UNIVERSIDADE CATÓLICA DOM BOSCO
Centro de Ciências Exatas e da Terra
Curso de Engenharia de Computação

Sistemas Multiagentes Cognitivos Modelando um Protótipo de Comércio Eletrônico B2C

Alexandre Rosa Camy

Wagner Tanaka Botelho

Profª Drª Maria das Graças Bruno Marietto
Orientadora

Relatório Semestral submetido como requisito parcial para a obtenção do grau de Engenheiro de Computação.

Campo Grande, MS
Junho - 2002

Índice

Lista de Figuras

Capítulo 1

Introdução

A área de Inteligência Artificial Distribuída (IAD) vem se destacando como geradora de teorias, metodologias e técnicas para o desenvolvimento de sistemas dinâmicos, com vários participantes e perspectivas, de grande escala e com interdependências mútuas. Como exemplo de sistemas com estas características citam-se Comércio Eletrônico (CE), Educação a Distância, sistemas de processamento de informação, etc.

Cada vez mais a Internet está sendo inserida em atividades comerciais, pessoais, acadêmicas, etc. Visando modelar sistemas de Inteligência Artificial Distribuída para aplicações na Web, uma tecnologia aplicada é a linguagem de programação Java. Esta linguagem permite o uso de componentes tais como servlets, applets, acesso remoto a Banco de Dados, criação de páginas Web com conteúdo interativo e dinâmico, aplicativos corporativos de larga escala, etc.

Este projeto propõe um estudo da aplicação da IAD na estruturação de um protótipo de comércio eletrônico, mais especificamente o desenvolvimento de um Sistema Multiagentes Cognitivos para a modelagem de um CE B2C.

No Capítulo 2 a linguagem de programação Java é apresentada mostrando as suas principais características e funcionalidades. No Capítulo 3 será apresentado os conceitos e principais características de Programação Orientada a Objeto. A tecnologia Java Servlets está descrita no Capítulo 4, mostrando-se características tais como eficiência, persistência, portabilidade, robustez, intensibilidade e segurança. No Capítulo 5 a área de Inteligência Artificial Distribuída é apresentada, destacando conceitos de agentes, Sistemas Multiagentes, mecanismos e protocolos de comunicação. No Capítulo 6 conceitos de CE são abordados já interligando-os com a tecnologia de IAD. No Capítulo 7

será apresentado uma Análise de Requisistos para sistemas de CE, apresentando nesta etapa requisitos funcionais que modelam o comportamento de tais sistemas. No Capítulo 8 tem-se as considerações finais desse trabalho.

Capítulo 2

Linguagem de Programação Java

Neste capítulo tem-se uma introdução à linguagem de programação Java, apresentando seu histórico e principais características. Na Seção 2.1 apresenta-se um breve histórico da linguagem Java. A Seção 2.2 faz referência a suas principais características. Na Seção 2.3 são apresentados alguns tipos de programas que podem ser implementados em Java.

2.1 Breve Histórico

Em 1990 a empresa Sun Microsystems iniciou o desenvolvimento de um projeto que resultou na construção da linguagem Java. A equipe inicial deste projeto, denominada *Green Team,* era formada por Patrick Naughtom, Jim Gosling, Bill Joy e Mike Sheridam [24]. Uma tendência prevista pela equipe em 1990, na área de computação, correspondia à eletrônica de produtos destinados aos consumidores finais, onde pequenos aparelhos eletrônicos teriam *software* embarcados para controlar e executar uma gama de ações.

O primeiro projeto da equipe estava relacionado à criação de um dispositivo que controlasse vários produtos eletrônicos. Para tanto, o *Green Team* decidiu construir uma linguagem de programação para o desenvolvimento de *software* embarcados em dispositivos eletrônicos. O mentor desta idéia foi Gosling. A linguagem C++ era a mais próxima das necessidades da equipe, mas tinha o problema de se preocupar principalmente com o desempenho, e não com a confiabilidade. Os programas em C++ tendiam a ter proble-

mas de execução em momentos imprevisíveis. Já na eletrônica de *software* embarcados, a confiabilidade é mais importante do que a velocidade. Com todas estas evidências, Gosling concluiu sobre a necessidade de desenvolver uma nova linguagem de programação [24].

Enquanto Gosling desenvolvia uma nova linguagem, Naughtom desenvolvia a interface do dispositivo, contendo uma série de animações gráficas. Em 1991 Gosling concluiu a estruturação da nova linguagem, denominada Oak, em homenagem ao carvalho que podia ser visto da janela de sua casa [24].

Em 1992 o *Green Team* apresentou ao presidente da Sun Microsystem, Scott McNealy, o protótipo do primeiro produto desenvolvido pela equipe. Tratava-se de um dispositivo sem teclado, *mouse* e nem botões. Não parecia ser um computador, possuindo apenas uma pequena tela. O *software* desenvolvido para aquele aplicativo era o guia de uma casa, onde era possível andar por toda sua extensão e até programar os filmes que se quisesse assistir num videocassete virtual. McNealy considerou que o produto poderia ser um sucesso comercial e, em setembro de 1992, o *Green Team* se transformou em uma empresa subsidiária à Sun, denominada *First Person*. Entretanto, devido ao alto custo de produção dos *chips* utilizados, a *First Person* se dissolveu em 1994 [24].

Com o surgimento da *World Wide Web*, Bill Joy viu uma grande oportunidade para o ressurgimento da Oak. A proposta era lançar a linguagem Oak gratuitamente no mercado, para que se transformasse em uma linguagem padrão na Web. Joy sugeriu que o lucro viesse da venda de licenças comerciais sobre o produto.

Em janeiro de 1995 a linguagem Oak foi renomeada para Java, pois já havia no mercado uma linguagem com o nome Oak. Em 23 de maio de 1995 a Sun lança formalmente a linguagem Java, em conjunto com o *browser* HotJava no evento SunWorld 95 [30].

Atualmente, a linguagem Java é utilizada na criação de páginas Web com conteúdo interativo e dinâmico, no desenvolvimento de aplicativos corporativos de larga escala, no aprimoramento da funcionalidade de servidores da *World Wide Web*, em aplicativos para dispositivos destinados ao consumidor final tais como telefones celulares, *pagers*, *Personal Digital Assistants* (PDA), etc [6].

2.2 Características da Linguagem Java

A empresa Sun define Java como uma linguagem simples, orientada a objetos, distribuída, interpretada, potente, segura, de arquitetura neutra, portável, de alto desempenho, multiencadeada e dinâmica [24].

Nas próximas subseções estão descritas algumas características da linguagem Java.

2.2.1 Distribuição

Java oferece a funcionalidade de compartihar informações e processamento de dados. A ampla biblioteca embutida na linguagem permite que os aplicativos trabalhem com protocolos tais como TCP/IP, HTTP e FTP [24].

2.2.2 Compilação e Interpretação

No processo de criação de um programa Java, em primeiro lugar o programador deve compilar o código com extensão .java, utilizando um compilador Java. Após a compilação, este código é transformado em código de *bytes* (*bytecode*) com extensão .class, código este independente de plataforma. A interpretação do *bytecode* é feita por um processador virtual Java denominado *Java Virtual Machine* (JVM). A Figura 2.1 ilustra o processo de compilação e interpretação do ambiente Java.

2.2.3 Independência de Arquitetura

A incompatibilidade de versões em diferentes arquiteturas é resolvida em Java através do conceito de *bytecode*. Esta característica resolve o problema dos programadores que tinham de desenvolver versões diferentes de um mesmo aplicativo para as várias arquiteturas existentes no mercado.

2.2.4 Consistência

A consistência de uma linguagem de programação é a baixa taxa de problemas de execução dos programas desenvolvidos na mesma. Java possui recursos que diminuem as possibilidades de erro, tais como [24]:

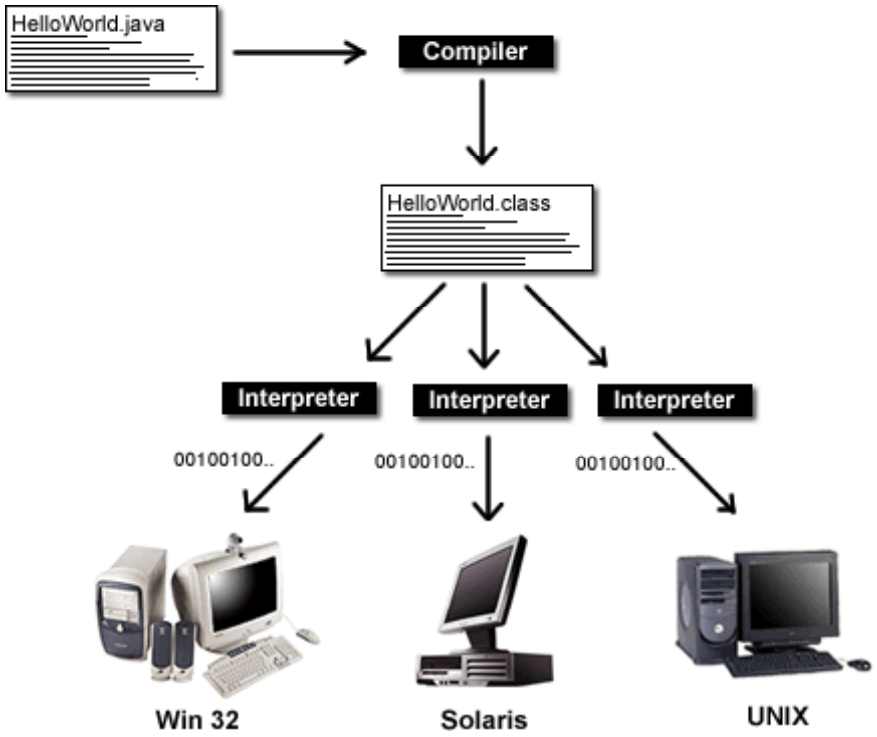

Figura 2.1: Arquitetura de Compilação, Interpretação (Execução) de Programas Java

- Java é uma linguagem fortemente tipada (do Inglês *strongly typed)*, característica que permite uma ampla verificação em tempo de compilação, facilitando o encontro de *bugs;*
- Java não permite o acesso à memória do computador.

2.2.5 Diferenças entre Java e C++

Apesar da linguagem C++ não ter servido para os propósitos dos programadores da Sun, a linguagem Java inicialmente baseou-se nela. Isto porque a intenção era tornar a linguagem mais familiar, mais compreensível e também encurtar o tempo necessário de aprendizado. Na época cerca de 90% dos programadores utilizam C ou C++. A equipe de desenvolvimento da Sun manteve os recursos que facilitariam o desenvolvimento, implementação e manutenção do *software*, e eliminou os itens que atrapalhariam [24].

Por exemplo, a equipe *Green Team* eliminou a característica de herança múltipla da linguagem C++, restringindo o número de heranças em apenas uma classe. Tal decisão evitou que uma classe herde outras com comportamentos contraditórios.

Outro problema da linguagem C ou C++ é o gerenciamento de memória, onde o programador precisa estar atento à alocação, liberação e quantidade de memória que o programa está utilizando. Quem programa em Java não precisa se preocupar com estas questões, pois Java possui um coletor de lixo automático (*garbage collection*) que simplifica a programação e diminui o número de *bugs.*

2.3 Programas Java

Dentre os tipos de programas que a linguagem Java permite criar tem-se os applets e aplicações. Um applet Java é um programa que roda a partir de um *browser Web.* Uma aplicação Java, por sua vez, é um programa independente do *browser* e pode ser executado como um programa isolado [4].

2.3.1 Aplicações

Para escrever uma aplicação Java deve-se primeiro especificar um método main(). Este método é executado quando o aplicativo é inicializado. Dentro do método main() especifica-se a funcionalidade executada pelo aplicativo.

Esta subseção será explanada com um exemplo simples de uma aplicação Java, ilustrada na Figura 2.2.

```
1 public class BemVindo
2 {
3     public static void main (String args[])
4     {
5         System.out.println ("Bem Vindo à Programação Java");
6     }
7 }
```

Figura 2.2: Exemplo de uma Aplicação em Java

A seguir tem-se uma explicação do código desta aplicação:

- A linha 1 inicia a definição da classe BemVindo. Cada programa consiste de pelo menos uma definição de classe, definida pelo programador. A palavra reservada class introduz a definição de uma classe em Java, sendo imediatamente seguida pelo nome da classe;
- A linha 2 possui uma chave, que inicia o corpo da definição da classe;
- A linha 3 apresenta o método main(), que é a característica de um programa Java como um aplicativo. Em outras palavras, todo aplicativo Java deve ter um método main(). Quando da inicialização de um aplicativo, este é o primeiro método a ser executado. A palavra void indica que este método realizará uma tarefa, mas não retornará um valor quando completá-la;
- A linha 4 possui uma chave esquerda, que inicia o corpo do método;
- A linha 5 imprime uma string de caracteres contidos entre as aspas duplas. O objeto System.out imprime o parâmetro no dispositivo padrão de saída;
- As linhas 6 e 7 encerram os blocos do método e da classe, respectivamente.

Compilação

Após a digitação do código da Figura 2.2, o próximo passo é compilar o programa, criando desta forma um arquivo em *bytecodes.* Se o código foi digitado em um editor de texto normal, então o usuário deve digitar no *prompt* o seguinte comando:

javac BemVindo.java

Se o código foi digitado em qualquer ambiente de desenvolvimento, o usuário deve utilizar o botão, item de menu ou tecla de atalho que dispara a execução do compilador Java. Caso não haja problemas de sintaxe, o arquivo BemVindo.class é gerado.

Execução

O último passo refere-se à interpretação da aplicação. Se o arquivo foi compilado no prompt, agora é necessário utilizar o interpretador Java, pois o *bytecode* é executado na JVM. Para a interpretação, digite o seguinte comando no *prompt*:

java BemVindo

Se o código foi compilado em um ambiente de desenvolvimento, deve ser selecionado o botão, item de menu ou tecla de atalho que dispara a execução do interpretador Java. A mensagem "Bem vindo à Programação Java" aparecerá no dispositivo padrão de saída.

2.3.2 Applets

Um applet é um programa Java executado a partir de um *browser* Web. O ciclo de vida de um applet é composto dos seguintes métodos [4]:

- **init():** Método opcional, sendo o primeiro a ser chamado pela JVM ao abrir pela primeira vez uma página Web que contém um applet. Ele executará suas operações de inicialização necessárias uma só vez, tais como: criar e inicializar objetos, obter valores de parâmetros de um arquivo HTML, etc;

- **start():** Método chamado cada vez que o usuário voltar à página Web do applet, sem que tenha fechado o *browser*. A diferença entre o método start() e init() é que Java chama uma só vez o método init() durante a vida do applet. No entanto, o método start() poderá ser chamado

```
1  import java.applet.*;
2  import java.awt.*;
3
4  public class Hello_World extends Applet
5  {
6      public void paint (Graphics g)
7      {
8          g.drwString("Hello World", 5, 10);
9      }
10 }
```

Figura 2.3: Exemplo de Applet

várias vezes enquanto o usuário estiver navegando, saindo e entrando no applet [4];

- **stop():** Método chamado pela JVM cada vez que o usuário sair da página Web;
- **destroy():** Método chamado pela JVM quando um usuário decide sair da página que contém o applet. Auxilia em recursos específicos ou na limpeza de quaisquer itens que o applet tiver alocado durante sua vida;
- **paint():** Método chamado pela JVM toda vez que um applet tem de redesenhar o conteúdo de sua janela [24].

Para melhor entender a estrutura de applet, na Figura 2.3 tem-se o exemplo do applet Hello_World. A seguir o código de Hello_World é comentado:

- A linha 1 importa o pacote java.applet, necessário para criar um applet. Este pacote oferece a possibilidade de manipular diferentes eventos de interface com o usuário e opções de desenho na tela. Para importar pacotes em Java utiliza-se o comando import;
- A linha 2 importa o pacote java.awt, pois a classe Graphics deste pacote será utilizada;
- A linha 4 define a classe Hello_World, através da palavra chave class. A palavra chave extends, seguida por um nome de classe, indica a classe da

qual herda características. Neste exemplo a classe Hello_World herda todas as carcterísticas da classe Applet;

- A linha 5 possui uma chave, que inicia o corpo da definição de classe;
- A linha 6 inicia a definição do método paint(). A lista de parâmetros do método paint() indica que ele requer um objeto da classe Graphics (denominado g) para realizar a tarefa. O objeto de Graphics é usado por paint() para desenhar imagens gráficas no applet;
- A linha 7 possui uma chave, que inicia o corpo do método paint();
- A linha 8 é uma chamada ao método drawString(), da classe Graphics, que mostrará o texto dentro da janela applet;
- As linhas 9 e 10 encerram os blocos do método e da classe, respectivamente.

Compilação

A compilação de um applet é similar à de uma aplicação, podendo ser utilizado o compilador Java (javac) no *Prompt* ou qualquer ambiente de desenvolvimento.

Execução

A execução de um applet pode ser realizada de duas formas:

- Executando o *appletviewer* com o seguinte comando:

  appletviewer Hello_World <ENTER>

- Carregando o arquivo Hello_World.class em um *browser*. Neste caso é necessário criar um arquivo com extensão html que irá carregar o applet *browser*. Como exemplo tem-se o arquivo Hello_World.html, apresentado na Figura 2.4. Após ter criado o arquivo HTML, basta abri-lo em um *browser*.

```
<HTML>
    <HEAD>
        <TITLE>Hello World Applet</TITLE>
    </HEAD>
    <BODY>
        A seguir tem-se a saída applet:
        <APPLET CODE="Hello_World.class" width=200 HEIGHT=40 </APPLET>
    </BODY>
</HTML>
```

Figura 2.4: Exemplo de programa em HTML para Carregar um Applet

Capítulo 3

Conceitos de Programação Orientada à Objeto

Neste capítulo tem-se uma introdução à Programação Orientada a Objetos(POO). Para tanto na Subseção 3.1 os conceitos principais da POO são apresentados.

3.1 Conceitos Principais da POO

A seguir são apresentadas algumas vantagens de ser utilizada POO:

- Redução no custo de manutenção do *software*. Na POO existem certas características, tais como herança e encapsulamento, que em caso de necessidade de alteração do código pode-se modificar apenas o componente que necessita desta alteração;
- Aumento da reutilização do código. A POO fornece a possibilidade de um objeto acessar e usar como se fossem seus os métodos e a estrutura de uma classe.

3.1.1 Classes

Segundo [24], as classes definem o estado e o comportamento de um objeto. Cada classe é formada por duas partes: as propriedades que a definem e a capacidade de gerenciar estas propriedades.

A seguir são descritas algumas vantagens de se utilizar classes na POO [24]:

- Através das classes é possível lidar com herança. Assim, é possível criar novas classes com base em classes anteriores;
- As classes são utilizadas para agrupar dados e métodos.

A forma geral de declaração de uma classe está colocada a seguir:

```
[modificadores] class [nome classe] extends [nome super]
```

Como exemplo tem-se o seguinte trecho de código:

```
public class extends SuperClasse
{
  //código da classe
}
```

A seguir cada um dos elementos definidores de uma classe são apresentados.

Modificadores

Os modificadores de uma classe determinam como ela será manipulada mais tarde no decorrer do desenvolvimento do programa. Embora, no geral, eles não sejam tão relevantes para o desenvolvimento da classe propriamente dita, sua importância aumenta quando se decide criar outras classes, interfaces e exceções que envolvam aquela classe. A seguir tem-se a especificação de cada um dos modificadores:

- **public:** Indica que uma classe pode ser acessada por todos os objetos, independentemente do pacote em que estejam;
- **friendly:** Se nenhum modificador de classe for especificado, então a classe será considerada friendly. Neste caso, apenas os objetos que estiverem no mesmo pacote poderão usar esta classe;
- **final:** Uma classe final pode ser instanciada, mas não pode ser derivada. Isto é, não pode ser superclasse de uma subclasse;

- **abstract:** É uma classe em que pelo menos um método não está concluído. Desse modo uma classe abstrata não pode ser instanciada, mas pode ser derivada. Nesses casos a subclasse deve prover o corpo do método para que possa ser instanciada. Isto é muito útil quando deseja-se definir em uma classe regras gerais de comportamento, para que mais tarde as regras específicas sejam introduzidas por subclasses.

Uma observação importante é que geralmente cada classe será armazenada em um arquivo. É possível ter várias classes em um mesmo arquivo, porém só a uma delas deverá ser atribuído o modificador public. O nome do arquivo deve ser o nome da classe public.

Superclasses

Ao se extender uma superclasse através da palavra-chave extends, a classe está se tornando uma nova cópia daquela classe, mas permitindo uma especialização.

Caso o resto da classe esteje em branco, a nova classe irá se comportar de forma idêntica à classe original. A nova classe terá todos os campos e métodos declarados ou herdados da classe original. Ao contrário de C++, Java não permite herança múltipla, fazendo com que suas classes possam ter apenas uma superclasse.

3.1.2 Objetos

Um objeto é uma variável do tipo classe, sendo chamado de instância de uma classe. De forma esquemática pode-se dizer que as classes são abstrações, enquanto os objetos são concretizações destas abstrações [4].

É possível criar uma instância declarando uma variável e atribuindo-a a um objeto criado com o operador new. Veja no exemplo a seguir:

```
public Fruta ()
{
    Fruta manga; //Declara uma variável
    manga = new Fruta(); //manga aponta para um objeto
    Fruta pera = new Fruta();
}
```

Segundo [33] um objeto é um ente independente composto por:

- **Estado Interno:** Uma memória interna em que valores podem ser armazenados e modificados ao longo da vida do objeto.
- **Estado Externo:** Um conjunto de ações pré-definidas (métodos), através das quais o objeto responderá a demanda de processamento por parte de outros objetos.

Cilclo de Vida de um Objeto

O ciclo de vida engloba o momento em que um objeto é declarado até sua eliminação. No instante em que o objeto é declarado, é alocado um espaço em memória para ele e automaticamente é executado seu construtor, estando então pronto para ser usado. Sua eliminação pode ser de 2 formas [33]:

- Elimina-se o objeto no final do programa se ele for global, no final de um módulo se ele for local e no final de um bloco se ele for declarado dentro deste. Esta forma de eliminação todas as linguagens utilizam;
- A segunda forma, denominada *garbage collection,* não é implementada em todas as linguagens e não é uma característica somente de Orientação a Objetos. *Garbage collection* elimina através do compilador o objeto/variável depois de sua última utilização. A partir do momento em que ele não é mais referenciado, passa a não existir mais na memória. *Garbage collection* está implementado em Lisp, SmallTalk e Java, enquanto em C++ e Pascal não.

3.1.3 Métodos

Os métodos são as funções que pertencem a uma classe, devendo sempre ser declarado dentro dela [4]. Eles são responsáveis pelo gerenciamento de todas as tarefas que serão realizadas pelas classes [24].

Um método entra em ação no momento em que é chamado. Isto pode ocorrer explicitamente ou implicitamente. A chamada explicita se dá por ordem do programador, através da execução de um comando contendo o nome do método. A chamada implícita ocorre quando o interpretador chama um método por sua própria deliberação. A chamada do método `main()` é um exemplo de chamada implícita.

Embora a implementação real do método esteja contida dentro do corpo do mesmo, como com as classes, um grande volume de informações importantes é definido na declaração do método [24]. A sintaxe completa para a declaração de um método é a seguinte:

[moderadores de acesso] [modificador] [tipo do valor de retorno] [nome] ([argumentos]) throws [lista de excessões]
{
    [corpo do método]
}

Moderadores de Acesso

A exemplo das classes, é importante limitar o acesso aos métodos de uma determinada classe. Os moderadores de acesso são empregados para restringir este acesso ao método. Independentemente do moderador usado, um método é sempre acessível a partir de qualquer método contido na mesma classe. Os moderadores de acesso existentes são os seguintes:

- **public:** É o moderador mais flexível que pode ser aplicado a um método, pois pode ser chamado a partir de métodos contidos em qualquer outra classe, independentemente do pacote a que pertence;
- **protected:** É idêntico ao public, exceto pelo fato de que não pode ser acessado por objetos fora do pacote atual;
- **private:** Corresponde ao nível máximo de proteção que pode ser aplicado a um método, pois é privativo da classe que o contém, seu uso é vedado a qualquer outra classe;
- **friendly:** O uso do método é permitido dentro da classe que tenha sido derivada desta classe, ainda que esteja fora do pacote. Caso o modificador não seja especificado, o método será considerado friendly;
- **private protected:** Este moderador dá acesso tanto à classe quanto a qualquer subclasse, mas não dá acesso ao resto do pacote nem por qualquer outra classe que esteje fora do pacote atual.

Modificadores de Métodos

Os modificadores de métodos permitem que se defina as propriedades do método, determinando como as classes derivadas podem ou não redefinir ou alterar métodos, e de que forma este método será visível. Os modificadores existentes são [4]:

- **static:** Permite que variáveis e métodos sejam acessados sem o uso de um objeto para referenciá-los;
- **abstract:** Possibilita um método de ser declarado mas sem a necessidade de ser implementado na classe atual. Funciona como uma espécie de lembrete para que alguma classe derivada complete a declaração. Portanto, é responsabilidade das subclasses de substituir e implementar estes métodos.
- **final:** Evita que qualquer subclasse substitua o método. Os métodos declarados como static ou private são implicitamente final. Um método final deve ser obrigatoriamente seguido de um corpo.
- **native:** Como no caso abstract, declara apenas a assinatura do método, designando um método implementado em outra linguagem, como C++ por exemplo.

Tipo de Valor de Retorno

O retorno de informações permite retornar qualquer tipo de dados, desde os mais simples até os objetos mais complexos [24]. É especificado por uma palavra-chave ou nome de classe na declaração do método, determinando o valor que deve retornar. Um método não declarado como void deve retornar um valor através do comando return.

Argumentos

Os argumentos, ou lista de parâmetros, são as informações que serão passadas para o método, podendo ser composta por quantos parâmetros forem necessários.

3.1.4 Construtores

Em geral, os métodos construtores são chamados no momento da instanciação da classe e usados para inicializar objetos, campos de classe e realizar tarefas como: conectar-se a um servidor ou fazer alguns cálculos iniciais. São identificados por terem o mesmo nome que a classe e não especificarem valor de retorno, pois não são chamados por outro método.

Os construtores não podem ser declarados como native abstract, static, syncronized nem final.

3.1.5 Variáveis

O modo como as variáveis são empregadas nos programas, os processos necessários para acessá-las e o grau de proteção que deseja fornecer a elas são muito importantes e devem ser levados em consideração [24]. A capacidade de acessar uma determinada variável depende de dois fatores:

- Dos modificadores de acesso usados durante a criação da variável;
- A localização da declaração da variável dentro da classe.

A exemplo dos modificadores de classes e de métodos, os modificadores de acesso definem o quanto certas variáveis poderão ser acessadas por outras classes. Os modificadores de acesso se aplicam apenas aos campos globais[1]. Os modificadores são:

- **friendly:** Permite o acesso dos campos por outras classes dentro do mesmo pacote e não permite o acesso para subclasses ou classes que estejam fora do pacote;
- **public:** Torna os campos visíveis para todas as classes ou subclasses, independentemente do pacote em que estão;
- **protected:** Permite o acesso aos campos por todas as subclasses da classe atual, mas não podem ser acessados por classes que estejam fora do pacote atual;
- **private:** É o grau máximo de proteção. Os campos podem ser acessados por todos os métodos dentro da classe atual. Mas não podem ser acessados por qualquer outra classe ou subclasse da classe atual;

[1] São variáveis declaradas antes de qualquer método da classe.

- **private protected:** Os campos podem ser acessados tanto dentro das subclasses quanto da classe atual;
- **static:** Torna o campo estático, cujo valor é o mesmo em todas as intâncias da classe. Os campos estáticos podem ser alterados em métodos estáticos e não-estáticos;
- **final:** Faz com que o valor do campo não possa ser alterado durante a execução do programa. É necessário definir o valor de todos os campos do tipo final quando eles forem declarados.

3.1.6 Encapsulamento

Encapsulamento é a capacidade de controlar o acesso a variáveis e métodos de classes e objetos. Este controle pode se dar através do uso de modificadores que definirão uma política de acesso a tais componentes.

3.1.7 Abstração

Abstração é a capacidade de analisar em um sistema real o que deve ser considerado quando de sua modelagem computacional.

3.1.8 Polimorfismo

O polimorfismo é utilizado para descrever a situação onde um nome pode referir-se a diferentes métodos [4]. Existem dois tipos de polimorfismo:

- Polimorfismo de Sobrecarga;
- Polimorfismo de Sobreposição.

Polimorfismo de Sobrecarga

Acontece quando se tem os mesmos nomes de métodos em uma classe. Isto se torna possível caso os métodos tenham diferentes números ou diferentes tipos de parâmetros. O uso de diferentes tipos de retorno não descaracteriza a ambigüidade entre os métodos, causando assim erro na compilação do programa [4].

A seguir é mostrada uma classe que contem métodos sobrecarregados, ilustrando o polimorfismo:

```
class Comprar
{
    int Escolhe(float Preco);
    int Escolhe(float Preco, String Vendedor);
    int Escolhe(float Preco, String Vendedor. String Marca);
}
```

Polimorfismo de Sobreposição

O polimorfismo por sobreposição acontece quando um método de classe tem o mesmo nome e assinatura (número, tipo e ordem de parâmetros) que um método em uma superclasse. Na classe em que os métodos têm os mesmos nome e assinatura, o método da classe derivada sempre sobreporá o método na sua classe mãe. Quando os métodos são sobrepostos, a linguagem Java determina os métodos adequados para chamar o programa durante a execução e não no momento da compilação.

3.1.9 Herança

A herança é um dos recursos mais importantes da POO. Trata-se de um processo de construir sobre uma classe já definida, estendendo de certo modo a classe já existente. Sua principal vantagem está no reaproveitamento de código que ela proporciona, melhorando a organização e facilitando a compreensão do mesmo [33]. A Figura 3.1 ilustra uma hierarquia de classes, onde as inferiores foram derivadas das superiores.

Uma subclasse herda todos os métodos e variáveis de suas classes mãe. Mas os modificadores de acesso como `private` podem impor restrições ao uso de itens herdados [4]. A Figura 3.2 ilustra as possibilidades de se fazer uso de herança de acordo com seus modificadores.

Diferente de outras linguagens orientadas a objeto como C++, Java permite que seja derivada apenas uma classe mãe, não podendo assim derivar uma nova subclasse a partir de duas ou mais classes mãe.

Uma subclasse tanto pode adicionar novos métodos e atributos não existentes na classe base, como também modificar a implementação de alguma operação por questões de eficiência.

Uma classe é derivada de outra utilizando a palavra reservada `extends`, conforme o trecho simplificado de código a seguir:

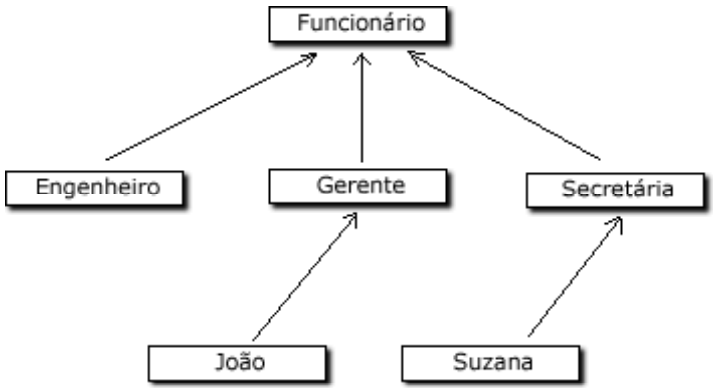

Figura 3.1: Uma Herança de Classes

Métodos e Atributos da Classe	**Instância da Classe**	**SubClasse da Classe**	**Instância da SubClasse**
privados (*private*)	não	não	não
protegidos (*protected*)	não	sim	não
pacote (*package*)	sim (no mesmo pacote)	sim (no mesmo pacote)	sim (no mesmo pacote)
públicos (*public*)	sim	sim	sim

Figura 3.2: Especificadores de Acesso do Java

```
public class SuperClass
{
    ...
}

public class SubClass extends SuperClass
{
    ...
}
```

Capítulo 4

Java Servlets

Quando a linguagem Java foi lançada pela Sun Microsytems, seu propósito era de fornecer interatividade através do uso de *applets* [15]. Isto porque o *bytecode* do applet é carregado na máquina do cliente e executado pelo *browser*.

Apesar de adicionarem interatividade, os applets estão crescendo cada vez mais, em complexidade e tamanho, tornando assim o tempo de *download* inviável. Há também problemas de compatibilidade, pois para rodar um applet é necessário ter um *browser* compatível, caso contrário o cliente não pode visualizá-lo. Estas questões, dentre outras, levaram a equipe da Sun a desenvolver uma tecnologia Java que rodasse os programas no lado do servidor ao invés do cliente.

Quando os programas são executados no lado do servidor, os problemas de compatibilidade e demora no carregamento são eliminados. As aplicações Java no lado do servidor apenas enviam ao cliente , através do protocolo HTTP, pequenos pacotes de informação para que possam ser visualizados. Java Servlets é uma tecnologia Java que roda no lado do servidor [15].

Dentre as características do Java Servlets citam-se:

- **Eficiência:** o código de inicialização de um servlet é executado apenas na primeira vez em que o servidor *Web* o carrega. Esta técnica é mais eficiente do que o carregamento completo de um novo executável com todas as solicitações;

- **Persistência:** servlets podem manter uma condição de espera entre as chamadas, pois quando é carregado permanece na memória enquanto

atende a chegada de solicitações. Esta característica pode melhorar em muito a performance de suas aplicações;

- **Portabilidade:** o que caracteriza a portabilidade dos servlets é o fato de serem desenvolvidos em Java, podendo ser movidos para outros sistemas operacionais sem alterar o código;
- **Robustez:** servlets são robustos pelo fato de terem acesso total ao JDK, que se trata de uma biblioteca de classes que inclui suporte de rede, de arquivos, acesso a Banco de Dados, componentes de objetos distribuídos, segurança e muitas outras classes;
- **Extensibilidade:** servlets são desenvolvidos em uma linguagem orientada a objetos. A herança e o polimorfismo são algumas das características da POO que podem ajudar a gerar novos objetos que ajudem no desenvolvimento do sistema;
- **Segurança:** servlets rodam no lado do servidor, herdando assim a segurança fornecida pelo mesmo.

4.1 Arquitetura Web

A arquitetura da *Web* é do tipo cliente/servidor, onde o cliente é constituído de um *browser* que faz solicitações a um servidor *Web*. Esta comunicação é feita através de pares solicitação/resposta (do Inglês *request/response*), sendo iniciada sempre pelo cliente.

A melhor forma de se estruturar programas que armazenam e gerenciam dados na Web é utilizando a arquitetura de três camadas, conforme ilustrado na Figura 4.1. As camadas consideradas são:

- Interface com o usuário;
- Lógica Computacional;
- Armazenamento de Dados.

Figura 4.1: Modelo de Três Camadas

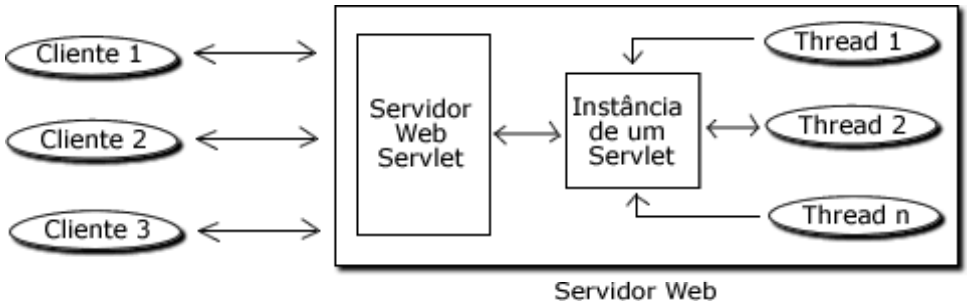

Figura 4.2: Execução de Servlets

4.2 Arquitetura e API Servlet

Servlets trabalham com o modelo de processamento solicitação/resposta, onde o cliente envia uma mensagem com a solicitação ao servidor, que por sua vez envia uma mensagem de resposta ao cliente. Qualquer protocolo que siga o modelo de solicitação/resposta pode ser utilizado como por exemplo: HTTP, FTP e SMTP.

Todo servlet é executado em uma aplicação servidor específica, denominada Servlet Container. De acordo com a Figura 4.2 pode-se perceber a seguinte linha de execução:

- Primeiro o cliente envia uma solicitação ao servidor *Web*;
- O Servlet Container lê o Servlet e cria um *thread* para o processo. O servlet é instanciado apenas na primeira solicitação;

- O servlet faz a consulta ao Banco de Dados, constrói a resposta e envia ao Servlet Container, que o reenvia ao servidor *Web*;

- O servidor *Web* responde ao cliente.

Quando há diversas requisições para um servlet, o servidor Servlet manipula cada solicitação criando um novo *thread* para dada solicitação. Este *thread* é usado cada vez que o mesmo cliente faz uma solicitação ao servidor.

A API Java Servlet é definida como uma extensão do padrão JDK, na qual as extensões estão em pacotes que começam com javax. Esta API é formada por um conjunto de classes que define uma interface padrão entre um servidor *Web* e um Servlet Container. A API encapsula as solicitações do cliente como objetos para que o servidor *Web* possa transmiti-las ao Servlet Container. Este processo é similar no sentido contrário, onde o servlet Container encapsula as respostas para retransmiti-las ao servidor *Web,* que posteriormente repassará ao cliente.

A API dá suporte ao gerenciamento do ciclo de vida do servlet, ao acesso do contexto do servlet, à classes de utilidades e a classes de suporte HTTP. Ela é composta por dois pacotes [15]:

- javax.servlet: este pacote contém classes e interfaces genéricas que podem ser implementadas e extendidas por todos os servlets.

- javax.servlet.http: este pacote contém as classes que são extendidas quando se cria servlets específico do tipo HTTP, ou seja, que trabalhem com o protocolo HTTP e a geração de páginas HTML.

A interface javax.servlet.Servlet é o coração desta arquitetura por fornecer uma estrutura de trabalho para todos os servlets [15]. Ela define 5 métodos:

- init(): Método chamado automaticamente apenas uma vez na inicialização do servlet durante seu ciclo de execução [6];

- getServletConfig(): Método que retorna uma referência para um objeto que implementa a interface ServletConfig. Tal objeto fornece acesso às informações de configuração do servlet;

- service(ServletRequest request, ServletResponse response): Este é o primeiro método a ser chamado em cada servlet, recebe e responde à chamadas de clientes;

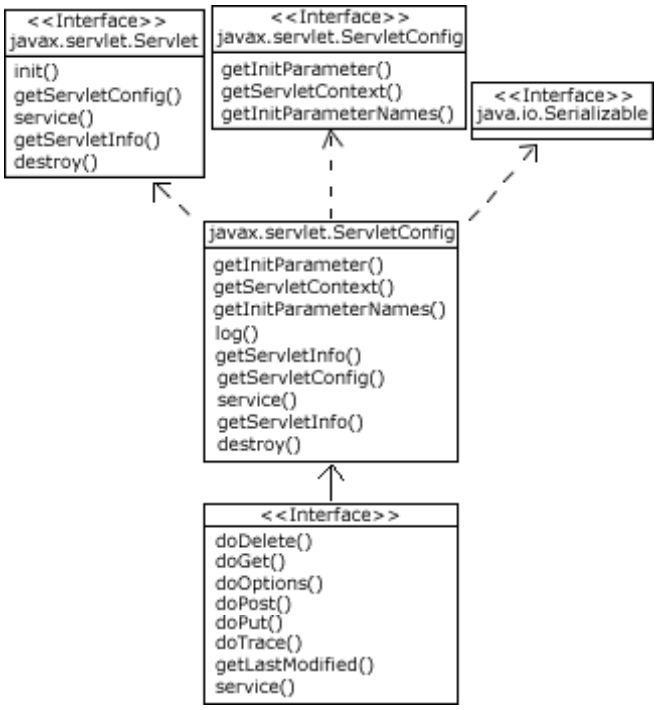

Figura 4.3: Estrutura de um Servlet

- getServletInfo(): Método que retorna uma String contendo informações do servlet, tais como o autor e versão;
- destroy(): Método que é chamado quando um servlet é finalizado pelo servidor em que está sendo executado [6];

Segundo [15], todos os servlets ou devem ser implementados nesta interface ou então devem herdá-la. A Figura 4.3 representa um modelo do nível mais alto da estrutura de um Servlet.

4.2.1 Classe HttpServlet

Todos os servlets baseados na *Web* extendem a classe HttpServlet. Os dois tipos mais comuns de solicitação de HTTP [6]:

- GET: obtém informações do servidor, como por exemplo a busca de um documento HTML ou imagem;

- POST: envia dados ao servidor, como por exemplo o envio de informações a um servidor através de um formulário HTML, o envio de informações de autenticação, etc.

Esta classe define os métodos doGet() e doPost() para responder às solicitações GET e POST. Estes métodos são chamados pelo método service(), que primeiro determina o tipo de solicitação, e então chama o método apropriado.

Os métodos doGet() e doPost() são similares, tendo como única diferença as solicitações de seus serviços, o primeiro recebe solicitações do tipo GET e o segundo recebe solicitações do tipo POST. Ambos os métodos recebem os objetos HttpServletRequest e HttpServletResponse, que encapsulam o formato solicitação/resposta [15].

4.2.2 Interface HttpServletRequest

Segundo [6] cada chamada de um doGet() ou doPost() recebe um objeto que implementa a interface HttpServletRequest, que contém um objeto que encapsula a solicitação do cliente.

Uma variedade de métodos é fornecida para permitir ao servlet processar a solicitação do cliente. Alguns dos métodos mais utilizados são:

- getParameter (String name): Retorna o valor associado com um parâmetro enviado para o servlet como parte de uma solicitação GET ou POST;
- getParameterNames (): Retorna os nomes de todos os parâmetros enviados para o servlet como parte de uma solicitação POST;
- getParameterValues (String name): Retorna um *array* de strings contendo os valores para um parâmetro de servlets especificados;
- getCookies (): Retorna um *array* de objetos cookie armazenados no cliente pelo servidor;
- getSession (boolean create): Retorna um objeto HttpSession associado com a sessão de navegação atual do cliente.

4.2.3 Interface HttpServletResponse

Cada chamada de um método doGet() ou doPost() para um HttpServlet, envia um objeto que implementa a interface HttpServletResponse. Este objeto contém a resposta para o cliente [6].

Alguns dos métodos fornecidos para permitir ao servlet formular a reposta ao cliente são:

- addCookie (Cookie cookie): Adiciona um *cookie* ao cabeçalho da resposta para o cliente;
- getOutPutStream (): Obtém um fluxo de saída baseado em *byte* que permite que os dados binários sejam enviados para o cliente;
- getWriter (): Obtém um fluxo de saída baseado em caracter que permite que os dados de texto sejam enviados para o cliente;
- setContentType (String type): Especifica o tipo MIME[1] da resposta para o navegador.

4.2.4 Ciclo de Vida de um Servlet

A interface Servlet define os métodos do ciclo de vida do servlet. Estes métodos são o init(), o service() e o destroy() [15]. Os métodos init() e destroy() são chamados somente uma vez durante o ciclo de vida de um servlet, enquanto que o método service(), juntamente com os métodos associados doGet(), doPost(), serão executados quantas vezes o Servlet Container receber uma solicitação.

init()

É o método onde a vida do servlet começa, onde são criados e inicializados os recursos que estarão sendo usados quando do atendimento das solicitações. É chamado pelo servidor apenas uma vez, imediatamente após o servlet ser instanciado [15].

O método init () pode ter dois formatos. O primeiro é sem argumento, conforme exposto a seguir:

[1] O tipo MIME ajuda o navegador a determinar como exibir os dados. Por Exemplo: o tipo MIME "text/html" indica que a resposta é um documento HTML.

```
public void init () throws ServletException
{
    [corpo do método]
}
```

O segundo formato considera como parâmetro o objeto ServletConfig:

```
public void init (ServletConfig config) throws ServletException;
{
    super.init (config);
    [corpo do método]
}
```

Caso haja informação necessária, será utilizado o método acima, que terá como parâmetro uma referência para ServletConfig. Neste caso recomenda-se criar uma chamada para super.init (config) de tal forma que a superclasse registre as informações para um acesso posterior [15]. O formato a ser utilizado depende de quais informações são necessárias no momento da instanciação.

O método init() pode utilizar o ServletException se por alguma razão o servlet não inicializar os recursos necessários para atender às chamadas.

service()

O método service() atende a todas as solicitações enviadas por um cliente. Mas ele próprio não pode iniciar seu serviço antes que o método init() tenha sido executado [15]. Ele recebe e responde solicitações de clientes, podendo ser chamado em duas situações:

- Quando o método init() for completado com sucesso;
- Cada vez que o servidor *Web* receber uma solicitação para o servlet.

Este método implementa solicitações e respostas. O objeto ServletRequest contém informação sobre o serviço de solicitação, encapsulando a informação vinda do cliente. O objeto ServletResponse contém a informação que será retornada ao cliente [15].

A declaração do método service() é feita da seguinte forma:

```
public void service (ServletRequest req, ServletResponse res)
        throws ServletException, IOException
{
    [corpo do método]
}
```

destroy()

Este método significa o final do ciclo de vida do servlet. Quando o serviço é finalizado, o método destroy() do servlet é chamado, realizando assim as operações de fechamento da execução do servlet tais como *garbage collection*, liberação de recursos, etc.

A declaração do método destroy() é feita da seguinte forma:

```
public void destroy ()
{
    [corpo do método]
}
```

4.2.5 Executando Servlets

Servlets não são executados da mesma forma que applets e aplicações, pois oferecem funcionalidades que extendem um servidor. Para executar um servlet os seguintes passos são necessários:

- Instalar o servlet em um endereço apropriado. Servlets são alocados em qualquer diretório em uma máquina onde um servidor Servlet estiver sendo executado.
- Solicitar serviços de um servlet através da solicitação de um cliente. Existem duas formas de se solicitar estes serviços:
  - Referenciando o servlet pelo nome na URL. Nesta forma de invocação, o *default* é a chamada do método doGet(). A seguinte URL irá executar o servlet no servidor: http://localhost:8080/servlet/BasicServlet.
  - Criando uma página HTML que enviará a solicitação para o servlet usando o método POST. Isto irá invocar o método doPost do servlet

Partes do código de um servlet podem estar inseridas diretamentes em páginas HTML, usando a técnica denominada Java Server Pages (JSP).

Assim, um comando Java não é usado para executar um servlet. Ao invés disso usa-se um comando URL apontado para a localização do servlet.

Capítulo 5

Inteligência Artificial Distribuída

Estudos de Inteligência Artificial (IA) consideram como modelo de inteligência o comportamento individual humano, cuja ênfase é colocada na representação de conhecimento e métodos de inferência. Por outro lado, o modelo de inteligência utilizado pela Inteligência Artificial Distribuída (IAD) baseia-se no comportamento social, sendo a ênfase colocada em ações e interações entre agentes [27].

A IAD surgiu da junção entre as áreas de IA e Sistemas Distribuídos. Na IA a unidade de análise e desenvolvimento é um processo computacional com um único local de controle, um único foco de atenção e uma base de conhecimento, enquanto que a IAD analisa o desenvolvimento de grupos sociais de agentes que trabalham de forma cooperativa, objetivando resolver um determinado problema [22].

A IAD se diferencia da IA na medida em que traz novas e mais abrangentes perspectivas sobre representação do conhecimento, planejamento, resolução de problemas, coordenação, comunicação, negociação, etc [22]. Neste capítulo será apresentada uma visão geral sobre IAD, visando embasar teoricamente o presente projeto. Na seção 5.1 tem-se uma conceituação de agentes. Na seção 5.2 sistemas multiagentes reativos e cognitivos são abordados, enfatizando os sistemas multiagentes cognitivos pois estes serão utilizadas no desenvolvimento do CE neste trabalho.

5.1 Agentes

Atualmente na literatura há várias definições para o termo "agente". Por exemplo, Etzioni e Weld definem um agente como *"... um programa de computador que se comporta de forma análoga a um agente humano, tal como um agente de viagem ou um agente de seguro"* [9]. Por sua vez, Russel e Norvig definem agente como *"... qualquer coisa que pode perceber seu ambiente através de realizadores"* [25]. É possível observar que a definição contextual de Etzioni e Weld tende a considerar os agentes como assistentes pessoais, enquanto a de Russel e Norvig apresenta uma abordagem geral e didática [22].

Essa dificuldade em ter uma definição de agentes pode ser explicada pelas seguintes razões [22]:

- A interdisciplinariedade e a abrangência dos campos de pesquisa e comercial, nos quais a teoria de agentes pode ser estudada e aplicada;
- Agentes estão presentes no mundo real. Geralmente, os conceitos envolvidos com ambientes nos quais os agentes estão inseridos são imprecisos e incompletos. Esta característica torna difícil a categorização dos ambientes, e como conseqüência, também a categorização dos agentes que os representam.

Na tentativa de conceituar o termo agentes, em [35] há uma proposta de adoção das noções fraca e forte de agência. A noção fraca considera um conjunto de propriedades/atributos que um *software* ou *hardware* deve apresentar para ser considerado um agente. As propriedades consideradas em [35] são as seguintes [22]:

- Autonomia: Com esta propriedade agentes operam sem a intervenção direta de seres humanos ou outros sistemas, e têm algum tipo de controle sobre suas ações e estados internos;
- Habilidade Social: Com esta propriedade agentes interagem com outros agentes através de algum tipo de linguagem de comunicação de agentes;
- Reatividade: Com esta propriedade agentes percebem seu ambiente (mundo físico, a Internet, outros agentes, etc), e respondem a mudanças que ocorrem neste ambiente;

- Iniciativa (*pro-activeness*): Com esta propriedade agentes não agem simplesmente em resposta a seus ambientes, mas tomam iniciativa e por isto são capazes de exibir comportamento orientado a objetivos.

A definição da noção forte de agência considera, além das propriedades definidas na noção fraca, outras características relacionadas às características humanas, como por exemplo estados mentais (conhecimento, crença, intenção, obrigação), emoção, racionalidade, etc.

Neste trabalho será adotada a definição de agentes proposta por [7], que considera agentes como:

Definição: *"...resolvedores de problemas que podem trabalhar em conjunto para resolver problemas que estão além de suas capacidades individuais"* [7].

Na IAD existe uma divisão inicial dos agentes em reativos e cognitivos. Essa divisão será destacada nas próximas seções.

5.1.1 Agentes Reativos

Um agente reativo não usa raciocínio simbólico complexo, estruturas de memória e uma representação interna explícita do conhecimento. Assim, não possui um histórico de suas ações passadas, nem pode fazer previsão de atos futuros. Com essas restrições este agente somente percebe o ambiente externo e, baseado nos estímulos do ambiente, reage de uma forma pré-determinada [22].

Pode-se ter em uma sociedade somente agentes reativos, e nesse caso a comunicação se dá de forma indireta, através do ambiente externo. Ou em uma mesma sociedade pode-se ter agentes reativos e cognitivos, onde a comunicação pode ser feita por passagem de mensagem, mas o agente reage em um processo de estímulo-resposta.

Devido a esta arquitetura baseada em estímulo-resposta, agentes reativos não precisam de sofisticados mecanismos de coordenação e comunicação. O comportamento do sistema é explicado pela existência de uma **inteligência emergente** [22].

Ainda em [22] tem-se algumas vantagens na modelagem de agentes reativos como: economia cognitiva, robustez e tolerância a falhas.

5.1.2 Agentes Cognitivos

Agentes cognitivos são capazes de raciocinar a respeito de suas intenções e conhecimentos, criar planos de ação e executá-los. Possuem modelos explícitos do mundo externo, estruturas de memória que permitam manter um histórico de ações passadas e fazer previsões de ações futuras, e um sistema desenvolvido de cooperação e comunicação [22]..

Uma característica que diferencia do estímulo-resposta é que os agentes cognitivos possuem controle deliberativo, na medida em que conseguem deliberar qual ação será executada, e posteriormente executá-la.

5.2 Sistemas Multiagentes

Pesquisas em IAD são freqüentemente classificadas em duas grandes áreas: Resolução Distribuída de Problemas (RDP) e Sistemas Multiagentes (SMA). Nesse dois casos os problemas são resolvidos de forma cooperativa e distribuída, usando processos denominados agentes. Nas próximas subseções apresenta-se com mais detalhes os Sistemas Multiagentes reativos e cognitivos.

5.2.1 Sistema Multiagentes Reativos

A abordagem reativa foi introduzida por Brooks [2], no domínio da robótica. Um exemplo de sistema multiagentes reativos é uma colônia de formigas, onde cada formiga é uma entidade simples. Mas uma colônia de formigas pode realizar trabalhos tais como: procura de alimentos, transporte do alimento até o formigueiro, defesa da colônia, etc. A execução destes trabalhos é complexa, muito embora a estrutura de cada formiga seja simples.

A seguir são destacadas algumas características de agentes e sistemas multiagentes reativos:

- Não há representação explícita de conhecimento. O conhecimento dos agentes é implícito e se manifesta através do seu comportamento;
- Não há representação do ambiente. O seu comportamento se baseia no que é percebido a cada instante do ambiente, mas sem uma representação explícita deste;

- Não há memória das ações. Os agentes reativos não mantêm um histórico de suas ações, de forma que o resultado de uma ação passada não exerce nenhuma influência sobre suas ações futuras;

- Organização etológica. A forma de organização dos agentes reativos é similar a de animais como insetos e microorganismos, em oposição à organização social dos sistemas cognitivos;

- Grande número de membros. Os sistemas multiagentes reativos têm, em geral, um grande número de agentes, de ordem de dezenas, centenas ou mesmo milhões de agentes [27].

Existem alguns ambientes de desenvolvimento para sistemas multiagentes reativos, como o sistema SEIEME - *Simulateur Evènementiel Multi-Entités* [21], o sistema Swarm [23], e o ambiente SIMULA [13].

5.2.2 Sistema Multiagentes Cognitivos

Os SMAs cognitivos são baseados em modelos organizacionais humanos, como grupos, hierarquias e mercados [27]. Segundo [12] apud [27] as principais características dos agentes cognitivos são:

- Mantém uma representação explícita de seu ambiente e de outros agentes da sociedade;

- Podem manter um histórico das interações e ações passadas;

- A comunicação entre agentes é feita através do envio e recebimento de mensagens;

- Seu mecanismo de controle é deliberativo, ou seja, tais agentes raciocinam e decidem sobre quais objetivos devem alcançar;

- Seu modelo de organização é baseado em sistemas sociológicos, como as organizações humanas;

- Uma sociedade contém tipicamente poucos agentes.

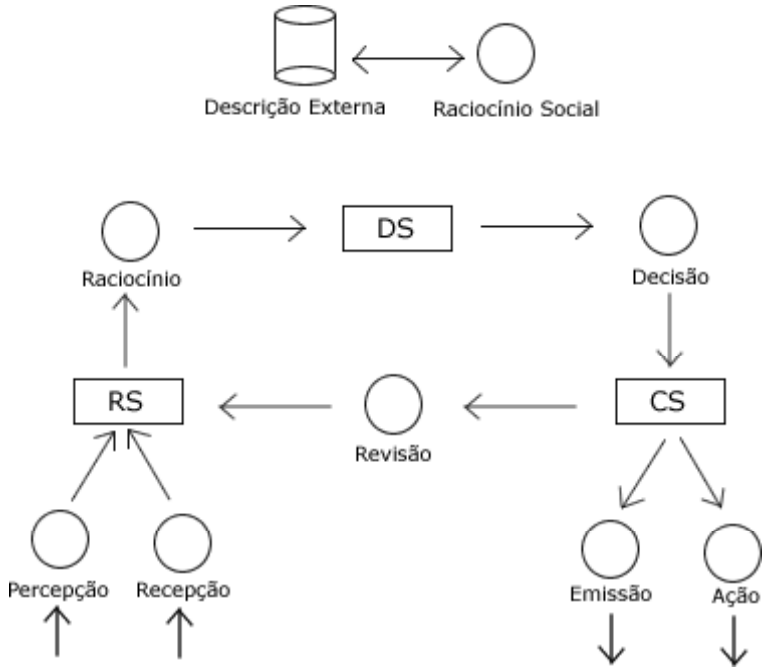

Figura 5.1: Arquitetura de um Agente

Arquiteturas de Agentes Cognitivos

Segundo [12] apud [27] uma arquitetura de agentes deve contemplar ao menos módulos responsáveis pelos mecanismos de comunicação, percepção, raciocínio e decisão.

A Figura 5.1 representa uma arquitetura de agente. Este agente é composto por mecanismos de percepção e recepção (entrada de dados), emissão e ação (saída de dados), raciocínio e decisão (funções internas) revisão (revisão de crenças do agente). Este agente raciocina sobre os outros agentes utilizando um raciocínio social. Este mecanismo utiliza as informações que o agente tem sobre os outros, armazenadas em uma estrutura denominada descrição externa. O agente também possui diversos estados internos como o estado de raciocínio (RS), de decisão (DS) e de engajamento (CS). Estes estados são modificados através da execução de diversos mecanismos internos ([28] apud [27]).

Organizações de Agentes

As organizações de agentes devem ser analisadas através da ocorrência de interações sociais. Estas interações permitem prever a formação dinâmica destas organizações. Caso existam restrições de comunicação entre os agentes, as interações sociais podem ser limitadas entre os diversos tipos de agentes do sistema.

Segundo [5] apud [27] existem duas classes de modelos para as interações sociais:

- Modelos **descendentes** (*top-down*)**;**
- Modelos **ascendentes** (*bottom-up*)**.**

Os modelos descendentes levam em consideração que de princípio os agentes ja têm um problema a resolver. Tendo a cooperação pré-estabelecida como uma hipótese de partida, as organizações sociais são limitadas por uma organização pré-existente, guiando os agentes para atingir seus objetivos. Neste modelo, a alocação de tarefas pode ser feita de modo estático ou dinâmico. Os diversos papéis de resolução são definidos em tempo de concepção. As desvantagens desta abordagem são a auxência de uma perspectiva dinâmica, um subjetivismo social e uma ênfase na comunicação como motor das interações sociais.

Nos modelos ascendentes os agentes não têm necessariamente um objetivo a atingir. As interações sociais são estabelecidas dinamicamente como forma de atingirem seus próprios objetivos. Estes modelos são mais utilizados em um SMA.

Interação entre Agentes

Praticamente todos os trabalhos em interações entre agentes cognitivos baseia-se na teoria de atos de fala (do Inglês *speech act theory)* [26] apud [27]. Esta teoria propõe uma categorização de primitivas de comunicação que são associadas às suas consequências. Alguns exemplos são: *inform, ask-to-do, answer, promisse, propose*, etc. A linguagem KQML é uma linguagem baseada em atos de linguagem, é uma das mais utilizadas na área de IAD.

Um ponto importante e freqüentemente abordado nos atos de fala são os protocolos de interação. Este protocolos servem para estruturar as trocas de mensagens entre agentes e são concebidos com um caráter genérico, pois podem ser utilizados em outras aplicações.

5.2.3 Linguagem KQML

Um dos aspectos mais importantes na sociedade entre agentes é a capacidade dos mesmos em cooperarem na solução de um problema comum. Para que ocorra a cooperação é necessário que haja comunicação [16].

Para a implementação de um mecanismo de comunicação entre agentes é preciso cumprir algumas exigências. Segundo [16] a linguagem deve:

- Prover um modelo simples, para se ajustar a uma ampla variedade de sistemas;
- Ser eficiente, ajustando-se bem a uma ampla variedade de sistemas existentes e provendo a possibilidade de uma implementação parcial;
- Prover segurança na comunicação entre agentes, garantindo o isolamento e a integridade dos dados, permitindo a manutenção de outros agentes e provendo mecanismos de troca de dados confidenciais.

Entre as linguagens de agentes encontradas na literatura que aderem estas características pode-se citar: KQML, Parla, e LALO. Na proxima subseção será apresentada a linguagem KQML, pois a mesma será utilizada no SMA desenvolvido neste projeto.

Knowledge Query and Manipulation Language (KQML)

A linguagem KQML foi desenvolvida dentro do *Knowledge Sharing Effort* (KSE) patrocinado pelo DARPA, em 1993. O objetivo do KSE é desenvolver técnicas e ferramentas para promover a distribuição do conhecimento entre agentes inteligentes [11].

KQML é uma linguagem de alto nível que caracteriza-se tanto pela formatação de mensagens quanto por um protocolo para a manipulação destas mensagens. Ela se identifica como uma linguagem de propósito geral, pois não soluciona ou sugere como deve estar estruturada a implementação dos agentes. Independentemente do ambiente ou estrutura dos agentes é possível utilizá-la para prover a comunicação num sistema multiagentes [11].

Segundo [22] a estrutura da linguagem é composta por três níveis: conteúdo, mensagem e comunicação. A Figura 5.2 ilustra a estrutura de camadas onde um conteúdo é encapsulado em uma camada de mensagem, que por sua vez está encapsulada em uma camada de comunicação.

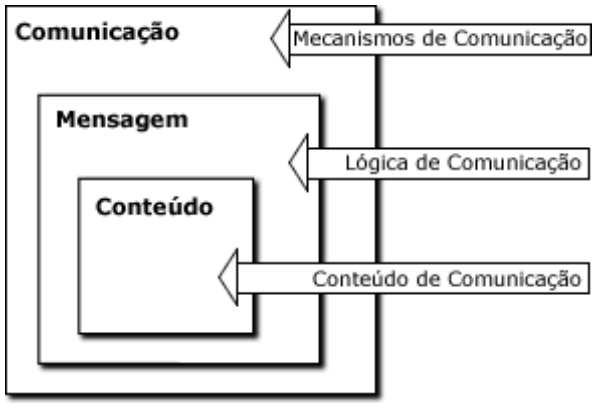

Figura 5.2: Camadas da linguagem KQML

Camada de Conteúdo Nesta camada está embutido o conteúdo da mensagem. Como a KQML não faz restrições a esta camada, pode-se usar qualquer tipo de linguagem de representação do conhecimento [22].

Camada de Mensagem Segundo [11] a camada de mensagem diz respeito aos tipos de mensagens que são enviadas entre os agentes durante a comunicação. Ela também pode ser reconhecida como uma camada de atos de linguagem, pois informa os tipos de atos de discurso que pode representar, tais como uma afirmação, uma pergunta, uma resposta, um erro e uma negação.

Devido ao fato da linguagem KQML possuir um conteúdo "opaco", a camada de mensagem adiciona um conjunto de características que descrevem o conteúdo, como por exemplo a linguagem na qual esta expresso, a ontologia que assume, etc.

Camada de Comunicação A camada de comunicação apresenta alguns parâmetros que servem de informação para que a mecânica de comunicação seja devidamente efetuada [11]. Nesta camada os agentes realizam troca de pacotes, onde são especificados alguns atributos de comunicação em baixo nível, tais como: a identidade do emissário e do destinatário e se a comunicação será síncrona ou assíncrona [22].

A linguagem KQML também utiliza agentes especiais, denominados facilitadores, que oferecem à sociedade de agentes suporte básico para uma

Performatives	Identificadores
Informação	TELL, DENY, UNTELL
Banco de Dados	INSERT, DELETE, DELETE-ONE, DELTE-ALL
Respostas	ERROR, SORRY
Consultas	EVALUATE, REPLY, ASK-IF, ASK-ABOUT, ASK-ONE, ASK-ALL, ASK-IN
Multi-resposta	STREAM-ABOUT, STREAM-ALL, EOS, STREAM-IN
Convencimento	ACHIEVE, UNACHIEVE
Pacotes	STANDBY, READY, NEXT, REST, DISCARD, GENERATOR
Definição de capacidade	ADVERTISE, IMPORT, EXPORT
Notificação	SUBSCRIBE, MONITOR
Rede	REGISTER, UNREGISTER, FORWARD, BROADCAST, PIPE, BREAK, TRANSPORT-ADDRESS, ROUTE
Facilitação	BROKER-ONE, BROKER-ALL, RECOMMEND-ONE, RECOMMEND-ALL, RECRUIT-ONE, RECRUIT-ALL

Figura 5.3: Tabela de *Performatives*

integração em rede tal como identificação, registro de serviços, passagem de mensagem, etc [22].

Performatives

A linguagem KQML enfoca um grande conjunto de mensagens pré-definidas, denominadas *performatives,* que definem as operações possíveis de serem executadas pelos agentes.

Uma mensagem KQML consiste do nome de uma *performative*, seus argumentos associados e um conjunto de argumentos opcionais. O nome *performative* abstrae o conteúdo da mensagem em um nível de atos de linguagem [22]. Elas são divididas em grupos de acordo com suas funções. A Figura 5.3 ilustra os grupos de *performatives*.

Os nomes das *performatives* são reservados, pois devem ter o mesmo significado em qualquer comunicação que utilize a linguagem KQML. Devido ao fato delas não serem necessárias e suficientes para todas as aplicações, os agentes podem considerar apenas um subconjunto e até mesmo trabalhar com *performatives* não pré-definidas.

Assim como as *performatives*, KQML também possui um conjunto de parâmetros chave reservados, sendo útil para estabelecer graus de uniformidade dos parâmetros e suporte para que os programas entendam *performatives* desconhecidas mas com parâmetros conhecidos. A Figura 5.4 ilustra os parâmetros reservados e seus significados.

Parâmetro Chave	Significado
:content	Informação sobre qual performativa expressa uma atitude
:force	Se o transmissor irá sempre negar o significado da performativa
:in-reply-to	O rótulo esperado em resposta
:language	O nome de uma linguagem contido no parâmetro :content
:ontology	O nome da ontologia usada no parâmetro :content
:receiver	O receptor atual da performativa
:reply-with	Se o transmissor espera uma resposta, caso sim, um rótulo para a resposta
:from	Indica o agente emissor original, usado quando há agentes intermediários
:to	Indica o agente receptor final, usado quando há agentes intermediários
:sender	O atual transmissor da performativa

Figura 5.4: Tabela de Parâmetros Reservados

Uma *performative* é expressa como uma *string* em ASCII, usando uma notação de lista baseada na linguagem LISP. Os parâmetros chave devem começar com dois pontos (:) e devem preceder o correspondente parâmetro. A seguir é mostrado um exemplo de mensagem KQML, onde no primeiro trecho há uma solicitação do agente de Interface para o agente de Informação, relacionado à inserção de informações de um usuário no Banco de Dados do CE.

```
(ask-about
      :sender InterfaceAg
      :content (Nome= "José Silva" E-Mail="zesilva@ec.ucdb.br")
      :receiver InformationAg
      :reply-with form01
      :language JAVA
      :ontology null)

(reply
      :sender InformationAg
      :content (Dados Cadastrados com Sucesso)
      :receiver InterfaceAg
      :in-reply-to form01
      :language JAVA
      :ontology null)
```

5.2.4 Protocolos de Comunicação

Um protocolo é um conjunto de regras para dar suporte à comunicação em uma rede. Quando a comunicação entre agentes se dá por passagem de mensagens, não há uma memória compartilhada e por isso devem existir regras que os agentes obedeçam, conhecidas como protocolos [22]. De acordo com [29] agentes participam de diferentes protocolos através de interações específicas com outros agentes, por exemplo respondendo a mensagens, realizando ações de seu domínio, etc.

Contract Net

Um dos protocolos de maior influência na área de Inteligência Artificial Distribuída é o *Contract Net* [32][31].

No *Contract Net* não há um controle centralizado, e os agentes podem cooperar através do compartilhamento de tarefas [22]. É um protocolo para cooperação entre os agentes e para solução de problemas por comunicação direta. É modelado sobre os mecanismos que governam troca de mercadorias e serviços. Provê uma solução para o chamado problema da conexão: encontrar um agente apropriado para fazer uma determinada tarefa [10].

Os agentes compartilham tarefas quando a tarefa é muito extensa ou o agente não tem conhecimento necessário para realizar uma determinada tarefa. Uma solução para tarefas extensas é o particionamento em subtarefas, para que se possa alocar estas subtarefas a outros agentes. Se o agente não tem conhecimento necessário, o agente procura outros agentes que estão aptos a executar ou gerenciar a tarefa a ser realizada.

Para que se possa compartilhar as tarefas, é necessário que essas tarefas sejam distribuídas entre os agentes, e para isso o *Contract Net* utiliza um processo de negociação e de estabelecimento de contratos. Como definido em [32] e [31], um contrato é um acordo explícito entre um agente que gera uma tarefa (chamado gerente), e um agente que executará a tarefa (chamado contratado). O gerente é responsável por monitorar a execução de uma tarefa. Os agentes não são designados a *priori* como gerentes ou contratados, gerando assim um sistema sem controle hierárquico.

Para que se possa determinar qual o agente é o contratado, é necessário estabelecer um contrato após um processo de negociação, onde o gerente de uma determinada tarefa avisa a existência de uma tarefa a outros agentes. Aqueles agentes que estão livres de qualquer tarefa, avaliam todos os anún-

cioas feitos por vários gerentes. Se algum contratado tiver potencial para realizar uma ou mais tarefas que foram anunciadas pelo gerente, é enviada uma licitação a este gerente, que tem como uma de suas funções analisar todas as licitações e escolher o agente mais apto para realizar a tarefa.

Do ponto de vista do gerente o processo é [10]:

- Anuncia uma tarefa que precisa ser executada;
- Recebe e avalia os "lances" dos contratantes potenciais;
- Decide por um contratante conveniente e envia o contrato para ele;
- Recebe e sintetiza os resultados.

Do ponto de vista do contratante, o processo é [10]:

- Recebe o anúncio de uma tarefa;
- Avalia sua própria capacidade de respondê-la;
- Responde negando ou fazendo um lance;
- Executa a tarefa se sua oferta é aceita;
- Envia os resultados.

A arquitetura de um agente do *Contract Net* está ilustrada na Figura 5.5.

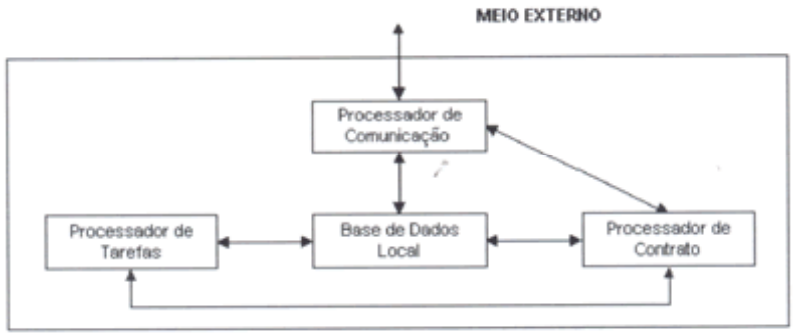

Figura 5.5: Arquitetura do Agente do *Contract Net.* Fonte [Smith 80]

A base de dados local contém o conhecimento individual e social do agente. O processador de comunicação é responsável pela comunicação com o ambiente externo e os demais agentes, através do recebimento e envio de mensagens. O processador de contrato tem a responsabilidade de analisar as tarefas submetidas à uma possível licitação, abrir licitações e finalizar contratos. O processador de tarefas é reponsável por executar as tarefas que o agente é capaz de realizar. Para tanto, ele recebe uma solicitação de execução do processador de contrato, consulta a base de dados local caso seja necessário, executa a tarefa e envia o resultado novamente ao processador de contrato [22].

Capítulo 6

Comércio Eletrônico

Nos últimos anos o ambiente empresarial vem passando por profundas mudanças, diretamente relacionadas com a Tecnologia da Informação (TI)[1]. A globalização, a integração interna e externa das organizações, dentre outros itens, têm confirmado a tendência da criação e utilização de sistemas de Comércio Eletrônico (CE) [1]. É esta situação que tem exigido um grande esforço das empresas para a adoção das tecnologias de informação referente ao CE em sua estratégia competitiva.

Este capítulo visa apresentar uma visão geral do CE. Para tanto, na Seção 6.1 tem-se os conceitos fundamentais da área. Na Seção 6.2 uma classificação dos tipos de CE é apresentada. Na Seção 6.3 são colocadas algumas aplicações na área. Finalmente, na Seção 6.4 apresenta-se a tecnologia de agentes no CE.

6.1 Conceitos Fundamentais

Nesta seção tem-se a apresentação de conceitos relacionados ao CE.

6.1.1 Definições de Comércio Eletrônico

O conceito de CE é muito amplo e possui vários pontos de vista e definições. Segundo [18] apud [1], dependendo da perspectiva, o CE pode ter as seguintes abordagens:

[1]Tecnologia da Informação (TI) é o termo usado para o conjunto dos conhecimentos que se aplicam na utilização da informática na estrutura de organização.

- Perspectiva de comunicações: CE de informações, produtos/serviços por meio de linhas telefônicas, redes de computadores ou qualquer outro meio eletrônico;

- Perspectiva de processo de negócios: CE é a aplicação de tecnologia para a automação de transações de negócio e fluxos de dados;

- Perspectiva de serviço: CE é uma ferramenta que endereça o desejo das empresas, consumidores e gerência para cortar custos de serviços, enquanto melhora a qualidade das mercadorias e aumenta a velocidade da entrega do serviço;

- Perspectiva *on-line*: CE provê a capacidade de comprar e vender produtos e informações na Internet e em outros serviços *on-line.*

De uma forma geral, segundo [1], CE é a realização de toda a cadeia de valores dos processos de negócio em um ambiente eletrônico, por meio da aplicação intensa das tecnologias de comunicação e de informação, atendendo aos objetivos de negócio, podendo ser realizado de forma completa ou parcial. São incluídas as transações negócio-a-negócio (do Inglês *Business to Business - B2B*), negócio-a-consumidor (do Inglês *Business to Consumer - B2C*) e o intra-organizacional (do Inglês *Intrabusiness)*, em uma infra-estrutura predominantemente pública de livre acesso e baixo custo.

6.1.2 Mercado Eletrônico

De acordo com [1], os mercados eletrônicos podem ser definidos como lugares de mercado colocados em ação por meio das TI. Eles apoiam todas as faces de transação e contribuem para a realização de um mercado econômico ideal como um lugar abstrato para trocas de informações. O mercado eletrônico é o meio utilizado pelo CE para realizar transações, operações, atividades, etc.

Os mercados eletrônicos são caracterizados pelas seguintes facilidades [1]:

- Onipresença, pelos mercados eletrônicos abertos 24 horas todos os dias, e por qualquer usuário ter acesso à rede independente de sua localização;

- Fácil acesso à informação;

- Baixo custo de transação.

Os mercados eletrônicos estão se tornando mais predominantes a cada dia. De acordo com [34] apud [1], os mercados tradicionais são caros por causa dos custos de coordenação. Mas a TI pode auxiliar as organizações a reduzir o custo de participação de mercado por meio dos conceitos de mercado eletrônico.

6.1.3 *E-Business*

O conceito de negócio eletrônico (do Inglês *eletronic business*, ou *e-business*), expressa uma maior abrangência em relação ao CE, principalmente por não se tratar apenas de compras e vendas, mas também da prestação de serviços ao consumidor, da colaboração com os parceiros de negócio e administração de transações eletrônicas dentro de uma organização.

6.1.4 Sistemas Interorganizacionais

Sistemas Interorganizacionais (do Inglês *Interorganizational Systems - IOS*) envolve fluxos de informações entre duas ou mais empresas, sendo construído usando rede acessível de forma pública ou privada. Através do tempo, a interação entre estas empresas evoluiu de simples sistemas ligando computadores e vendedores para mercados eletrônicos complexos, integrando fornecedores, produtores, canais intermediários e clientes. Os IOS tem por objetivo processar transações eficientes como transmissão de pedidos, faturas, pagamentos entre outros [1].

6.2 Classificação do Comércio Eletrônico

As aplicações de CE podem ser classificadas em três classes de transações, de acordo com a natureza destas transações [1]:

- B2B (Negócio-a-Negócio) do Inglês *Business-to-Business*: É realizado no ambiente entre organizações, facilitando as seguintes aplicações de negócios:
  - Gerenciamento de fornecedor;
  - Gerenciamento de estoque;
  - Gerenciamento de distribuição;

  - Gerenciamento de canal;
  - Gerenciamento de pagamento.

- B2C (Negócio-a-Consumidor) do Inglês *Business-to-Consumer*: É realizado no ambiente entre organizações e consumidores. São transações de vendas com computadores individuais. Nesta perspectiva, o CE facilita as seguintes transações econômicas:

  - Interação social;
  - Gerenciamento de finança pessoal;
  - Informações e compras de produtos.

- Aplicação Intra-Organizacional do Inglês *Intrabusiness*: É realizado no ambiente interno das organizações. Estas atividades normalmente são executadas em *intranets* envolvendo trocas de mercadorias, serviços ou informações, produtos corporativos para empregados, treinamento *on-line,* atividades de redução de custos, etc. Segundo [1], a finalidade deste tipo de CE é ajudar a empresa a manter relacionamentos que são críticos para entrega de valor ao cliente, sendo possível através da integração de várias funções em uma organização. CE facilita as seguintes aplicações:

  - Comunicações de grupo de trabalho;
  - Publicação eletrônica;
  - Produtividade da força de vendas.

6.3 Aplicações do Comércio Eletrônico

As aplicações visualizadas para o mercado de consumo pode ser classificada em [1]:

- Gerenciamento de finança pessoal e *home banking*. Estes serviços permitem aos clientes evitar filas longas e oferece a flexibilidade de fazer transações bancárias a qualquer momento. Evita também a construção de mais agências bancárias, cortando despesas de escritório.

- *Home Shopping*: Permite ao consumidor entrar em lojas *on-line*, analisar os produtos e comprar via *on-line*.

- *Home entertainment*: na área de entretenimento doméstico são desenvolvidos sistemas onde o cliente tem o controle sobre o que, quando e onde assistir. A programação pode ser escolhida através de um catálogo *on-line* de milhares de filmes, vídeos de música, documentários, novelas, concertos e eventos esportivos, dentre outros. Este mesmo cenário serve para categorizar jogos interativos.

- Microtransações de informações: esta categoria se refere a empresas provedoras de serviço que vendem informações digitais que podem ser enviadas por uma rede a um preço baixo, permitindo que o cliente efetue o pagamento por meio de um sistema eletrônico. Estas informações podem ser: dados, retratos, programas de computadores e serviços.

6.4 Agentes no Comércio Eletrônico

Existem várias tecnologias e teorias que podem ser utilizados na modelagem de sistemas sociais (humanos ou não) que envolvam interações entre os agentes. Pode-se citar as seguintes tecnologias PHP, ASP, Java, CGI, BASIC. Dentre as teorias tem-se Redes Neurais, Algoritmos Genéticos, Inteligência Artificial Distribuída, Sistemas Especialistas, *Machine Learning,* etc. A escolha da tecnologia e teoria a ser adotada depende em parte da filosofia do pesquisador, pois em última análise todas essas ferramentas da Ciência apresentam prós e contras e podem modelar problemas na área de computação. Entretanto cada teoria apresenta características próprias que melhor indica a modelagem de determinados grupos de problemas. Por exemplo, em [20] é dito que Algoritmos Genéticos são mais indicados para desenvolver adaptações especializadas para ambientes específicos.

Especificamente a teoria relacionada à IAD é indicada para modelar sistemas sociais pois propõe desde seu início na década de 70 o trabalho com aspectos sociais e políticos de sistemas computacionais, apresentando novas e mais abrangentes formas para a resolução de problemas, linguagens de programação, planejamento, representação do conhecimento, etc. Em decorrência a essa adequação do uso de IAD para modelar sistemas sociais, nesta seção analisado o uso de SMA para desenvolver sistemas de CE.

Uma das vantagens na utilização de agentes no CE é a possibilidade de personalização na implementação de um sistema. Os sistemas de agentes forneceriam infraestrutura necessária à uma conexão confiável, servindo também como um motor ou escudo para transações entre as bases de dados envolvidas nas diferentes aplicações [19].

A cada dia consumidores estão se preocupando em buscar maior quantidade de informações sobre os diversos produtos que existem no mercado. Por sua vez, os comerciantes estão preocupados em atrair clientes, oferecer vantagens para que o cliente sinta-se satisfeito, como por exemplo, suporte sobre seus produtos, etc. Agentes inteligentes podem ser empregados nesse exemplo de várias maneiras, como:

- Agentes que compram para o consumidor;
- Agentes que coletam informações sobre um determinado produto;
- Agentes que retornam para o consumidor sugestões de compras;
- Agentes que fornecem conselhos sobre os produtos, bem como solucionam problemas e dificuldades que o usuário possa ter no momento da compra.

As regras de negócio contidas nos sistemas de CE devem poder ser rapidamente modificadas, uma vez que novos produtos e promoções estarão sendo dinamicamente incluídos. As regras de negócio podem ser encapsuladas em agentes inteligentes programados em Java, responsáveis por atividades como determinar o perfil do cliente, auxiliá-lo na configuração de produtos, sugerir promoções e produtos complementares, etc [17].

6.4.1 Exemplos de Aplicação de Agentes no Comércio Eletrônico

A seguir tem-se exemplos de sistemas que foram implementados utilizando agentes no CE [19]:

- Kasbah, desenvolvido no MIT por Chavez e Maes, é um *Web site* onde o usuário cria agentes autônomos [19]. Os agentes criados têm como objetivo procurar compradores ou vendedores com potencial, negociando

com eles dependendo do interesse do criador do agente. Uma característica chave do Kasbah é que está aberto a adicionar novos tipos de agentes, usando diferentes estratégias de venda.

- *Broad Area Syndicated Information System* (BASIS), desenvolvido por Genesereth na Universidade de Stanford. É um agente que é usado em duas áreas de aplicação:
  - *CommerceNet*, que dá suporte à indústria de eletrônica na área do vale do silício com catálogos e serviços de informação [19].
  - *Health Care*, que prevê ligações entre diversas fontes de informação (médicos, enfermeiras, administração, seguradores, laboratórios, etc..) registro *on-line*, ferramentas de suporte à decisão, etc [14].

Capítulo 7

Análise de Requisitos para Sistemas de Comércio Eletrônico

Análise de Requisitos é uma metodologia utilizada na definição de um sistema computacional, transformando uma representação vaga do escopo do mesmo em uma compreensão mais completa. Sua realização é muito importante na etapa inicial de um projeto, pois ainda não há um consenso sobre um quadro detalhado da estrutura do sistema. Ela define ao final características que devem ser oferecidas e os padrões gerais de funcionamento.

Este processo propicia um ambiente de discussão, reflexão, amadurecimento e integração de pontos fundamentais do sistema. Os requisitos podem ser:

- **Requisitos Funcionais:** descrevem como o sistema implementado funcionará;
- **Requisitos Não-Funcionais:** descrevem restrições, qualidades e/ou benefícios associados ao sistema.

Neste relatório parcial, são apresentados requisitos funcionais, pois o objetivo nesta etapa do projeto é determinar os comportamentos principais de um CE. Na próxima seção tem-se a apresentação dos requisitos funcionais detectados.

7.1 Requisitos Funcionais

Requisitos funcionais especificam o comportamento de um sistema, apresentando o que o usuário espera que o sistema faça. Segundo [8] requisitos funcionais são:

- Especificações da funcionalidade do projeto;
- Ações que este produto deve executar, averiguar, calcular, registrar, recuperar;
- Derivados da finalidade fundamental do sistema.

A seguir tem-se a apresentação de requisitos funcionais a serem considerados quando do desenvolvimento de um CE.

7.1.1 Seleção do Produto

Este requisito oferece serviço para a busca e seleção de produtos que serão comprados pelo usuário. Esta etapa permite ao consumidor limitar a gama de produtos que melhor correspondem às suas necessidades, possibilitando assim uma maior agilidade na escolha e consequentemente uma maior satisfação do cliente. Para oferecer esta funcionalidade, o sistema deveria oferecer [3]:

- Um Banco de Dados com os produtos e suas características;
- Mecanismos de busca por palavras-chave e ou características do produto;
- Ferramentas que oferecem recomendações baseadas no perfil do usuário, bem como em opiniões fornecidas por outros usuários com interesses semelhantes;
- Mecanismos para visualizar informações dos produtos, por exemplo através de catálogos dinâmicos.

7.1.2 Seleção do Vendedor

Em compras via *Web,* os consumidores normalmente comparam os produtos de vários vendedores. Para oferecer tal funcionalidade um sistema de CE deveria disponibilizar:

- Mecanismos para a integração com agentes de outros CE;
- Mecanismos para permitir a comunicação dos usuários com estes agentes;
- Mecanismos para automatizar a comparação de diferentes alternativas de vendedores, personalizando os parâmetros de comparação (preços, tempo de entrega, etc).

7.1.3 Negociação

Neste requisito os termos da transação dos produtos são determinados, como por exemplo o preço e o prazo de entrega, quantidade, forma de pagamento, etc. Tais termos poderão ser negociados dinamicamente ao invés de serem fixados. Normalmente este tipo de negociação é mais utilizada em leilões virtuais, definindo suas próprias estratégias. Ele irá permitir aos agentes negociar segundo vários critérios, tais como: garantia, tempo de entrega, condições de pagamento entre outros [3].

Para oferecer esta funcionalidade o sistema de CE deverá oferecer serviços tais como:

- Uma base de dados com o perfil dos usuários;
- Agentes responsáveis pela intermediação entre os interesses de sistemas de CE e os interesses de compra do usuário (agentes de negociação);
- Um repositório com técnicas de negociação utilizadas em áreas como Administração, Economia.

7.1.4 Oferta de Produtos

Um dos usos do CE é prover uma grande quantidade de informações de diversos tipos de produtos aos clientes, por meio de brochuras eletrônicas e guias de compra *on-line.* Isto pode ser visto como um canal de marketing adicional [1].

Para oferecer este requisito o CE deve procurar oferecer:

- Uma base de dados com o perfil dos usuários;
- Repositório com técnicas de Marketing;
- Agentes de comunicação para apresentar de forma atrativa as ofertas aos usuários.

7.1.5 Determinar o Perfil do Usuário

A contribuição da tecnologia na aprendizagem sobre os clientes é sua capacidade de gravar todos os eventos do relacionamento, tais como os pedidos de informações sobre um produto, a compra de um produto, serviço à cliente requerido etc. Por meio dessas interações, seja por telefone, pessoalmente , etc, as necessidades do cliente são identificadas e alimentarão os futuros esforços de marketing [1].

Para oferecer este requisito um CE deveria disponibilizar:

- Uma base de dados com o perfil do usuário;
- Mecanismos de busca para obter informações personalizadas do usuário;
- Serviço para permitir ao usuário alterar ou excluir seu perfil.

7.1.6 Forma de Pagamento

Existem vários métodos de pagamento nos sistemas de CE. Os métodos tradicionais como cheques não são adequados para pagamento em tempo real [1]. As empresas estão se preocupando em cada vez mais facilitar a forma de pagamento para o usuário, podendo realizar uma compra com todas as opções de escolha e efetuar o pagamento on-line com segurança. Para oferecer serviços relacionados à pagamentos *on-line* um CE deve oferecer:

- Serviços relacionados a segurança;
- Serviços de conexão com instituições financeiras;
- Serviços para finalizar o processo de pagamento.

7.1.7 Logística

Uma das atividades da pós-venda é permitir ao usuário acompanhar o processo de efetivação da entrega do produto adquirido. Para disponibilizar esta funcionalidade o CE deve oferecer os seguintes serviços:

- Mecanismos de comunicação de tópicos como confirmação da compra, envio do produto etc;
- Histórico de compras efetivadas.

7.1.8 Acompanhamento do Usuário

Para definir o perfil de cada usuário, objetivando personalizao o atendimento, é necessário acompanhar todos os passos do usuário no sistema. Para oferecer esta funcionalidade, um CE deve oferecer mecanismos para captar e armazenar as seleções e entradas do usuário no sistema.

Capítulo 8

Considerações Finais

Até o presente momento foi realizado um levantamento bibliográfico das principais tecnologias existentes que poderão ser utilizadas neste projeto, tais como Java, Java Servlet, Inteligência Artificial Distribuída, Comércio Eletrônico e Engenharia de Software. Este levantamento envolvem a leitura e análise de livros, artigos, etc, bem como o desenvolvimento de programas Java e Java Servlet, com acesso a banco de dados, para obter um conhecimento necessário à implementação do Sistemas Multiagentes para CE na *World Wide Web*.

Os próximos passos para a conclusão do projeto pode ser observados nos itens abaixo:

- Modelagem da arquitetura dos agentes do SMA;
- Modelar o Banco de Dados dos agentes e da agência;
- Modelagem da arquitetura do SMA;
- Escolher a plataforma de agentes;
- Realizar testes com a plataforma escolhida;
- Implementar agentes de teste, bem como suas interações;
- Implementar o Banco de Dados dos agentes e da agência;
- Implementar os agentes do estudo de caso;
- Implementar o SMA cognitivo;

- Realizar a verificação do sistema desenvolvido;
- Realizar a validação do sistema.

Bibliografia

[1] A. L. Albertin. *Comércio Eletrônico: Modelo, Aspectos e Contribuições de suas aplicações.* Atlas, São Paulo, 2001.

[2] R. Brooks. *A Robust layered Control System for a Mobile Robot.* IEEE Journal of Robotics and Automation, São Francisco, USA, 1986.

[3] H. D. A. L. Cardoso. *Sistema Multi-Agente para Comércio Eletrônico.* Acessado em 17 de maio de 2002, http://www.fe.up.pt/ eol/PROJECTS/SMACE/relatorio.html, 1999.

[4] S. I. A. Chan, M. C.; Griffith. *Java 1001 Dicas de Programação.* Makron Books, são Paulo, 1999.

[5] C. C. Conte, R. *Mind is not enough: precognitive bases of social interactions.* Proc. of the 1st. Symposium of Simulating Societies, Guildford, UK, 1992.

[6] P. J. Deitel, H. M.; Deitel. *Java Como Programar.* Campus, Porto Alegre, 2001.

[7] L. V. R. C. D. D. Durfee, E. H. *Trends in Cooperative Distributed Problem Solving.* IEEE Transaction Knowledge Data Eng., 1989.

[8] G. G. Dutekevicz, I. C. *Requisitos Funcionais.* Acessado em 15 de maio de 2002, http://www.eps.ufsc.br/ kern/06ReqFunc.pdf, 2000.

[9] W. Etizioni. *Inteligent Agents on the Internet: Fact, Fiction, and Forecast.* IEEE Expert, 1995.

[10] X. M. Eudenia. *Contract Net.* http://www.ime.usp.br/ eudenia/jaia/texto/node25.html, 2001.

[11] R. A. Faraco. *Protocolo e Linguagem de Comunicação KQML.* Acessado em 09 de março de 2002, http://www.eps.ufsc.br/disserta98/faraco/cap3.htm, 1998.

[12] G. L. Ferber, J. *Intelligence artificielle distribuée.* XI International Workshop on Expert Systems and their applications, Avignon, France, 1991.

[13] R. Frozza. *SIMULA - um Ambiente para o Desenvolvimento de Sistemas Multiagentes reativos.* CPGCC/UFRGS, RS/Brasil, 1997.

[14] M. Genesereth. *Center for Information Technology (CIT).* Acessado em 14 fev. 2002, http://logic.stanford.edu/cit/cit.html, 2000.

[15] J. Goodwill. *Developimg Java Servlets.* SAMS, Indiana - USA, 1999.

[16] C. A. R. Hubner, J. F. *Avaliação de comunicação entre agentes utilizando KQML.* Acessado em 09 de março de 2002, http://www.lti.pcs.usp.br/ jomi/publicacoes/1998/seminco/kqml/arigo.html, 1998.

[17] JumpNet. *E-Commerce.* Acessado em 13 dez. 2001, http://www.jumpnet.com.br/pages /ecommerce.asp, 2001.

[18] W. A. Kalakota, R. *Eletronic Commerce: a Manager's Guide.* Addison-Wesley, New York, 1997.

[19] B. S. Kerschberg, L. *The DPSC Eletronic Marketspace.* Acessado em 13 dez. 2001, 2001.

[20] J. S. Lansing. *Artificial Societies and the Social Sciences, Relatório Técnico.* EUA, 2002.

[21] L. Magnin. *SIEME - Simulateur dtenvironnement pour systèmes multi-agents.* http://www-laforia.ibp.fr/ magnin/these/sieme, 1996.

[22] M. G. B. Marietto. *Definição Dinâmica de Estratégias Instrucionais em Sistemas de Tutoria Inteligente: Uma Abordagem Multiagentes na WWW.* These de Doctorat de 1'INPG, ITA, Brasil, 2000.

[23] B. R. L. C. A. M. Minar, N. *The SWARM Simulation System: A Toolkit for Building Multi-Agent Simulations.* http://www.santafe.edu/projects/swarm/overview/overview, 1996.

[24] A. e. a. Newman. *Usando Java.* Campus, Rio de Janeiro, 1997.

[25] N. P. Russel, S. *Artificial Intelligence - a modern approach.* Cours, 1995.

[26] J. Searle. *Speech acts.* Cambridge, USA, 1969.

[27] A. L. O. Sichman, J. S. *Introdução aos Sistemas Muitagentes.* UFRGS, Instituto de Informática and USP Escola Politécnica, Porto Alegre, RS and São Paulo, SP, 2001.

[28] J. S. Sichman. *Du raisonnement social chez les agents: une approche fondeé sur la théorie de la dependance.* These de Doctorat de 1'INPG, Grenoble, France, 1995.

[29] M. Sigh. *On the Semantics of Protocols Among Distributed Intelligent Agents.* Proceedings of Eleventh Annual International Phoenix Conference on Computers and Communications, Austin, Texas, 1992.

[30] M. E. T. S. Silva. *História do Java.* Acessado em 07 nov. 2001, http://www.ualg.pt/uceh/ceduc/cadeiras/met2/discentes /trabalhos/19981999/paginas/manuel/java.htm, 2001.

[31] D. R. Smith, R. G. *Frameworks for Cooperation in Distributed Problem Solving.* IEEE Transaction on Systems, Man, and Cybernetic, 1981.

[32] R. G. Smith. *The Contact Net Protocol: High-Level Communication and Control in a Distribuited Problem Solver.* IEEE Transaction and Computers, 1980.

[33] Unifra. *Programação Orientada a Objeto.* Acessado em 22 fev. 2002, http://cpd-srv03.unifra.br/ espode/UNIFRA/LingProgII/default.htm, 2000.

[34] O. E. Willianson. *The Economic Instituctions of Capitalism.* Free Press, New York, 1985.

[35] J. N. Wolldridge, M. *Agents Theories, Aplication and Languages.* 1st International Conference on MultiAgent Systems, São Francisco, USA, 1995.