UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

IMPLEMENTAÇÃO DE UMA LIVRARIA VIRTUAL UTILIZANDO AGENTES BDI ATRAVÉS DA LINGUAGEM AGENTSPEAK(L)

DANIEL DALCASTAGNE

BLUMENAU
2006

2006/1-04

DANIEL DALCASTAGNE

IMPLEMENTAÇÃO DE UMA LIVRARIA VIRTUAL UTILIZANDO AGENTES BDI ATRAVÉS DA LINGUAGEM AGENTSPEAK(L)

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Jomi Fred Hübner, Doutor - Orientador

BLUMENAU
2006

2006/1-04

IMPLEMENTAÇÃO DE UMA LIVRARIA VIRTUAL UTILIZANDO AGENDES BDI ATRAVÉS DA LINGUAGEM AGENTESPEAK(L)

Por

DANIEL DALCASTAGNE

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Jomi Fred Hübner, Doutor - Orientador, FURB
Membro:	Prof. Paulo César Rodacki, Doutor – FURB
Membro:	Prof. Mauro Marcelo Mattos, Doutor – FURB

Blumenau, 06 de julho de 2006

Dedico este trabalho a todos aqueles que persistem diante das adversidades.

AGRADECIMENTOS

À Deus, pelo seu imenso amor e graça.

Ao meu pai Dário e minha mãe Rosane, por serem nada mais, nada menos que os Pais e pela intensa dedicação e forte apoio dedicado durante toda minha vida.

À namorada Adriana, não apenas pela compreensão de minha ausência durante o desenvolvimento deste trabalho, mas pelo constante apoio e cobranças.

Ao meu orientador, Jomi Fred Hübner, pelo conhecimento compartilhado, pelo Jason e pelo Saci.

Eu estou sempre fazendo aquilo que não sou capaz, numa tentativa de assim aprender como fazê-lo.

Pablo Ruiz Picasso

RESUMO

Este trabalho apresenta um Sistema Multiagentes (SMA) atuando sobre um sistema de livraria virtual. A livraria é composta por quatro agentes básicos: *Agent Sales Assistant,* que negocia com o cliente de maneira análoga ao assistente real de uma loja qualquer; *Agent Delivery Manager*, que organiza todos os aspectos relacionados à entrega do produto ao cliente; *Agent Customer Relations,* que é responsável por todas as transações off-line; *Agent Stock Manager* que é responsável pelo estoque de livro da loja. A arquitetura dos agentes é baseada em modelo de cognição fundamentado em três principais atitudes mentais que são as crenças, os desejos e as intenções (abreviadas por BDI, do inglês *beliefs*, *desires* e *intentions*, respectivamente). O objetivo geral do trabalho é o desenvolvimento de um sistema com características comerciais utilizando o paradigma de agentes. Para especificação do sistema é utilizada uma nova metodologia para especificação de Sistema Multiagentes denominada Prometheus. Para implementação dos agentes é utilizada a linguagem *AgentSpeak(L)* através do interpretador *Jason*. Ao final é feita uma análise de como ocorreu o desenvolvimento do sistema utilizando os recursos mencionados acima.

Palavras-chave: Sistema multiagentes. *AgentSpeak(L). Jason.* Prometheus.

ABSTRACT

This work presents a Multi-Agent System (SMA) acting on a system of virtual bookstore. The bookstore is composed by four basic agents: Agent Sales Assistant, who negotiates with the way customer similar to the real assistant of any store; Agent Delivery Manager, that organizes all aspects related to the delivery of the product to the customer; Agent Customer Relations, who is responsible for all off-line transactions; Agent Stock Manager who is responsible for the stock of books of the store. The agents' architecture is based on cognition model based in three main mental attitudes that are the faiths, the desires and the intentions (abbreviated by BDI, of the English beliefs, desires and intentions, respectively). The general objective of the work is the development of a system with commercial characteristics using the agents' paradigm. For specification of the system a new methodology is used for specification of Multi-Agent System denominated Prometheus. For the agents' implementation the language is used AgentSpeak(L) through the interpreter Jason. At the end it is made an analysis of as it happened the development of the system using the resources mentioned above.

Keywords: Multi-Agent System. Bookstore. AgentSpeak(L). Jason. Prometheus

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

SMA – Sistema Multiagentes

BDI – Beliefs, Desires e Intentions

JASON – Java-based AgentSpeak interpreter used with SACI for multi-agent distribution Over the Net

SACI – Simple Agent Communication Infrastructure

KQML – Knowledge Query and Manipulation Language

GPL – General Public License

LGPL – Library General Public License

SPL – Sun Public License

UML – Unified Modeling Language

AOSE – Agent Oriented Software Engineering

SUMÁRIO

1 INTRODUÇÃO

Atualmente quatro paradigmas de programação são amplamente estudados e fortemente definidos no meio científico e acadêmico da ciência da computação: o paradigma imperativo que é baseado em comandos e procedimentos; o paradigma orientado a objetos que representa o mundo em classes que por sua vez são instanciadas na forma de objetos; o paradigma funcional que é baseado em funções e o paradigma lógico que é baseado em interferência lógica.

Entretanto, segundo Wooldridge (1999, p. 27), no final da década de 90, uma nova maneira de conceber sistemas computacionais começou a ganhar representatividade. Estes sistemas são por sua vez formados por vários agentes autônomos, denominados Sistemas Multiagentes (SMA). Wooldridge (2002, p. 14) define um agente autônomo como um sistema computacional que está situado em um ambiente, e que é capaz de realizar ações autônomas para alcançar seus objetivos.

Conforme proposto por Shoham (2003, p. 2), essa nova maneira conceitual de ver um sistema de software foi então denominada de programação orientada a agentes. Uma das principais linguagens que empregam o paradigma de programação orientada a agentes, em particular utilizando a arquitetura *beliefs, desires e intentions* (BDI), e que possibilita a implementação de softwares com as características mencionadas é a linguagem AgentSpeak(L), que possui a sintaxe baseada nos conceitos de programação em lógica (RAO, 1996, p. 2).

Segundo Hübner, Bordini e Vieira (2004, p. 19), poucas aplicações foram desenvolvidas com AgentSpeak(L) até o momento, dado que a sua implementação prática é muito recente e ainda requer um trabalho mais aprofundado de experimentação para que se torne uma linguagem de programação de uso mais amplo.

A tecnologia de sistemas multiagentes encontra-se num estágio ainda inicial, e há necessidade de se implementar sistemas com a finalidade de dar sustentação e fornecer parâmetros para construção de uma linguagem de programação que apresente características adequadas para agentes, permitindo sua utilização de forma eficiente e eficaz na construção de softwares.

Levando-se em consideração a premissa acima, propõe-se neste trabalho um estudo de caso através da implementação de uma livraria virtual, detalhada em Padgham e Winikoff (2004) através da metodologia Prometheus e que se encontra parcialmente implementada através da linguagem de programação de agentes JACK (PADGHAM; WINIKOFF, 2004, p. 3).

Uma das ferramentas que contemplam os requisitos necessários para realização deste trabalho é a ferramenta Jason (*Java-based AgentSpeak interpreter used with SACI for multi-agent distribution Over the Net*). Jason é interpretador multi-plataforma para a linguagem AgentSpeak(L).

Com a intenção de analisar as potencialidades do interpretador Jason, no desenvolvimento de um sistema computacional semelhante à realidade da maioria dos aplicativos comerciais desenvolvidos na atualidade, e nortear novos trabalhos que venham a utilizar este paradigma, é que surge a idéia do desenvolvimento de um aplicativo “comercial”, utilizando uma modelagem baseada em agentes.

A livraria consiste de um sistema web onde o usuário conta com recursos necessários para localizar, obter informações e comprar livros, contando ainda com atendimento personalizado, mantido através de um registro com suas preferências, entre outras opções.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é realizar um estudo da viabilidade da implementação de um sistema computacional convencional, utilizando o paradigma da programação orientada a agentes, tendo como ferramenta a linguagem AgentSpeak(L), através do interpretador Jason. Averiguando desta maneira os recursos disponíveis por estas ferramentas, apontando suas limitações (se houver) e fornecendo parâmetros para seu aperfeiçoamento.

1.2 ESTRUTURA DO TRABALHO

O trabalho está organizado em cinco capítulos. No capítulo seguinte é descrita a fundamentação teórica utilizada para embasar este trabalho. São apresentados os conceitos básicos relacionados ao desenvolvimento de SMA, em seguida é feita a apresentação da linguagem AgentSpeak(L) e do interpretador Jason, posteriormente é apresentado a metodologia Prometheus utilizada para especificação do sistema e por fim são apresentados alguns trabalhos correlatos.

A capítulo 3 descreve a especificação da ferramenta. A especificação ocorre através da metodologia Prometheus, porém de maneira mais textualizada. O capítulo 4 apresenta como se desenvolveu a implementação, ao final do capítulo são apresentados os resultados alcançados a partir da experiência vivenciada no desenvolvimento do trabalho.

Por fim o capítulo 5 contém a conclusão do trabalho, juntamente com sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais conceitos inerentes ao desenvolvimento deste trabalho, através de uma revisão bibliográfica dos seguintes temas: SMA, arquitetura BDI, a linguagem AgentSpeak(L), o interpretador Jason e a metodologia Prometheus.

2.1 SISTEMAS MULTIAGENTES (SMA)

Com o objetivo de facilitar a visualização e compreensão de problemas estruturalmente desconexos do ponto de vista prático numa sociedade, como são em geral os sistemas computacionais, é inerente à natureza humana buscar formas de conceber estes problemas de maneira que se aproximem da realidade. Assim surgiram a orientação objetos, as redes neurais, entre outras. De forma análoga, a área de Sistemas Multiagentes (SMA) é influenciada pela sociologia e, portanto, tem possibilitado uma concepção de sistemas com propriedades que até então somente sociedades possuíam (HÜBNER, 2003, p. 23).

O paradigma orientado a agentes surge potencialmente da observação de comportamentos sociais, na qual se percebe o surgimento de uma organização a partir da interação de seus elementos. Tomando como exemplo uma colônia de formigas, verifica-se que as ações individuais de cada formiga não representam grande complexidade, porém ao analisar a colônia como um todo se percebe um sistema verdadeiramente complexo, concluindo-se desta maneira que o formigueiro é um sistema mais inteligente do que as formigas que o compõe (HÜBNER, 2003 apud JOHNSON, 2001, p. 24).

Um componente indispensável na construção de SMA são os agentes. Entretanto é preciso ressaltar que ainda não existe uma definição de agente, capaz de ser adotada como um consenso, pois se trata de uma área de pesquisa muito recente. Porém, pode-se definir um

agente como uma entidade, que está inserida em um ambiente, podendo perceber, agir, ser persistente nas suas metas e comunicar-se com outros agentes assumindo comportamentos autônomos (PADGHAM; WINIKOFF, 2004, p. 3).

O exemplo da colônia de formigas representa o enfoque principal em SMA, onde se tem como objetivo prover mecanismos para a criação de sistemas computacionais a partir de entidades de software autônomas (agentes), que interagem através de um ambiente compartilhado por um grupo organizado de agentes, que cooperam na resolução de problemas que estão além das capacidades de resolução de cada um individualmente.

Esta maneira de visualizar um SMA constitui o que se chama de agentes reativos, que de modo geral possuem um comportamento muito simples, escolhem suas ações basead[1]os unicamente nas percepções que têm do ambiente e constituem organizações do tipo etológico[1]. As sociedades são formadas por muitos agentes, e o interesse está voltado para a emergência de um comportamento global a partir da interação de um grupo grande de agentes (HÜBNER, 2003, p. 3).

De maneira oposta aos agentes reativos, existem os agentes cognitivos, que possuem um estado mental e funcionam racionalmente, isto é, raciocinam para construir um plano de ações que leva a um objetivo pretendido. Apresentam características particulares como: têm autonomia funcional, pois podem alterar seu funcionamento a fim de adaptar-se melhor ao seu ambiente (HÜBNER, 2003 apud RAO; GEORGEFF, 1995, p. 5); estão continuamente em funcionamento; são sociáveis (possuem a capacidade de comunicação e de modelagem dos outros); o mecanismo de controle é deliberativo (o agente raciocina sobre quais ações realizar); tem memória e as sociedades são formadas por poucos agentes.

Segundo Bordini, Vieira e Moreira (2001, p. 2) no processo de desenvolvimento de SMA duas abordagens são bastante comuns:

[1] Ciência descritiva dos costumes e das tradições dos animais no seu ambiente natural.

a) *Top-down*: inicia-se definindo as aspectos coletivos, como organização e comunicação, que são refinados até a definição dos agentes. Esta é normalmente, a ênfase dada pelas metodologias de *Agent Oriented Software Engineering* (AOSE). Nesta abordagem o projetista olha o sistema como um todo, analisando primeiramente a coletividade e posteriormente a individualidade;

b) *Bottom-up*: inicia-se definindo os aspectos individuais, relacionados aos agentes, de tal forma que ocorra a emergência dos aspectos coletivos. A interação e a organização são definidas do ponto de vista dos agentes. Analisa-se o sistema como se estivesse "dentro dos agentes", buscando primeiramente definir a individualidade do agente e posteriormente a coletividade na sociedade.

Hübner, Bordini e Vieira (2004, p. 4) apresentam algumas aplicações de sistemas multiagentes:

c) controle de tráfego aéreo: é uma aplicação reconhecidamente complexa. A aplicação de sistemas multiagentes mais comentada é um sistema para controle de tráfego aéreo que está sendo testado para uso no aeroporto de Sydney na Austrália;

a) indústria: modelar uma linha de produção através de agentes cooperantes é uma área de aplicação bastante grande;

b) gerência de negócios: agentes podem ser empregados em sistemas de *workflow* (i.e., sistemas que visam garantir que as tarefas necessárias serão feitas pelas pessoas certas nas horas certas);

c) ambientes de aprendizagem: estes sistemas podem empregar agentes tanto com o mesmo objetivo mencionado acima como para a modelagem dos alunos em tutores inteligentes;

d) entretenimento: agentes podem ser utilizados para aumentar o realismo de personagens de jogos e sistemas para a indústria de entretenimento em geral;

e) aplicações distribuídas: para domínios naturalmente distribuídos, a metáfora de agente é bastante adequada; telecomunicações, transportes e sistemas para a área de saúde são alguns exemplos.

Padgham e Winikoff (2004, p. 3) ressaltam ainda que um agente inteligente é um fragmento de software que é:

a) situado: existe num ambiente;

b) autônomo: independente, não é controlado externamente;

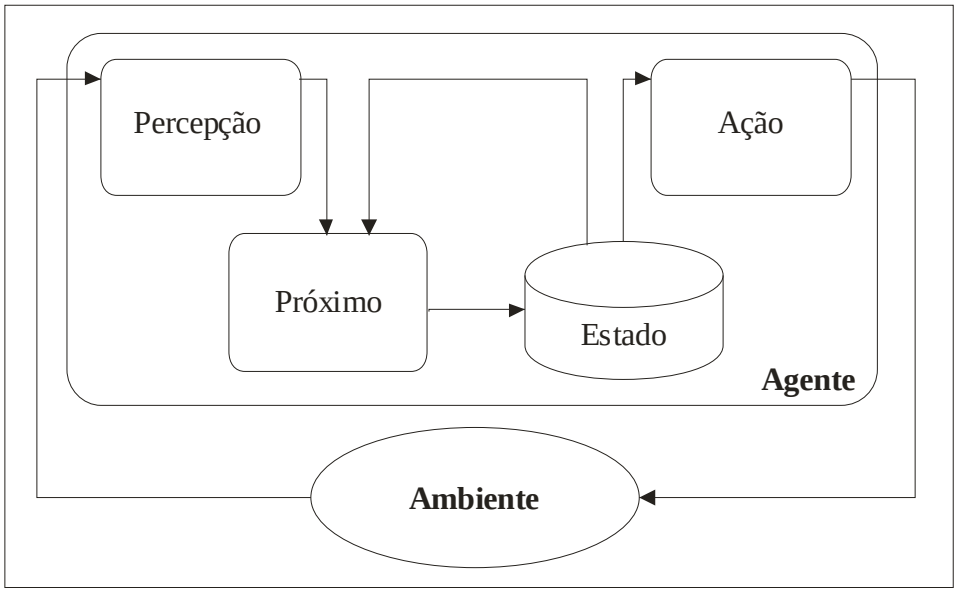
c) reativo: responde (em um tempo aceitável) a mudanças que ocorrem no ambiente;

d) proativo: encontra-se constantemente na procura de metas;

e) flexível: possui múltiplos caminhos para alcançar metas;

f) robusto: recupera-se de falhas;

g) social: interage com outros agentes.



Fonte: Wooldridge (2002, p. 6).

Figura 1 - Modelo geral de agente

A figura 1 apresenta o modelo geral de agente proposto por Wooldridge (2002), onde um agente é capaz de perceber alterações no ambiente, que são provenientes das ações que os agentes realizam constantemente. Uma das ações possíveis de um agente é comunicar-se com outros agentes da sociedade que compartilham o mesmo ambiente.

2.2 A ARQUITETURA BDI

Segundo Bordini e Vieira (2003, p. 8), as mais importantes arquiteturas de agentes deliberativos são baseadas em um modelo de cognição fundamentado em três principais atitudes mentais que são as crenças, os desejos e as intenções (abreviadas por BDI, do inglês *beliefs*, *desires* e *intentions*, respectivamente).

Crença, desejo, expectativa, capacidade e intenção são exemplos de estados intencionais. Esta intencionalidade, encontrada nos estados mentais indica uma propriedade de direcionamento do mundo para o agente e vice-versa. Como exemplo, a afirmação “a porta está fechada” é uma crença sobre a porta, do mundo para o agente e “entrar na sala” é um desejo, do agente para o mundo. Logo, intencionalidade não está ligada ao estado mental “intenção”, mas sim a todos os estados que possuem a propriedade de direcionalidade (CARVALHO, 2004, apud ZAMBERLAM; GIRAFFA, 2001, p. 51).

O nome dado a esta abordagem que utiliza estes estados para modelagem de agentes deliberativos é abordagem mentalista (CARVALHO, 2004, apud ZAMBERLAM; GIRAFFA, 2001, p. 50).

Ainda segundo Carvalho (2004, p. 51), a crença contém a visão fundamental do agente com relação ao seu ambiente. Um agente pode ter crenças sobre o mundo, sobre outros agentes, sobre interações com outros agentes e crenças sobre suas próprias crenças. As crenças podem inclusive ser contraditórias.

Os desejos representam estados do mundo que o agente quer atingir (quer que passem a ser verdadeiros). Em tese, podem ser contraditórios, ou seja, pode-se desejar coisas que são mutuamente exclusivas do ponto de vista prático (BORDINI; VIEIRA, 2003, p. 9). Os desejos motivam o agente a agir de forma a realizar metas, tais ações são realizadas através das intenções causadas pelos desejos.

Carvalho (2004 apud ZAMBERLAM; GIRAFFA, 2001, p.7) com base na Teoria das Situações, um desejo *D* de um agente *A* é apresentado como um estado mental intencional e motivador pela seguinte situação representada no quadro 1.

***D* |= {<*Des*, *A*, *P*, *e*, l_0, t_0, *v*>}**, em que:

Des	é a relação para a representação de desejo;
A	representa o agente que possui o desejo;
P	é uma proposição;
e	é uma situação que informa sobre a satisfação, não satisfação, urgência, intensidade e insistência do desejo *D*;
l_0	é a localização espacial (local) associada à ocorrência do desejo *D*;
t_0	é o tempo associado à ocorrência do desejo *D*;
v	1, se ocorre o desejo *D* ao agente *A* 0, se não ocorre o desejo *D* ao agente *A*

Fonte: adaptado de Carvalho (2004, p. 52).

Quadro 1 – Representação de desejos com base na Teoria das Situações

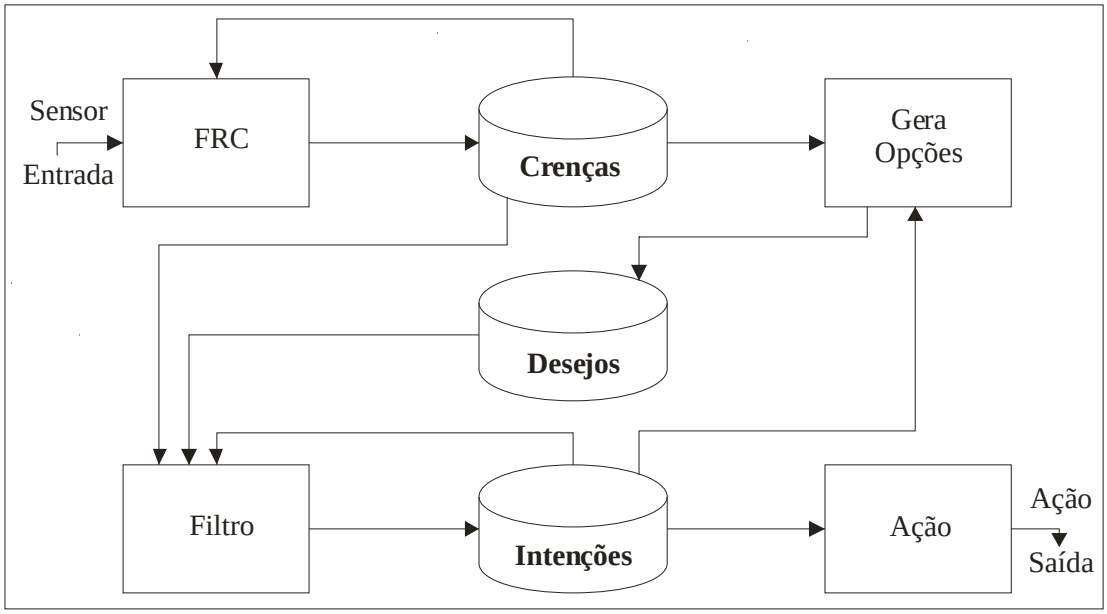
As intenções dos agentes são um subconjunto dos desejos. Se um agente decide seguir uma meta específica, então esta meta torna-se uma intenção.

Em Carvalho (2004 apud ZAMBERLAM; GIRAFFA, 2001, p.52) indica-se que a base para a definição do conceito de intenção está fortemente associada aos trabalhos filosóficos de Michael Bratman. Este autor claramente distingue o conceito de fazer as coisas intencionalmente, na forma de uma ação, e possuir intenção de fazê-las, ou seja, tendo como base um estado mental.

As intenções determinam o processo de raciocínio prático, pois a propriedade mais óbvia das intenções é que elas determinam as ações a serem realizadas. E enquanto se tem uma intenção particular haverá consideração por ações que são consistentes para a realização desta intenção (CARVALHO, 2004, p. 52). Por exemplo, se uma pessoa está em Blumenau e tem a intenção de ir a Jaraguá no Sul, ela deverá excluir opções que dificultem ou impossibilitem a realização desta intenção, como entrar em um ônibus para Ilhota.

2.2.1 Arquitetura BDI genérica

De forma esquemática, uma arquitetura BDI genérica pode ser apresentada como na figura 2, conforme proposto em Wooldridge (2002).



Fonte: adaptado de Wooldridge (2002, p. 36).

Figura 2 – Arquitetura BDI genérica

A função FRC (Função de Revisão de Crenças) recebe as informações do ambiente, podendo ler e atualizar a base de crenças do agente. Com as alterações do estado do ambiente, podem ser gerados novos objetivos. A função Gera Opções verifica quais estados devem ser atingidos de acordo com o estado atual e as intenções com que o agente está comprometido. A função filtro serve para atualizar o conjunto de intenções do agente com base nas crenças e desejos que ele possui. A função de Ação representa a escolha de uma determina ação para ser executada.

2.3 A LINGUAGEM AGENTSPEAK(L)

A linguagem AgentSpeak(L) foi projetada para a programação de agentes BDI na forma de sistemas de planejamento reativos. Sistemas de planejamento reativos são sistemas que estão permanentemente em execução, reagindo a eventos que acontecem no ambiente em que estão situados através da execução de planos que se encontram em uma biblioteca de planos parcialmente instanciados (BORDINI; VIEIRA, 2003, p. 16). Os programas escritos em AgentSpeak(L) possuem sintaxe fortemente baseada em programas escritos utilizando o paradigma de lógica, como Prolog por exemplo.

Segundo Bordini e Vieira (2003, p. 16), um agente AgentSpeak(L) corresponde à especificação de um conjunto de crenças que formarão a base de crenças inicial e um conjunto de planos. Um átomo de crença é um predicado de primeira ordem na notação lógica usual (ou fatos, no sentido de programação em lógica) e literais de crença são átomos de crenças ou suas negações.

A linguagem AgentSpeak(L) distingue dois tipos de objetivos: objetivos de realização e objetivos de teste. Objetivos de realização e teste são predicados, tais como crenças, porém com operadores prefixados “!” e “?” respectivamente. Objetivos de realização expressam os desejos do agente e objetivos de teste retornam a unificação do predicado de teste com uma crença do agente ou pode falhar quando não existir nenhuma crença que seja satisfeita (BORDINI; VIEIRA, 2003, p. 16).

Um plano é formado por um evento ativador (denotando o propósito do plano), seguido de uma conjunção de literais de crença representando um contexto que deve ser conseqüência lógica do conjunto de crenças do agente no momento em que o evento é selecionado para o plano ser considerado aplicável. O resto do plano é uma seqüência de

ações básicas ou subobjetivos que o agente deve atingir ou testar quando uma instância do plano é selecionada para execução.

```
+userLogon(UserName, Password)
     :    userPassword(UserName, Password, UserCode) &
          clientBookstore(UserCode, Name, Address) &
          userProfile(UserCode, ProfileCode)
     <-   welcome(Name);
          !ShowBookProfile(UserCode).
```

Quadro 2 – Exemplo de plano em AgentSpeak(L)

O quadro 2 apresenta um exemplo de plano AgentSpeak(L). Em síntese o plano especifica que o agente recebeu uma percepção proveniente do ambiente no qual o usuário informa um *user name* e um *password* para acessar uma página web de vendas de livros, resultando na adição de uma crença *userLogon(UserName, Password)*. Se o usuário possuir um *user name* e uma *password* válidos e for cadastrado como cliente, então o agente pode executar o plano que consiste de: executar a ação básica de apresentar boas vindas ao usuário e a execução de um subplano para apresentar os livros preferidos do usuário. (Assume-se que apresentar boas vindas é uma ação básica que o agente é capaz de executar no ambiente em questão).

A especificação de um agente em AgentSpeak(L) deve ser feita de acordo com a gramática apresentada no Quadro 3.

ag	*::=*	*bs*	*ps*						
bs	*::=*	$b_1 \ldots b_n$							$(n \geq 0)$
at	*::=*	$p(t_1, \ldots, t_n)$							$(n \geq 0)$
ps	*::=*	$p_1 \ldots p_n$							$(n \geq 0)$
p	*::=*	*te : ct* ← *h*							
te	*::=*	*+at*	|	*-at*	|	*+g*	|	*-g*	
ct	*::=*	*at*	|	¬*at*	|	*ct* ^ *ct*	|	*T*	
h	*::=*	*a*	|	*g*	|	*u*	|	*h;h*	
a	*::=*	$A(t_1, \ldots, t_n)$							$(n \geq 0)$
g	*::=*	*!at*	|	*?at*					
u	*::=*	*+at*	|	*-at*					

Fonte: Hübner, Bordini e Vieira (2004, p. 10).

Quadro 3 – Sintaxe AgentSpeak(L)

Na gramática apresentada no quadro 2 as abreviações correspondem a:

a) *ag*: é um agente especificado por um conjunto de crenças *bs* (*beliefs*) correspondendo à base inicial de crenças do agente, e um conjunto de planos *ps* que forma a biblioteca de planos do agente;
b) *at*: são fórmulas atômicas da linguagem, que são predicados onde P é um símbolo predicativo e $t_1,...,t_n$ são termos padrão da lógica de primeira ordem;
c) *p*: é um plano em AgentSpeak(L), onde *te* (*triggering event*) é o evento ativador, *ct* é o contexto do plano (uma conjunção de literais de crença) e *h* é uma seqüência de ações, objetivos ou atualizações de crenças;
d) *te*: é um evento ativador e corresponde à adição/remoção de crenças da base de crenças do agente (+*at* e -*at*, respectivamente), ou à adição/remoção de objetivos (+*g* e -*g*, respectivamente);
e) *g*: são os objetivos e podem ser de realização (*!at*) ou de teste (*?at*);
f) *u*: corresponde a atualização da base de crenças.

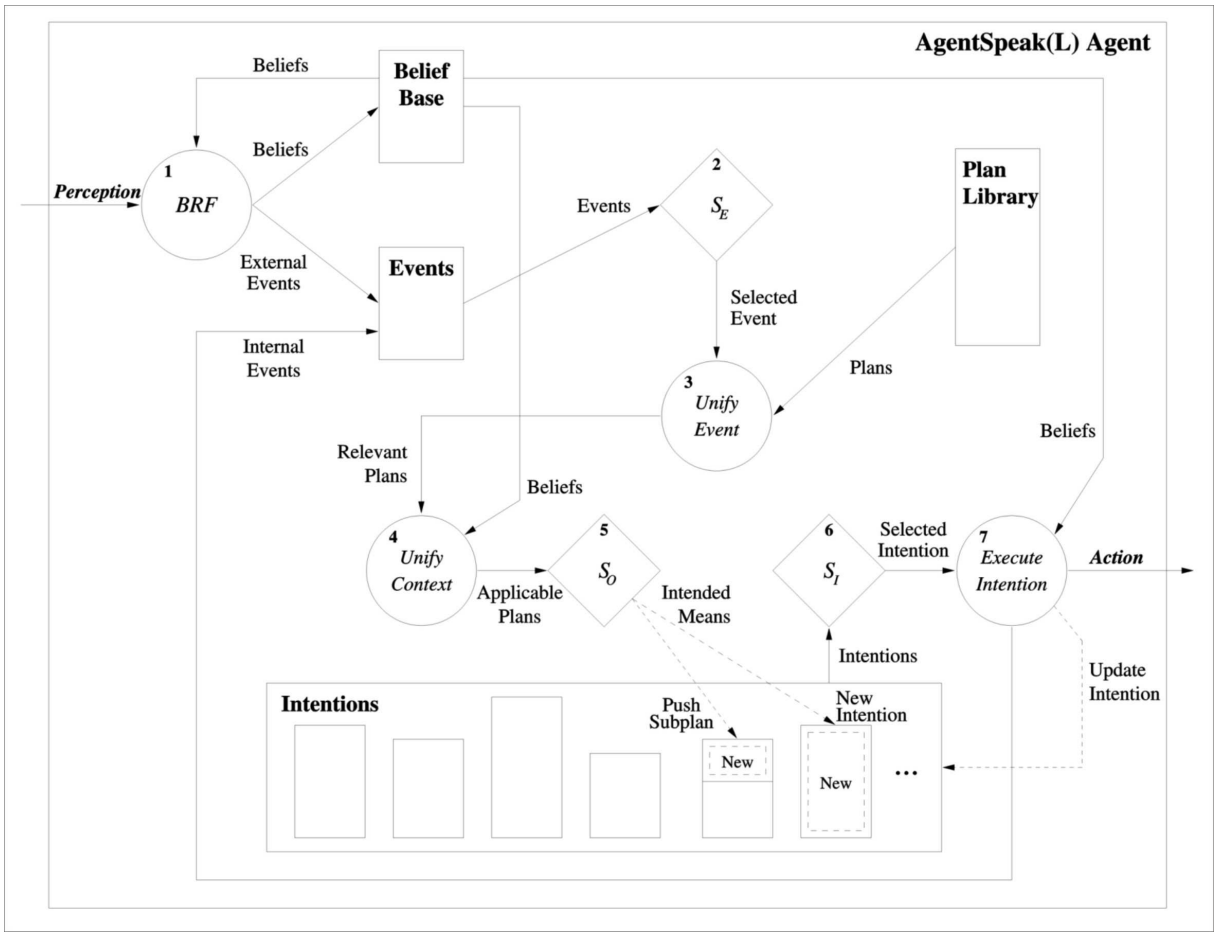
2.4 O INTERPRETADOR JASON

Jason é um interpretador multi-plataforma para a linguagem *AgentSpeak(L)*, desenvolvido em Java e disponível sob a licença *General Public License* (GPL) / *Library General Public License* (LGPL). Um SMA desenvolvido na ferramenta Jason, possui um ambiente onde os agentes estão situados e um conjunto de instâncias de agentes *AgentSpeak(L)*. O ambiente dos agentes deve ser desenvolvido na linguagem Java.

A forma que os programas *AgentSpeak(L)* são interpretados é detalhada nos parágrafos subseqüentes com auxilio da figura 3 reproduzida de Machado e Bordini (2002).

O conjunto de crenças, eventos, planos e intenções são representados por retângulos. Losangos representam a seleção de um elemento de um conjunto. Círculos representam alguns dos processos envolvidos na interpretação de programas AgentSpeak(L).

A função de revisão de crenças (BRF, com o rótulo 1) recebe as percepções do ambiente, adicionando um evento na lista de eventos e uma crença a cada percepção recebida. O conjunto de eventos é atualizado através das percepções do ambiente ou através da execução de novas intenções (i.e., quando subobjetivos são especificados no corpo do plano). A cada adição ou remoção de crença é gerado um evento. Eventos também podem ser gerados quando o agente se compromete a realizar um objetivo.



Fonte: Machado e Bordini (2002).

Figura 3 – Ciclo de interpretação de um programa *AgentSpeak(L)*.

Após ter selecionado um evento S_E (rótulo 2), o interpretador deve unificar o evento selecionado com eventos ativadores das cabeças dos planos, para isso é necessário verificar a biblioteca de planos do agente (rótulo 3). Depois é gerado um conjunto de todos os planos

relevantes para o evento escolhido, através da unificação do contexto dos planos. É verificado o contexto de cada plano para gerar um conjunto de planos aplicáveis (planos que podem ser aplicados no estado atual do agente) e descartados os planos que não podem ser aplicados, *i.e.*, que o contexto falha (rótulo 4). Depois S_O (rótulo 5) escolhe um único plano do conjunto de planos aplicáveis e coloca o plano no topo de uma intenção existente (se o evento for interno) ou cria uma nova intenção no conjunto de intenções (se o evento for externo, i.e. gerado pela percepção do ambiente) (BORDINI; HÜBNER, 2004).

Ainda segundo Bordini, Hübner (2004), a função S_I (rótulo 6) seleciona uma intenção do conjunto de intenções do agente (as intenções do agente são instâncias de planos). E então é executado o plano (rótulo 7) selecionado pela S_I . Isso implica em uma ação básica a ser executada no ambiente, na geração de um evento (em caso da fórmula ser um objetivo de realização) ou na execução de um objetivo de teste (verificando a base de crenças).

Se a intenção for uma ação básica ou um objetivo de teste, o conjunto de intenções precisa ser atualizado. No caso de objetivo de teste, é percorrida a base de crenças, procurando por um átomo de crença que unifique com o predicado de objetivo de teste. Se for possível a unificação do predicado, podem ocorrer instanciações das variáveis nos planos parcialmente instanciados que contenham objetivo de teste e deve ser removido o objetivo do conjunto de intenções, pois ele já foi resolvido. Nos casos em que as ações básicas são selecionadas, o interpretador avisa o ambiente para executar a ação e remove a ação do conjunto de intenções.

Quando todas as fórmulas do corpo do plano forem removidas (isto é, forem executadas) o plano é removido da intenção, assim como o objetivo de realização que o gerou, se esse for o caso. O ciclo de interpretação termina e AgentSpeak(L) começa um novo ciclo, verificando as percepções do ambiente, gerando os eventos necessários e continuando o ciclo de raciocínio do agente como descrito acima (BORDINI; VIEIRA, 2003).

Conforme afirmam Hübner, Bordini e Vieira (2004, p.20), além de interpretar a linguagem AgentSpeak(L) original, o Jason possui os seguintes recursos:

a) negação forte (strong negation), portanto tanto sistemas que consideram mundo-fechado (closed-world) quanto mundo-aberto (open-world) são possíveis;
b) tratamento de falhas em planos;
c) comunicação baseada em atos de fala (incluindo informações de fontes como anotações de crenças);
d) anotações em identificadores de planos, que podem ser utilizadas na elaboração de funções personalizadas para seleção de planos;
e) suporte para o desenvolvimento de ambientes (que normalmente não é programada em AgentSpeak(L); no Jason o ambiente é programado em Java);
f) a possibilidade de executar o SMA distribuidamente em uma rede (usando o SACI);
g) possibilidade de especializar (em Java) as funções de seleção de planos, as funções de confiança e toda a arquitetura do agente (percepção, revisão de crenças, comunicação e atuação);
h) possuir uma biblioteca básica de “ações internas”;
i) possibilitar a extensão da biblioteca de ações internas.

2.4.1 Utilizando o Jason

Nos parágrafos seguintes é demonstrado um exemplo do uso da ferramenta Jason. O código implementado se refere a um robô aspirador de pó, ou seja, quando o robô perceber sujeira no ambiente (o ambiente é representado por coordenadas X e Y, como se fosse um

grid), ele tem como objetivo parar, para poder executar a ação básica no ambiente de aspirar a sujeira. O objetivo da implementação é a demonstração da sintaxe da linguagem AgentSpeak(L), como criar o ambiente e configurar agentes no Jason.

O arquivo de configuração do projeto (arquivo com extensão .mas2j) de SMA utilizado na ferramenta Jason é mostrada no quadro 4. Deve ser informado qual o tipo da arquitetura (no caso, arquitetura centralizada, *i.e.*, execução em uma única máquina), qual é a classe java que simula o ambiente (a classe Mundo) dos agentes e quais são os agentes (o agente robô que tem seu código no arquivo robo.asl). Para ver mais detalhe sobre a BNF veja (BORDINI; HÜBNER et al., 2004).

```
MAS robotAspirador{
     architecture:
                Centralised
     enviroment:
                Mundo
     agents:
                robo  robo.asl;
```

Fonte: Appio (2004, p. 26).

Quadro 4 – Configuração do arquivo de projeto na ferramenta Jason.

O ambiente onde o robô está atuando é apresentado no quadro 5. É simulado uma sala 2*x*2 contendo sujeira em dois lugares. O robô consegue perceber sujeira através dos seus sensores. O método `executeAction` simula os pedidos de atuação do robô, no caso do quadro 6, estas ações podem ser andar (move o agente para uma nova posição *x*, *y*) ou aspirar (remove o lixo do ambiente). Quando o robô faz percepção o simulador do ambiente lhe envia onde tem sujeira e a sua posição no ambiente, através do método `getPercepts`.

As percepções do ambiente do agente robô são: a posição que ele se encontra no ambiente através do predicado `pos(X,Y)` (ver marca (3) no quadro 5) e a crença que tem lixo onde ele se encontra, representada pelo predicado `temLixo` (ver marca (4)). As ações que o agente executa no ambiente são: a ação de aspirar ao lixo (ver marca (2)) na coordenada

onde ele se encontra e mover (ver marca (1)) para uma outra posição no mundo representadas pelos predicados `aspirarLixo` e `andar(posLivre)` respectivamente.

No quadro 6 é apresentado o código AgentSpeak(L) do robô aspirador de pó. Os planos são denotados por um *label* (P*n*) para que se possa referir ao plano no texto que segue.

O plano P1 é executado sempre que o agente recebe do ambiente uma percepção `pos(X, Y)`, ou seja, quando ele perceber uma nova posição, estiver procurando por lixo e não tem lixo na posição que ele se encontra, então ele tem como intenção executar o corpo do plano que é apenas executar a ação no ambiente `andar(posLivre)` movendo o robô para a próxima posição no ambiente.

O plano P2 é selecionado para execução quando o agente receber do ambiente o predicado `temLixo` e o robô estiver procurando por lixo, as intenções do agente são: parar de procurar por lixo (pois já encontrou), aspirar o lixo e continuar a procurar.

Sendo que o corpo do plano é constituído por objetivos de realização, ou seja, o robô tem como primeiro objetivo executar o plano P3 que remove a crença `procurandoLixo`, depois o agente tem como objetivo executar o plano P4 que apenas executa a ação no ambiente de aspirar o lixo e por fim executa o plano P5 que adiciona a crença `procurandoLixo` e move o robô para uma nova posição. Ficando neste ciclo até que o robô percorra todo o ambiente e não encontre mais lixo.

```
import jason.*;
public class Mundo extends Environment {
     private boolean[][] ambiente = new   boolean[2][2];
     ...
     private int x = 0, int y = 0
     Term posAtual = null;
     public Mundo() {
          for(int i=0; i < 2; i++)
               for(int j=0; j < 2; j++)
                    ambiente[i][j] = false;
     //sugeira
     ambiente[0][1] = true;
     ambiente[1][0] = true;
     posAtual = TermImpl.parse("pos("+x+","+y+")");
     addPercept(posAtual);
}
public boolean executeAction(String ag, Term action) {
     if (action.equals(andar)) { //(1)
          y++;
          if (y == 2) {
               y = 0;
               x++;
          }
          if (x == 2) //acabou grid
               return true;
          System.out.println("robo x="+x+"; y="+y);
     } else if (action.equals(aspirar)) { //(2)
          if ( ambiente[x][y] ) {
               ambiente[x][y] = false;
               removePercept(temLixo);
               ambiente[x][y] = false;
          }
     }
     removePercept(posAtual);
     posAtual = TermImpl.parse("pos("+x+","+y+")");
     addPercept(posAtual); //(3)
     if (ambiente[x][y])
          addPercept(temLixo); //(4)
     return true;
     }
}
```

Fonte: Appio (2004, p. 27).

Quadro 5 – Ambiente do agente robô aspirador de pó

```
procurandoLixo.
+ pos(X, Y) : procurandoLixo & not (temLixo)    (P1)
     ← andar(posLivre).
+ temLixo : procurandoLixo                      (P2)
     ← !parar;
     !executeAcaoAsp;
     !continar.
+!parar : true                                  (P3)
     ← - procurandoLixo.
+!executeAcaoAsp : true                         (P4)
     ← aspirarLixo.
+!continar : true                               (P5)
     ← + procurandoLixo;
     andar(posLivre).
```

Fonte: Appio (2004, p. 28).

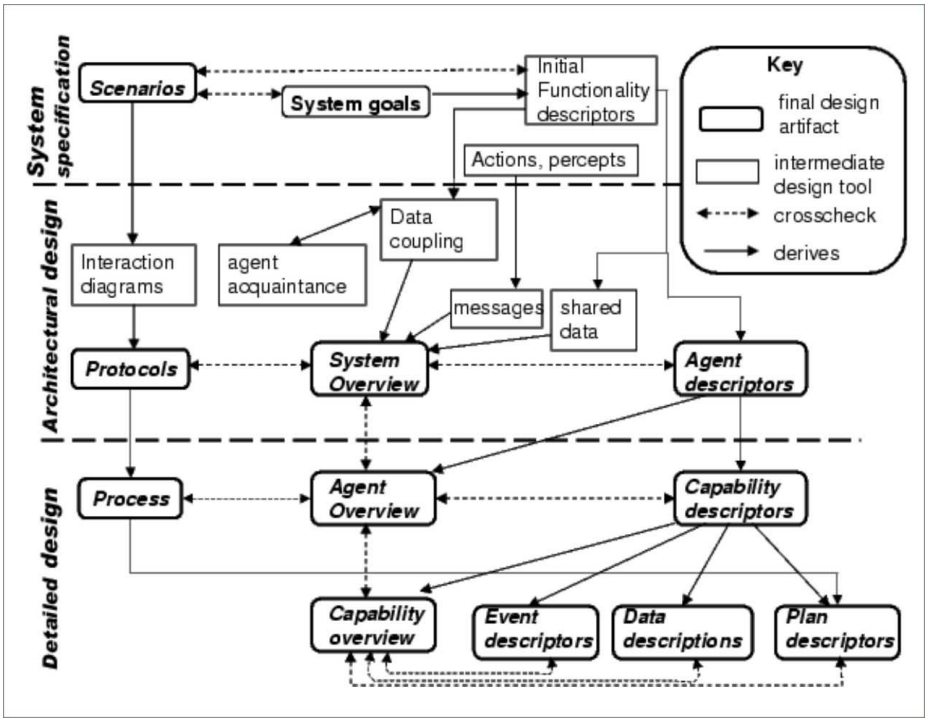
Quadro 6 – Planos AgentSpeak(L) do agente robô aspirador

2.5 A METODOLOGIA PROMETHEUS

Tendo em vista que as diversas linguagens de modelagem de software, principalmente as baseadas no paradigma da programação orientada a objetos como a *Unified Modeling Language* (UML), nem sempre atendem de forma adequada as necessidades para a modelagem de SMA, Padgham e Winikoff (2004) definiram a metodologia Prometheus, que por sua vez facilita a especificação, projeto, implementação e testes de aspectos característicos de SMA.

A metodologia Prometheus é apresentada na obra *Developing intelligent agent systems: a practical guide*, onde os autores realizam um estudo de caso através da modelagem de uma livraria virtual, onde o usuário por meio de uma página web tem a possibilidade de localizar, obter informações e comprar livros pela internet.

Conforme descrito em Padgham e Winikoff (2004, p. 23), a metodologia Prometheus consiste de três fases, ilustradas na figura 4.



Fonte: Padgham e Winikoff (2004, p. 23).

Figura 4 – Fases da metodologia Prometheus

A seguir é apresentada uma breve descrição das três fases da metodologia, sendo que alguns detalhes são omitidos, pois se pretende nesse momento prover uma visão global do funcionamento da metodologia e não necessariamente sua total compreensão. É importante ressaltar nesse momento que todos os conceitos e diagramas apresentados nesta fase se referem unicamente a metodologia Prometheus, sendo que não são feitas referências a UML.

2.5.1 Especificação do sistema

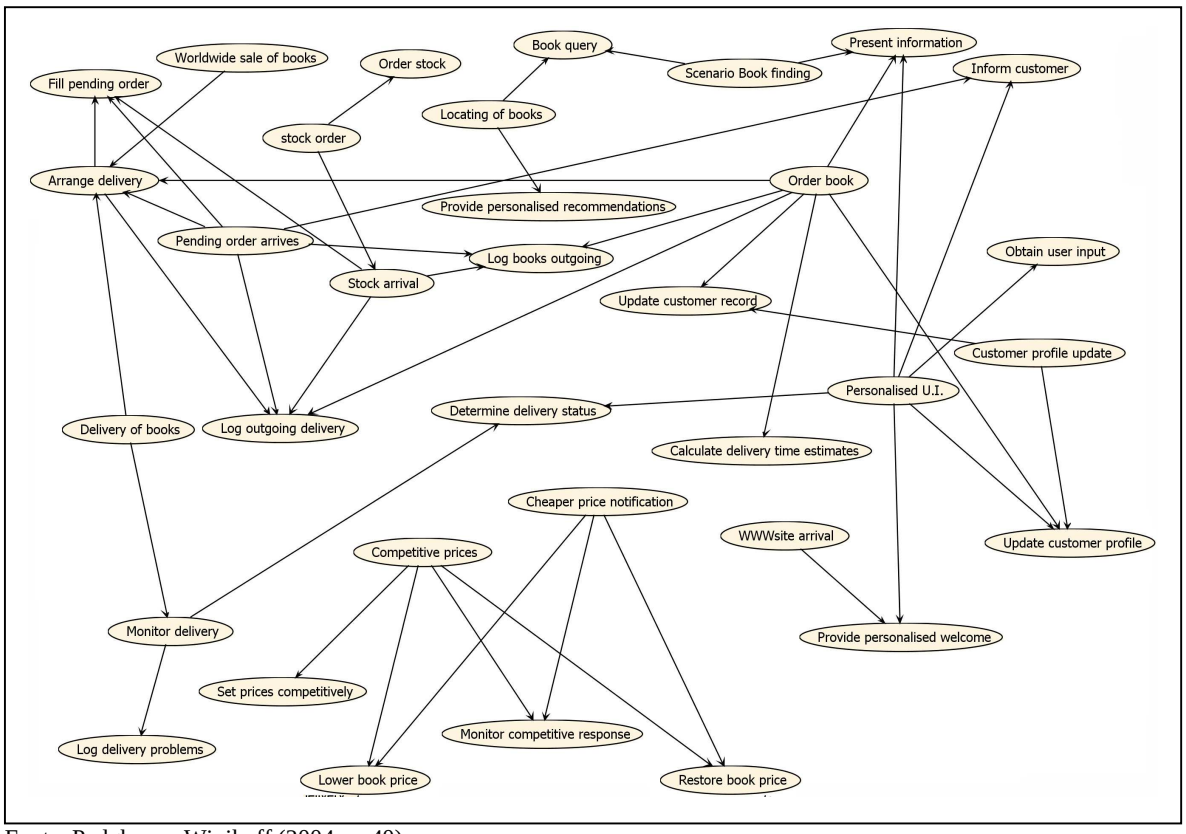
A fase de Especificação do sistema (*System specification)* tem como foco: identificar as metas, desenvolver os cenários de caso de uso ilustrando a operação do sistema, identificar as funcionalidades e especificar uma interface entre o sistema e o ambiente em termos de ações e percepções (PADGHAM; WINIKOFF, 2004, p 24).

As metas são a base para o desenvolvimento de SMA, elas especificam o que o sistema precisa ser capaz de fazer, e não sofrem muitas mudanças no decorrer da

implementação, ao contrário das funcionalidades do sistema (PADGHAM; WINIKOFF, 2004, p 25).

Podem-se extrair, no caso de uma livraria virtual, as seguintes metas: disponibilizar venda mundial de livros, interação completamente on-line com o cliente, disponibilizar uma grande quantidade de livros para venda, prover uma interface personalizada, prover maneiras de localizar os livros, realizar vendas, entregas e possuir preços competitivos.

Depois de identificadas as metas, elas devem ser refinadas e posteriormente extraídas submetas conforme a digrama demonstrado na figura 5, onde por exemplo, para a meta preços competitivos *(competitive prices)*, pode-se extrair submetas para alcançar a meta principal como fixar preços competitivos (*set prices competitively*), reduzir o preço dos livros (*lower book price*), monitorar o efeito dos preços competitivos (*monitor competitive response*) e restaurar o preço dos livros (*restore book price*).

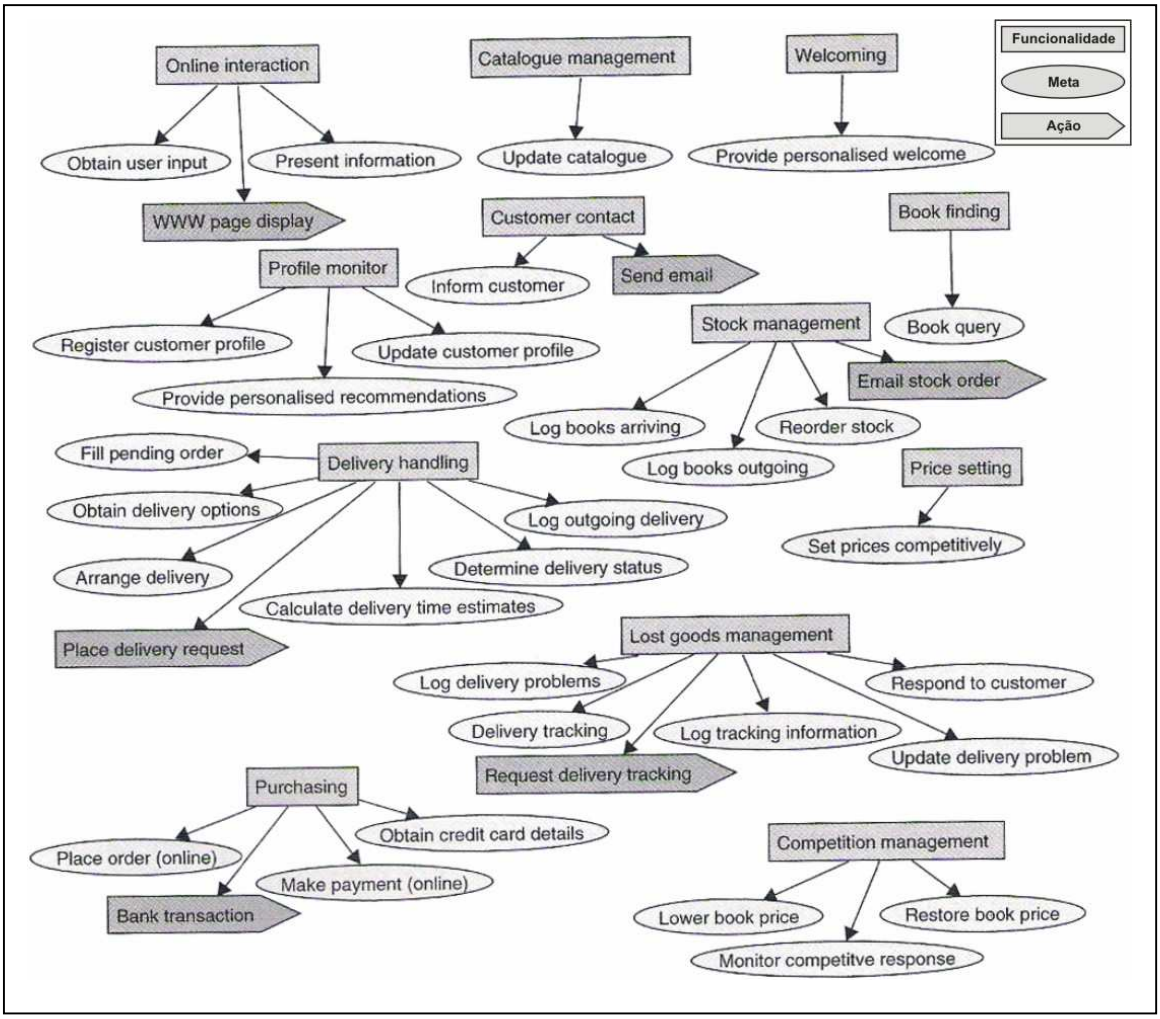


Fonte: Padgham e Winikoff (2004, p. 40).

Figura 5 – Metas para livraria virtual

Os casos de uso são herdados da orientação-objeto, com algumas diferenciações, e são exemplos da operação do sistema. De um ponto de vista mais amplo, um caso de uso consiste de uma seqüência de passos que ocorrem durante a execução do sistema, incluindo possíveis exceções (PADGHAM; WINIKOFF, 2004, p 25).

Funcionalidades são fragmentos de comportamentos que são relacionados com as metas, dados, percepções e ações conforme ilustra a figura 6. Como exemplo, na definição de uma livraria eletrônica, pode-se incluir funcionalidades como “entrega”, “reposição de estoque”, etc (PADGHAM; WINIKOFF, 2004, p 25).



Fonte: Padgham e Winikoff (2004, p. 42).

Figura 6 – Funcionalidades para a livraria virtual

Tomando como exemplo a funcionalidade gerência de estoque *(stock management)* da figura 6, que tem por finalidade monitorar as entradas e saídas de estoque é possível identificar as metas: registrar saídas de livros, registrar a entrada de novos livros, organizar o estoque e a ação enviar e-mail.

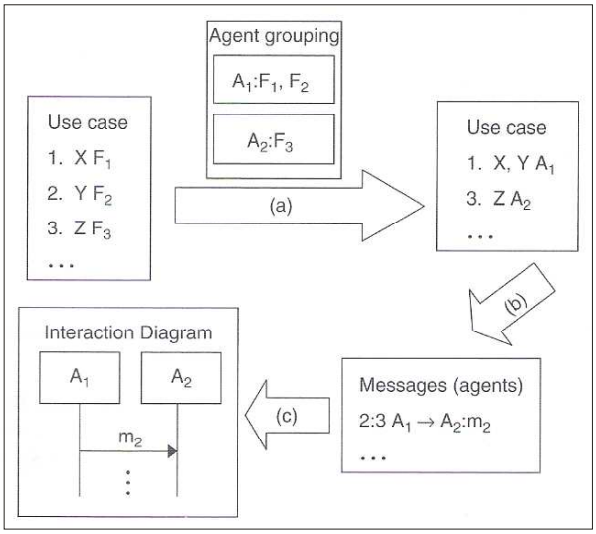
Neste momento define-se ainda o ambiente em que cada agente será situado. Definidas as ações (o que agente efetua no ambiente) e percepções (informação vinda do ambiente).

2.5.2 Desenvolvimento da arquitetura

A fase de Desenvolvimento da arquitetura (*Architectural design*) utiliza as saídas da fase de Especificação do sistema para determinar quais agentes existirão e como os mesmos irão interagir. Esta fase envolve três atividades: definição dos tipos de agentes, definição da estrutura do sistema e definição das interações entre os agentes.

O principal objetivo desta fase é decidir que tipo de agente deve-se implementar. Isso é feito agrupando-se funcionalidades dentro de tipos de agente, de modo que cada tipo de agente consiste de uma ou mais funcionalidades (PADGHAM; WINIKOFF, 2004, p 26).

Segundo Padgham e Winikoff (2004, p. 68), uma vez decididos os tipos de agentes do sistema, deve-se especificar a interação entre eles, capturando os aspectos dinâmicos do sistema. Conforme demonstra a figura 7, o processo de desenvolvimento do diagrama de interação consiste em capturar os cenários obtidos na fase de especificação do sistema e (a) substituir cada funcionalidade pelo agente que a possui; (b) inserir uma comunicação entre os agentes e (c) expressar o resultado em um diagrama de interação.

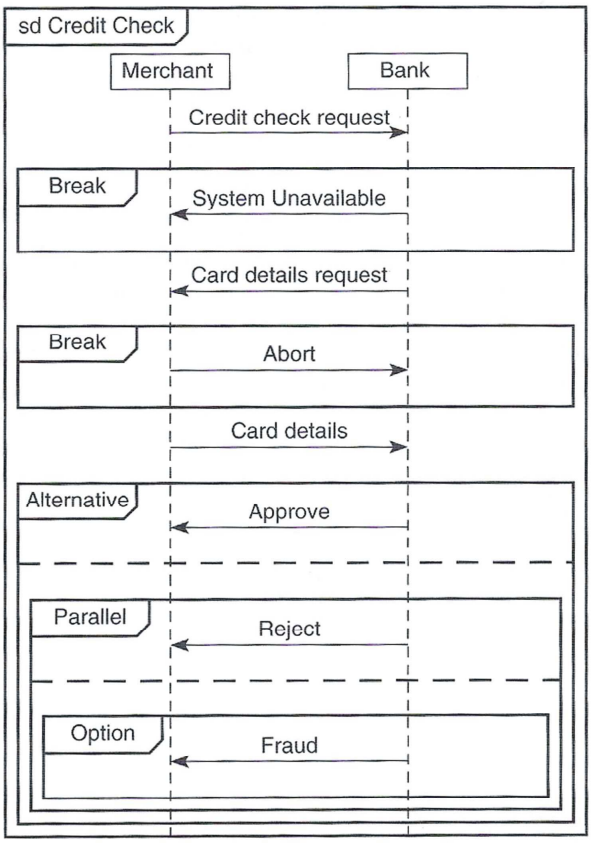


Fonte: Padgham e Winikoff (2004, p. 68).

Figura 7 – Conversão de um caso de uso para um diagrama de interação

Após o desenvolvimento do diagrama de interação deve-se definir exatamente como será feita a comunicação entre os agentes, isto é feito através de diagrama de protocolo que define exatamente a seqüência dessas interações no sistema.

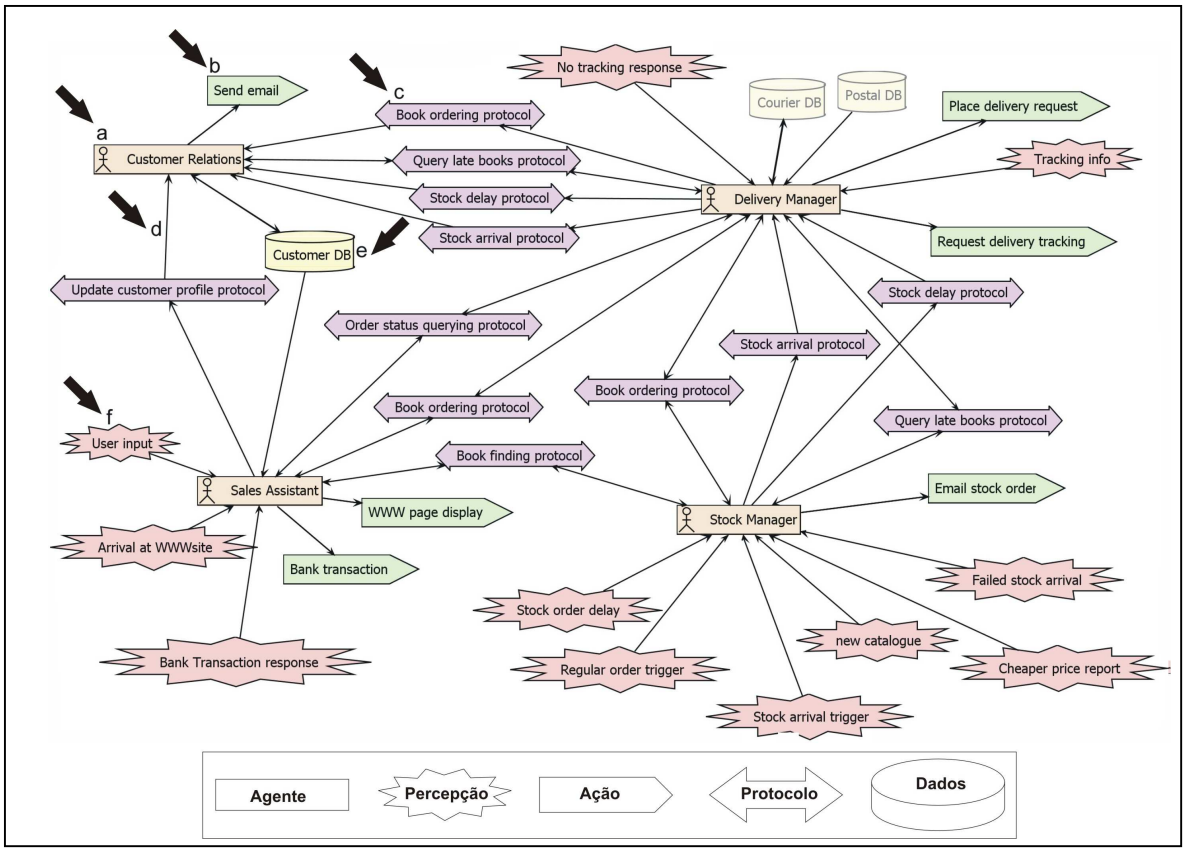
Na figura 8 é apresentado o diagrama de protocolo para o processo de checagem do cartão de crédito. Neste diagrama são apresentados os personagens comerciante *(merchant)* e banco *(bank)* e o processo de interação entre eles. O protocolo começa enviando para o banco uma mensagem para checagem do cartão, o banco responde com os detalhes do cartão de crédito. O protocolo segue até que o banco aprove, rejeite ou informe que trata-se de um cartão fraudulento. Conforme ilustra a figura 8 o envio de *reject* e *fraud* pode acontecer simultaneamente.



Fonte: Padgham e Winikoff (2004, p. 75).

Figura 8 – Diagrama de protocolo para checagem do cartão de crédito

Tendo especificado os agentes dentro do sistema e a comunicação entre eles, deve-se apresentar esta informação no *System overview diagram*, no qual captura-se na forma de um diagrama a arquitetura global do sistema, apresentando qual agente reage para cada percepção, assim como quais ações o agente executa no ambiente (PADGHAM; WINIKOFF, 2004, p 26). Na figura 9 estão representados os agentes (a), as ações (b), os protocolos (c), as mensagens com resposta (d), os dados (e) e as percepções (f) existentes em um SMA.



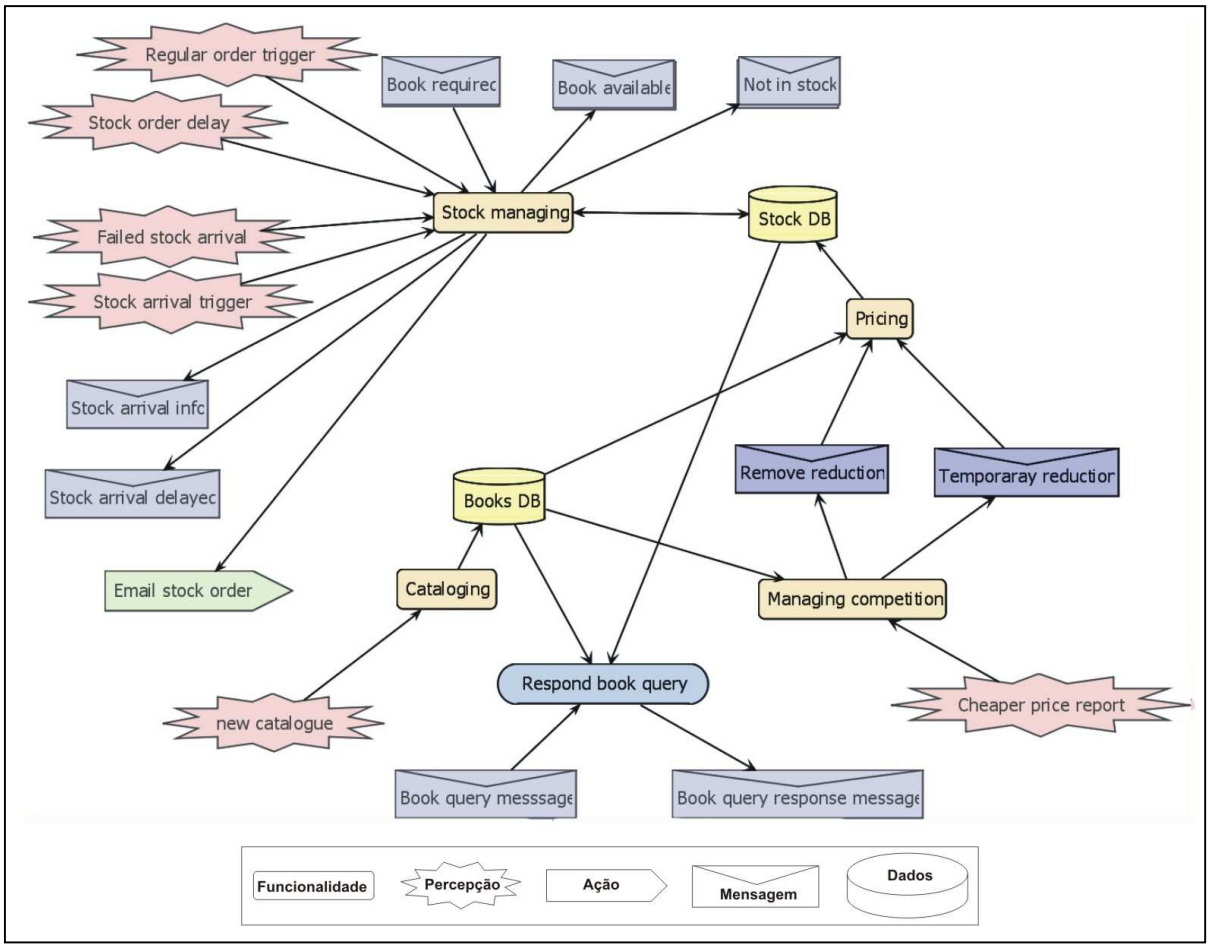
Fonte: Padgham e Winikoff (2004, p. 154).

Figura 9 – System Overview Diagram

2.5.3 Projeto detalhado

A fase de Projeto detalhado (*Detailed design)* foca o desenvolvimento da estrutura interna de cada agente e como ele realizará sua tarefa dentro do sistema. Progressivamente refina-se cada agente definindo capacidades (módulos dentro do agente), eventos internos, planos e detalhes da estrutura de dados (PADGHAM; WINIKOFF, 2004, p 26).

A fase de Projeto detalhado é divida em duas partes. A primeira parte trata do refinamento dos agentes em termos de capacidades, originando o *Agent overview diagram*, que mostra o relacionamento entre capacidades possibilitando uma visão interna dos agentes.



Fonte: Padgham e Winikoff (2004, p. 103).

Figura 10 – Agent overview diagram: gerente de estoque

A figura 10 demonstra um *Agent overview diagram* para o agente Gerente de Estoque (*Stock Manager*) da livraria virtual. O diagrama apresenta as interfaces do agente, descrevendo as entidades externas como eventos, ações e percepções. O agente em questão possui a capacidade de gerenciar o estoque, preço e catálogos.

A segunda parte da fase *Detailed design* trata como os planos serão organizados dentro de uma capacidade, os eventos que serão gerados e controlados por esse plano, a especificação do algoritmo, tanto quanto associado dados (ou crenças), levando-se em consideração a plataforma de desenvolvimento, pois deve-se nesse momento fornecer artifícios para a implementação (PADGHAM; WINIKOFF, 2004, p 27).

Neste estágio, cada capacidade é decomposta em capacidades adicionais, ou eventualmente, em um conjunto de planos que provê os detalhes de como reagir a situações, ou alcançar metas (PADGHAM; WINIKOFF, 2004, p 27).

2.5.4 As ferramentas Jack e Prometheus

Atualmente existem duas ferramentas que utilizam o Prometheus. O ambiente de desenvolvimento JACK, que inclui uma ferramenta de modelagem para a construção dos diagramas, resultando na geração do código na linguagem de programação JACK. O JACK Development Environment (JDE) fornece suporte à metodologia Prometheus pelo fato dos conceitos utilizados por JACK corresponderem aos artefatos gerados na fase de projeto detalhado da metodologia.

A outra ferramenta é o Prometheus Design Tool (PDT). O PDT possui as seguinte funcionalidades: realiza a checagem de inconsistências, gera automaticamente um conjunto de diagramas de acordo com a metodologia e gera automaticamente a descrição do projeto (HyperText Markup Language - HTML), o que inclui descritores para cada entidade, um dicionário para o projeto e os diagramas.

2.6 TRABALHOS CORRELATOS

No que se refere ao software desenvolvido, Padgham e Winikoff (2004) disponibilizam como material base de sua obra, alguns trechos de implementação de uma livraria virtual utilizando a linguagem JACK. Um agente Jack pode ser implementado no *Jack Intelligent Agents* que é um ambiente de desenvolvimento integrado com Java que inclui

todos seus componentes, assim como oferece extensões específicas de implementação e comportamento de agentes, empregando os conceitos do modelo BDI.

No que tange a utilização da linguagem AgentSpeak(L), várias implementações já foram feitas nas mais diversas áreas. Appio (2004) apresenta um sistema para criar estratégias de armadilha em um jogo tipo Pacman, onde os personagens fantasmas são concebidos como agentes que criam e cooperaram na execução das armadilhas, dificultando a vitória do personagem come-come, que é controlado por um usuário.

Calcín, Okuyama e Dias (2004) apresentam uma simulação do processo de decisão na compra e vendas de mercadorias, onde é feito um estudo de um SMA que objetiva simular este processo em um ambiente virtual, a decisão do consumidor está baseada nas características do produto e na reputação do vendedor.

3 ESPECIFICAÇÃO DO SISTEMA

As seções seguintes descrevem a especificação da livraria virtual, utilizando os conceitos apresentados anteriormente.

3.1 VISÃO GERAL

Os principais requisitos para o desenvolvimento deste estudo de caso são:

a) implementação de um sistema web na linguagem Java (JSP) que deve constituir o agente responsável pela apresentação web e pela interação com os agentes da aplicação desenvolvidos em Jason através da linguagem AgentSpeak(L);

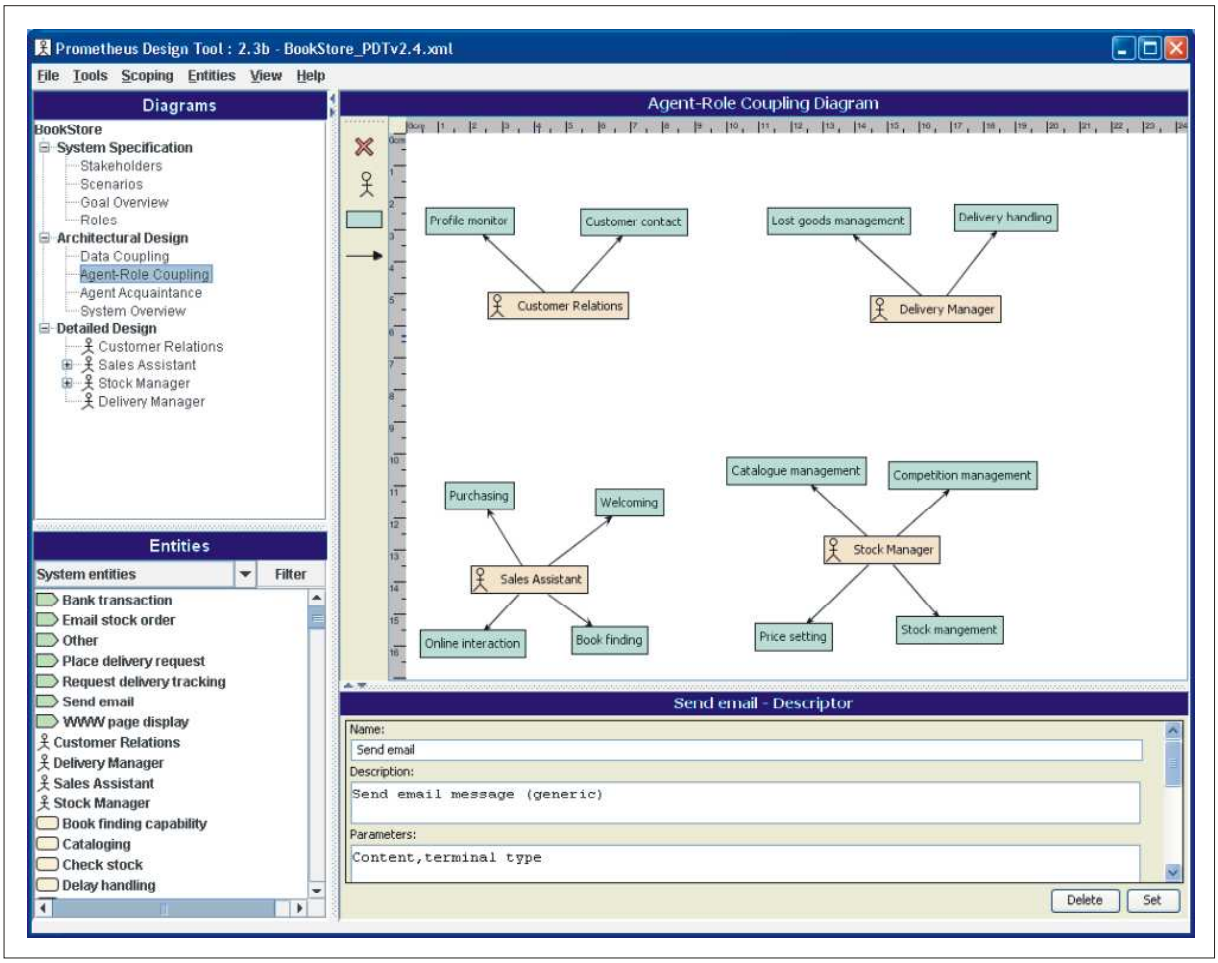
b) utilizar o paradigma de programação orientada a agentes através da linguagem AgentSpeak(L), interpretada pela ferramenta Jason, para desenvolver a camada lógica da aplicação;

c) desenvolver agentes BDI que devem atuar sobre um sistema de livraria virtual, de modo a possibilitar a localização, escolha e compra de livros através de sistema web.

3.2 PROMETHEUS DESIGN TOOL

Conforme já mencionado anteriormente, a especificação do sistema implementado neste trabalho foi previamente realizada por Lin Padgham e Michael Winikoff, através da metodologia Prometheus, na obra *Developing intelligent agent systems: a practical guide*. Para isto foi utilizada a ferramenta *Prometheus Design Tool*, conforme ilustra a figura 11, que por sua vez possibilita o desenho de todos os diagramas apresentados anteriormente e suas

respectivas documentações.



Fonte: Padgham e Winikoff (2004, p. 103).

Figura 11 – Prometheus Design Tool

O processo de especificação de sistemas multiagentes através desta metodologia, consiste de um trabalho extremamente detalhado. Deste o momento de especificação do sistema (*System specification*), passando pela definição da arquitetura (*Architectural design*) até o projeto detalhado (*Detailed design*), produziu-se um documento de noventa e nove páginas, contendo um material rico em detalhes para o desenvolvimento da aplicação.

Nas sessões seguintes serão apresentados os principais diagramas para compreensão do funcionamento do sistema.

3.3 ESPECIFICAÇÃO

Nesta sessão são apresentadas as funcionalidades que o sistema deverá possuir e a descrição dos principais cenários que demonstram as interações entre o cliente e a página web.

3.3.1 Funcionalidades

As funcionalidades inerentes à livraria virtual são as seguintes:

a) interação on-line: esta funcionalidade é responsável por gerenciar as interações com o usuário via website, ela é ativada no momento que usuário entra na página. Para esta interação é necessário acessar os registros de clientes e pedidos. Basicamente, obtém-se dados do usuário e apresentam-se as informações pertinentes;

b) boas-vindas: é necessário apresentar mensagens de boas vindas quando o usuário fizer *logon* no website. Para isto é necessário manter um registro dos usuários;

c) gerenciar estoque: é fundamental controlar o estoque de livros. Deve-se monitorar informações como quantidade disponível, pedidos e novas encomendas. Estas informações são obtidas através do registro de estoque, de compras e vendas;

d) localizar livros: deve-se disponibilizar a possibilidade de localizar livros através de parâmetros de buscas informados pelo usuário;

e) gerenciar entregas: esta funcionalidade é responsável por gerenciar as entregas aos clientes. Para isso deve registrar as vendas, calcular prazos de entrega e informar o status de pedidos;

f) ser competitivo: é necessário que em determinados momentos o preço de alguns livros sejam temporariamente reduzidos para se tornarem competitivos em relação aos concorrentes. No momentos conveniente os preços devem ser restaurados.

3.3.2 Cenários

Abaixo são apresentados os principais cenários envolvidos na navegação do website. Alguns cenários demonstrando situações mais triviais são omitidos, pois conforme mencionado anteriormente a especificação completa é extremamente detalhada, de modo que se tornaria inoportuna neste trabalho a apresentação na sua totalidade.

3.3.2.1 Cenário localizar livro

O cenário localizar livro é disparado quando usuário necessita localizar um livro, para isso podem-se informar parâmetros como título, autor e gênero. Este cenário é composto por duas metas (localizar os livros e apresentar as informações) e uma ação (apresentar a página web) conforme ilustra o quadro 7.

Name	Scenario Book finding scenario						
Description	Finds book(s) as requested by the user and displays the result.						
Priority	Not Specified						
Stakeholders							
Initiated by	System						
Trigger							
Steps	**#**	**Type**	**Name**	**Role**	**Descrip**	**Data used**	**Data prod**
	1	Goal	Book query	Book finding		Stock DB	
	2	Goal	Present information	Online interaction		Books DB	
	3	Goal	WWW page display	Online interaction			
Variation	No books found that match request.Provide message and suggest changes to the user.						

Fonte: Padgham e Winikoff (2004, p. 145).

Quadro 7 – Cenário localizar livro

3.3.2.2 Cenário comprar livro

Um dos principais cenários envolvidos na navegação do website é momento que o usuário decide por comprar um livro. Neste momento é preciso que o sistema verifique as opções e tempo de entrega e apresente estas informações ao usuário.

Name	Order book scenario					
Description	An order is received from WWW page interface (goal Place order).Information is obtained in order to place the order and order is placed.					
Priority	Not Specified					
Stakeholders						
Initiated by	System					
Trigger						
Steps	#	**Type**	**Name**	**Role**	**Data used**	**Data prod**
	1	Goal	Obtain Delivery options	Delivery handling	Courier DB, Postal DB	
	2	Goal	Calculate delivery time	Delivery handling	Courier DB, Postal DB	
	3	Goal	Present information	Online interaction		
	4	Percept	User input	Online interaction		
	5	Goal	Arrange delivery	Delivery handling		
	6	Action	Place delivery request	Delivery handling		
	7	Goal	Log outgoing delivery	Delivery handling		Customer Orders
	8	Goal	Log books outgoing	Stock mangement		Stock DB
	9	Goal	Update customer record	Profile monitor		Customer DB
Variation						

Fonte: Adaptado de Padgham e Winikoff (2004, p. 146).

Quadro 8 – Cenário localizar livro

Após a confirmação da compra, é preciso que o sistema organize esta informação para que a compra possa ser enviada, para isso é necessário registrar a venda do livro, atualizar o estoque e o registro do cliente a respeito da nova aquisição.

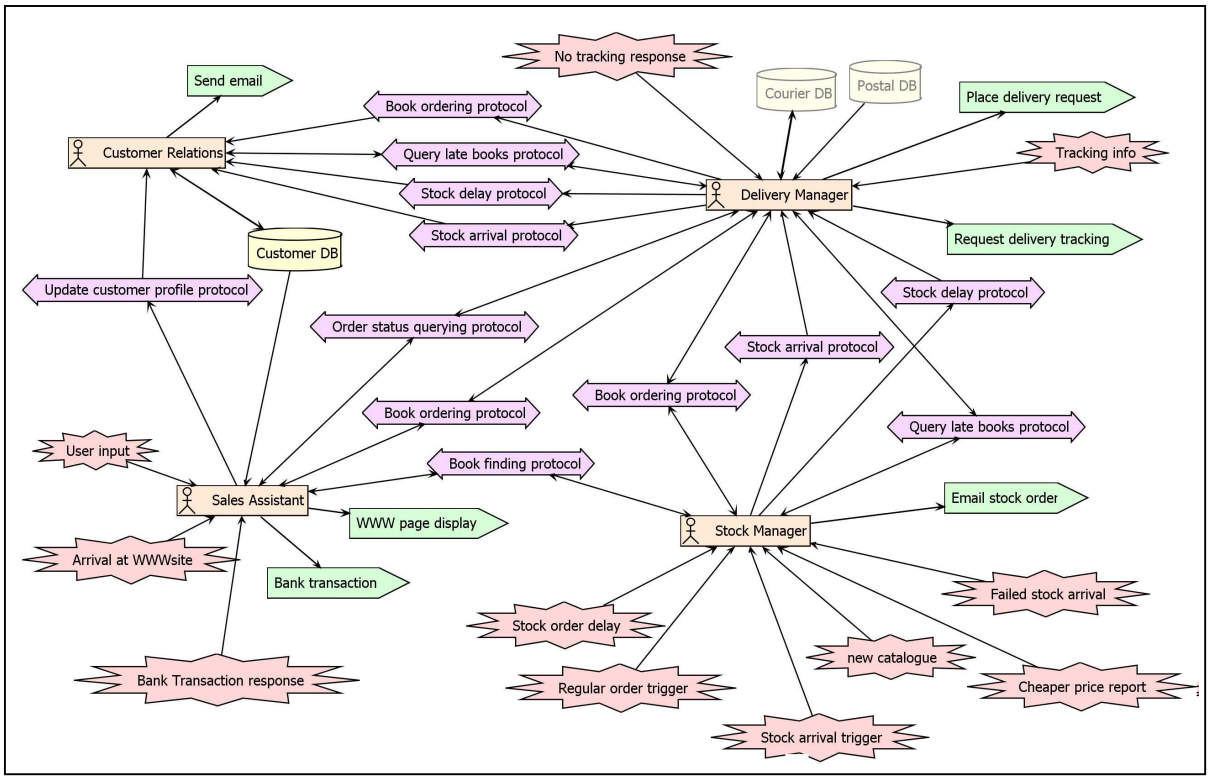
Conforme ilustrado no quadro 8, para realização das metas e ações descridas neste cenário são obtidas informações da base de dados (ou crenças) de entrega e de clientes, e atualizado os dados de pedidos, estoque e clientes.

3.3.3 Definição da arquitetura

Nesta seção são apresentados três artefatos fundamentais na especificação do sistema: os agentes, as percepções e as ações que devem ser realizadas. A forma de apresentação destes artefatos segue conforme a metodologia de especificação utilizada neste trabalho, porém de maneira mais objetiva.

Para os agentes é apresentado o nome, a descrição das suas funcionalidades, e de maneira sintética, suas percepções, ações e metas, sendo que posteriormente é feita a especificação mais detalha dos mesmos.

A figura 12 (*System overview diagram*) apresenta de maneira esquemática os itens abordados nesta seção, demonstrando graficamente os agentes, as percepções e as ações detalhadas a seguir.



Fonte: Padgham e Winikoff (2004, p. 154).

Figura 12 – System overview diagram

3.3.4 Agentes

A seguir são exibidos os agentes que o compõe o sistema.

3.3.4.1 Agente assistente de vendas (*Agent Sales Assistant*)

Descrição: Este agente negocia com o cliente através de interações on-line, de maneira análoga ao assistente real de uma loja qualquer. Isto inclui a ajuda ao cliente a procurar um livro adequado ou se necessário apurar as devidas informações.

Percepções: A entrada do usuário no web site. Seleção de um item na página.

Ações: Mostrar o conteúdo da página web.

Metas: Localizar livros. Obter as opções de entrega. Registrar a informações fornecidas pelo usuário. Apresentar informações. Prover uma interação personalizada para cada cliente. Registrar um arquivo com as preferências do usuário obtidas através da navegação pela página web. Atualizar o registro do cliente.

3.3.4.2 Agente gerente de entrega (*Agent Delivery Manager*)

Descrição: Organiza todos os aspectos relacionados à entrega do produto ao cliente. Negocia qualquer problema relacionado à entrega, inclusive notifica o agente de relações com o cliente.

Percepções: Caso não houver uma resposta da chegada da mercadoria após certo tempo isto é percebido pelo agente.

Ações: Enviar uma solicitação ao serviço de entrega para identificar a localização de uma determinada mercadoria que ainda não foi entregue.

Metas: Atender uma ordem pendente. Obter opções de entrega. Organizar a entrega. Calcular o tempo estimado de entrega. Determinar o status da entrega. Registrar a saída da entrega. Registrar os problemas na entrega. Registrar informações da entrega. Atualizar os problemas de entrega.

3.3.4.3 Agente relações com cliente (*Agent Customer Relations*)

Descrição: Este agente é responsável por todas as transações off-line com o cliente, inclusive da manutenção da base de dados com informações sobre o cliente.

Percepções: (Sem percepção).

Ações: Enviar e-mail.

Metas: Informar o consumidor. Prover recomendações personalizadas. Registrar o perfil do cliente. Responder ao consumidor. Atualizar o registro do consumidor.

3.3.4.4 Agente gerente de estoque (*Agent Stock Manager*)

Descrição: É responsável pelos livros disponíveis na loja, assegurando que os livros estão disponíveis. Avaliar, reordenar, monitorar entregas, etc.

Percepções: Novo catálogo. Relatório de preços mais baratos. Chegada de um estoque.

Ações: Acionar um provedor.

Metas: Registrar saídas de livros. Registrar chegadas de livros. Ordenar o estoque. Abaixar o preço dos livros. Monitorar as ações do concorrente. Restaurar o preço dos livros. Criar preços competitivos. Gerenciar os estoques.

3.3.5 Percepções

A seguir é feita a descrição de como ocorre as percepções recebidas pelos agentes

a) chegada no website: indica quando um usuário efetuou *login* no website. Nesse momento deve-se obter informações de identificação do usuário e registrar a visita. O agente que responde a esta percepção é o Assistente de vendas (*Sales assistant*). Espera-se que esta percepção não aconteça mais de dez vezes por minuto;

b) entradas do usuário: acontece quando o usuário entra com informações no website através de cliques do mouse ou digitação em campos. O agente que responde a esta percepção é o Assistente de vendas (*Sales assistant*). Espera-se que esta percepção aconteça no máximo em terno de uma ou duas vezes por segundo;

c) novo catálogo: acontece quando se obtém a informação de que existe um novo catálogo de livros. Deve-se então atualizar as informações a respeito de livros, preços, etc. A agente que receberá a percepção é o Gerente de estoque (*Stock manager*). Não deve acontecer mais que dez vezes por mês;

d) chegada de estoque: acontece quando se obtém a informação de que houve a aquisição de novos livros para repor o estoque. As informações são provenientes de um revendedor. Espera-se que não acontece mais que dez vezes por mês;

e) relatório de preços mais baixos: esta percepção acontece quando é informado que um concorrente esta vendendo um livro com o preço mais baixo. O agente que vai realizar as ações convenientes é Gerente de estoque (*Stock manager*);

3.3.5.1 Ações

A principal ação inerente ao website é apresentar a página com o conteúdo pertinente. Esta ação é gerada de diversas maneiras, como por exemplo, quando o agente percebe que o usuário efetuou *login* no website, uma possível ação é apresentar os livros de sua preferência.

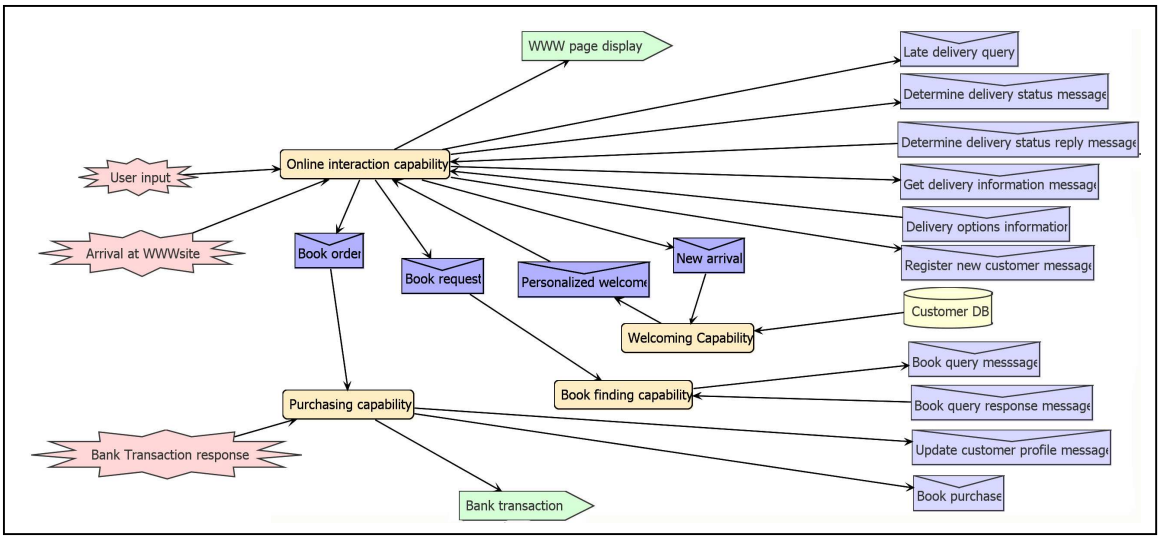
Na especificação original, outras ações são executadas pelos agentes, como efetuar transações bancárias e entrar em contato com empresas de transporte, porém estes itens não são contemplados neste trabalho, portanto são omitidos na especificação.

3.3.6 Projeto detalhado

Na fase de projeto detalhado são especificadas as capacidades de cada agente, que compõe o principal componente desta fase de especificação. Abaixo são apresentadas as principais capacidades de cada agente.

3.3.6.1 Agente assistente de vendas (*Agent Sales Assistant*)

Conforme ilustra figura 13, o agente Assistente de vendas possui quatro capacidades fundamentais: localizar livros (*bookfinding*), possibilitar as interações on-line (*online interaction*), compra (*purchasing*) e boas vindas (*welcoming*). A seguir são descritas de maneira mais detalhada estas capacidades:



Fonte: Padgham e Winikoff (2004, p. 182).

Figura 13 – Capacidades de agente Assitente de vendas

a) boas vindas: quando o usuário efetua *login* no website o agente deve apresentar uma mensagem de boas vindas e listar livros de acordo com o perfil do cliente, que é mantido através de um registro com as preferências do usuário;

b) interação on-line: de maneira mais genérica, o agente Assistente de vendas é responsável pelas interações do usuário no website. Toda solicitação do usuário, como localizar livro, verificar preço ou comprar um livro deve ser atendida pelo agente Assistente de vendas;

c) localizar livros: o agente deve possuir a capacidade de localizar livros. Para isso o usuário tem a possibilidade de informar parâmetros de busca, como título do livro, autor, ou gênero. Através destes parâmetros o agente vai consultar sua base de crenças e verificar se algum livro corresponde aos parâmetros informados. Ao localizar um livro, o agente deve armazenar em uma lista as informações sobre a obra, como título, autor, editora e preço para posteriormente enviar para o agente responsável pela página web para ser apresentado ao usuário;

d) compra: o usuário tem a possibilidade de optar por comprar um livro. Nesse momento o agente deve apresentar as informações sobre o livro, preço, calcular o frete e tempo de entrega e posteriormente adicionar uma crença de que o

usuário comprou o livro, para que posteriormente o livro possa ser entregue.

3.3.6.2 Agente gerente de entrega (*Agent Delivery Manager*)

O agente Gerente de entrega é constituído de duas capacidades básicas que são:

a) calcular frete: baseado no CEP de entrega dos livros, no peso e no preço do quilo por quilômetro, o agente deve calcular o preço do frete da compra. Para isso é multiplicada a distância pelo peso em quilos pelo preço por quilômetro;
b) determinar status da entrega: o agente deve possuir um conjunto de crenças para determinar o status da entrega de um pedido (aguardando envio, enviado, entrega atrasada, entregue), de modo que cliente possa consultar a qualquer momento em que status se encontra a compra.

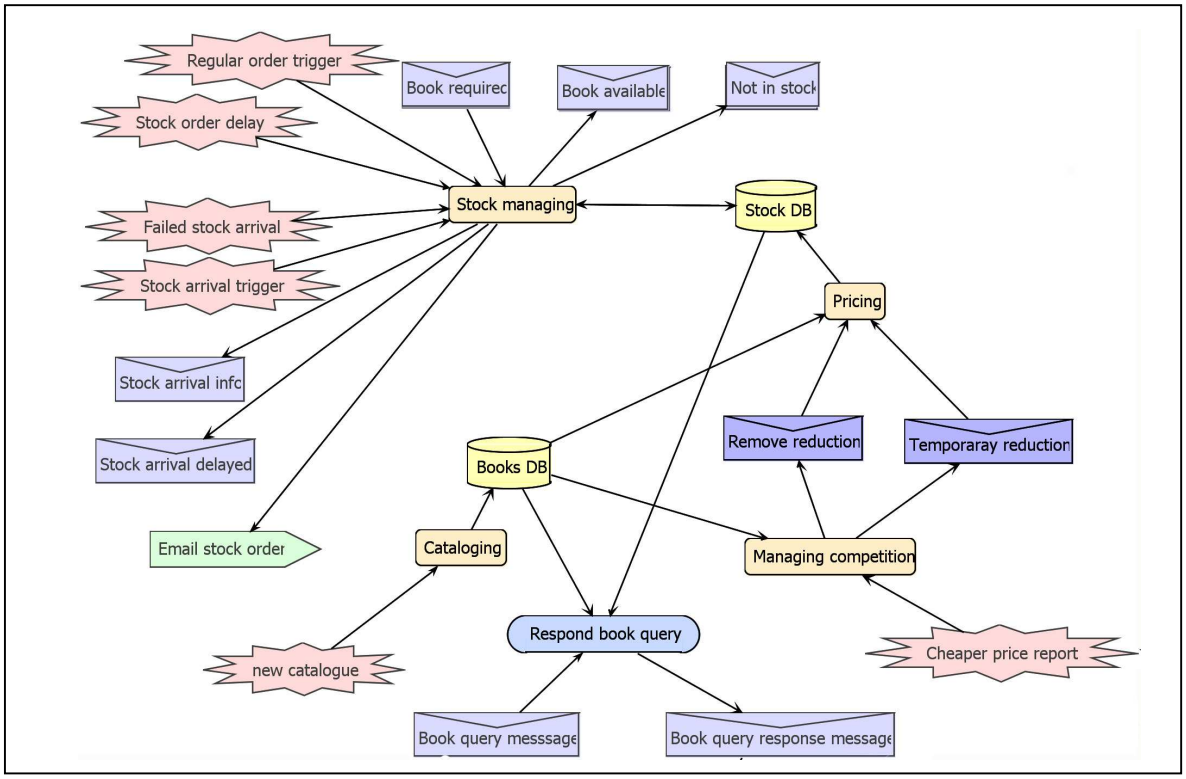
3.3.6.3 Agente relações com cliente (*Agent Customer Relations*)

Este agente é responsável pelas transações off-line com o cliente, de modo que o agente tem a capacidade de enviar recomendações, ou aviso de novos livros que estão disponíveis para compra. Não há participação direta deste agente na navegação do website.

3.3.6.4 Agente gerente de estoque (*Agent Stock Manager*)

A figura 14 apresenta as percepções, as ações e os dados envolvidos nas capacidades do agente Gerente de estoque. Este agente possui de um ponto de vista mais abrangente, a capacidade de gerenciar todos os aspectos relacionados ao estoque dos livros, como por

exemplo, saber a quantidade, registrar a chegada de estoque, saída e preços dos livros. A seguir são especificadas as principais capacidades.



Fonte: Padgham e Winikoff (2004, p. 182).

Figura 14 – Capacidades de agente Gerente de estoque

a) gerenciar o estoque: é preciso garantir que a quantidade de livros em estoque esteja sempre satisfatório, para isso é preciso registrar as saídas e entradas de livros, de modo que sempre que um livro é vendido este precisa ser debitado do estoque, ou de maneira inversa sempre que há novas aquisições, este precisa ser creditado no valor do estoque;

b) catalogar: quando se têm a informação de que existem novos títulos a venda, deve-se de registrar essa informação, para possibilitar que o usuário tenha acesso a essa informação através do website;

c) gerenciar preços: o agente Gerente de estoque deve possuir uma base de crenças com a informação de todos os preços dos livros cadastrados, e ainda possuir a capacidade de baixar os preços temporariamente, para possuir um preço mais competitivo em relação ao concorrente.

4 IMPLEMENTAÇÃO

Nesta seção são abordados detalhes relativos à implementação do sistema, demonstrando sua operacionalidade, técnicas e ferramentas utilizadas no desenvolvimento.

4.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

Três ferramentas possibilitaram o desenvolvimento do protótipo: o Jason 0.8, o Saci e o NetBeans 4.1. A figura 15 apresenta de maneira esquemática a utilização das ferramentas na construção do protótipo.

Os agentes foram desenvolvidos utilizando a ferramenta Jason, que é um interpretador multi-plataforma para a linguagem AgentSpeak(L), disponível sob a licença *General Public License* (GPL) / *Library General Public License* (LGPL).

O Saci possibilitou a comunicação distribuída entre os agentes e o ambiente, que por sua vez foi desenvolvido na linguagem Java (JSP) na ferramenta de desenvolvimento NetBeans 4.1, desenvolvido pela Sun Microsystems Inc., sob a licença *Sun Public License* (SPL).

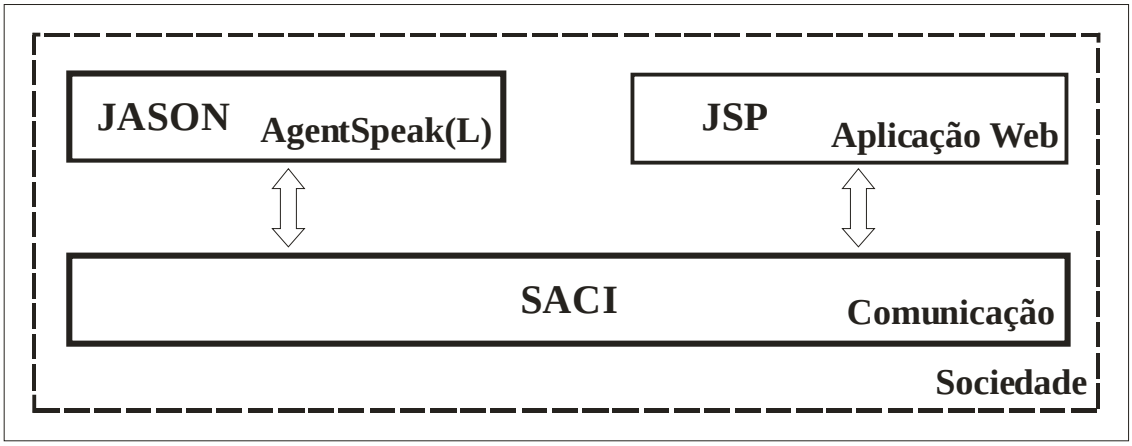


Figura 15 – Camadas da aplicação

4.2 IMPLEMENTAÇÃO DOS AGENTES (JASON / AGENTSPEAK(L))

O desenvolvimento de um SMA através do interpretador Jason inicia-se criando um projeto, representado pela extensão ".mas2j", onde são definidos a infra-estrutura do SMA e os agentes, conforme ilustra o quadro 9.

A infra-estrutura pode ser definida como `centralised`, onde os agentes são executados localmente, ou `saci`, opção utilizada neste trabalho, onde os agentes podem ser executados de maneira distribuída.

O identificador MAS representado no quadro indicada o nome da sociedade dos agentes, e a palavra `agents` é usada para definir o conjunto de agentes que irão fazer parte do SMA. Definindo o projeto, inicia-se a programação dos agentes AgentSpeak(L) que é feita em arquivos com extensão ".asl".

```
MAS eletronicBookstore {
     infrastructure: Saci
     agents:
          deliveryManager;
          customerRelations;
          salesAssistant;
          stockManager;
```

Quadro 9 – Projeto em Jason

De maneira geral, como em qualquer ferramenta comercial, a maior parte das ações executadas pelo usuário na utilização da ferramenta consiste de inserir, atualizar e consultar dados (crenças tratando-se de SMA), isso ocorre através de mensagens que os agentes recebem da interface.

O exemplo demonstrado no quadro 10 apresenta uma seqüência de ações que devem ser executadas pelo agente Assistente de vendas (*Sales assistant*) para que um novo usuário seja cadastrado como cliente da livraria virtual. No momento que é recebido do agente responsável pela interface a mensagem `addClient(...)`, verifica-se se na base de crenças

não há um registro com o mesmo nome de usuário (*username*) informado para o novo cliente. Caso não exista deve-se alcançar o objetivo de realização `!generateSeqCust`, que gera um número de seqüência para que o usuário possa ser adicionado na base de crenças através da ação `+clientBookstore(...)`. O símbolo "_" utilizado no código indica um termo do literal que não armazena informação alguma.

Caso o teste `not clientBookstore(...)` falhe, há um segundo plano que se responsabiliza por executar a ação básica de enviar uma mensagem, informando ao usuário que o *username* informado não é válido.

```
// Adicionado um novo cliente
+received(InterfaceAg, askOne, addClient(UserName, Password, Name,
            Address, Uptown, City, State, Phone, EMail, RG), M)
      :     not clientBookstore(_,UserName,_,_,_,_,_,_,_,_,_)
      <-    !generateSeqCust(Sequence);
            +clientBookstore(Sequence, UserName, Password, Name,Address,
                  Uptown, City, State,Phone, EMail, RG);
            .send(InterfaceAg, tell, "Congratulations! Register OK", M);
            .print("Add client: ", Name, " code: ", Sequence).

// Nome de usuario já existe. Cliente não adiciondo
+received(InterfaceAg, askOne, addClient(UserName,_,_,_,_,_,_,_,_,_), M)
      :     true
      <-    .send(InterfaceAg, tell, "Invalid username", M);
            .print("Invalid username").
```

Quadro 10 – Adição de uma crença em AgentSpeak(L)

O quadro 11 apresenta o código fonte para o plano `userLogon,` implementado para que o usuário efetue *login* no website e sejam apresentados os livros de sua preferência.

Ao receber a mensagem `userLogon(UserName, Password)`, o agente verifica se o usuário realmente existe, se o *username* e o *password* estiverem corretos o agente assistente de vendas envia para o agente web uma mensagem com o nome do usuário.

```
+received(InterfaceAg, askOne, userLogon(UserName,Password), M)
      :     clientBookstore(_,UserName,Password,Name,_,_, _, _, _, _, _)
      <-    .send(InterfaceAg, tell, Name, M);
            .print(Name, " login").

+received(InterfaceAg, askOne, userLogon(UserName, Password), M)
      :     not clientBookstore(_,UserName,Password, Name,_,_,_,_,_,_,_)
      <-    .send(InterfaceAg, tell, "Username or password invalid!", M);
            .print("Username or password invalid!").
```

Quadro 11 – Usuário efetua *login* na livraria virtual

O código apresentado no quadro 12 demonstra a implementação do plano `+!findBook(...)` executado pelo agente Gerente de estoque (*Stock manager*) para localizar livros. O contexto para execução desta ação é descrita no *Scenario Book finding scenario*, onde o usuário decide localizar algum livro.

O plano recebe como parâmetro o nome de um autor, título, editora e um gênero de livro. A ação interna `.findall` é utilizada para localizar os livros e agrupá-los numa lista.

```
+!findBook(    Author, Title, Publisher, Gender)
     :         true
     <-        .findall( CodeBook, book(CodeBook, Title,
                         Author, Publisher, Gender),
               ListBookRet).
```

Quadro 12 – Código fonte para execução de plano de busca

A ação interna `.findall` é composta por três parâmetros. O primeiro parâmetro indica o que deve ser agrupado, no exemplo apresentado no quadro 12 deve-se agrupar o código do livro (*CodeBook*), que por sua vez deve ser um termo do literal de crença passado como segundo parâmetro e o terceiro parâmetro retorna uma lista de *strings* com os códigos dos livros correspondentes a crença `book(...)`.

De acordo com a especificação, uma das funcionalidades do agente Gerente de estoque, é controlar as entradas e saídas de livros. No quadro 13 é apresentada a implementação da ação básica `+!bookOutgoing(CodeBook, Quantity)`. Esta ação é executada quando o usuário clica no botão *Confirm* na tela de vendas.

```
+!bookOutgoing(CodeBook, Quantity)
     :    stockBook(CodeBook,BookCase,BookCaseName,QuantAct)
                & QuantAct >= Quantity
     <-   -stockBook(CodeBook, BookCase, BookCaseName,
                QuantAct);
          +stockBook(CodeBook, BookCase, BookCaseName,
                QuantAct-Quantity);
          .print("Book outgoing - CurrentBalance: ",
                QuantAct-Quantity).
```

Quadro 13 – Ação básica para registrar a redução do estoque

Para a execução da ação, deve-se primeiramente verificar se há em estoque a quantidade de livros que está sendo vendida, caso verdadeiro, através do literal `stockBook(...)`, precedido do sinal de "-" retira-se da base de crenças do agente a quantidade atual de livros e posteriormente adiciona-se a crença da nova quantidade, através do literal precedido do sinal de "+".

O agente Gerente de vendas (*Delivery manager*) é responsável por apresentar e gerenciar todos as informações inerentes às vendas dos livros. O quadro 14 demonstra os planos que devem ser executados quando o cliente opta por comprar um livro.

```
+received(InterfaceAg, askOne, informPurchase(CodeBook, Quantity,
                ZipCode), M)
     :    true
     <-   .send(stockManager, askOne,
               book(CodeBook,Title,_,_,_,Price,_,Weight),
               book(CodeBook,Title,_,_,_,Price,_,Weight));
          !calculateRate(ZipCode, Weight, CodeBook,FreightPrice);
          .send(InterfaceAg, tell,
               [Title,Price,Quantity*Price,FreightPrice,
               (Quantity*Price)+FreightPrice], M).


+!calculateRate(ZipCode, Weight, CodeBook, FreightPrice)
     :    zipCode(ZipCode, City, State) &
          zipCodeRate(ZipCode, Rate, Distance, Days) &
          Weight > 0
     <-   FreightPrice = (Distance*Rate*Weight).
```

Quadro 14 – Apresentação das informações para compra

Quando o agente recebe a mensagem `informPurchase(...)`, primeiramente ele envia uma mensagem ao agente Gerente de estoque questionando o título, o preço e o peso o do livro que está sendo comprado. Possuindo as devidas informações, executa-se então a meta `+!calculateRate(...)`, que calcula o preço total do frete para entrega do livro. Por fim envia-se uma mensagem ao agente responsável pela interface com uma lista de todas as informações que devem ser apresentadas ao usuário antes que ele confirme a compra do livro.

A ação `.send` utilizada na execução do plano do quadro 14, é utilizada para a comunicação entre agentes. A ação é formada por três parâmetros: o primeiro é simplesmente

o nome do agente receptor, o segundo parâmetro corresponde a uma *illocutionary force*, que em síntese consiste de um ato de fala feito em uma expressão e finalmente o terceiro parâmetro é um literal, que representa o conteúdo da mensagem.

4.3 IMPLEMENTAÇÃO DO AGENTE WEB (NETBEANS / KQML)

O principal componente que viabiliza a comunicação entre o agente responsável pela aplicação web e os agentes da aplicação Jason é o MBox. Um componente MBox serve como uma interface entre o agente e a sociedade. Sua principal função é entregar uma mensagem ao receptor e gerar mecanismos de transporte remoto numa rede de maneira transparente para o desenvolvedor. O quadro 15 apresenta o código fonte para criação do MBox, implementado no ambiente NetBeans.

```
saci.MBoxSAg mbox;

public void jspInit() {

    System.out.println("Iniciando página");
    Config c = new Config();
    c.set("society.name", "eletronicBookstore");
    try {
        mbox = new MBoxSAg("eletronicBookstore ",c);
        mbox.init();
        System.out.println("MBox criado!");
        getServletContext().setAttribute("mbox", mbox);
    } catch (Exception e) {
        System.err.println("Erro iniciando jsp "+e);
    }
}
```

Quadro 15 – Rotina para criação do MBox

No quadro 17 é demonstrada a utilização de um MBox no desenvolvimento das páginas. Primeiramente é enviada uma mensagem `getCodeBookSW(...)` para o agente Gerente de estoque solicitando os códigos dos livros que estão na vitrine. Após receber a resposta com os respectivos códigos de livro, o agente interface executa um laço de repetição,

obtendo as informações de cada livro (`showShopWindow(...)`) e apresentando as informações na tela.

Entretanto, para que seja possível a criação do Mbox é fundamental importar as bibliotecas do Saci e do Jason, conforme apresentado no quadro 16.

```
<%@page import="saci.*"%>
<%@page import="jason.asSyntax.*"%>
```

Quadro 16 – Importação as bibliotecas do Jason e Saci

Para a apresentação gráfica propriamente dita é utilizado o HMTL (Hiper Text Markup Language) convencional.

```
<% Message     msg, msgCB;
Literal     cont, contCB;
Message     answer, answerCB;
String      content, contentCB, codeBookLis;
ListTerm    listBooksCB;
Iterator    iterCB;
Term        terms;

msgCB       = new Message("(askOne :receiver stockManager)");
contCB      = Literal.parseLiteral(msgGetBook);

msgCB.put("content", contCB);
answerCB    =   mbox.ask(msgCB);
contentCB   =   answerCB.get("content").toString();
listBooksCB =   ListTermImpl.parseList(contentCB);
iterCB      =   listBooksCB.iterator();

while (iterCB.hasNext()) {
    codeBookLis =   iterCB.next().toString();
    msg         =   new Message("(askOne :receiver stockManager)");
    cont        =   Literal.parseLiteral("showShopWindow("+codeBookLis+")");
    msg.put("content", cont);
    answer      =   mbox.ask(msg);
    content     =   answer.get("content").toString();

    terms       = Term.parse(content);

    String  codeBook    =   terms.getTerm(0).toString().replaceAll("\"","");
    String  image       =   "image/" + terms.getTerm(1).toString().replaceAll("\"","");
    String  title       =   terms.getTerm(2).toString().replaceAll("\"","");
    String  author      =   terms.getTerm(3).toString().replaceAll("\"","");
    String  publisher   =   terms.getTerm(4).toString().replaceAll("\"","");
    String  price       =   terms.getTerm(5).toString().replaceAll("\"",""); %>

    <img src=       <%= image %>>       <p>
    <b> Title:      <%= title %>        </b><br>
    <b> Author:     <%= author %>       </b><br>
    <b> Publisher:  <%= publisher %>    </b><br>
    <b> Price: $    <%= price %>        </b><br></p>

    <% if (!codeClient.equals("")) { %>
        <p><button value="Buy" type="button" onclick="JavaScript: window.location.href =
'buyBook.jsp?codeBook=<%= codeBook %>&codeClient=<%= codeClient %>';">Buy</button></p><%
    }
}
%>
```

Quadro 17 – Utilização do MBox no ambiente de desenvolvimento NetBeans

4.4 OPERACIONALIDADE DA IMPLEMENTAÇÃO

A organização gráfica e textual do sistema web segue os padrões já conhecidos, de modo que é possível navegar de maneira intuitiva. Apesar de não ter havido grande preocupação em desenvolver um sistema com uma produção gráfica mais arrojada, procurou-se dar particular atenção à forma como a informação é apresentada (*layout* da página), garantindo a legibilidade e a navegabilidade de maneira organizada.

Conforme apresenta a figura 16, no momento que o usuário acessa a página web são listados todos os livros de vitrine, que são apresentados a todos os usuários que acessam a página. A figura 17 demonstra como os livros são apresentados.

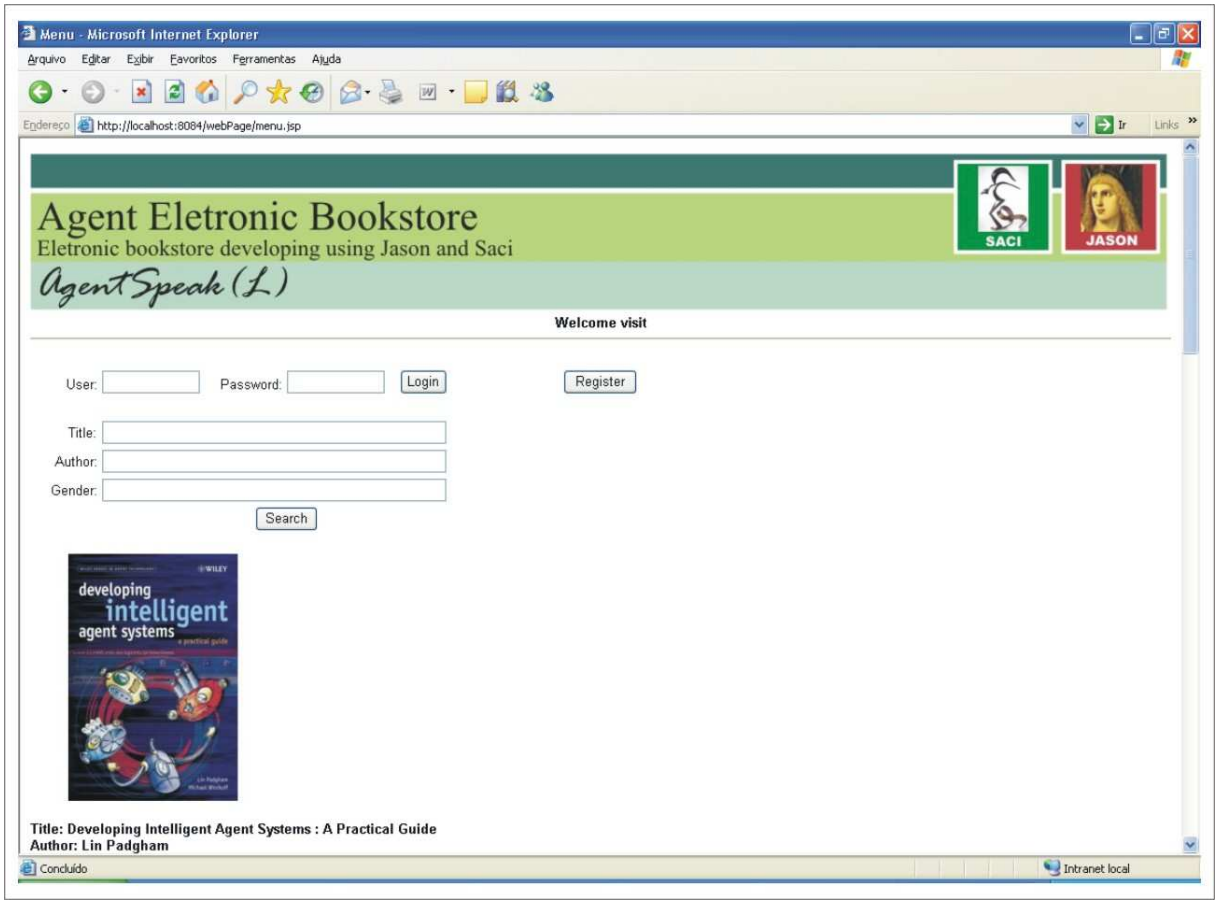


Figura 16 – Tela inicial

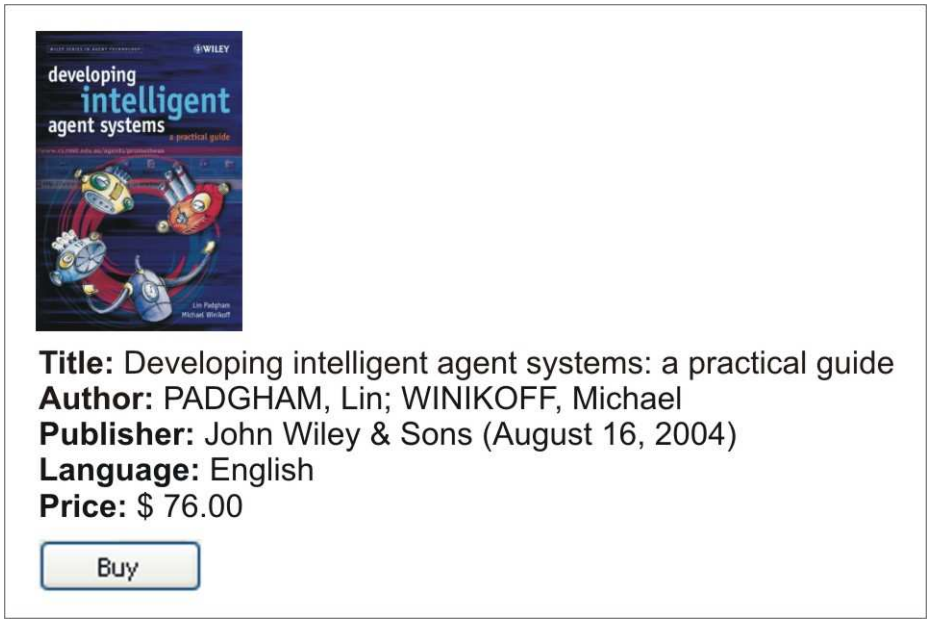


Figura 17 – Maneira como os livros são apresentados na página web

Se o usuário for um cliente cadastrado da livraria virtual, após o momento que o mesmo efetua *login* (figura 17) no website, os livros de vitrine são substituídos por livros do gênero preferido do usuário, que é obtido através das compras anteriores do cliente. Por exemplo: se um cliente comprou um livro de Sistema Multiagentes, na próxima vez que o mesmo efetuar *login*, serão apresentados livros deste tema.

Figura 18 – *Login* no website

Caso o usuário não for cadastrado, existe a possibilidade de se cadastrar, conforme o formulário demonstrado na figura 19, onde o usuário informa seus dados pessoais, cria um nome de usuário e senha. Após o processo de castrado o usuário pode efetuar *login* no website, sendo que a partir disso está apto a adquirir livros.

Figura 19 – Formulário para registro do usuário

Através de um sistema de busca, podem-se localizar outros livros, informando parâmetros de busca como título, autor ou gênero, de acordo com a figura 18.

Figura 20 – Opções de busca

Para efetuar a busca o usuário pode informar um ou mais parâmetros. No caso de ser informado mais de um item, o sistema lista os livros que obedeçam a todos os parâmetros solicitados, constituindo um “e” lógico.

Se o usuário optar por adquirir um livro clicando sobre o botão *Buy* demonstrado na figura 17, é apresentada a tela de compra (figura 21).

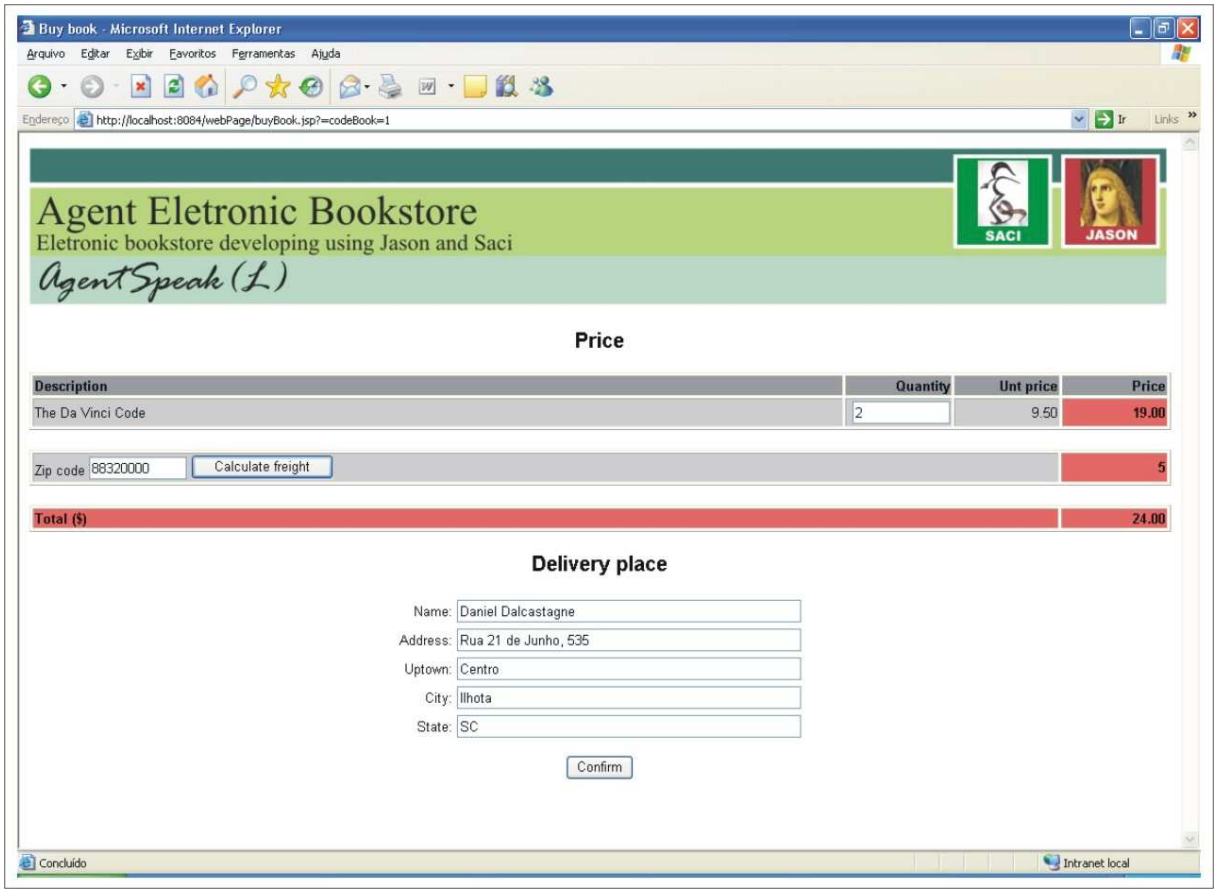


Figura 21 – Adquirindo um livro

Na tela de compra é apresentado o valor do livro que está sendo adquirido. Se o cliente desejar comprar mais que um volume do mesmo livro, basta informar a quantidade no campo Quantity, e em seguida é calculado o novo preço baseado na quantidade informada.

Em seguida é apresentado o preço do frete para a entrega do livro, sendo que para isso o usuário deve informar o CEP (Zip code). Por fim é apresentado o valor do livro mais o valor do frete que representa o valor total que deve ser pago pelo cliente.

Os campos listados abaixo dos cálculos indicam onde a mercadoria deve ser entregue, por *default* o sistema apresenta o endereço informado no cadastro do cliente.

A figura 22 corresponde a tela do menu principal das função pertinentes a gerência da livraria virtual. Neste módulo podem-se extrair relatórios dos clientes cadastrados, livros e vendas, podendo ainda cadastrar novos livros e informar se o preço de algum livro deve ser alterado.

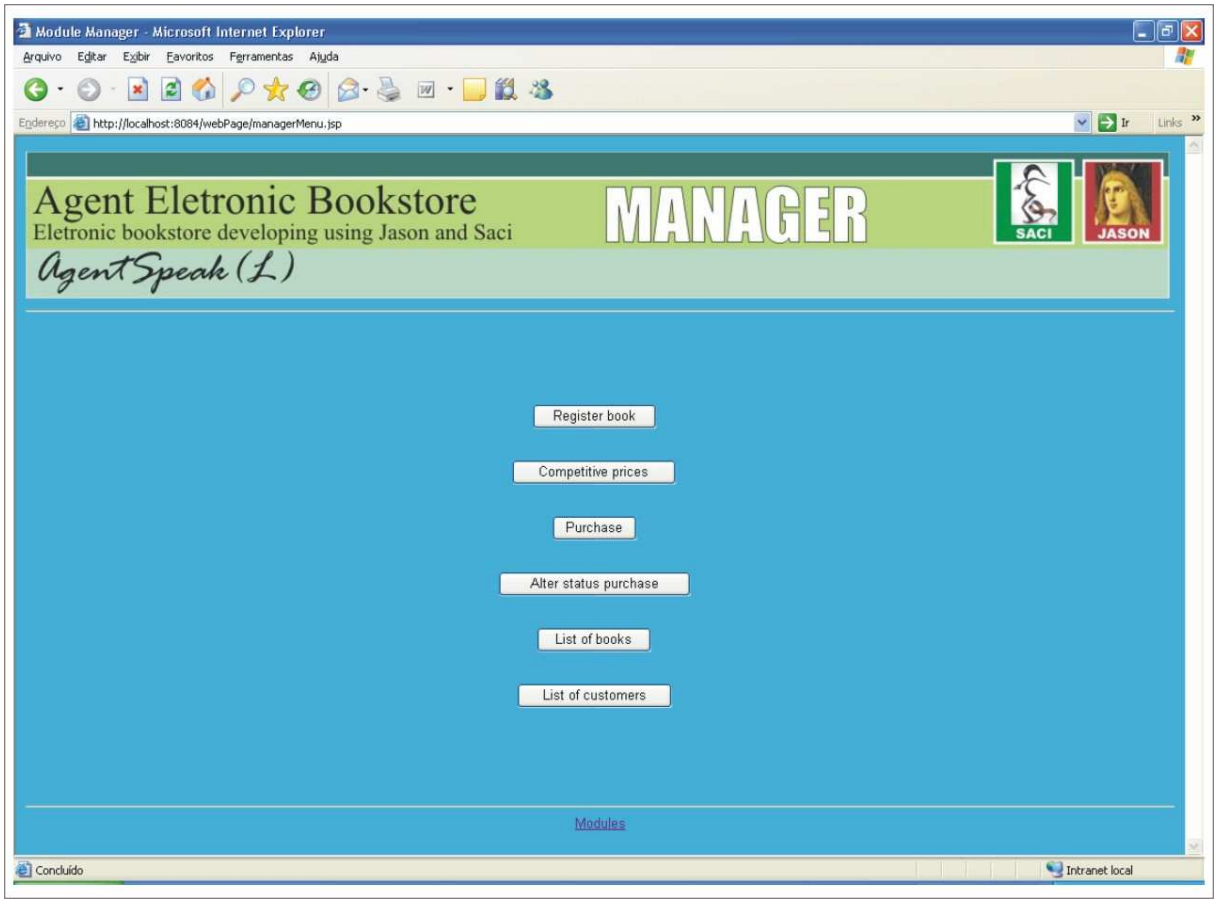


Figura 22 – Módulo de gerência

4.5 RESULTADOS E DISCUSSÃO

Como inicialmente proposto, algumas constatações foram feitas no desenvolvimento deste trabalho, no que diz respeito à construção de softwares de cunho comercial utilizando a linguagem AgentSpeak(L) e o interpretador Jason.

Em relação ao desenvolvimento de código AgentSpeak(L), o mesmo proporciona uma excelente abstração na maneira de visualizar os componentes de software como sendo agentes, em especial no caso da livraria, onde se possui de maneira bem clara o papel de cada um deles.

O SACI através do padrão *Knowledge Query and Manipulation Language* (KQML), possibilitou a comunicação entre o ambiente desenvolvido em NetBeans 4.1 e os agentes de

maneira transparente e de alto nível para o programador.

O desenvolvimento da parte lógica da aplicação em AgentSpeak(L) também ocorreu de maneira muita interressante, através de pouco código foi possível organizar de maneira adequada as ações necessárias para que os objetivos fossem alcançados. Por exemplo no momento que usuário efetua *login* na página, de maneira muito simples verifica-se se o usuário é cadastrado, consultando a base de crenças, se for cadastrado, executa-se a ação básica de apresentar boas vindas, caso contrário mostra-se uma mensagem informando que usuário e senha não conferem.

A maior deficiência encontrada no desenvolvimento da aplicação foi na questão de persistência e manipulação das crenças (ou dados no caso de um banco de dados relacional). Atualmente o Jason não possibilita a persistência das crenças dos agentes, de modo que elas são mantidas quando declaradas explicitamente no código ou armazenadas de maneira volátil na memória principal do computador, impossibilitando dessa maneira o desenvolvimento de uma aplicação profissional.

Ainda sobre a questão de manipulação das crenças, verificou-se uma dificuldade quando se trabalha com literais de crenças com muitos termos, como por exemplo a crença `clientBookstore`, que é composta por um número considerável de atributos (comparando a um banco de dados convencional), pois sempre que se faz uma adição, atualização ou consulta a essas crenças, é necessário passar todos os termos como parâmetro, dificultando o desenvolvimento de aplicações que possuam muitas crenças com essa característica.

Por fim a especificação através da metodologia Prometheus possibilitou uma visualização clara de como deveriam funcionar todos os aspectos relacionados aos agentes, de modo que a especificação através desta metodologia foi fundamental para o sucesso deste trabalho.

A especificação dos agentes detalhada nas três fases da modelagem possibilitou que após a compreensão da metodologia, do paradigma orientado a agentes, e da linguagem AgentSpeak(L), fosse possível que de maneira clara e objetiva fossem desenvolvidos os agentes propostos, tornando-se simples a visualização das percepções, planos e ações que cada agente deveria executar, de maneira que provavelmente a UML não conseguiria abstrair.

5 CONCLUSÕES

Este trabalho apresentou um estudo sobre o desenvolvimento de uma aplicação caracterizada por aspectos comerciais (livraria virtual), implementada utilizando um novo paradigma de programação denominado de programação orientada a agentes, que até então era utilizada apenas para implementação de aplicações de cunho científico.

A especificação da livraria virtual ocorreu através de uma nova metodologia para especificação de SMA denominada Prometheus. Para implementação utilizou-se a linguagem de programação de agentes chamada AgentSpeak(L), que por sua vez é interpretada pela ferramenta Jason. Para viabilizar a comunicação entre os agentes principais e o agente para apresentação web, desenvolvida em NetBeans, se vez uso da ferramenta SACI.

Constatou-se com este estudo que a linguagem AgentSpeak(L), através do interpretador Jason, pode se tornar uma excelente opção para desenvolvimento de aplicações comerciais, pela capacidade de abstração, em especial quando se possui bem definido o papel dos agentes envolvidos na aplicação. Bastando para isso fornecer mecanismos para persistir e manipular as crenças do agente com maior facilidade.

Do ponto de vista de especificação a metodologia Prometheus se apresentou bastante adequada, possibilitando uma fácil implementação a partir de uma boa especificação.

De maneira geral conclui-se que o trabalho possibilitou que um primeiro passo fosse dado para que a linguagem AgentSpeak(L) através do interpretador Jason e o SACI se tornem uma ferramenta de uso mais amplo e acessível.

5.1 EXTENSÕES

Levando-se em consideração que poucas aplicações foram desenvolvidas até o

momento utilizando a linguagem AgentSpeak(L) e o interpretador Jason, a extensão deste trabalho pode ocorrer de diversas maneiras.

Considerando o conceito de livraria virtual, pode-se dividir uma possível extensão em dois segmentos: o aperfeiçoamento do módulo de apresentação (a livraria propriamente dita), e o melhoramento do módulo de gerenciamento.

O aperfeiçoamento do módulo de apresentação pode ocorrer através da melhoria da qualidade gráfica do website, da implementação do conceito de carrinho de compras e o desenvolvimento de meios efetivos de comunicação com os clientes, através de envio automático de e-mail, por exemplo.

O melhoramento do módulo de gerenciamento tem como objetivo prover uma gestão efetiva sobre as informações envolvidas no negócio, como controle de clientes, gerenciamento das vendas e estoque, faturamento entre outros fatores envolvidos diretamente em qualquer ferramenta gestão.

Um terceiro segmento para extensão deste trabalho seria através da implementação de um comportamento mais sofisticado para o agente Assistente de vendas, de modo que possa ser simulado o comportamento humano, com o objetivo de tornar a navegação no website mais atrativa para o usuário, para isso, poderiam ser utilizadas técnicas de inteligência artificial.

REFERÊNCIAS BIBLIOGRÁFICAS

APPIO, Alisson R. **Sistema multiagentes utilizando a linguagem AgentSpeak(L) para criar estratégias de armadilha e cooperação em um jogo tipo pacman**. 2004. 49 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

BORDINI, Rafael H.;VIEIRA, Renata. Linguagens de programação orientadas a agentes: uma introdução baseada em AgentSpeak(L). **Informática Teórica e Aplicada**, Porto Alegre, n. 1, ago. 2003. Disponível em <http://www.inf.ufrgs.br/~revista/Numpub.html>. Acesso em: 10 set. 2005.

BORDINI, Rafael H.;VIEIRA, Renata; MOREIRA, Álvaro F. Fundamentos de sistemas multiagentes. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 21., 2001, Fortaleza. **Anais...** Fortaleza: [s.n.], 2001. p. 3-45. Disponível em: <http://www.inf.unioeste.br/~cbrizzi/FSMA-Bordini.pdf>. Acesso em: 23 ago. 2005.

CALCÍN, Oscar P.; OKUYAMA, Fabio Y.; DIAS, Aurélio M. Simulación del proceso de compra de artículos en un mercado virtual con agentes BDI. In: CONFERENCIA LATINOAMERICANA EN INFORMÁTICA, 30., 2004, Perú. **Anais...** Perú: [s.n.], 2004. p. 214-223.

CARVALHO, Felipe G. de. **Comportamento em grupo de personagens do tipo black&white**. 2004. 102 f. Dissertação (Mestrado), Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

HÜBNER, Jomi F. **Um modelo de reorganização de sistemas multiagentes**. 2003. 246 f. Tese (Doutorado em Engenharia Elétrica), Escola Politécnica de Universidade de São Paulo, São Paulo.

HÜBNER, Jomi F.; BORDINI, Rafael H.; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. In: ESCOLA DE INFORMÁTICA DA SBC, 12., 2004, Guarapuava. **Anais...** Guarapuava: Unicentro, 2004. p. 51-89. Disponível em: <http://www.inf.furb.br/~jomi/pubs/2004/Hübner-eriPR2004.pdf >. Acesso em: 01 set. 2005.

PADGHAM, Lin; WINIKOFF, Michael. **Developing intelligent agent systems:** a practical guide. Melbourne: John Wiley & Sons, 2004.

WOOLDRIDGE, Michael. **An introduction to multiagent systems**. New York: John Wiley & Sons, 2002.

WOOLDRIDGE, Michael. Intelligent agents. In: WEISS, Gerhard (Ed.). **Multiagent Systems–A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA: MIT Press, 1999. p. 27–77.

RAO, Anand S. AgentSpeak(L): BDI agents speak out in a logical computable language. In: WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD (MAAMAW'96), 7., 1996, Eindhoven, The Netherlands. **Anais...** London: Springer-Verlag, 1996. p. 42–55.

REIS, Dalton S. dos et al. Uma arquitetura para simulação de agentes autônomos com comportamento social. In: SBC SYMPOSIUM ON VIRTUAL REALITY, 1., São Paulo. **Anais...** São Paulo: [s.n.], 2004, p. 39-50.

SHOHAM, Yoav. **Agent-oriented programming**. Stanford, 2003. Disponível em: <www.ncat.edu/~esterlin/c7902s02/Notes/Shoham.pdf>. Acesso em: 29 abr. 2006.