	*pessimas*	*bom*	*0,2*
bom	*pessimas*	*ruim*	*0,8*
ruim	*otimas*	*bom*	*0,95*
ruim	*otimas*	*ruim*	*0,05*
ruim	*pessimas*	*bom*	*0,4*
ruim	*pessimas*	*ruim*	*0,6*

Tabela A.4: Modelo quantitativo da variável *Idade*

Idade	***P(Idade)***
menor	*0,6*
maior	*0,4*

Tabela A.5: Modelo quantitativo da variável *Caracteristica*

Caracteristica	***P(Caracteristica)***
arrojado	*0,35*
conservador	*0,65*

Tabela A.6: Modelo quantitativo da variável *Capital*

Caracteristica	***Idade***	***Capital***	***P(Capital)***
arrojado	*menor*	*muito*	*0,9*
arrojado	*menor*	*pouco*	*0,1*
arrojado	*maior*	*muito*	*0,4*
arrojado	*maior*	*pouco*	*0,6*
conservador	*menor*	*muito*	*0,75*
conservador	*menor*	*pouco*	*0,75*
conservador	*maior*	*muito*	*0,2*
conservador	*maior*	*pouco*	*0,8*

Tabela A.7: Modelo quantitativo da variável *Capital*

Caracteristica	***Idade***	***Capital***	***P(Capital)***
arrojado	*menor*	*muito*	*0,9*
arrojado	*menor*	*pouco*	*0,1*
arrojado	*maior*	*muito*	*0,4*
arrojado	*maior*	*pouco*	*0,6*
conservador	*menor*	*muito*	*0,75*
conservador	*menor*	*pouco*	*0,75*
conservador	*maior*	*muito*	*0,2*
conservador	*maior*	*pouco*	*0,8*

APÊNDICE B CÓDIGO-FONTE DOS AGENTES $INVESTOR$ E $BROKER$ DO ESTUDO DE CASO DO AGENTE INVESTIDOR

Agente *investor*:

```
1  /* Initial beliefs */
2  caixa(2000).
3  investido(0).
4  cotacao(0).
5
6  /* Initial goals */
7  !registerDF.
8  !acao.
9
10 /* Plans */
11 +!registerDF
12   <- .print("Olah! Sou um investidor.");
13     copa.ia.register("JAmplia-BN", "JADE-Agent").
14
15 +!kqml_received(server, tell, Belief, _)
16   <- .print("Msg recebida do SERVER; contendo: '", Belief, "'");
17     copa.ia.addBelief(Belief).
18
19 +!kqml_received(broker, tell, Belief, _)
20   <- .print("Msg recebida do BROKER; contendo: '", Belief, "'");
21     copa.ia.addBelief(Belief);
22     !atualizaGanhos;
23     !acao.
24
25 // Atualiza o investimento com a cotacao atual
26 +!atualizaGanhos
27   <- ?cotacao(Co);
28     ?investido(I);
29     -+investido(I + ((I * Co) / 100)).
30
31
32 // O investidor pode...
33 // ficar parado somente se nao tem informacoes para decidir
34 +!acao[source(self)]
35   : true
36   <- .print("O melhor mesmo eh ficar na minha");
37     !imprimiValores.
38
39 // investir 1/10
```

```
40 +!acao[source(self)]
41   : investimento(medio) & caracteristica(arrojado) & caixa(C) & (C >
         20)
42   <- ?investido(I);
43     -+caixa(C * 9/10);
44     -+investido(I + (C * 1/10));
45     .print("Aplicando um pouco");
46     !imprimiValores.
47
48 // retirar 1/10
49 +!acao[source(self)]
50   : investimento(medio) & caracteristica(conservador) & investido(I) &
         (I > 5)
51   <- ?caixa(C);
52     -+caixa(C + (I * 1/10));
53     -+investido(I * 9/10);
54     .print("Retirando um pouco");
55     !imprimiValores.
56
57 // investir 1/4 caso tenha mais de $50 em caixa
58 +!acao[source(self)]
59   : investimento(alto) & atual(ruim) & projecao(bom) & caixa(C) & (C >
         50)
60   <- ?investido(I);
61     -+caixa(C * 3/4);
62     -+investido(I + (C * 1/4));
63     .print("Vamos investir um monte!");
64     !imprimiValores.
65
66 // retirar 1/4 retirar caso tenha algo investido
67 +!acao[source(self)]
68   : investimento(baixo) & atual(bom) & projecao(ruim) & investido(I) &
         (I > 5)
69   <- ?caixa(C);
70     -+caixa(C + (I * 1/4));
71     -+investido(I * 3/4);
72     .print("Acho que vai cair... Melhor retirar bastante");
73     !imprimiValores.
74
75
76 // Plano auxiliar para imprimir valores atuais
77 +!imprimiValores
78   <- ?caixa(C);
79     ?investido(I);
80     .print("Caixa: $", C, "; Investido: $", I).
```

Agente *broker*:

```
1  /* Initial beliefs */
2  cotacao(0).
3
4  /* Initial goals */
5  !registerDF.
6
7  /* Plans */
8  +!registerDF
9    <- .print("Olah! Sou corretor.");
10     copa.ia.register("JAmplia-BN", "JADE-Agent").
11
```

```
12 +!kqml_received(server, tell, Belief, _)
13   <- copa.ia.addBelief(Belief);
14     !calcula.
15
16 // O corretor deve calcular a cotacao da bolsa no dia.
17 // Nao precisa calcular se nao tem informacoes basicas
18 +!calcula
19   : not atual(bom) & not atual(ruim)
20   <- .print("Nao sei!").
21
22 // Melhor situacao
23 +!calcula
24   : atual(ruim) & projecao(bom) & noticias(otimas)
25   <- .random(R);
26     -+cotacao(R * 2);
27     copa.ia.getBelief(cotacao(_), Co);
28     .print("Hoje teve cotacao POSITIVA: ", Co);
29     .send(investor, tell, Co).
30
31 // Pior situacao
32 +!calcula
33   : atual(bom) & projecao(ruim) & noticias(pessimas)
34   <- .random(R);
35     -+cotacao((R - 1) * 2);
36     copa.ia.getBelief(cotacao(_), Co);
37     .print("Hoje teve cotacao NEGATIVA: ", Co);
38     .send(investor, tell, Co).
39
40 // Situações com pequena variação
41 +!calcula
42   : atual(bom) & projecao(bom) & noticias(otimas)
43   <- .random(R);
44     -+cotacao(R / 2);
45     copa.ia.getBelief(cotacao(_), Co);
46     .print("Hoje teve pequena cotacao POSITIVA: ", Co);
47     .send(investor, tell, Co).
48
49 +!calcula
50   : atual(ruim) & projecao(ruim) & noticias(pessimas)
51   <- .random(R);
52     -+cotacao((R - 1) / 2);
53     copa.ia.getBelief(cotacao(_), Co);
54     .print("Hoje teve pequena cotacao NEGATIVA: ", Co);
55     .send(investor, tell, Co).
```

APÊNDICE C ARTIGO ACEITO NO CONGRESSO CISIS 2011