UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Uma arquitetura para coordenar a interação de agentes na Internet

por

SÍLVIO CÉSAR CAZELLA

Dissertação submetida à avaliação, como requisito parcial
para a obtenção do grau de Mestre
em Ciência da Computação

Prof. Dr. Luís Otávio Campos Alvares
Orientador

Porto Alegre , março de 1997.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Cazella, Silvio César

Uma arquitetura para coordenar a interação de agentes na Internet / por Silvio César Cazella. - Porto Alegre: CPGCC da UFRGS, 1997.

115 f.:il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul. Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, BR-RS, 1997. Orientador: Alvares, Luis Otávio Campos

1. Inteligência Artificial. 2. Inteligência Artificial Distribuida. 3. Sistemas Multiagentes. 4. Uma arquitetura para coordenar a interação de agentes na Internet. I. Alvares, Luis Otávio Campos. II.Título.

"....Não sabendo que aquilo era impossível de ser feito, ele foi lá e fez. - Jean Cocteau."

"...Não é o que acontece com você. É a maneira como você se comporta a respeito. - W. Mitchell"

"...Use todos os talentos que você possui: os bosques ficariam muito silenciosos se nenhum pássaro cantasse, a não ser os que cantassem melhor. - Henry Van Dyke"

Agradecimentos

Gostaria de agradecer a muitas pessoas que neste tempo de mestrado, deram-me muito auxílio e confiança. Tentarei listar algumas, mas como sempre, por sermos humanos, podemos nos esquecer de alguém, sendo assim, obrigado a todos em geral.

Começo pelas pessoas do próprio Instituto de Informática: a minha grande amiga Maria do Carmo Toscani, uma das responsáveis pelo meu mestrado, pois foi uma das minhas maiores incentivadoras iniciais; ao funcionário Volnei da Rocha Delgado pelas informações sobre os tramites burocráticos do Instituto de Informática; a grande equipe da biblioteca do Instituto por todo o apoio e esclarecimentos prestados durante o periodo de mestrado, apresentando sempre um atendimento educado e dedicado; ao pessoal responsável pela administração da rede do Instituto, agradeço a todos em geral por todo o apoio e paciência na tentativa de responder todas as minhas dúvidas sobre o funcionamento de redes de computadores; e por último deixo meu muito obrigado a uma grande pessoa de simpatia e astral incomparáveis, sempre calma, bem educada e preocupada com os (como ela mesmo denomina) seus filhos do Instituto de Informática, a funcionária Eliane Ricardo Iranco.

Gostaria de estender os meus agradecimentos aos amigos do doutorado, que sempre mostraram-se dispostos a ajudar aqueles que estavam começando na vida acadêmica, meu muito obrigado ao amigo Marcelo Ladeira, ao amigo Jugurta Lisboa Filho, a minha amiga e ex-professora Lúcia Maria Martins Giraffa, e a amiga Patricia Alejandra Behar.

Não poderia deixar de esquecer de agradecer alguns professores do Instituto, por terem demonstrado sempre dedicação e paciência na hora de tirar dúvidas, e orientar, muito obrigado a professora Rosa Maria Viccari, por ter praticamente adotado-me no primeiro semestre do curso, ao professor Antônio Carlos da Rocha Costa pelas dicas e informações prestadas, e ao professor Luís Otávio Campos Alvares pela atuação exemplar como orientador.

Por fim, deixo meu muito obrigado a minha família que soube ter paciência, e companheirismo nos intermináveis momentos em "que tive de sair do ar", para entrar nos livros e no computador, não podendo dar-lhes toda a atenção merecida. Agradeço a toda a turma (ou seria melhor dizer amigos ou equipe) dos anos de 1994, 1995 e 1996 do mestrado, com os quais tive o privilégio de conviver nestes anos de mestrado, e que nos momentos de crise em que eu olhava para a situação educacional do país e me perguntava se tanto sacrifício realmente era valido, sempre deram-me apoio para ir em frente e lutar por algo melhor, pois como dizia a grande poetiza Cora Coralina, "*.....mais vale lutar, do que colher dinheiro fácil....*".

Já ia esquecendo, muito obrigado a mim, pelo esforço empreendido.

Sumário

Lista de abreviaturas

BSA	Agente Básico de Software (*Basic Software Agent*)
DNS	Sistema de Nomes de Referência (*Domain Name System*)
FDDI	Interfaces descentralizadas de fibra para dados (*Fiber Distributed Data Interfaces)*
FTP	Protocolo de transferência de arquivos (*File Transfer Protocol)*
GUI	Interface Gráfica do Usuário (*Graphic User Interface*)
HTML	Linguagem de Anotação de Hipertexto (*HyperText Markup Language*)
IA	Inteligência Artificial
IAD	Inteligência Artificial Distribuída
KQML	Linguagem de Consulta e Manipulação do Conhecimento (*Knowledge Query and Manipulation Language*)
OOP	Programação Orientada a Objeto (*Object Oriented Programming*)
PPP	Protocolo Ponto a Ponto (*Point to Point Protocol*)
RDP	Resolução Distribuída de Problemas
RFC	Solicitação para comentários (*Requests for comments*)
SLIP	Protocolo Internet para Linha Serial (*Serial Line Internet Protocol)*
SMA	Sistemas Multiagentes
TCP/IP	Protocolo de controle de Transmissão / Protocolo da Internet (*Transmission Control Protocol / Internet Protocol)*
URL	Localizador Uniforme de Recurso (*Uniform Resource Locator*)
WWW	Rede mundial de computadores (*Wourld Wide Web*)

Lista de figuras

Lista de tabelas

Resumo

O grande salto tecnológico ocorrido nos últimos decênios em áreas como a informática e as telecomunicações já começa a causar uma verdadeira revolução social. Com o advento da Internet, a possibilidade de substituir a presença física de pessoas por assistentes inteligentes ou agentes é uma realidade que começa a tomar forma.

O objetivo principal deste trabalho é definir como deve ser feita a interação entre agentes, nas sociedades de agentes dentro da Internet. Para tanto, é apresentada uma proposta de arquitetura geral para coordenar a interação de agentes na Internet, de forma a possibilitar, entre outros, a localização de agentes, a conexão entre os agentes, a identificação de agentes (nomes de agentes, função e endereços destes na rede) sem que ocorra a repetição de nomes na sociedade, a entrada e saída de agentes da sociedade, a troca de mensagens entre os agentes e a procura de agentes com determinadas características.

Inicialmente, é apresentada uma visão geral sobre Inteligência Artificial Distribuída, área em que o trabalho se insere, e alguns exemplos de aplicações agentes. A seguir, as principais características da linguagem Java são introduzidas, por ser a linguagem utilizada nas implementações realizadas. Estas implementações foram feitas, a nível de protótipos, para permitir a obtenção de experiência no trabalho com agentes. Toda a programação destes agentes foi feita utilizando a linguagem Java, e a sua escolha como linguagem para estas implementações, foi feita devido a características próprias. Por exemplo, a linguagem Java é independente da plataforma de trabalho, o que no caso de programação para Internet, torna-se um característica muito desejável.

Após, dois sistemas para o desenvolvimento de agentes na Internet são apresentados em suas principais características do ponto de vista da arquitetura geral. Estes sistemas são o SodaBot e o sistema Java Agent Template. Este último totalmente implementado com a utilização da linguagem Java.

Finalmente, apresenta-se em detalhes a arquitetura proposta, com um exemplo de utilização e perspectivas para a continuação do trabalho. O resultado obtido com todos os estudos realizados diz respeito a proposta de uma arquitetura que tem por objetivo permitir coordenar a interação entre os agentes que vierem a ser implementados na Internet. Toda a proposta foi acompanhada por estudos reais da viabilidade da implementação futura da arquitetura.

Palavras-Chave: Inteligência Artificial, Inteligência Artificial Distribuída, Sistemas Multiagentes, Arquitetura para coordenar a interação de agentes na Internet.

TITLE: "AN ARCHITECTURE TO COORDINATE THE INTERACTION OF AGENTS IN THE INTERNET"

Abstract

Technological breakthroughs which occurred in the last decades in areas such as Computer Science and Telecommunications have started to cause an undeniable social revolution. With the Internet, the possibility of replacing the physical presence of people by intelligent assistants or agents is now becoming a reality.

This thesis aims at defining how the interaction among agents in the Internet should be achieved. An architecture to coordinate the interaction of agents in the Internet is presented, so as to make it possible, among other things, the localization of agents, the connection among agents, the identification of agents (their names, function, and addresses in the net), the way in and the way out of agents of the society, the exchange of messages among agents and the search for agents with specific characteristics.

First, a general view on Distributed Artificial Intelligence, the area in which this work is inserted, and some examples of agents are given. Next, the main characteristics of the Java language are introduced, since it is the language used in the implementations carried out. These implementations accomplished were prototypes which aim at gaining experience in the work with agents. All the programming of these agents was carried out using the Java language, which was chosen for these implementations due to its characteristics. For example, the Java language is Architecture Neutral, which in the case of programming for the Internet, becomes very desirable.

After that, the main characteristics of two systems for the development of agents in the Internet, concerning their general architecture, were presented. These systems are SodaBot and Java Agent Template, the latter being totally implemented in the Java language.

Finally, the architecture which was proposed is described in detail with an example of utilization and perspectives for a continuation of the work. The result obtained from this work is related to the proposal of an architecture which aims at making it possible to coordinate the interaction among agents which will be implemented in the Internet. The entire proposal was accompanied by studies of the feasibility of a future implementation of this architecture.

Keywords: Artificial Intelligence, Distributed Artificial Intelligence, Multiagent Systems, Architecture to coordinate the Interaction of agents in the Internet.

1 Introdução

Com o advento da Internet, a possibilidade de substituir a presença física pela eletrônica no trabalho, realmente começou a tomar forma. A Internet possibilitou através de seus recursos, que os mais variados profissionais pudessem comunicar-se a longas distâncias e consultar informações provenientes dos mais variados endereços eletrônicos através do mundo.

Muito do tempo útil de trabalho humano que acaba sendo perdido com tarefas simples, poderia ser realizado por assistentes inteligentes junto a Internet, assistentes que teriam a finalidade de auxiliar o usuário da rede. Cada assistente inteligente constitui-se em um agente modular e autônomo, e encontra seu embasamento teórico na Inteligência Artificial e na Inteligência Artificial Distribuída respectivamente.

Os sistemas de Inteligência Artificial tradicionais concentravam sua atenção em um único agente, o qual seria provido de alguma espécie de inteligência, e sozinho seria especialista na realização de uma tarefa específica.

Atualmente o enfoque tende a ser outro, fala-se em Inteligência Artificial Distribuída. Nesta subárea da IA o enfoque recai sobre o aspecto de integração de agentes providos de alguma inteligência. São especialistas na atividade que desempenham, não trabalham mais isolados, mas de forma cooperativa tentam resolver um problema da melhor forma possível, caracterizando uma área da Inteligência Artificial Distribuída, a área dos Sistemas Multiagentes.

O termo agente é muito utilizado, tendo virado uma marca de qualidade em termos mercadológicos. Praticamente no mercado de hoje todo o programa é dito como incluindo *agent technology*. Processadores de textos que possuem flexibilidade, programas que verificam quando algo é digitado, alertando quando uma palavra está mal escrita. Estes programas são agentes? Na verdade, deve-se tomar algum cuidado ao definir se um determinado *software* é ou não um agente, pois o mundo dos *softwares* que hoje vivem sobre a rubrica dos agentes é multi-facetado. Mas os agentes apresentam algumas características próprias que os diferenciam dos demais softwares.

Os agentes são entidades autônomas, e encontram-se, geralmente, imersos em uma sociedade. Conceitualmente são entidades capazes de reagir e/ou agir a favor de algo ou alguém, influenciando seu ambiente. Este ambiente necessariamente inclui outros agentes. Interagir com este ambiente, ou mais especificamente com outros agentes, é o elemento chave de um Sistema Multiagente.

Os agentes podem atuar em nome de alguém em particular, isto é, eles podem tomar algumas atitudes as quais representam de maneira apropriada os interesses de outros; portanto os agentes também devem ser robustos e capazes de manipular com segurança informações privadas. Os agentes são entidades altamente interativas - eles despendem muito de seu tempo comunicando-se com outros agentes e com seres humanos, são participantes altamente ativos em seu mundo computacional, isto é , eles reagem a algo e causam mudanças em todo o estado do sistema.

No caso de agentes inteligentes, está-se falando de entidades de *software* que são capazes de demonstrar um comportamento autônomo orientado a um objetivo, em um ambiente computacional heterogêneo. Um agente pode interagir com uma sociedade de agentes - tanto humanos como eletrônicos - em alguma tarefa. Dependendo da natureza do ambiente e da tarefa, a arquitetura de um agente de *software* inclui a capacidade de representar o conhecimento, alcançar metas, interagir com o ambiente, e tratar com ocorrências inesperadas. Estes também podem requerer habilidade de coordenação e colaboração com outros agentes (humano ou eletrônico), utilizando uma determinada linguagem **[HED 95]**.

O propósito destes assistentes inteligentes (agentes), é o de auxiliar o usuário na realização de tarefas específicas dentro da Internet. Estes assistentes realizam tarefas como: vasculhar a rede na busca de informações e endereços eletrônicos de interesse do usuário, coordenação de tarefas entre um grupo de pessoas, troca de informações entre os agentes que auxiliam os componentes de um grupo de trabalho, etc.

Os assistentes inteligentes (agentes) são entidades úteis e práticas. Abaixo são descritos dois tipos de agentes de *software* úteis:

Assistentes personalizados on-line:

Estes agentes geralmente pertencem a uma pessoa em particular e atuam como secretárias. Eles realizam atividades como:

a) Automaticamente respondem pedidos de marcação de encontros através da consulta de seus próprios horários de compromissos.

b) Contatam seus proprietários de maneira apropriada fazendo uso de suas localizações, por exemplo, via a apresentação de uma janela na sua estação de trabalho, mandando um fax ou correio eletrônico, ou até mesmo realizando uma chamada telefônica.

c) Filtrar e classificar a chegada de correio eletrônico e *fax* baseado em suas próprias preferências.

Agentes de aplicação:

Estes agentes coordenam a transferência e processamento de informações entre pessoas e outros agentes. Agentes de aplicação incluem **[COE 94]**:

a) *Time schedulers*, para marcação de encontros de grupos ou individuais através da negociação entre seus agentes pessoais.

b) Sistema de processamento de texto, que permitem um processamento de documentos envolvendo muitas pessoas em diferentes *sites*.

c) Agentes recepcionistas que aceitam pedidos e determinam o destino apropriado pela interação com outros agentes (e talvez com pessoas).

Muitos dos trabalhos realizados na área dos agentes podem ser colocados dentro de uma das seguintes categorias **[COE 94]**:

1) Tratamento altamente teórico das intenções e capacidades dos agentes.

2) Construção de agentes específicos.

Os agentes apresentam, todas as características relatadas acima; mas se estes podem interagir entre si, convivendo em uma chamada sociedade de agentes, como deve ser a arquitetura de interação entre estes indivíduos? De que maneira um agente consegue estabelecer um contato direto com o agente da sociedade de seu interesse? E se reportarmos estas sociedades para o imenso ambiente da Internet, como deve ser o procedimento para que estes agentes encontrem-se neste emaranhado de redes? O trabalho realizado nesta dissertação procura fornecer respostas a estas questões.

A motivação maior na realização desta dissertação, foi a possibilidade de unir algumas tecnologias em um único estudo: estudo dos Sistemas Multiagentes, estudo da estrutura da Internet, estudo de arquiteturas para implementação de agentes inteligentes, estudo de agentes, e o estudo da linguagem Java para permitir a implementação de alguns agentes assistentes, que acompanham as características dos agentes propostos pela Inteligência Artificial Distribuída dentro do ambiente da Internet.

O objetivo desta dissertação é propor uma arquitetura que possibilite implementar agentes inteligentes no ambiente da Internet, possibilitando que haja a interação entre estes agentes, nas mais diversas sociedades. Em outras palavras, o objetivo é definir uma arquitetura de agentes que permita a implementação de Sistema Multiagentes na Internet.

Para atingir-se este objetivo foram traçadas algumas diretrizes básicas para o trabalho, entre estas pode-se citar o estudo de outras arquiteturas existentes, o estudo de alguns agentes já implementados na Internet ou agentes simples que não fazem uso dos recursos da Internet, e o estudo da linguagem Java para construção e implementação de alguns agentes de teste.

Durante toda a etapa de estudo e implementação de agentes, e de estudo de outras arquiteturas foram sendo levantados vários pontos que deveriam ser estudados para definição de uma arquitetura. Um destes pontos referia-se a necessidade de um agente central na sociedade de agentes, um agente que teria a função de coordenar toda a interação entre os membros da sociedade.

A fase de implementação de agentes foi de profunda importância para permitir que a arquitetura fosse proposta, observando-se as possibilidade reais de implementação desta.

A dissertação está estruturada nos capítulos abaixo apresentados.

O capítulo 2 apresenta uma visão geral do significado do termo Inteligência Artificial Distribuída (IAD) e agentes, destacando a relação da IAD com os Sistemas Multiagentes.

No capítulo 3 são apresentados alguns exemplos de agentes. Este capítulo procura demonstrar que estes agentes já existem e muitas vezes são utilizados sem que o usuário tome conhecimento deles.

O capítulo 4 enfatiza a utilização da linguagem Java na implementação de agentes inteligentes. Este capítulo procura dar uma rápida visão sobre esta linguagem, e apresentar as propostas dos agentes implementados.

No capítulo 5, é apresentado o estudo feito sobre dois sistemas que se propõem a permitir a implementação de agentes. Os dois sistemas apresentados são o SodaBot e o JAT respectivamente.

No capítulo 6, finalmente, é apresentada em detalhes uma proposta de arquitetura que permite a implementação de agentes inteligentes na Internet.

Espera-se que este trabalho, através dos resultados obtidos, sirva como estímulo para aumentar as pesquisas no campo de Sistemas Multiagentes (SMA), bem como aumentar o número de aplicações reais utilizando SMA.

Espera-se, também, que esta arquitetura propicie o surgimento de vários assistentes que venham a auxiliar os usuários da Internet.

2 Inteligência Artificial Distribuída

A Inteligência Artificial Distribuída (IAD) constitui-se em uma das áreas de pesquisa da Inteligência Artificial tradicional. A IAD procura estudar o conhecimento e as técnicas de raciocínio que podem ser úteis para que um grupo de agentes computacionais integrem uma sociedade de agentes.

A IAD desenvolve técnicas de IA para tentar solucionar problemas de maneira distribuída. Estas técnicas passam a ser úteis na solução de problemas, amplos onde o domínio da questão é inerentemente distribuído no espaço, como por exemplo, pode-se citar os problemas de controle de tráfego aéreo, distribuição de energia elétrica, etc.

Para podermos entender melhor e até mesmo justificar a existência das técnicas de Inteligência Artificial Distribuída, podemos comparar esta com a própria sociedade humana. Na sociedade humana para termos um determinado problema solucionado, muitas vezes dependemos da interação de mais de um agente (ser humano) e da comunicação destes para que a meta (solução do problema) seja atingida de maneira satisfatória.

Sob o aspecto da Resolução Distribuída de Problemas em Inteligência Artificial Distribuída (IAD), uma tarefa global é inicialmente definida e o problema principal é projetar as entidades distribuídas, para permitir a execução desta tarefa global. O objetivo é estudar a distribuição e a resolução colaborativa de uma determinada tarefa.

Em IAD, entidades individuais que contribuem para solução de um problema são chamadas de agente. Agentes quando agrupados podem formar uma comunidade do tipo cooperativa com o objetivo de atingir uma meta comum (solucionar um problema global).

2.1 Áreas da Inteligência Artificial Distribuída

Podemos dividir o mundo da IAD em dois enfoques principais **[BON 88][BIT 96]**: a Resolução Distribuída de Problemas (RDP) e Sistemas Multiagentes (SMA). Vários autores citam esta divisão no mundo da IAD, mas temos que observar que a linha que separa as duas áreas de pesquisa ainda é muito tênue, ou seja, não é muito clara a relação entre estas duas áreas.

2.1.1 Resolução Distribuída de Problemas (RDP)

Neste o enfoque principal é o *problema*. Seus objetivos principais são fazer uso da capacidade de processamento oferecida pela tecnologia de redes para atacar problemas naturalmente distribuídos, como a decomposição e a alocação de tarefas em uma rede de computadores.

Os agentes envolvidos em RDP são programados para cooperar, dividir tarefas, comunicar-se de maneira confiável, etc.

Existem uma série de aplicações para RDP, como por exemplo, Interpretação Distribuida - redes de sensores, diagnóstico de falhas de redes de comunicação, Planejamento e Controle Distribuídos - controle de tráfego aéreo, robôs cooperantes, veículos com controle remoto, controle distribuído de processos em manufatura, Sistemas Especialistas Cooperantes - controle de veículos autônomos, negociação entre especialistas, etc **[BIT 96]**.

2.1.2 Sistemas Multiagentes (SMA)

Neste caso, a pesquisa preocupa-se com a *coordenação* do comportamento inteligente entre uma coleção de agentes autônomos inteligentes, nos seguintes aspectos: como eles podem coordenar seus conhecimentos, objetivos, habilidades e planejar juntamente como realizar ações ou solucionar problemas.

Os agentes em um Sistema Multiagente podem compartilhar conhecimentos sobre problemas e soluções, abordando, também , os processos de coordenação entre os agentes. Nesta área os agentes já existem, e o problema é distribuido para que estes o resolvam.

Em Sistemas Multiagentes a tarefa de coordenação pode ser bastante difícil, pois podem existir situações (em sistema abertos) onde não existe possibilidade de um controle global **[BON 88]**.

Quanto a sociedade de agentes podemos dividi-la em dois grandes grupos: Sistema Multiagente aberto, e Sistema Multiagente fechado. No Sistema Multiagente aberto, os agentes podem dinamicamente entrar e sair da sociedade. No caso do Sistema Multiagente fechado, os agentes, a prior, são sempre os mesmos, ou seja, não existe uma migração dinâmica dos agentes para dentro ou para fora da sociedade, estabelecendo o que podemos denominar de uma sociedade fechada. A figura 2.1 exemplifica os dois sistemas:

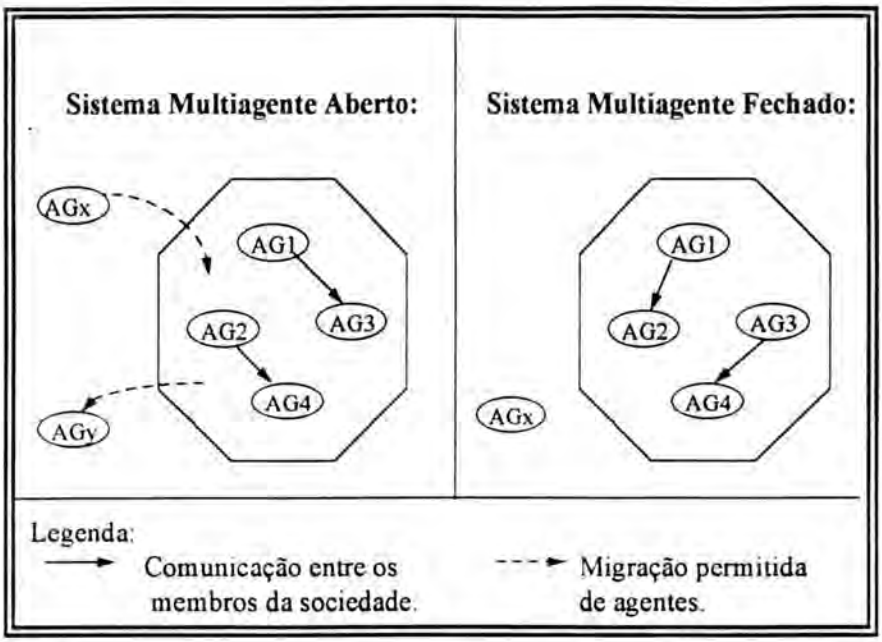

FIGURA 2.1 - Sistema Multiagente aberto ou fechado

Como os agentes devem fazer para atingir juntos uma certa meta?

1. Um agente escolhe uma meta a ser atingida e um plano para ela.

2. Se o agente pode realizar as ações necessárias para o plano, ele as faz, caso contrário, ele ativa seu mecanismo de inferência social para selecionar o(s) outro(s) agente(s) necessários.

3. O agente inicia um protocolo qualquer de interação com os outros agentes para estabelecer uma coalizão com o objetivo de atingir a meta. Se os outros concordarem em colaborar, a coalizão é firmada e os agentes iniciam a execução do plano.

4. Se os outros não quiserem colaborar, o agente tenta localizar outros parceiros.

Se não for possivel localizar outros parceiros, ele considera a meta como inatingível e elabora outra meta.

2.2 Relação da IAD com Sistemas Multiagentes

A Inteligência Artificial Distribuída (IAD) , aborda a atividade de um agente autônomo em um ambiente multiagente. Ainda não existe uma definição uniforme para o termo *agente* dentro da IAD, a seguir serão fornecidas três possíveis definições:

1."*Chama-se de agente uma entidade abstrata que é capaz de agir sobre ela mesma e sobre seu próprio ambiente, que dispõe de uma representação parcial deste ambiente, e que, em um universo Multiagentes, pode comunicar-se com outros agentes, e cujo comportamento é conseqüência de suas observações, de seu conhecimento e das interações com outros agentes*". **[FER 91]**

2. *"Um agente é uma entidade a qual pode-se associar uma identidade única, e que é capaz de realizar cálculos formais. Um agente pode ser considerado como um meio que produz um certo número de ações a partir dos conhecimentos e mecanismos internos que lhe são próprios"*. **[GAS 92]**

3. *"Agente é a palavra utilizada para designar uma entidade "inteligente" e autônoma. A palavra autônoma refere-se ao fato de que cada agente possui sua própria existência, a qual não é dependente da existência de outros agentes"*. **[FRO 95]**

Devemos notar no caso da última definição, que o atributo de inteligência não é necessariamente uma propriedade útil para caracterizar e distinguir agentes de outros tipos de *softwares* que existem no mercado, como por exemplo, os Tutores Inteligentes. Para alguns, o termo agente apenas significa entidades "autônomas" que atuam em nome de alguém ou próprio, para atingir uma determinada meta. Para outros autores, um agente é necessariamente "inteligente" e "autônomo", sendo que a autonomia é um pouco mais detalhada pelos aspectos de requerer ação periódica, execução espontânea, e iniciativa, sendo que os agentes devem ter habilidade de realizar ações independentes e antecipar-se a ações, as quais irão beneficiar o usuário **[PET 96]**.

A IAD baseia-se no comportamento social, com ênfase nas ações e nas interações dos agentes, os quais podem ser desde extremamente simples até extremamente complexos. Um comportamento social inteligente pode surgir no caso de membros "inteligentes" da sociedade (conhecidos como Agentes Cognitivos) ou no caso de membros não "inteligentes" da sociedade (conhecidos como Agentes Reativos) **[FRO 95]**.

2.2.1 Atributos dos Agentes de Software

Esta é uma lista de qualidades que são desejáveis que os *agentes de software*[1] possuam **[ETZ 95]**:

1) Autônomos:

Um agente é habilitado a tomar iniciativa e exercer um grau não trivial de controle sobre suas próprias ações.

2) Orientação de meta:

Um agente aceita pedidos de alto-nível, e é responsável por decidir *como* e *quando* satisfazer o pedido.

3) Colaborativo:

Um agente não obedece comandos cegamente, possuindo a habilidade de modificar pedidos, solicitar esclarecimentos de questões, ou até mesmo recusar certos pedidos.

[1] Nesta dissertação o termo agente de software será utilizado como sinônimo do termo agente.

4) Flexibilidade:

As ações de um agente não seguem um roteiro, estes são habilitados a realizar escolhas dinâmicas de quais ações realizar, e em qual seqüência, em resposta a uma estado do ambiente externo.

5) Inicialização própria:

Diferente dos programas padrões que são diretamente *disparados* pelos usuários, um agente pode perceber modificações em seu ambiente e decidir quando atuar.

6) Continuidade temporal:

Agentes são processos que estão em execução continuamente, não como um instante computacional que mapeia uma única entrada para uma única saída, e termina.

7) Comunicatividade:

Um agente é habilitado a ocupar-se com uma comunicação complexa com outros agentes, incluindo pessoas, com o intuito de obter informações ou conseguir ajuda para comprir seus objetivos.

8) Adaptativo:

Um agente adapta-se automaticamente a mudanças no seu ambiente. Devido a mudanças no ambiente, o agente é levado a tomar decisões concernentes a quem ele vai propor ou aceitar colaboração para atingir suas metas.

9) Móvel:

Um agente está habilitado a transportar-se de uma máquina especifica para outra máquina, e através de diferentes sistemas de arquiteturas, e plataformas.

10) Inferência social:

Um agente deve ser capaz de inferir sobre as atividades de outros agentes.

2.3 Agentes Cognitivos

Os agentes cognitivos são baseados nas organizações sociais humanas, como grupos, hierarquias. Os agentes possuem uma representação explícita do ambiente e dos outros agentes e dispõem de uma memória, sendo graças a esta que estes agentes podem planejar suas ações futuras. Estes podem comunicar-se entre si através da troca de mensagens.

A sociedade de agentes cognitivos têm, em geral, um pequeno número de membros. Estes agentes cognitivos dispõem de uma capacidade de raciocinio sobre uma base de conhecimento e aptidões para tratar de informações diversas, ligadas ao dominio da aplicação e relativas às interações com os outros agentes e com o ambiente.

2.4 Agentes Reativos

Os Agentes Reativos estão baseados em modelos de organizações biológicas ou etológicas[2], como por exemplo as sociedades de formigas. Neste caso podemos afirmar que uma única formiga não apresenta um aspecto muito inteligente, realizando somente algumas tarefas básicas inerentes a sua função, mas o formigueiro como um todo apresenta um comportamento que pode ser dito como inteligente.

Os agentes reativos não possuem memória de seus atos, não planejam suas ações futuras e não se comunicam com outros agentes, cada agente toma conhecimento das ações de outros agentes apenas pelas modificações no ambiente em que está inserido (apenas percebem o ambiente através de estimulo - resposta).

A principal vantagem de um sistema reativo sobre os planejadores tradicionais é que ele opera robustamente em dominios onde uma modelagem completa e precisa é dificil. Os sistemas reativos dispensam qualquer tipo de modelagem e baseiam suas ações diretamente na percepção que têm do mundo.

Uma outra vantagem dos sistemas reativos é que eles são de resposta extremamente rápida, pelo fato já mencionado de basearem-se em estímulo-reposta.

2.5 Problemas em IAD

Quando o trabalho é feito por uma coleção de agentes, de forma coordenada, é importante a questão da divisão e organização deste trabalho: "*Quais agentes devem realizar quais tarefas, e quando?*". Uma distribuição de tarefas entre agentes exige que as mesmas sejam formuladas e descritas. Tarefas que exigem mais conhecimento, do que um agente possui, devem ser decompostas.

---

[2] Etologia refere-se: *2. Biologia*: Parte da ecologia que trata dos hábitos dos animais e da acomodação dos seres vivos às condições do ambiente.

2.5.1 Processo de Decomposição

Em um processo de decomposição, uma tarefa global (principal) que pode, por exemplo, exigir uma grande quantidade de conhecimento e recurso de um agente (o qual talvez não apresente todas estas qualificações) deve ser decomposta em subtarefas menores, as quais exigem menor quantidade de conhecimento e recursos para a sua solução.

Após a decomposição da tarefa inicial em tarefas mais simples, estas devem ser alocadas para os agentes de real competência para sua realização, ou seja, estas devem ser distribuídas para aqueles agentes capazes de executá-las satisfatoriamente.

2.5.2 Coordenação e Cooperação

O maior problema a ser enfrentado no projeto de qualquer sistema de raciocínio distribuído é como as ações de cada agente podem ser coordenadas para que funcionem eficientemente em conjunto. Há várias abordagens que podem ser adotadas aqui, incluindo as seguintes **[RIC 93]**:

- Um agente é responsável. Esse agente-mestre faz um plano e distribui partes do plano para os outros agentes "escravos", que cumprem suas tarefas e comunicam os resultados. Eles também podem comunicar-se com os outros agentes "escravos", caso isto seja necessário, para que os objetivos sejam atingidos.

- Um agente é responsável e este agente decompõe o problema em subproblemas, mas depois ocorre uma negociação para decidir que agentes ficarão responsáveis por quais subtarefas.

- Não há nenhum agente responsável, embora exista um objetivo comum para todos os agentes. Eles têm de cooperar, tanto na formação do plano quanto na sua execução.

- Não há nenhum agente responsável e não há garantia de que haverá um objetivo comum para todos os agentes. Eles podem até competir entre si.

2.6 Vantagens da IAD

A IAD apresenta algumas vantagens sobre a IA comum, vantagens do tipo:

- Modularidade do sistema:

É mais fácil criar e manter uma coleção de módulos quase independentes do que um único módulo básico (grande). Se um sistema inteligente pode ser construído de maneira distribuída, onde cada parte pode ser desenvolvida separadamente por um especialista

em um tipo de conhecimento ou domínio particular, um sistema de inteligência distribuída pode ser utilizado **[BON 88]**.

- Eficiência ou velocidade:

Nem todos os conhecimentos são necessário para todas as tarefas. Quando o modularizamos, ganhamos a possibilidade de focalizar os esforços do sistema na solução do problema do modo mais compensador possível. Concorrência pode aumentar a velocidade de computação e raciocinio.

- Raciocínio heterogêneo:

As técnicas para solução de um problema e os formalismos de representação do conhecimento que mais se prestam para trabalhar em uma parte de um problema podem não ser os melhores para trabalhar em uma outra parte **[BON 88]**.

- Múltiplas perspectivas:

O conhecimento exigido para solucionar um problema pode não estar na cabeça de uma única pessoa. É muito difícil juntar diversas pessoas para criar uma única base de conhecimento coerente, e às vezes essa reunião é até mesmo impossível, pois os modelos que cada uma delas possui sobre o domínio em questão são inconsistentes.

- Isolamento / autonomia:

Para proteção e para controle, partes do sistema podem ser separadas e isoladas uma da outra.

- Problemas distribuídos:

Alguns problemas são inerentemente distribuídos. Alguns problemas são melhor descritos como uma coleção de agentes separados; existe uma melhor forma para o problema ou domínio, porque os elementos são naturalmente distribuídos.

- Confiabilidade:

Se um problema está distribuído entre agentes de sistemas diferentes, a solução pode continuar mesmo que um dos sistemas apresente falhas **[RIC 93]**.

Possibilita uma redução no tamanho do domínio de entrada, o que significa uma menor complexidade computacional, tornando possível que os componentes sejam mais simples e confiáveis **[JEN 93]**.

- Simplificação:

A própria atividade de decompor um problema em problemas mais simples, ocasiona uma simplificação do problema e maior facilidade de compreensão do mesmo **[JEN 93]**.

3 Exemplos de aplicações agentes

Atualmente, as aplicações ditas **agentes inteligentes** têm sido alvo de grandes estudos, e inúmeros projetos e *softwares* têm surgido nesta área. Grandes corporações como a IBM®, tem investido amplamente no desenvolvimento deste tipo de *software*. Neste capítulo da dissertação, procura-se apresentar alguns agentes, sua concepção, sua utilização e finalidade. Serão apresentados agentes simples, como as *spiders* que buscam informações ao longo da Internet, e alguns agentes mais elaborados, como os voltados a aplicações industriais e ensino.

3.1 Agentes auxiliares na Rede

Na tabela abaixo, são apresentados algumas propostas de agentes, que servem para auxiliar o usuário com atividades junto a Internet.

TABELA 3.1 - Agentes Auxiliares

Nome do *software*	Características
Ahoy!	Trata-se de um serviço isento de custos oferecido na Internet, foi desenvolvido pela Universidade de Washington. Este serviço torna mais rápido o encontro de páginas pessoais no Web. Este utiliza um agente simples para criar uma ampla lista de páginas candidatas e emprega várias técnicas para escolher as páginas mais promissoras, deixando para o usuário um número viável de escolhas.
Webbie da IBM	O WBI (*WEB Browser Intelligence*, chamado de Webbie) é um agente que atua como um *proxy*[3] *WWW* entre um *browser* e o resto do Web. O WBI pode relembrar os endereços que foram consultados na Internet, e o que foi encontrado nestes endereços, e pode auxiliar a encontrar qualquer página já visitada. Este agente pode avisar o usuário, antes que este acesse uma página, se o *site* não está disponível ou se o acesso está lento,

[3] Proxy: é um programa tradutor, utilizado para permitir a integração entre redes que utilizam diferentes protocolos de comunicação.

Nome do *software*	Características
	auxiliando desta maneira na produtividade do trabalho do usuário. O WBI, também, pode auxiliar o usuário, informando se o conteúdo de uma página de interesse, já acessada, teve seu conteúdo alterado.
IBM Alter Ego	Trata-se de um agente baseado em regras que podem manipular *mails* e outras tarefas rotineiras do usuário. O agente Alter Ego pode responder e reagir a eventos assim como receber e enviar *mails* em momentos pré-determinados do dia. O Alter Ego apresenta vantagens do tipo **[JAN 96]**: - Editor de regras: simplifica a ação do usuário fornecendo *templates* que indicam como os *mails* podem ser manipulados. - Implementação centrada à rede: caso o usuário do Alter Ego estiver conectado a uma rede, ele servirá ao usuário durante 24 horas, sete dias por semana. *Mails* podem ser manipulados imediatamente. - Máquina de Inferência: uma máquina de Inferência com encadeamento para frente suporta regras que podem classificar mails. - *Log* de atividade: o Alter Ego retém um *log* das atividades que ele realiza para o usuário. O usuário pode visualizar o *log* e verificar quais instruções ele deseja descartar.

3.1.1 Agentes de busca na Internet

O WWW é descentralizado, dinâmico, e diversificado; a navegação algumas vezes torna-se difícil e encontrar informações pode ser um desafio. A razão deste desafio é o fato do usuário do WWW estar navegando para encontrar algo através de seqüências de *links* de hipertextos. Como o Web continua a crescer, os usuários precisam passar por muitos *links* para encontrar o que realmente estão desejando. Sendo assim, os usuários passaram a depender de algumas ferramentas de busca, para auxiliar na filtragem de informações do Web.

Construção de Robôs (agentes) para o Web:

Existe um número muito grande de ferramentas de busca disponíveis no próprio Web, cada uma destas utiliza um método diversificado para construir seus próprios bancos de dados.

Agentes para Web (robôs), também são conhecidos como aranhas (*spiders*) ou andarilhos (*wanderers*), são agentes de *software* (alguns bastante simples) que atravessam o espaço de informações do *world wide Web* seguindo *links* de hipertextos e buscando documentos Web através do protocolo padrão HTTP. Agentes Web requerem que as decisões não sejam feitas de maneira centralizada e que não haja a participação de um administrador individual de um *site* Web **[CHE 95]**.

Um exemplo de Agentes para Web pode ser observado na maneira em que é feito o cadastro de uma home-page no *site* do aplicativo Alta Vista[4]. No momento em que se deseja cadastrar uma página o agente do Alta Vista somente solicita que se informe o endereço correto da página, se o endereço estiver correto, este buscará sozinho informações sobre a página a ser cadastrada, através de um método de varredura dos *tags* (marcas internas a uma página HTML). Passado um período de 24 horas, esta página passará a fazer parte do banco de dados de páginas indexadas pelo agente do Alta Vista.

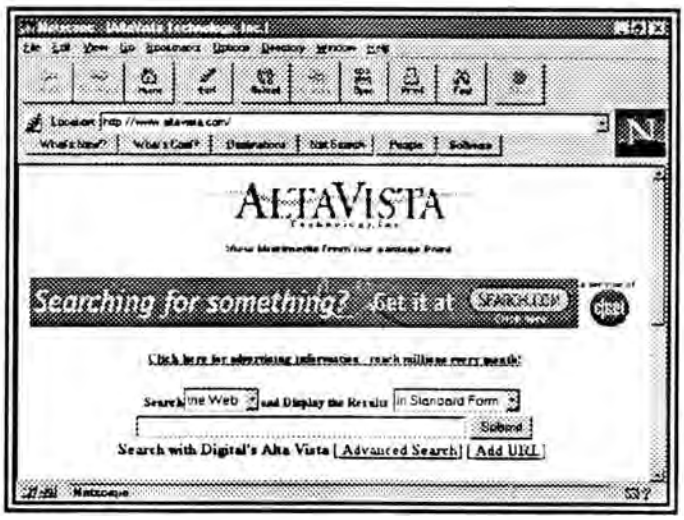

FIGURA 3.1 - Página principal do *site* referente ao Alta Vista

A figura 3.1, apresenta a página de acesso ao *site* do Alta Vista. A figura 3.2, abaixo, apresenta a página do Alta Vista que propicia que seja cadastrada uma nova URL, no banco de dados do Alta Vista.

[4] Alta Vista: é um *site*, que disponibiliza uma home-page na Internet, e tem como objetivo auxiliar na busca de assuntos desejados pelo usuário, esta também propicia que se cadastre uma página em seu banco de dados. http://www.altavista.com

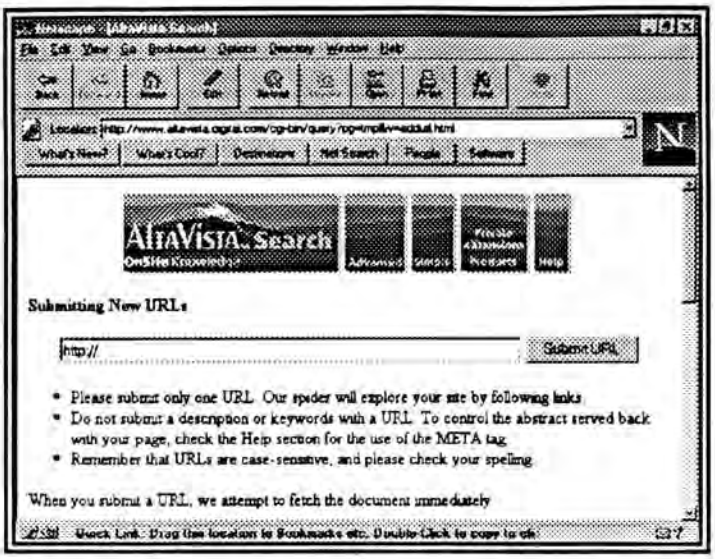

FIGURA 3.2 - Página do Alta Vista para cadastro de URL

***Spiders* para indexação do Web:**

Existe uma grande variedade de *spiders* que se "movimentam" pelo Web coletando informações sobre o que encontram. Esta *spiders* fazem uso dos *hiperlinks* colocados nas páginas Web para deslocarem-se de maneira automática pelo Web, movimentando-se de um documento HTML para outro através dos endereços colocados nos *tags* referentes a *URL*. As informações coletadas pelas *spiders* podem ser utilizadas para diferentes propósitos, como por exemplo construir um index para auxiliar o usuário com a procura de informações através do uso de palavras chaves.

Uma das primeiras *spiders* que foi amplamente utilizada foi projetada por Oliver McBryan e foi diponibilizada em março de 1994. Esta foi a precursora de uma série de *spiders* que existem hoje na Internet. Ela construia um index de titulos e URLs a partir de uma coleção de mais de 100.000 documentos, além disto, fornecia para o usuário uma interface de busca para seus bancos de dados **[CHE 95]**.

A seguir serão apresentados um resumo sobre algumas *spiders* existentes.

3.1.2 WEBCRAWLER

O WEBCRAWLER **[CHE 95]** é um projeto desenvolvido por Brian Pinkerton da Universidade de Washington em Seattle. Ele consiste em uma ferramenta para busca de informações na Internet, e fornece um amplo caminho para encontrar informações através da manutenção de um índex do Web. Este contém informações de documentos oriundos de 6.000 servidores.

Ele responde a uma média de 6.000 consultas por dia e é atualizado semanalmente. O WEBCRAWLER é capaz de realizar as seguintes funções:

1. Construir indexes do Web;

2. Navegar de maneira automática pelo Web.

A arquitetura do WEBCRAWLER é constituída por alguns componentes:

1. **Mecanismo de busca:** este dirige as atividades do WEBCRAWLER e é responsável por decidir quais os novos documentos que serão explorados e pela recuperação destes.

2. **Os agentes:** Estes são responsáveis por buscar documentos da rede pela indicação do mecanismo de busca.

3. **O Banco de dados:** Este armazena algumas informações sobre os documentos.

4. **O servidor de perguntas:** Este implementa um serviço de perguntas fornecidas pela Internet.

A máquina de busca do WEBCRAWLER:

A máquina de busca do WEBCRAWLER determina quais documentos e quais tipos de documentos devem ser visitados. Os arquivos não indexáveis, como é o caso das figuras, arquivos de som, PostScript, ou arquivos binários, não são verificados. Quando o WEBCRAWLER é executado no modo indexador e quando é executado no modo de busca em tempo real, a sua máquina de busca utiliza estratégias diferentes para descobrir os documentos. No modo indexador, o objetivo é simplesmente construir um index do Web, tanto quanto for possível em um espaço limitado de armazenamento.

No modo de busca em tempo real, onde a meta é encontrar documentos que sejam o mais similar possível as perguntas do usuário, o WEBCRAWLER utiliza um algoritmo diferente de busca.

Os agentes do WEBCRAWLER:

Os agentes são invocados pelo mecanismo de busca com o objetivo de reaver documentos do Web. No WEBCRAWLER os agentes são executados como processos em separado, e o WEBCRAWLER emprega até 15 agentes executando em paralelo.

Para cada novo documento do Web que deve ser recuperado, o mecanismo de busca encontra um agente livre, e pede para o agente recuperar um documento representado por uma URL. O agente neste caso pode tanto responder ao mecanismo de busca com um objeto contendo todo o conteúdo do documento ou uma explicação do porque o documento não pode ser recuperado. Depois do agente ter respondido, este fica livre novamente e pode ser escalado para um novo trabalho.

Banco de dados do WEBCRAWLER:

O banco de dados do WEBCRAWLER contém índex de texto completo. O banco de dados é armazenado em disco e aberto para a adição de novos documentos.

Servidor de perguntas:

O servidor de perguntas implementa o serviço de procura do WEBCRAWLER via um formulário em HTML. Esta interface simples é poderosa e pode encontrar e relatar um documento com facilidade. O seu funcionamento é simples, o usuário simplesmente deve fornecer as palavras chaves para busca, então títulos, e URLs de documentos relacionados com as palavras chaves são retornadas e apresentadas para o usuário ordenadas por critério de relevância.

3.1.3 LYCOS

O projeto Lycos **[CHE 95]** foi encabeçado pelo Dr. Michael Mauldin da Universidade de Carnegie Mellon. Este auxilia o usuário a localizar documentos Web que contenham palavras chaves especificas fornecidas pelo usuário. Devido a abrangência de seu bando de dados , o Lycos rapidamente tornou-se muito popular junto aos usuários que necessitavam fazer buscas especificas no Web. Por meados de Julho de 1995, o Lycos acumulava as seguintes quantidades de dados:

- 5.077.834 URLs diferentes;
- 1.777.750 documentos (perfazendo um total de 8.703.484.067 bytes);
- 3.900.084 URLs inexploradas com suas descrições;
- 1.834.323.446 bytes de sumários;
- 1.078.127.917 bytes de indexes.

O banco de dados do Lycos tem crescido rapidamente, de 634.000 referências em agosto de 1994 para mais de 5.6 milhões em agosto de 1995. Desta maneira o Lycos oferece um amplo banco de dados, capaz de localizar documentos através de qualquer combinação de palavras fornecida pelo usuário.

A Interface do Lycos é apresentada na figura 3.3 abaixo:

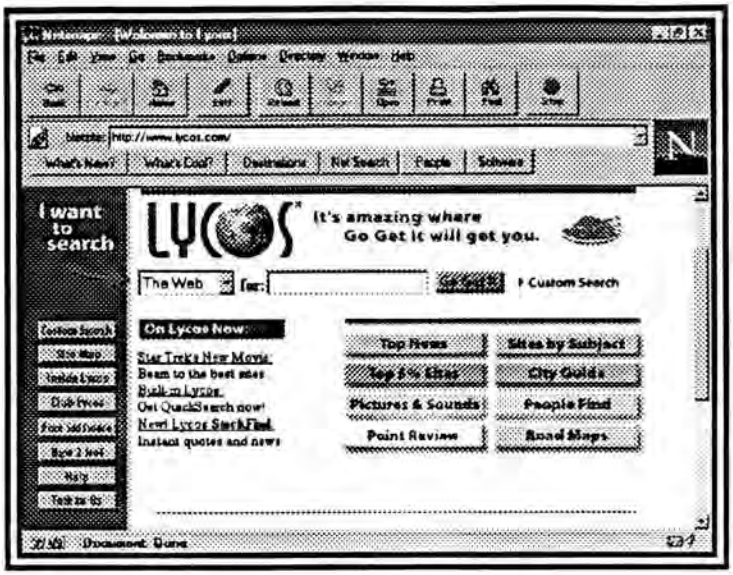

FIGURA 3.3 - Interface da página do Lycos

Espaço de busca do Lycos:

O Lycos é capaz de buscar documentos nos seguintes espaços de busca: HTTP, FTP e GOPHER[5].

Index do Lycos:

Para a reduzir a quantidade de informações que necessitam ser armazenadas, o Lycos extrai os seguintes *pedaços* de informações de cada documento que necessita ser armazenado:

- Título;
- Cabeçalho e sub cabeçalho;
- As primeiras 20 linhas do texto;
- As 100 palavras mais importantes;
- O tamanho em bytes do documento e número total de palavras.

As 100 palavras mais importantes são obtidas usando um algoritmo de pesagem específico, o qual considera o posicionamento das palavras dentro do texto e a freqüência das mesma, além de uma série de outros fatores.

[5] Gopher é um aplicativo cliente/servidor, disponível para consultas na Internet.

Como o Lycos movimenta-se no Web:

O componente *andarilho* do Lycos, é originalmente derivado de um programa chamado *Longlegs*, escrito por John Leavitt e Eric Nyberg da Universidade de Carnegie Mellon.

O Lycos utiliza uma inovação, um esquema probabilístico para saltar de um *site* para outro no WebSpace. Os passos básicos do algoritmo são os seguintes:

1. Quando um novo recurso de URL é trazido, o Lycos examina seu conteúdo para uma nova referência de URL, a qual é adicionada a uma fila interna.

2. Para escolher a próxima URL a ser explorada, o Lycos faz uma escolha randômica entre uma referência HTTP, GOPHER, e FTP na fila baseada em preferências.

3.2 Agentes na Indústria: ARCHON

Uma proposta, mais elaborada, para a utilização de agentes, é o uso deste voltado para aplicações industriais. Neste sentido existe um grande projeto na Europa, denominado de ARCHON[tm]. Neste projeto trabalha-se a nível de comunidades de agentes, que interagem entre si.

ARCHON[tm] (*Architecture for Cooperative Heterogeneous on-line Systems*) constitui-se em um sistema de Inteligência Artificial Distribuida para aplicações Industriais. O ARCHON[tm] foi projetado para uma arquitetura de propósito geral, e sua metodologia tem sido usada para suportar o desenvolvimento de sistemas de Inteligência Artificial distribuída, em diversos domínios da indústria. Alguns exemplos de aplicações para as quais este sistema foi satisfatoriamente aplicado incluem: fornecimento e distribuição de eletricidade, controle de fornos de cimento, controle de aceleradores e controle de robôs, entre outros **[JAN 96]**.

As entidades responsáveis pela solução de problemas no ARCHON[tm] são os agentes, os quais possuem habilidade de controlar suas próprias resoluções de problemas e de interagir com outros membros da comunidade. As interações envolvem comunicações e cooperação entre agentes a fim de engrandecer a solução de problemas.

3.3 Agentes no processo de ensino-aprendizagem

Uma área que está mostrando-se muito rica para a utilização de agentes, como auxiliares, é a área de ensino-aprendizagem. Os agentes nesta área são utilizados como simples *softwares*, os quais propõem o uso de agentes para o auxilio na atividade educativa, até projetos mais robustos, que propõem utilizar os agentes e a Internet como meios para o ensino a distância. A seguir será descrito um exemplo de *software* para ensino de crianças.

3.3.1 KIDSIM (Ambiente de simulação para crianças)

O KIDSIM (ambiente de simulação para crianças) constitui-se em um *kit* de ferramentas que permitem que as crianças construam simulações simbólicas. Crianças podem modificar a programação de objetos de simulação já existentes e definir novos. As simulações do KIDSIM consiste em **[JAN 96]**:

- um tabuleiro de jogo dividido em espaços discretos como um tabuleiro de xadrez;

- um relógio no qual o tempo é dividido em espaços discretos;

- um ou mais objetos de simulação;

- uma caixa de cópia, que é a fonte para novos objetos de simulação;

- um editor de regras, onde regras são definidas e editadas; etc..

O tabuleiro de jogo apresenta a simulação de um micromundo. É neste ambiente onde os objetos de simulação interagem uns com os outros. Sendo dividido em quadrados definidos, fica fácil para as crianças comunicarem suas intenções ao computador. O relógio começa e termina uma simulação em andamento. Dividindo o tempo em porções discretas é fácil para as crianças controlarem suas simulações. O relógio fornece o controle sobre o tempo e a capacidade de rodar em modo reverso. Se algo de errado acontece, pode-se fazer um *backup* do relógio até este ponto.

A caixa de cópia é um depósito para objetos de simulação que fazem cópias automáticas de coisas dentro dele. Se uma criança tira um objeto para fora da caixa de cópia, o sistema faz uma cópia do original e o devolve para a caixa. Isto fornece uma fonte infinita de novos objetos.

Agentes no KIDSIM:

No KIDSIM, os objetos ativos nas simulações são agentes. Durante cada fatia de tempo, os agentes se movem ao redor do tabuleiro, interagindo uns com os outros. Metaforicamente, eles são como personagens de um micromundo. Os agentes Kidsim possuem três atributos:

1. *Aparência:* a criança pode desenhar suas próprias aparências para os agentes, através do pensamento metafórico.

2. *Propriedades:* as crianças podem definir seus próprios dados e características para os agentes. Por exemplo: características para um macaco poderiam ser: nome, idade, peso, altura, sexo, etc.

3. *Regras:* as crianças podem definir as regras de comportamento dos agentes. O conjunto de regras para um agente constituem seu programa.

Os agentes do KIDSIM podem ser vistos como objetos no sentido da programação orientada à objetos. Eles possuem estado (propriedades), comportamento (regras), e uma aparência **[KID 96]**.

4 A Linguagem Java

A linguagem Java é uma linguagem projetada pela empresa ***Sun Microsystems*** ® e tem como característica principal o fato de ser uma linguagem propícia para o desenvolvimento de programas para o ambiente da Internet. Hoje na consulta de várias *home-pages* pelo mundo, pode-se observar a aplicação em massa desta linguagem no desenvolvimento de vários aplicativos que são incorporados às páginas, o que vem demonstrar que esta linguagem está tornando-se a plataforma de desenvolvimento de aplicações para a Internet.

A linguagem apresenta várias peculiaridades, que são responsáveis pelo seu sucesso. Entre estas pode-se citar o fato de Java ser uma linguagem orientada a objeto ou OOP (Object Oriented Programming). Esta não é uma novidade entre as linguagens existentes, mas é uma característica muito útil para facilitar a atividade de programação.

A história da linguagem Java começou no início dos anos 90, numa época em que muito se trabalhou pelo conceito de casa inteligente. O projeto original da SUN Microsystems - chamado OaK- era criar uma linguagem leve, com códigos pequenos e que servisse para controlar produtos eletrônicos de consumo. Em 1994, a Sun percebeu que a Internet, mais especificamente a WWW, teria espaço para essa linguagem, caracterizada pela **independência** de plataformas e sistemas operacionais e concebida para redes **[INF 96]**.

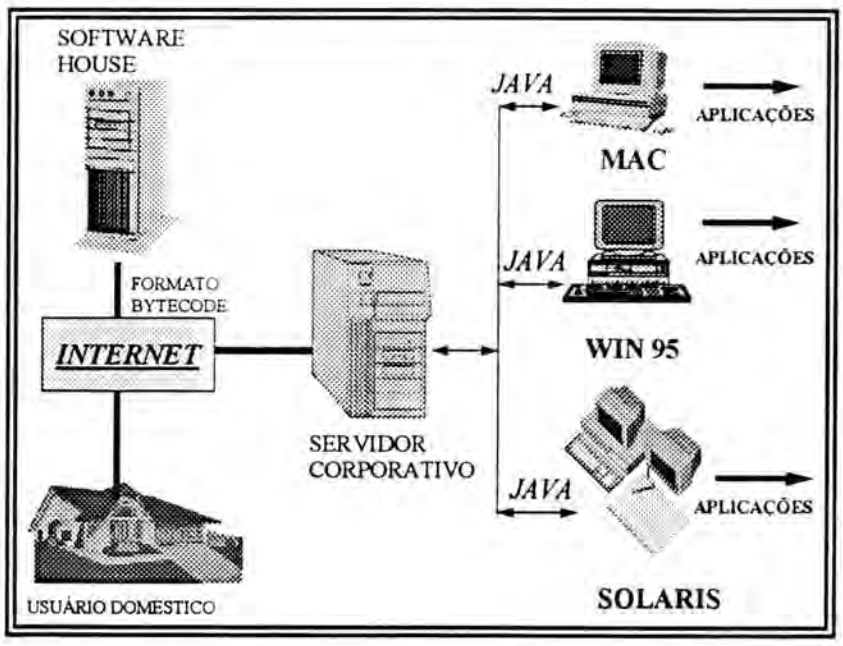

FIGURA 4.1 - A Linguagem Java

4.1 Características da Linguagem

A linguagem Java surge como o mais novo sucesso entre as linguagens de programação e como uma linguagem muito promissora no que tange ao trabalho de programação junto a Internet. Utilizando *applets*[6] desenvolvidas em Java pode-se incorporar às páginas do Web, por exemplo, animações e sons.

Claro que o uso de Java não se limita a realizar trabalhos somente de movimento e sons. Java propõe-se a ser muito mais ampla e robusta, propiciando inclusive a ligação de seus programas a bancos de dados.

Java é uma linguagem de alto nível, similar a C e C++, Pascal e Modula-3. Algumas caracteristicas desta linguagem tentam justificar o porque deste sucesso, e o porque de preferir-se programar com uma nova linguagem e não utilizar outras já consagradas como a linguagem C++ (por exemplo). Algumas caracteristicas que podem ser citadas são: linguagem multiplataforma (independente de arquitetura), simples, utiliza tipos de dados estáticos, multilinha (*multithreaded*), coleta de lixo automático da memória (*garbage collector*), orientada a objetos , segura, compilada e possui tratamento de exceções.

4.1.1 Simplicidade

Java é muito semelhante a linguagem C++, mas apresenta vários pontos que tornam a programação com seu uso mais simples do que a utilizando C++. Por exemplo, não há conversão implicita de tipos, não há aritmética de ponteiros, não existe uniões e nem estruturas.

4.1.2 Orientada a Objetos

O código Java é organizado em classes. Cada classe define um conjunto de métodos que formam o comportamento de um objeto. Uma classe pode herdar o comportamento de uma outra classe. Na raiz da hierarquia está sempre a classe Objeto (object). A linguagem Java dá suporte a uma hierarquia de classes de herança simples. Isto significa que cada classe só pode herdar de uma classe por vez. Java, também, dá suporte a construção de interfaces, através do uso de suas classes abstratas. Isto permite aos programadores definir métodos para interfaces sem que tenham de preocupar-se, imediatamente, como os tais métodos serão implementados.

---

[6] *Applets* são pequenos programas escritos em Java que são embutidos em páginas Web para produzir algum tipo de efeito.

4.1.3 Compilada

Quando um programa em Java é compilado este é transformado no que chama-se de *bytecodes*. *Bytecodes* são bastante similares a instruções de máquina, de forma que programas em Java podem ser bastante eficientes. Entretanto, *bytecodes* não são específicos para um modelo de computador em particular, de forma que programas em Java podem ser executados em vários tipos de computadores distintos sem que seja necessário um processo de recompilação.

4.1.4 Multiplataforma

Como os programas em Java são compilados para *bytecodes*, eles podem rodar em qualquer plataforma que dê suporte a Java. Como está se falando em uma linguagem voltada para programação na Internet, deve-se observar o quanto esta característica de independência de arquitetura é importante, pois na Internet, não se tem como saber qual o computador que o usuário está utilizando para visualizar uma página.

O funcionamento da linguagem Java para garantir esta independência de arquitetura é o seguinte:

1. Após ter-se compilado o programa e gerado o arquivo *.class* (que possui os *bytecodes*), podemos inserir este arquivo em uma página HTML através do uso de um *tag* específico (<*applet* code=.....>). Quando o usuário acessa esta página o arquivo *.class* é levado junto para a memória do microcomputador do usuário, onde é interpretado pelo *browser*[7] que o usuário está utilizando. Pode-se dizer que quando o programa em Java é compilado, o arquivo resultante possui cerca de 80% do código final necessário para sua execução, quando este é interpretado, é somado ao seu código os 20% restantes, que referem-se as características da plataforma que está sendo utilizada. A Figura 4.2 demonstra a execução de um programa em Java:

[7] Browser: são categorias de programas que permitem visualizar um documento feito em um certo padrão, no caso HTML (hipertext markup language). Atualmente os browsers têm tornado-se complexos devido a quantidade de padrões existentes (ex: imagens .gif .jpg. etc).

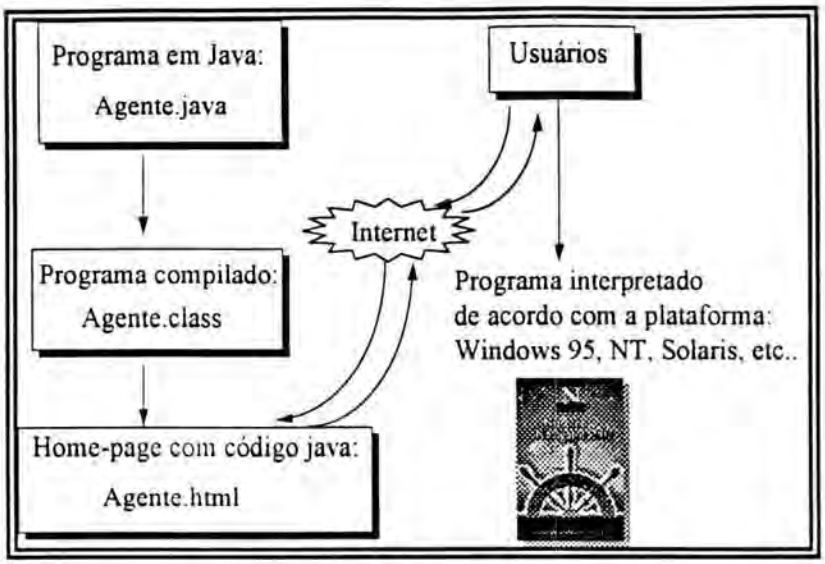

FIGURA 4.2 - Programação em Java

4.1.5 Multilinha (*multithread*)

Um programa em Java pode ter mais de uma linha (*thread*) de execução. Por exemplo, ele poderia executar algum cálculo extenso em uma linha de execução (processo em andamento), enquanto outras linhas de execução interagem com o usuário. Dessa, forma um usuário não precisa parar de trabalhar aguardando que os programas Java completem operações extensas.

4.1.6 Coletor de lixo (*Garbage Collector*)

Quando trabalha-se com C e C++ existe a preocupação com a porção de memória utilizada e na alocação e desalocação destes recursos da memória.

Em Java, os programadores não precisam se preocupar com o gerenciamento da memória. O sistema Java possui um programa embutido chamado *garbage collector*, que varre a memória e libera o que não esteja mais sendo referenciado (utilizado).

4.1.7 Exceções (*exception*)

Para quem programa, um dos principais problemas no uso dos computadores é exatamente o momento em que o programa *trava* (impede a continuidade na execução de um ou mais programas) a máquina. Os programas feitos em Java não *travam* a máquina, pois o sistema Java verifica cuidadosamente cada acesso à memória e garante que ele é permitido e não causará nenhum problema. Se algo inesperado ocorrer na execução de um programa em Java uma exceção é lançada. O programa encontrará estas exceções e tentará tratá-las.

4.1.8 Segurança

Como as *applets* Java são automaticamente descarregadas para o computador e executadas ao visualizar páginas com recursos Java, poder-se-ia pensar que existem riscos de um vírus infectar o computador. Isto não ocorre. Nenhuma *applet* Java é capaz de roubar informações ou danificar o computador, ou dados, de nenhuma forma **[HOF 96]**.

A razão pela qual as *applets* Java são seguras é que programas em Java são compilados em instruções *bytecodes*, que podem ser verificadas em busca de violações de segurança. Isto é possível porque os *bytecodes* Java possuem informações adicionais que são usadas para verificar se o programa original não foi corrompido.

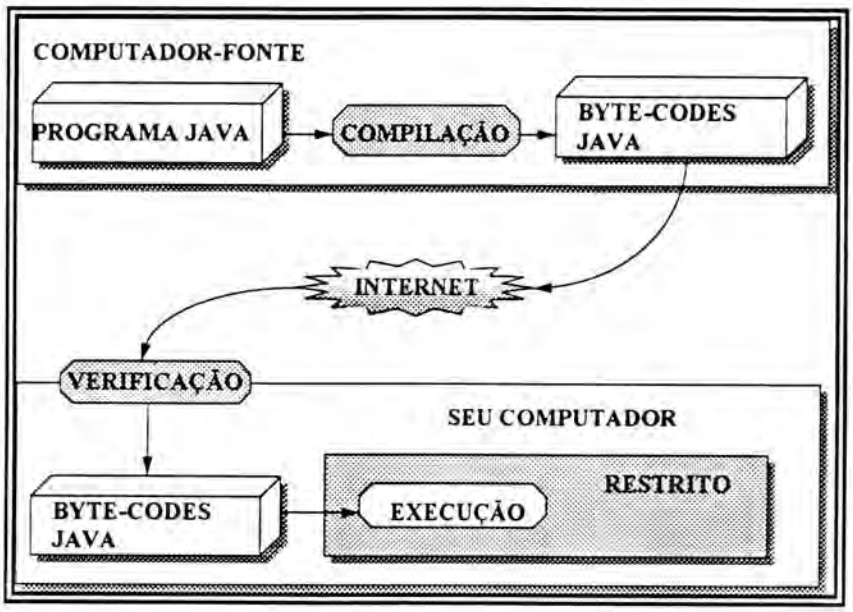

FIGURA 4.3 - Processo de Verificação dos bytecodes da Linguagem Java

Uma vez que a *applet* tenha sido verificada, ela será executada em um ambiente restrito. As *applets* não podem executar certas funções perigosas neste ambiente a não ser que sejam autorizadas a fazê-lo. Por causa do processo de verificação, a *applet* não é capaz de "fugir" deste ambiente restrito. Uma vez que os *bytecodes* Java tenham sido verificados, o interpretador Java pode transformar os nomes de funções em endereços, exatamente como programas tradicionais. Isto quer dizer que a linguagem Java pode ser eficiente e segura ao mesmo tempo.

4.2 Programando Agentes utilizando a Linguagem Java

Para chegar-se ao objetivo final desta dissertação, foi proposta a implementação de alguns agentes assistentes.

Estes agentes foram programados utilizando a própria Linguagem Java, aproveitando-se todas as características e facilidades listadas no item anterior. Os agentes foram escritos para serem programas *standalone*, e não *applets* em páginas HTML. Abaixo serão descritas detalhadamente as propostas dos dois agentes implementados.

4.2.1 Agente para Atualização de Arquivos

O Agente para Atualização de arquivos tem como propósito principal, o de liberar o usuário da tarefa de ficar sempre reportando-se a um *site* especifico para verificar se o arquivo já adquirido, apresenta uma versão mais atualizada . O funcionamento deste agente é exemplificado na figura 4.4.

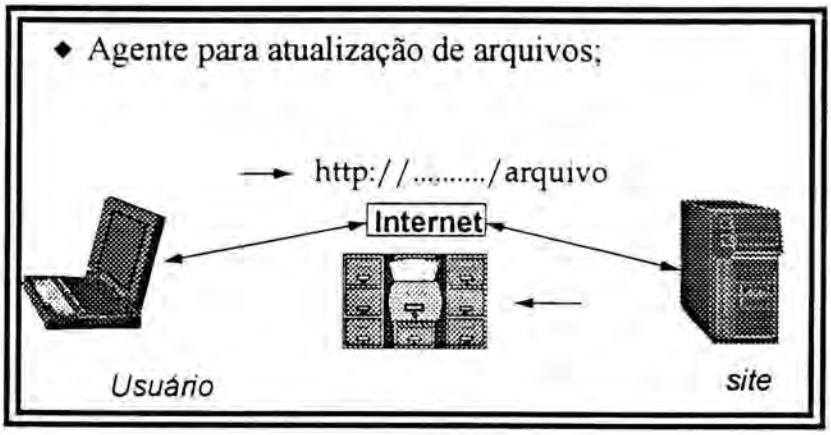

FIGURA 4.4 - Especificação do Agente de Atualização

O funcionamento do agente assume as seguintes características:

1) O usuário copia para o seu disco rigido algum arquivo especifico de seu interesse, por exemplo, um tutorial que não se encontra totalmente acabado.

2) O usuário tendo noção de que este tutorial está sofrendo atualizações constantes no *site* de origem, solicita que o seu Agente de Atualização fique monitorando o *site*.

3) O agente, de posse do endereço em que se encontra o tutorial, diariamente conecta-se a este *site* e verifica se a data do tutorial constante do *site* é mais moderna do que a do usuário, ou se os tamanhos dos arquivos estão discordando.

4) Caso o agente *conclua* que o tutorial do *site* é mais moderno, este deve de maneira autônoma e automática copiar uma nova versão para a máquina do usuário.

5) Após a cópia estar pronta na máquina do usuário, o agente deve enviar um e-mail para o usuário informando que houve a respectiva atualização.

6) A menos que o usuário informe para o agente monitorar outro local, este permanecerá monitorando o *site* do tutorial indefinidamente.

O agente de atualização é um agente único, ou seja, não faz parte de uma sociedade. Ele possui uma função simples e é capaz de cumpri-la na íntegra sem a necessidade de auxílio de outros agentes. A listagem completa do programa fonte deste agente encontra-se no Anexo 1 desta dissertação.

4.2.2 Agente para Agendamento

O agente para agendamento, apresenta uma proposta um pouco mais complexa. Ele tem como objetivo permitir que um grupo de trabalho marque encontro para uma reunião sem que haja a necessidade da interação com o ser humano, mas apenas ocorra a interação entre os agentes pessoais deste grupo de trabalho.

O funcionamento do agente é explicado abaixo e representado na figura 4.5:

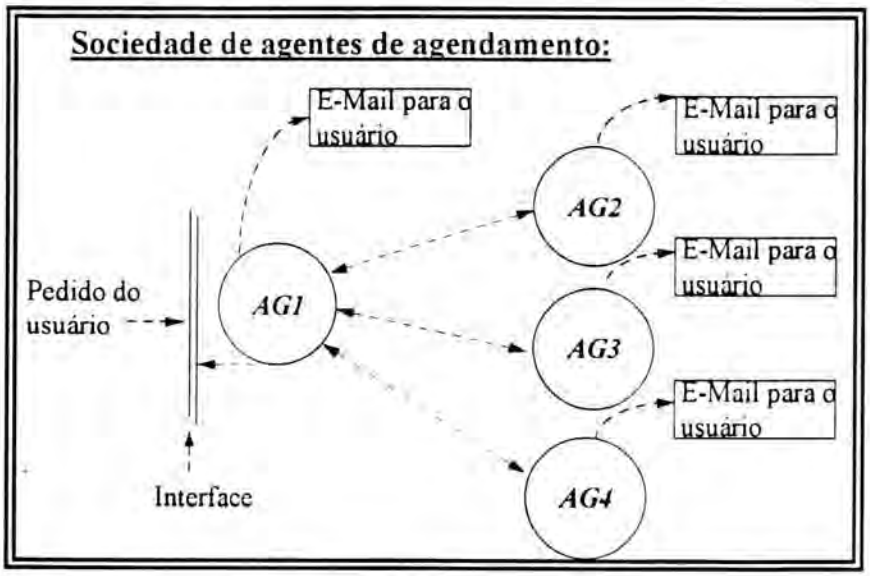

FIGURA 4.5 - Funcionamento do Agente de Agendamento

1) Vamos, inicialmente, supor que um dos componentes do grupo de trabalho está interessado em marcar uma reunião no dia "x" com o seu grupo. Sendo assim, ele pede para que seu agente de agendamento (AG1) faça a marcação da reunião.

2) O agente de agendamento de posse da data desejada, entra em contato com todos os agentes dos outros componentes do grupo (AG2, AG3, AG4) e pede que estes informem a possibilidade de marcar uma reunião para o dia "x".

3) Todos os agentes verificam suas agendas locais e retornam uma resposta para o agente solicitante (AG1).

4) Caso todos os agentes (AG2, AG3, AG4) respondam que o dia "x" está disponível para marcar reunião, o agente solicitante (AG1) deve dar um retorno para todos os agentes confirmando a marcação da reunião. Caso não seja

possível marcar por algum motivo, o agente solicitante (AG1) deve dar um retorno para todos informando que a marcação não será feita e terminar as negociações. Nas duas hipóteses acima o agente AG1 deve mandar um e-mail para o seu proprietário, informando que marcou a reunião pedida ou não, e neste caso pedir para o usuário fornecer uma nova data.

5) No caso da reunião ter sido marcada com todo o grupo de trabalho, cada agente que compõe esta sociedade deve mandar um e-mail para o seu usuário informando que a agenda foi alterada com a reunião do dia "x".

Implementação dos agentes de agendamento:

Para a implementação do agente de agendamento foi proposta a seguinte estrutura, apresentada na figura 4.6.

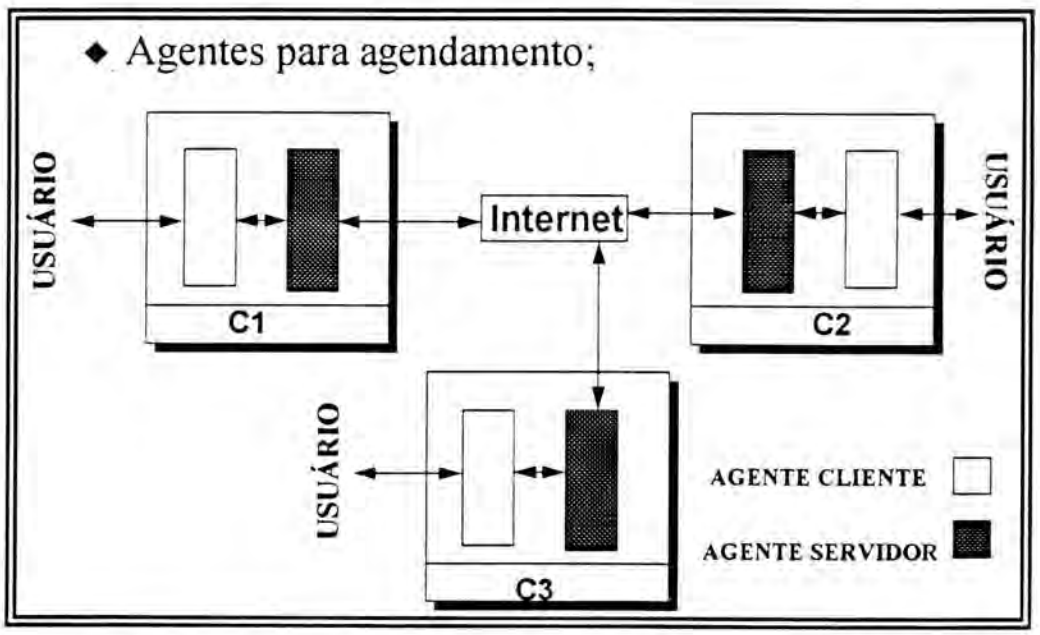

FIGURA 4.6 - Estrutura para Implementação dos Agentes de Agendamento

Na implementação, utilizando a linguagem Java, foram criados dois processos principais, um denominado de agente cliente e outro de agente servidor. Cada usuário teria em seu computador (c1, c2 e c3, no exemplo acima) estas duas estruturas básicas.

O agente cliente:

Este processo seria executado somente quando o usuário necessitasse marcar uma reunião com o seu grupo de trabalho. Este agente terá a função de apresentar alguma interface para que o usuário informe a data desejada para a reunião. Após o usuário informar a data desejada este agente informará a data para os agentes servidores do grupo e pedirá para que estes verifiquem a possibilidade de marcar a reunião. A listagem do programa fonte do agente cliente encontra-se no Anexo 2.

O agente servidor:

Este é um processo que está sempre em estado de escuta (*listening*) no computador, ou seja, ele está sempre em execução e observando as informações que chegam através de uma porta lógica X.

Quando o agente servidor recebe o pedido do agente cliente para que seja verificada uma determinada data, ele consulta a agenda e informa se a data está ocupada ou desocupada.

O agente cliente após receber a resposta de todos os servidores, informa se a reunião pode ser marcada. Caso todos tenham respondido que podem realizar a reunião na data pedida, a reunião é marcada.

No caso da reunião ser marcada, todos os agentes servidores da sociedade, devem mandar um e-mail para seu dono informando que a agenda foi alterada. Caso a reunião não seja marcada, somente o agente solicitante deve mandar um e-mail informando para seu proprietário que a data almejada não está disponível.

A listagem do programa correspondente ao agente servidor encontra-se em sua totalidade no Anexo 3. Aqui cabe ser feita uma ressalva, as implementações dos agentes cliente e servidor constantes nos seus respectivos anexos, não apresentam toda a riqueza de detalhes apresentados nesta proposta de implementação (principalmente no que tange ao aspecto de *strings* de comunicação), pois durante a implementação o que realmente interessava era verificar como estabelecer a devida comunicação entre os agentes.

4.2.3 Especificação de um Protocolo Básico de Interação

Para que haja comunicação entre os agentes, eles devem ter um protocolo básico de interação. A seguir estão os itens que poderão compor este protocolo, no caso específico do exemplo do sistema de agendamento descrito na seção anterior.

1)Conexão entre os agentes:

A conexão sempre será feita através de *sockets*[8] de comunicação, bastando-se para criar a conexão fornecer o número da porta lógica e o endereço TCP/IP da máquina desejada. O nome dos agentes será formado pela seguinte estrutura: scc@inf.ufrgs.br|agenda|143.002....|4321, onde a primeira parte antes da barra horizontal especifica o e-mail do proprietário do agente, a segunda a função do agente, a terceira o endereço TCP/IP da máquina que está hospedando o agente e o último número especifica a porta lógica de comunicação na máquina.

[8] Um socket é um dos pontos finais de uma comunicação entre dois programas que estão sendo executados em uma determinada rede. Para cada serviço oferecido pela Internet é especificado uma determinada porta lógica (ex: e-mail = 25, telnet=13), para que nosso micro entre em contato com um determinado serviço de um servidor, precisamos estabelecer um *socket* de comunicação com esta máquina através da porta desejada.

String de comunicação:

NOME DO AGENTE : SC	Agente **solicita conexão** com outro agente.
NOME DO AGENTE : AC	Neste *string* o agente solicitado está respondendo que aceita começar negociações (**Aceita conexão**).
NOME DO AGENTE : NC	Neste *string* o agente solicitado está informando que não aceita começar negociações (**Não conexão**). Neste caso o agente solicitante deverá aguardar um determinado tempo até fazer nova tentativa de conexão.

2)Conexão estabelecida:

Caso a conexão via *socket* seja estabelecida sem problemas, inicia-se a conversação entre os agentes.

Solicitação de marcação de reunião:

NOME DO AGENTE : MR *DATA	O agente solicitante envia para cada um dos agentes que compõem a sociedade, uma *string* solicitando **Marcar reunião** na data especificada na variável *DATA.

Resposta para o pedido de marcação de reunião:

NOME DO AGENTE : DD	O agente solicitado, após verificar sua agenda particular, informa que a data solicitada é uma **data disponível**.
NOME DO AGENTE : DND	Neste caso não houve disponibilidade na data solicitada, sendo assim, o agente informa para o solicitante que **data não está disponível**.

Resposta do solicitante:

NOME DO AGENTE : SMR	Caso todos os membros da sociedade tenham respondido que a data solicitada está disponível, o agente solicitante envia uma *string* informando que todos podem marcar a reunião(**Sim Marcar Reunião**).
NOME DO AGENTE : NMR	Caso, pelo menos, um dos membros da sociedade tenha informado que não pode marcar na data solicitada, então o solicitante

	manda para toda a sociedade uma *string* informando que a reunião não deve ser marcada (**Não Marcar Reunião**).

Respostas dos agentes solicitados:

NOME DO AGENTE :BYE	A conexão com o agente solicitante somente é encerrada quando o solicitado remete uma *string* para finalizar a conexão. Esta *string* significa que o agente solicitado esta a par de toda a negociação que ocorreu na sociedade e está pronto para encerrar a conexão.

5 Sistemas para desenvolvimento de agentes na Internet

Neste capítulo está se enfocando os sistemas ou ambientes, desenvolvidos para dar suporte a implementação de sociedades de agentes na Internet. Serão descritos dois sistemas: SodaBot e JAT.

5.1 O sistema SodaBot

Neste capítulo da dissertação, será introduzido o sistema SodaBot **[COE 94]**, desenvolvido por Michel Coen no laboratório de Inteligência Artificial do MIT. Constitui-se em um ambiente para criação e uso de agentes de *software* distribuídos. O sistema SodaBot é composto por uma máquina virtual denominada de Agente Básico de Software (Basic Software Agent) - um *framework* para construção de agentes que é essencialmente um sistema para operação de agentes, constituindo-se na parte principal do sistema. Também é apresentada uma linguagem para programação do Agente Básico de Software. Através desta linguagem, os usuários podem facilmente implementar um amplo conjunto de aplicações típicas de agentes de *software*. Como exemplo pode-se citar assistentes pessoais on-line e agentes de agendamento de compromissos. Na figura 5.1 é apresentado o sistema SodaBot:

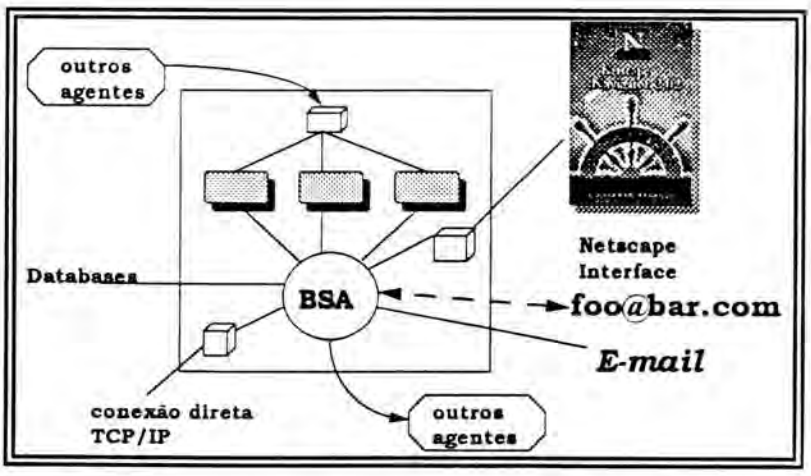

FIGURA 5.1 - Apresentação do SodaBot **[COE 94]**

Na figura acima, pode-se identificar o *BSA* que seria como o coração de toda a estrutura, sendo o responsável pela a execução de cada processo agente que foi projetado pelo seu usuário, e também, é responsável por ficar em estado de escuta (*listening*) em uma série de canais para verificar a chegada de novos dados. No topo, pode-se observar três retângulos sombreados que representam processos em estado de escuta, os quais aceitam estabelecer conexões TCP com outros agentes. Estes processos

de escuta são de extrema importância, para o atendimento de conexões, pois enquanto o ***BSA*** está ocupado (desta maneira impossibilitado de receber informações) são eles que atendem as conexões até o ***BSA*** ser liberado.

O SodaBot pode utilizar um *Browser* Netscape dedicado, para sua interface com o usuário. O *Browser* Netscape conecta-se ao ***BSA***, via uma porta de alta-prioridade, para permitir respostas rápidas às entradas do usuário. Os ***BSAs*** podem comunicar-se através de e-mail.

Nome dos agentes:

SodaBot implementa seu próprio esquema de nomes. Cada ***BSA*** tem um nome que parece com um endereço de e-mail, por exemplo, *foo@bar.com*, mas os nomes de usuário e *host*, não necessitam corresponder a entidades endereçáveis da Internet.

Os ***BSAs*** são interconectados via três camadas de servidores dispostos de maneira hierárquica. *Site servers* fornecem informações de todos os ***BSAs*** que estão sendo executados em um *site* em particular (ou grupo de *sites*), e servidores globais são utilizados para que *site servers* possam localizar outros *site servers*. Por exemplo, para contactar foo@bar.com, deve-se primeiramente fazer-se um contato com o server@bar.com, o qual poderia fornecer informações para contactar o foo@bar.com. Se não for sabido, como entrar em contato com server@bar.com, poderia-se pedir para o servidor global localiza-lo.

5.1.1 Agentes de Software

Paradigma do agente de *software* SodaBot:

No SodaBot cada usuário possui seu próprio ***BSA*** que é executado tipicamente em *background*. O ***BSA*** é um sistema operacional para agentes, e em uma mesma máquina pode haver mais de um ***BSA*** em ação. O ***BSA*** é programado na linguagem denominada de Linguagem de programação de agentes SodaBot (SodaBotL[9]).

Um ***BSA*** executa programas SodaBotL originados tanto de seu proprietário como de outras pessoas. O ***BSA*** é implementado utilizando um algoritmo de *time-sharing scheduling*. Sendo assim, apenas um ***BSA*** é necessário para executar uma série de agentes de aplicação simultaneamente para um usuário particular.

O ***BSA*** executa um agente até que necessite aguardar alguma informação, por exemplo, entrada de informação ou comunicação com outro agente. Neste caso o ***BSA***, pode colocar o agente em um estado de *sleep*, e atender outro agente no seu lugar.

O usuário e seu ***BSA*** interagem através da interface gráfica do usuário SodaBot. Na tela, uma janela central contém a interface principal do ***BSA***, que permite que o usuário monitore e controle as atividades do ***BSA***. No topo da tela é apresentado o agente editor

[9] Pronuncia-se "Soda-Bottle"

do SodaBot que permite ao usuário criar, compilar, e instalar aplicações de agentes. A parte inferior da tela contém janelas abertas por vários agentes de *software* que estão rodando no BSA. A figura 5.2, apresenta a estrutura do BSA:

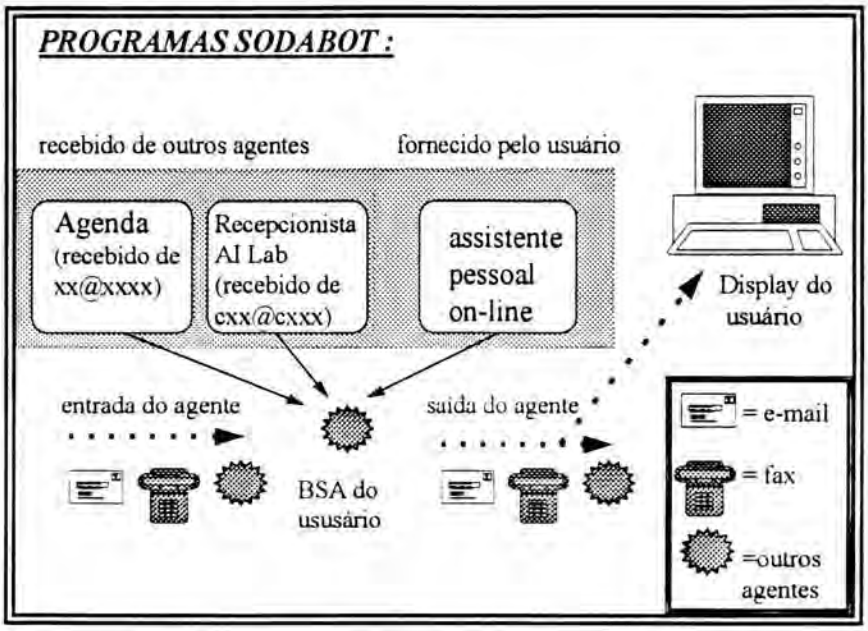

FIGURA 5.2 - Agente Básico de Software **[COE 94]**

5.1.2 A Linguagem de Programação de Agentes de Software

A linguagem SodaBotL oferece primitivas de alto-nível e estruturas de controle, abstraindo os detalhes de baixo-nível na implementação dos agentes. Por exemplo, em um ambiente Unix tipico, os criadores de agentes, que utilizam o SodaBotL, estão livres de preocupar-se com detalhes como servidores de e-mail, *sockets*, and X-windows. Isto, portanto, torna mais fácil criar um agente. A seguir alguns exemplos de primitivas da linguagem são apresentados, maiores detalhes sobre a linguagem podem ser encontrados no Anexo 4:

1) Fazer uma pergunta:

```
Ask {prompt "Time"} "When are you returning?";
```

2) Apresentar uma mensagem:

```
Display {prompt "Read it?"; Choices (yes, no)} "Mail from $sender: $subject";
```

3) Contactar outro agente:

```
Contact Agent <Receptionist; queries> {users: $inferred} "Do you know who in the AI LAB is responsible for $topic?"
```

4) Lidar com o tempo:

```
Wait until Tuesday before $date: { Display "Reminder, you have an appointment with $person on $date";}
```

A Linguagem SodaBot foi projetada visando a construção de dois tipos principais de agentes de *software*:

5.1.3 Assistentes Pessoais

Como já foi mencionado anteriormente, cada usuário possui o seu próprio BSA o qual pode funcionar como uma simples secretária eletrônica. O usuário também pode programar seus agentes para que venham a realizar atividades como: filtrar e-mail, notificá-lo sobre eventos particulares, ou manipular pedidos de maneira automática.

5.1.4 Agentes de Aplicação

Os usuários podem projetar agentes que venham a oferecer vários serviços específicos[10]. Como exemplo, é apresentado abaixo, um agente simples *Pollster Agent* (agente de pesquisa). Este agente solicita a opinião de um grupo de pessoas através de mensagens diretas aos seus agentes pessoais.

```
Agent {Pollster};

Get response {prompt "And what's your opinion?"; time-out in 10 minutes}
                $message_body;

Reply with  $response;
```

5.1.5 Distribuição Automática de Agentes

O ambiente SodaBot é composto por uma sociedade de agentes básicos de *software* (BSA) os quais são conectados via Internet (e/ou uma rede local).

[10] Agentes de aplicação, também, são conhecidos como SodaBots.

O agente básico de *software*:

Quando nenhum agente está sendo executado, o BSA fica a maior parte de seu tempo inativo. Entretanto, quando está ocioso, ele periodicamente ativa-se (por exemplo, a cada 05 segundos) e verifica:

1) Chegada de e-mail;

2) Atividade do usuário na GUI;

3) Contacta com outro agente.

Para garantir a receptividade durante a interação do usuário, a GUI executa como um processo separado (o qual responde a vários eventos X-windows). A figura 5.3 demonstra a arquitetura geral do sistema SodaBot:

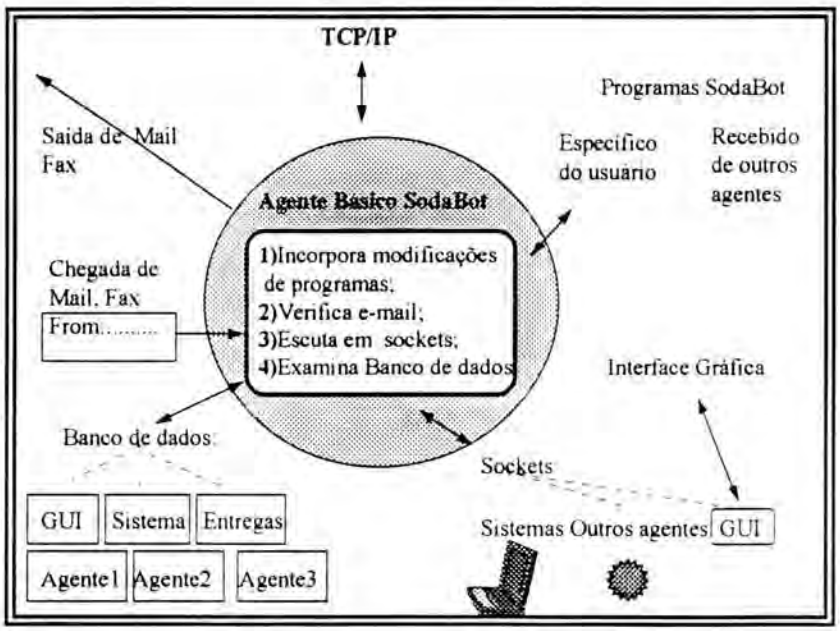

FIGURA 5.3 - Arquitetura geral do sistema SodaBot **[COE 94]**

O ***BSA*** é conectado aos recursos disponíveis do sistema, e todas as aplicações de agente acessam estes recursos através do ***BSA***. Desta maneira, por exemplo, o ***BSA*** é o único responsável pela apresentação de informações na tela do usuário e pelo tratamento do e-mail.

Devido ao fato das aplicações de agentes não terem acesso direto aos recursos do sistema, o ***BSA*** é totalmente livre para, em tempo de execução, reinterpretar os pedidos feitos a ele. Por exemplo, podemos supor que uma aplicação agente deseja apresentar na tela uma mensagem para o usuário, e o ***BSA*** tem conhecimento de que o usuário não está utilizando a *workstation*, mas que o fax deste está disponível. Neste caso, o ***BSA*** pode fazer uso do fax em vez de apresentar a mensagem na tela.

O ***BSA*** pode ser visto como o cerne do *time-sharing* de um sistema operacional de agente. Programas em SodaBotL são compilados dentro do método nativo de operação de linguagem do próprio ***BSA***, o qual é interpretado pelo ***BSA*** quando este executa um agente. O compilador de programas agente é dividido em múltiplas seções as quais podem ser paradas e reiniciadas pelo agente de *scheduler*[11] do ***BSA***.

O Agente de *scheduler* permite que um único ***BSA*** responda por uma variedade de atividades concorrentes do sistema. Não é uma atitude razoável deixar o ***BSA*** perder grande quantidade de tempo aguardando pelo acontecimento de algum evento. Por exemplo, digamos que um agente está sendo executado e em determinado nível de sua trajetória fique dependente de alguma resposta informativa do usuário. Neste caso, o ***BSA*** aguarda um determinado *time out* , antes de colocar o agente em estado de *sleeeping*. Agentes no estado de *sleeping* são armazenados em um dos três bancos de dados de *sleeping agent*, onde ficam em estado de espera até que um determinado evento particular ocorra, ou um determinado limite de tempo passe.

Quando o ***BSA*** recebe alguma informação do usuário através da Interface do usuário, verifica o banco de dados GUI em busca do registro que está aguardando a resposta fornecida. Este registro pode conter:

1) o nome do agente que está em estado de *sleeping;*

2) a posição de processamento deste agente;

3) o dado que este agente estava processando anteriormente.

Caso o registro adequado seja encontrado, o agente é realocado e continua seu processamento até terminar ou até que novo acontecimento gere um estado de espera.

5.1.6 Agentes Distribuídos no SodaBot

No SodaBot, uma aplicação agente geralmente não é executada como um programa único em um BSA. Até um certo ponto, várias seções de um agente são automaticamente distribuídas e executados em BSAs que constituem o ambiente SodaBot. A atividade destes agentes é manifestada pela interação coordenada destas seções de programas.

O ambiente SodaBot é formado por um grupo de agentes básicos de *software* que implementam de maneira cooperativa uma aplicação agente. Estes agentes podem estar sendo executados na mesma máquina ou eles podem estar distribuídos através da rede, cada um sendo executado (em teoria) em uma plataforma ou sistema operacional diferente.

Uma aplicação agente é estruturada em várias seções, as quais são apresentadas no esquema da figura 5.4 e descritas abaixo:

---

[11] O SodaBot foi escrito em PERL, Extended TCL/TK, e C.

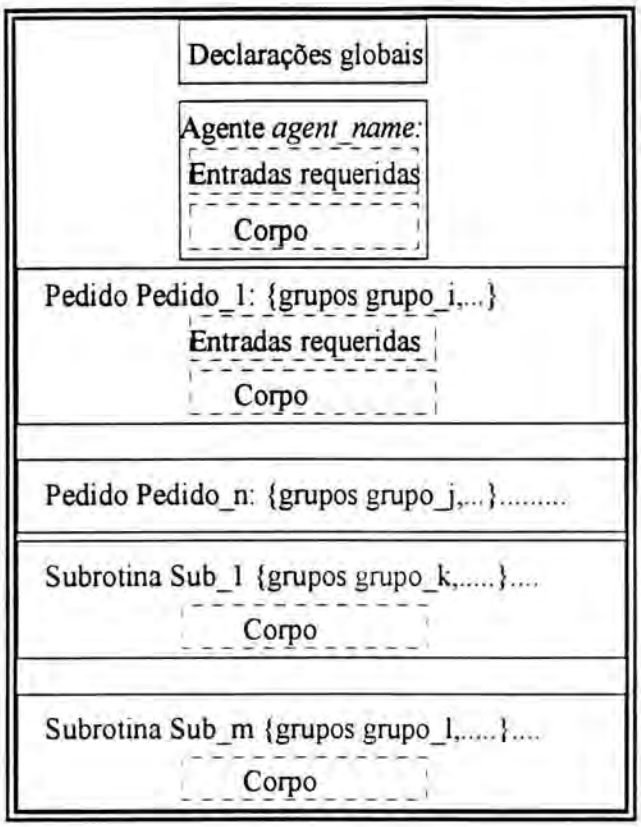

FIGURA 5.4 - A estrutura de um agente genérico SodaBot

1) Declarações globais:

São declarações que ficam distribuídas para cada BSA que executa o agente.

2) O agente principal:(Agente agent_name)

Esta seção especifica o que deve acontecer se o agente for requerido. Não existe um grupo de expressões especificadas aqui. As entradas requeridas especificam o formato da entrada que deve ser fornecida para o agente. O corpo refere-se a uma lista de expressões SodaBotL. Na atual implementação do SodaBot, agentes apenas podem ser invocados via e-mail.

3) Pedido Pedido_i:

Pedidos são partes atômicas de agentes que tornam-se distribuídas e executam nos membros deste ambiente; essencialmente, pedidos de uma aplicação agente são simplesmente o que diferentes BSAs podem perguntar um para o outro enquanto executam o agente; o grupo de adesão simplesmente determina que pedido um BSA está permitido a fazer.

Deve-se notar que cada pedido deve especificar a que grupo este pertence. Um BSA pode fazer um pedido para outro BSA através de e-mail para seu proprietário, como:

To: las@ai.mit.edu

SodaBot: <authorize;certify>

SodaBot - Parser: *< l:1; s:0; e:35. l:2; s:18; e:13.>*

.....

Michel H. Coen (mhcoen@ai.mit.edu) requests that you authorize the submisson in ~mhcoen/tr.ps

4) Subrotinas Sub_i:

Subrotinas são chamadas apenas por pedidos, e não por outras subrotinas.

Um exemplo de distribuição:

Suponha que um usuário chamado Patrick crie ou acesse um agente Pollster (agente para pesquisa de opinião) e então mande um mail para o Pollster de outro usuário chamado Gerry. Com a recepção desta mensagem, o BSA de Gerry verifica se este possui um agente Pollster. Então, o BSA de Guerry coloca a mensagem chegada do Pollster em seu banco de dados de distribuição, que consiste de mensagens que aguardam a chegada dos agentes de *softwares*. Este, então, pede para o BSA de Patrick fornecer o programa do agente Pollster. Quando este chega, o BSA de Guerry compila o programa Pollster que está em linguagem SodaBotL e então inicializa o Pollster.

Cada BSA possui duas estruturas chamadas de *RequestAgent* e *DeliveryAgent*. Quando o *RequestAgent* recebe o pedido de fornecer alguma seção de um agente de aplicação, este comprime, e remete um mail com o programa em SodaBotL correspondente a seção solicitada pelo *DeliveryAgent*. Quando o *DeliveryAgent* recebe o agente solicitado, este *unpack* (descomprime) o programa e usa o compilador SodaBot para instalar o agente. O descrito acima encontra-se esquematizado na figura 5.5 abaixo:

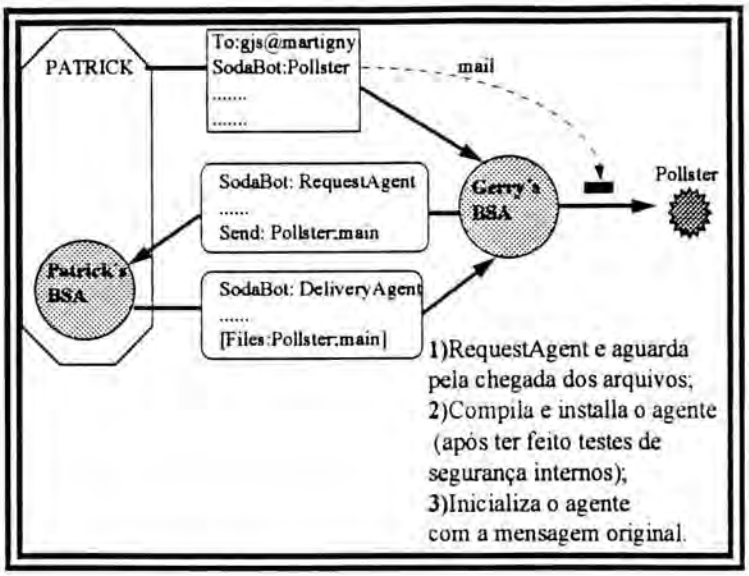

FIGURA 5.5 - Distribuição do Agente Pollster **[COE 94]**

O que aconteceria com o ambiente SodaBot quando o ambiente dos agentes for mais complexo, como por exemplo, seções de agentes podendo contatar-se com outros grupos de agentes dos quais não faz parte. Neste caso o autor do SodaBot fornece uma série de possiveis soluções que poderiam contornar este problema, como por exemplo a abaixo descrita **[COE 94]**:

Criar uma central que teria todos os agentes (um repositório de agentes) a qual guardaria todos os agentes em uma biblioteca, dispondo estes através de pedidos.

5.2 O sistema JAT

O JAT (*Java Agent Template*) foi desenvolvido por Rob Frost da Universidade de Stanford e fornece um modelo funcional completo (escrito inteiramente utilizando a linguagem Java), para construção de agentes de *software* que comunicam-se dois a dois, com uma comunidade de outros agentes distribuídos através da Internet. Embora o código que define cada um dos agentes seja portável, os agentes projetados no JAT, não são móveis (migratórios), tendo uma existência fixa em um determinado *host* **[JAT 96]**.

Atualmente, todas as mensagens provenientes de agentes, utilizam como padrão o KQML, como um protocolo de alto-nível. O JAT inclui facilidades para troca dinâmica de recursos, as quais podem incluir classes Java (por exemplo: novas linguagens e interpretadores, serviços remotos, etc.), arquivos de dados e informações embutidas nas mensagens KQML (maiores detalhes no item 6.4 do capitulo 6).

Os agentes do JAT podem ser executados tanto como aplicações *standalone* ou como *applets* através do uso do *appletviewer*[12] (*browsers* como o Netscape Navigator, não conseguem trabalhar com estes agentes, devido a suas restrições de acesso impostas

[12] *Appletviewer*: programa desenvolvido inteiramente em Java, o qual possibilita a visualização de *applets*, sem haver a necessidade do uso de um *browser*, como o Netscape Navigator, por exemplo.

por aspectos de segurança do próprio *browser*). Ambas as configurações suportam trabalhar com agentes gráficos ou não-gráficos.

A coordenação da sociedade de agentes é providenciada por um agente servidor de nomes (Agent Name Server). A arquitetura do JAT foi projetada especialmente para permitir a substituição e especialização dos principais componentes funcionais, incluíndo a GUI, controle de mensagens em baixo-nível, interpretação de mensagens e manipulação de recursos. Conseqüentemente, o JAT deve ser utilizado como uma plataforma para construção de um amplo conjunto de agentes para diferentes domínios de aplicação.

5.2.1. Arquitetura do JAT

Na figura 5.6 é apresentada uma visão geral da arquitetura do JAT:

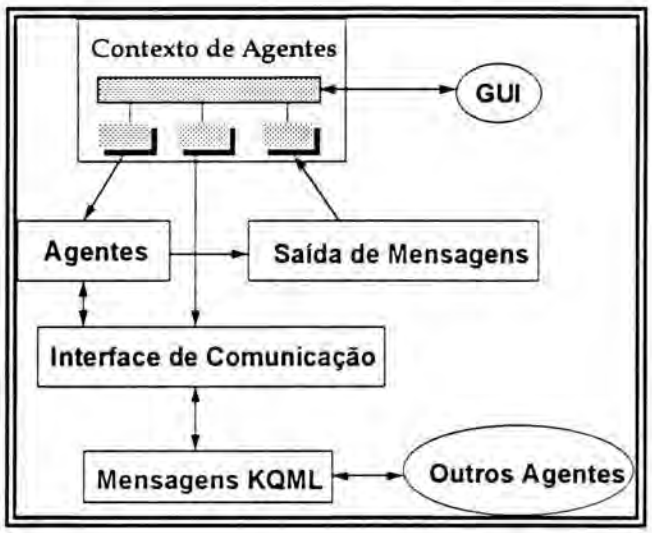

FIGURA 5.6 - Arquitetura básica do Java Agent Template **[JAT 96]**

Contexto de agente:

O Contexto de agente oferece um local executável para o agente e sua interface de comunicação associada. O contexto de agente é representado por uma interface *AgentContext* que especifica um conjunto de métodos abstratos que podem ser implementados por qualquer objeto que funciona como um contexto de agente. Na presente distribuição, a interface *AgentContext* é implementada pelas classes *AgentContext* e *ANSFrame* que oferecem contextos de agente com GUIs. Cada uma destas classes pode ser executada como uma aplicação ou como *applets*, através das classes *AgentApplet* e *ANSApplet*. Na verdade os agentes são recipentes para receber recursos (endereços, linguagens, *ontologies* e classes) e são habilitados a receber, interpretar e mandar mensagens KQML via uma interface de comunicação.

Agentes

O agente representa o elemento funcional primário na arquitetura. Um agente basicamente representa uma caixa preta que assincronamente recebe e envia mensagens KQML através de uma interface de comunicação. O conhecimento e a funcionalidade de um agente são representados pelos objetos Recurso (Resource). Recursos padrões incluem endereços, linguagens, *ontologies* e classes. Um agente comunica-se com o contexto de agentes através de um objeto *MessageOutput*.

Interface de comunicação

A Interface de comunicação fornece o mecanismo de baixo nivel para transmissão confiável de mensagens KQML. Na versão atual, a comunicação é feita usando uma interface baseada em *socket* (classe *SocketInterface*) que oferece às mensagens *multithreaded* (multilinha) para recebimento de transmissão.

Envio de mensagens

A comunicação entre o agente e seu contexto é feita usando um objeto *MessageOutput*. Este objeto permite ao agente enviar tanto mensagens de sistema como KQML e executar alguns métodos de contexto.

Agent Template: endereçamento

Um componente chave do *Agent Template* é o esquema utilizado para nomear e endereçar agentes. O *Agent Template* requer a presença de um Servidor de Nomes de Agentes (ANS) o qual armazena todos os nomes e endereços dos agentes existentes na sociedade de agentes. Os nomes dos agentes são únicos, e portanto por definição o nome do agente baseia-se em seu endereço (número do *host* e número da porta lógica). Pedidos sobre o endereço de outros agentes podem ser feitos para o ANS através do uso de uma mensagem KQML, utilizando a *ontology* "agent".

Quando um agente é criado este conecta-se a rede utilizando um *host* especifico e um número de porta os quais são registrados no ANS. Os endereços são mantidos de maneira dinâmica pelo ANS.

Endereço do agente

Sobre a maneira de distribuição atual, o endereço do agente é constituído pelo nome, completo do "*host*" (ex: piano.stanford.edu) e o nº da porta. Isto representa onde a interface *socket* do agente pode receber mensagens.

Arquivo init

Quando cada agente é criado um conjunto de mensagens é carregado para um arquivo init, localizado em algum lugar na Internet. A localização deste arquivo é

passada como um argumento de comando de linha (para execuções *standalone*) ou como um parâmetro para *applets*. No mínimo, este arquivo deve conter uma mensagem que forneça o endereço para um ANS. Cada linha deste arquivo contém uma mensagem KQML distinta.

Criação de agentes

Quando um agente é criado e conectado a rede, o usuário é requisitado a fornecer um nome, um número de porta e o domínio do *host* local. Se o nome for deixado em branco, então o nome é criado baseado no endereço (ex: piano.stanford.edu:5001). Se a porta é deixada em branco então o *socket* de interface do agente refere uma porta anônima. O domínio correto deve ser fornecido. O nome e endereço resultante do agente é relatado ao ANS listado no arquivo init.

Endereço desconhecido

Se um agente tentar mandar uma mensagem para outro agente, cujo endereço é desconhecido, a mensagem transmitida é bloqueada e uma mensagem de requisição de endereço é mandado para o ANS. Se o endereço é obtido com sucesso, a mensagem é remetida, em outro caso um erro é relatado. Este mecanismo utiliza métodos de sincronização e *multithreading* e é utilizado para toda recuperação de recurso (endereços, linguagem, etc...).

Terminação de um agente

Se um agente sabe que vai terminar, ou seja, que a sua execução será interrompida, este primeiro manda uma mensagem de remoção de endereço para o ANS, o qual ecoa esta mensagem para todos os outros agentes.

Comunicação

A função principal do Agente *Template* é tratar de maneira automática a troca de mensagens KQML assíncronas, entre agentes, os quais estão distribuídos através da Internet.

Socket Interface

A versão atual utiliza a classe *socketInterface*, uma subclasse de *CommInterface*, para troca de mensagens KQML. Quando o agente é conectado pela primeira vez à rede, uma *ServerThread* é disparada para criar um *socket server* na porta local especificada no endereço do agente. Quando o *socket* cliente tenta conectar-se com o *socket* servidor uma nova *ServerThread* é criada de maneira recursiva, esperando por mais conexões clientes. Quando o agente tenta mandar uma mensagem, uma *Clientthread* é disparada, a qual primeiramente verifica o endereço do receptor a mensagem.

Caso o endereço seja encontrado, então um *socket* cliente é criado e conectado ao *socket* servidor do recebedor, em outro caso, a transmissão da mensagem é bloqueada e uma mensagem pedindo o endereço é mandada para o ANS.

Devido a todas as transmissões de mensagens serem tratadas por *threads* independentes, a atividade dos agentes procede ininterruptamente.

Mensagens KQML

É assumido que todas as mensagens passadas entre agentes são, de alto nível, mensagens KQML. Também é assumido que toda mensagem KQML de alto nível terá o seguinte conjunto mínimo de campos: função (*performative*[13]), remetente, receptor, linguagem, *ontology* e conteúdo. Por exemplo: *performative*: "*avaluate*", *sender*: será preenchido automaticamente, *receiver*: nome do agente2, *language*: "KQML", *ontology*: "*test*", *content*:"(test2)". Campos adicionais podem ser acrescentados livremente.

Conteúdo das mensagens

O conteúdo das mensagens é definido tanto pela sintaxe (linguagem) e semântica (*ontology*). Quando uma mensagem é inicialmente recebida por um agente, o campo da semântica é verificado determinando se o agente entende a interpretação indicada pela *ontology*. Se o agente não entende a *ontology*, a mensagem é passada para o método *interpretMessage*() do *ontology*. Este método interpreta mensagens baseadas na linguagem do conteúdo do campo.

5.2.2 JAT v_3.0

O JAT na sua versão 3.0, inclui o *core JAT packages* (*JavaAgent.context*, *JavaAgent.agent* e *JavaAgent.resource*), os pacotes que dão suporte a serviços remotos (*RemoteService.context*, *RemoteService.agent*, *RemoteServices.resource* e *Image Selector*) e pacotes que implementam exemplos **[JAT 96]**.

A figura 5.7 apresenta em maiores detalhes a arquitetura completa do sistema JAT.

[13] *Performatives*: são as mensagens do KQML, que referenciam tanto as crenças como os objetivos de um agente.

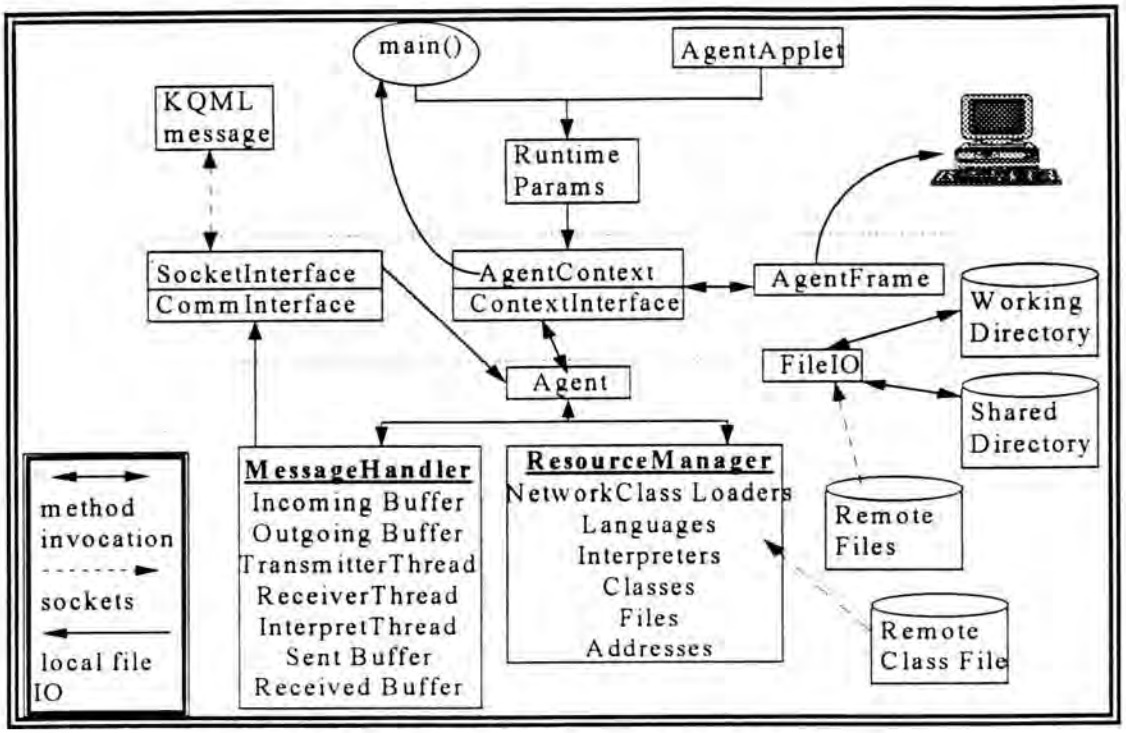

FIGURA 5.7 - Arquitetura detalhada do JAT **[JAT 96]**

5.2.3 Problemas detectados

Algumas falhas que este sistema apresenta, são apontadas pelo próprio autor. São elas:

1) Não trabalha no *Browser* Netscape devido a restrições de segurança do *browser*.

2) Requer um *browser* o qual permita que *applet*s façam conexões arbitrárias de conexões de *socket*s.

3) Uma das caracteristicas desejáveis deste sistema seria a de armazenar todos os agentes conhecidos, ainda se seu endereço é desconhecido.

4) Outra seria o fato do ANS guardar somente o endereço dos agentes. O correto seria o ANS guardar todas as informações sabidas sobre o agente.

5.2.4 Exemplo da utilização do JAT

Neste item será apresentado um exemplo do uso do Java Agent Template.

Exemplo:

Todos os agentes JAT mantém tanto classes quanto dados utilizando objetos que serão subclasses do *JavaAgent.resource.Resource*. Os recursos das classes Java incluem

linguagens e interpretadores (o interpretador é especificado utilizando o campo *ontology* KQML). É assumido que a implementação de cada classe de Interpretador é baseada em alguma especificação formal (*ontology*) da semântica da mensagem. Uma propriedade do JAT permite que estes recursos sejam dinamicamente trocados entre agentes em tempo de execução. Isto permite que um agente processe uma mensagem corretamente, mesmo desconhecendo tanto a linguagem como o interpretador, podendo adquirir de maneira dinâmica as classes referentes à linguagem e ao interpretador que necessita. Na seqüência será apresentada esta propriedade do JAT.

Vamos supor que temos um ANS e dois agentes já inicializados. O primeiro agente atuará como agente1 e o segundo como agente2 .

A seguir será descrito passo a passo como realmente ocorre a interação entre os agentes neste sistema:

• Primeiro, quando o agente1 tenta mandar sua primeira mensagem, caso não tenha o endereço do agente2, irá bloquear a transmissão da mensagem e no lugar desta, mandará uma mensagem para o ANS pedindo o endereço do agente2.

• O ANS pegará a mensagem proveniente do agente1 e remeterá uma de volta informando o endereço do agente2.

• De posse do endereço do agente2, o agente1 irá em frente e remeterá a mensagem *test* para o agente2.

• O agente2 irá receber a mensagem *test* e perceberá que não possui o interpretador *test* e que não é capaz de interpretar a mensagem. O agente2, então bloqueará a interpretação da mensagem e mandará uma mensagem para o agente1, requerendo a localização do código *TestInterpreter*. Entretanto, neste caso o agente2, não possui o endereço do agente1. O procedimento adotado neste caso, será o já descrito, este agente2 entrará em contato com o ANS requerendo o endereço do agente1.

• O ANS recebendo o pedido do agente2, irá responder com o endereço do agente1.

• O agente2 de posse do endereço do agente1, mandará, finalmente, uma mensagem solicitando a localização do interpretador.

• O agente1 recebendo o pedido do agente2, irá responder a localização correta do *TestInterpreter*.

• O agente2 de posse do endereço (localização fisica) do interpretador, irá carregar o código, criando uma nova instância da classe *TestInterpreter* e finalmente, interpretará a primeira mensagem recebida do agente1 (o objeto KQML*message* será passado como um argumento para o método *interpretMessage*() do objeto *TestInterpreter*).

• Devido a *performative* do conteúdo ser "test2", o método *test2Action*() será chamado.

6 Arquitetura para coordenar a interação de agentes na Internet

Após a implementação de alguns agentes assistentes, que serviram como experiência na construção deste tipo de agentes de *software*, e também, com o estudo de alguns sistemas e agentes já existentes, foram observadas algumas questões importantes que necessitavam de um estudo maior para a definição de uma arquitetura para coordenar a interação de agentes na Internet. Devido a alguns destes problemas abordarem questões sobre a conexão via TCP/IP, antes destes serem descritos, será feita uma breve descrição deste protocolo utilizado amplamente na Internet.

6.1 Protocolo da Internet : TCP/IP

Os computadores na Internet comunicam-se entre si enviando e recebendo pacotes de informações. Estes pacotes contêm, porções de dados, informações especiais de controle e endereçamento, necessários para levar os pacotes aos seus destinos e remontá-los em dados úteis. Tudo isso é realizado pelos *Transmission Control Protocol* e *Internet Protocol* (ou Protocolo de Controle de Transmissões e Protocolo da Internet), também conhecidos por TCP/IP **[EDD 94]**.

Um protocolo é uma regra ou um acordo para procedimentos, de comunicação em rede. Neste caso, o protocolo pode também indicar: como os pacotes são remontados em dados originais no computador central, como os dados são transmitidos pela linha telefônica, etc. O TCP/IP é o núcleo da Internet, mas refere-se a somente dois dos muitos protocolos nela existentes.

Os protocolos são freqüentemente descritos em termos de *RFCs, ou requests for comments* (solicitações para comentários). Os RFCs são os documentos de trabalho que a comunidade Internet emprega para desenvolver e registrar uma informação técnica. Muitas instalações computacionais da Internet armazenam essas informações eletronicamente.

O modelo de camadas da Internet mostra como diferentes camadas de protocolos são usadas para conectar o computador e o usuário à Internet, transformar os dados em pacotes e direcionar estes pacotes para o computador central (*host*) destinatário. Uma vez no computador central, os dados são extraídos dos pacotes e remontados em dados que podem ser utilizados pelo computador.

O esquema apresentado na ilustração abaixo detalha o modo como os pacotes são divididos ou remontados a cada estágio do processo de comunicação (figura 6.1).

O computador do usuário envia alguns dados para o computador central e recebe dados dele através da Internet. Por exemplo, um programa (aplicativo) como o Telnet (apresentado na figura 6.1, como um exemplo) divide os dados em pacotes. Os protocolos especificam como os pacotes devem ser colocados em camadas, ou embalados em pacotes menores. Diferentes camadas de pacotes têm acesso a uma variedade de programas e equipamentos para poder enviar as informações através de diferentes redes e conexões de comunicação.

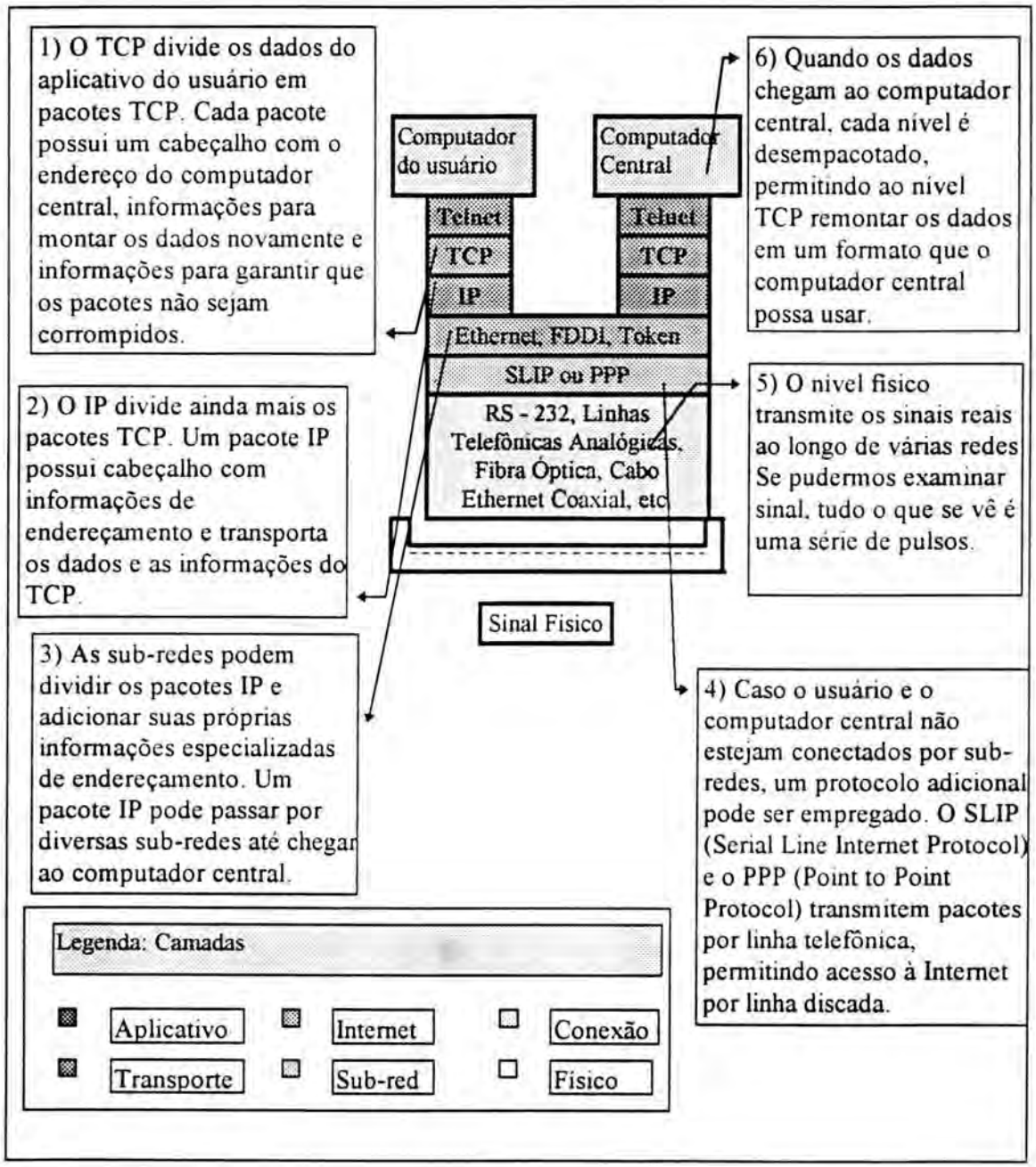

FIGURA 6.1 - Camadas da INTERNET

6.1.1 Interligando Redes

O que temos hoje na Internet são milhares de computadores ligados em rede e estas interligadas entre si. A ligação entre estas diferentes redes é feita através de computadores intermediários denominados de *routers* ou *gateways*. Esses computadores vão receber pacotes de informação, analisar se eles são endereçados a um computador de uma das redes a que ele esteja diretamente ligado e, caso não sejam, passá-los para frente **[DAM 96]**.

Todo pacote de informação enviado pela rede, possui em seu interior o endereço da rede destino e da máquina destino dentro da rede. Quando o pacote sai de um dado computador, este vai para o *gateway* da sua rede. Este *gateway* verifica se este pacote é para sua própria rede ou não. Caso este pacote não seja para a sua rede, o *gateway* em questão deve encaminhar o pacote para o *gateway* mais próximo para que este seja encaminhado até que a rede e a máquina desejadas sejam localizadas.

O *gateway* trabalha com o que chamamos de endereços lógicos de rede. Os endereços dos computadores de uma rede são um sub conjunto desse endereço lógico. Um pacote carrega consigo o endereço destino da rede e máquina, desta maneira o *gateway* consegue saber se este pacote é para uma das máquinas da sua rede.

6.1.2 Composição do endereço IP

Cada um dos computadores que compõem uma rede TCP/IP tem um endereço lógico chamado endereço IP[14]. Esse endereço tem 32 bits, com quatro conjuntos de 8 bits. Estes números em notação decimal, formam quatro números menores que 255, formando endereços IP do tipo: 136.47.238.169 (por exemplo) **[DAM 96]**.

Para facilitar o trabalho de roteamento dos *gateways* os endereços IP são hierárquicos. Desta maneira, uma rede pode ser (por exemplo) designada pela seguinte combinação numérica: 135.10.0.0 e um dos computadores que compõem esta rede pode ser designado pelo endereço 135.10.230.7. Quando o *gateway* recebe um pacote com esse endereço de máquina, se ele atende a rede 135.10.0.0, sabe que o pacote é seu. Temos então em um endereço IP do computador, uma parte que indica a rede e outra que é o endereço desta máquina dentro da rede em questão.

6.1.3 Endereços IP dinâmicos

No caso dos usuários que se associam a um determinado provedor de acesso à Internet, é comum estes não fornecerem um endereço de IP para cada um dos usuários.

---

[14] Deve-se observar que o termo IP refere-se a um protocolo da camada de transporte, quando na dissertação for feita referência ao termo "endereço IP", está sendo abordado o aspecto do formato do endereço que é utilizado por este protocolo. Esta denominação é utilizada nesta dissertação por ser de larga utilização em muitos textos consultados.

Os provedores têm uma série limitada de endereços IP e, cada vez, que um usuário se conecta à Internet, recebe um endereço diferente. O principal problema é descobrir qual o endereço designado para a máquina do usuário a cada interação com o provedor de acesso. Como pode uma máquina entrar em contato com a do usuário em questão, se este endereço muda a cada interação a critério do provedor de acesso?

6.1.4 Portas de comunicação

A maioria dos sistemas operacionais trabalha com multiprogramação. Sendo assim, a comunicação não é mais feita entre computadores, mas sim entre aplicativos. Desta maneira, algo que identifique a aplicação destinatária, deve vir junto com o pacote. Isto é feito através de uma abstração, denominada de porta lógica. Um determinado computador pode estar com uma ou várias portas lógicas abertas ao mesmo tempo.

Cada porta deve ser referenciada através de um número com 16 bits, podendo chegar até 65000. As portas abaixo de 1024 são reservadas para aplicações padrões, como é o caso das portas lógica de número 80, que atende a todas as aplicações HTTP; de número 25 que atende as aplicações de e-mail e a porta de número 13 que atende as aplicações de Telnet.

Assim, para uma aplicação mandar um pacote para outra, tem que saber, não só o endereço IP do outro computador, mas também o número da porta lógica a que esta aplicação está conectada.

6.1.5 Servidores DNS

Para facilitar o acesso a lugares na Internet, usualmente não se faz uso direto de uma combinação numérica, mas sim de um determinado nome que referencia uma determinada combinação numérica. Para tanto foi criada uma abstração denominada de DN (*Domain Name*). Quando acessamos uma página WWW, o texto (endereço) que referencia a página, refere-se a um número de máquina.

Na Internet existem computadores denominados de servidores DNS (*Domain Name System*), destinados a converter estes nomes em endereços IP.

O *Domain Name System* (DNS) estabelece uma hierarquia de domínios, referências, ou grupos de computadores. Estabelece um nome de referência (também conhecido como endereço da Internet) para cada máquina na Internet. As referências principais têm a responsabilidade de manter listas e endereços de outras referências do nível imediatamente inferior em cada grupo **[EDD 94]**

Para ver como funciona exatamente o DNS, como exemplo será utilizado o endereço Internet do Instituto de Informática da UFRGS: *inf.ufrgs.br*. A referência principal responsável por este nome é *br*, ou Brasil (indicando que esta rede está localizada fisicamente no Brasil). A referência seguinte, *ufrgs*, indica qual instituição é responsável por esta rede. A referência *inf* indica uma das redes mantidas pela UFRGS.

O endereço IP refere-se ao verdadeiro endereço do computador e da rede que conecta a UFRGS à Internet. Pode-se adicionar o símbolo @ para adicionar informações não-Internet a um endereço Internet.

6.2 Considerações Importantes (Problemas Analisados)

Abaixo serão listados três problemas principais que foram analisados para a proposta de uma arquitetura:

Problema número 1:

Um dos problemas que foram analisados e levados em consideração na proposta de uma arquitetura, foi o do tipo de sociedade de agentes que se está trabalhando.

A arquitetura proposta tem de ser independente do tipo de sociedade em questão, ou seja, ela deve apresentar um funcionamento que venha a suprir as necessidades de integração entre os agentes, independente do fato da sociedade ser aberta ou fechada. A figura 6.2 ilustra estes tipos de sociedade de agente:

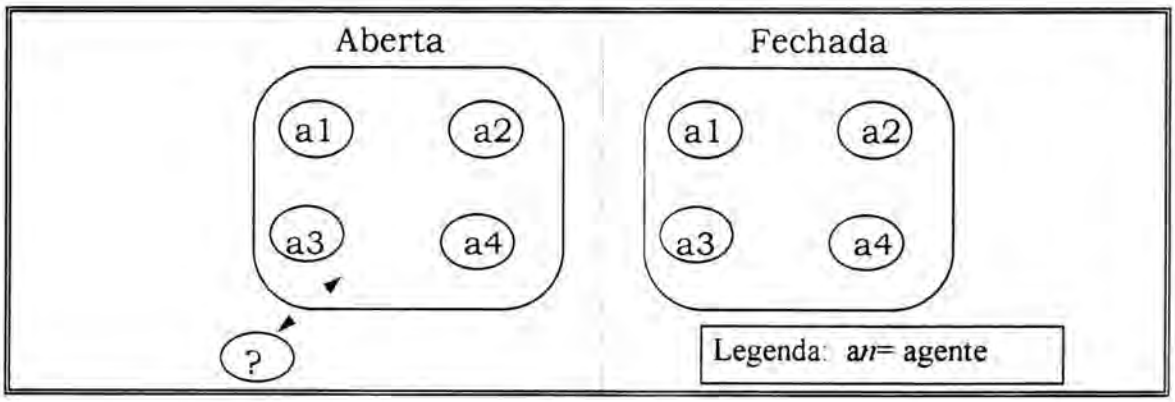

FIGURA 6.2 - Tipo de Sociedade

O problema que foi analisado sob o ponto de vista do tipo de sociedade pode ser descrito da seguinte maneira: a sociedade em questão é uma sociedade aberta, ou seja, a entrada e saída de indivíduos é permitida, ou esta sociedade apresenta-se sem alteração, ou seja, a migração de seus integrantes é nula. Este problema deve ser analisado principalmente no momento de propor a integração entre os agentes. A questão seria - como todos ficarão sabendo da entrada, ou saída de membros da sociedade?

Problema número 2:

Um outro problema que foi avaliado, é o fato da necessidade ou não da existência de um agente centralizador, como figura integradora de uma sociedade de agentes. A figura 6.3 ilustra este fato.

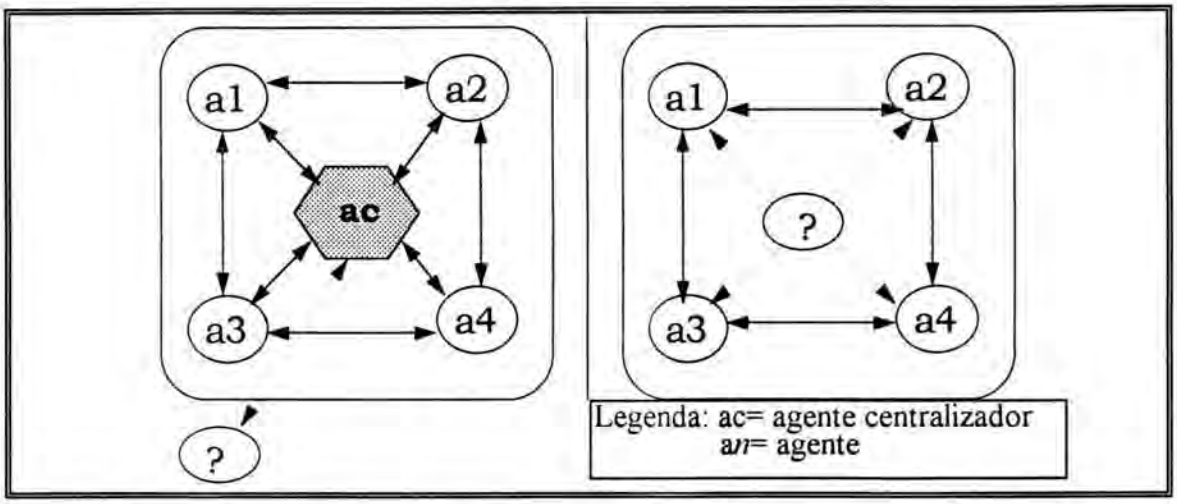

FIGURA 6.3 - Necessidade do Agente Centralizador de Endereços

É interessante que nesta sociedade de agentes exista um agente centralizador dos endereços de máquina de cada agente? Como deve ser a apresentação destes indivíduos dentro da sociedade? Duas soluções iniciais podem ser apresentadas:

1. Existe um agente central na sociedade, e este tem a função de atualizar as informações gerais da sociedade, ou seja, caso um "agente x" chegue e passe a integrar a sociedade, este deve primeiro dirigir-se ao agente centralizador e informar seus dados (função desempenhada, nome, endereço de IP, etc.). Logo após da apresentação ter sido feita, o agente centralizador deve informar a entrada do novo membro a sociedade. O mesmo deve ocorrer no caso da retirada, temporária ou não, de um agente da sociedade (neste caso a retirada do agente deve ocorrer de maneira consciente, ou seja, ele sai por vontade própria e não por uma queda de sua rede de trabalho, por exemplo).

2. Não existe a figura do agente central. Neste caso se um "agente x" quiser entrar em uma sociedade de agentes, este deve dirigir-se a cada membro integrante da sociedade (para encontrar os membros da sociedade, entende-se que os agentes terão de consultar um arquivo comum, onde todos os agentes deixam seus dados escritos). Esta solução é bastante viável, mas vai depender da quantidade de indivíduos que vierem a integrar esta sociedade.

No caso das duas possibilidades acima podemos observar tanto pontos a favor como contrários.

No caso da primeira solução proposta, temos como ponto a favor uma estrutura central, que ficaria localizada em uma determinada máquina da rede, e caso algum agente não saiba como entrar em contato com outro membro da sociedade, entraria em contato com este agente centralizador, e solicitaria o endereço de máquina do agente desejado (de maneira semelhante ao sistema JAT 3.0).

O ponto contrário a esta solução seria o fato de toda a sociedade ficar, até certo ponto, dependente de um agente centralizador fixo em uma máquina, ou seja, caso (por exemplo) a máquina que está hospedando este agente, venha a apresentar algum tipo de *pane* (perder o contato com a rede), toda a sociedade de agentes seria de certa maneira

atingida, pois os agentes não saberiam quais os membros que entraram ou deixaram esta sociedade de agentes, e nem saberiam como acessar uns aos outros.

No caso da segunda solução apresentada, pode-se destacar um ponto muito positivo, que é a não existência de um agente centralizador das informações, ou seja, não existe a dependência dos demais agentes em relação às informações prestadas pelo agente centralizador. Por outro lado, esta segunda solução deve ser avaliada de maneira ponderada, pois ela propõe que a cada entrada ou saída de agente de uma sociedade, este informe, este fato para todos os membros da sociedade.

Esta ação de manter uma comunicação com todos os membros, pode ser saudável no caso de termos poucos membros integrando esta sociedade, mas por outro lado, caso a sociedade seja integrada por muitos membros e/ou seja muito dinâmica, pode acabar ocorrendo um congestionamento na comunicação da rede, o que pode acarretar uma queda de seu desempenho geral.

Problema número 3:

O último problema levantado, para ser levado em consideração na definição de uma arquitetura para agentes, foi a existência dos IP dinâmicos de máquina. Conforme a figura 6.4, o agente de IP dinâmico (ad) deve apresentar-se ao agente central (ac) ou a toda a sociedade, e como trabalhar com as trocas de IP?

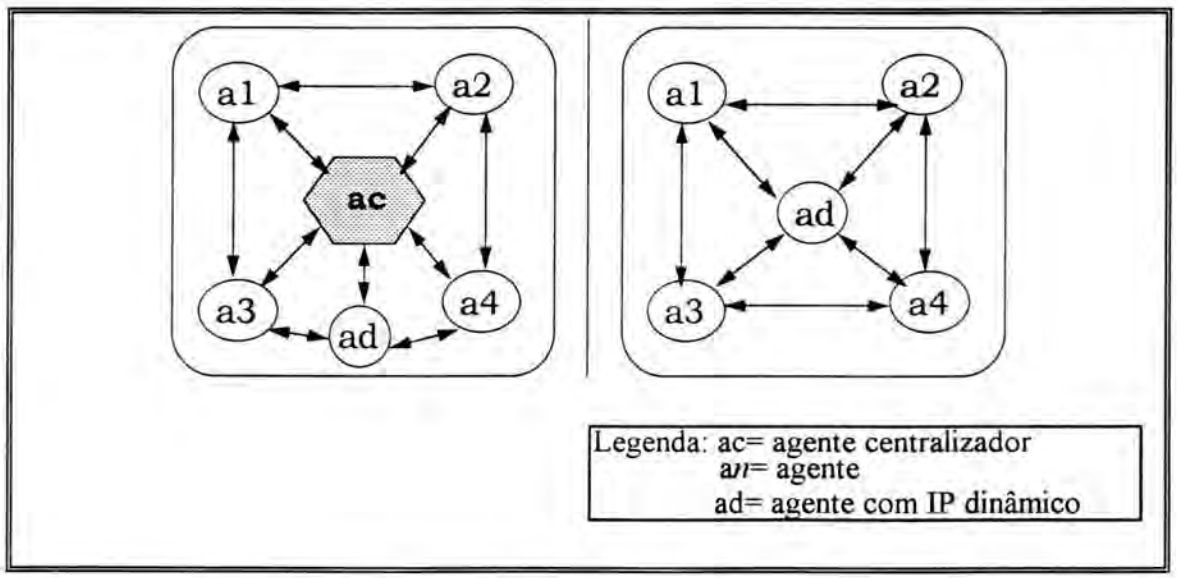

FIGURA 6.4 - Endereços de IP Dinâmico

Como resolver o problema dos IP dinâmicos? Esta questão é muito importante para que se realize uma integração sem maiores problemas entre os agentes.

Como foi explicado anteriormente, na Internet muitas máquinas não possuem um endereço de IP fixo, ou seja, cada vez que entram na rede, são designados endereços diversificados para identificar aquela máquina naquele determinado momento. Ao desconectar-se da rede o endereço é perdido, e ao reconectar-se novo endereço é fornecido.

Este é o caso das máquinas de usuários que conectam-se a servidores que disponibilizam os serviços da Internet. Por estarem dependendo deste servidor para conectar-se, é ele que designa a cada interação um endereço para a máquina do usuário.

Neste caso temos um problema bem mais complexo, pois temos os agentes nos mesmos locais fixos, mas não podemos utilizar sempre o mesmo endereço para achá-los, ou seja, não adianta manter um arquivo com o trinômio "agente, porta, endereço", pois o último item se modificará constantemente. A figura 6.5, exemplifica o acima exposto.

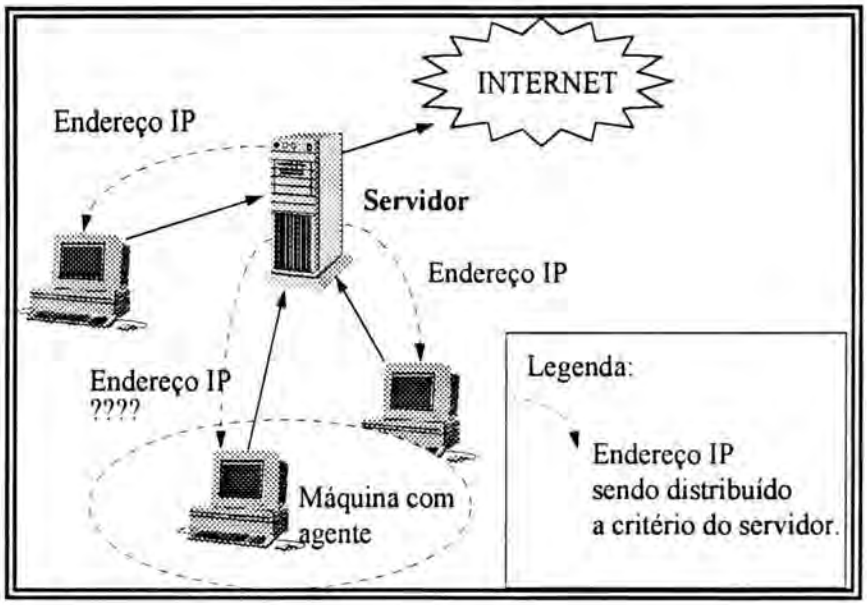

FIGURA 6.5 - Endereço IP distribuído pelo Servidor de Acesso à INTERNET

Para resolver este problema:

Quando for desfeita a conexão do agente com a Internet, primeiramente este entra em contato por *socket* de comunicação com o agente central pedindo para continuar cadastrado e que somente os seus dados fiquem bloqueados até seu retorno a rede. O agente centralizador informa o número cadastral deste dentro do arquivo (agente cadastrado número 50, por exemplo). Quando este agente volta a ser conectado à Internet, prontamente entra em contato como agente centralizador e através de seu número cadastral, pede que seus dados sejam atualizados e que ele seja reintegrado a sociedade. Caso não exista a figura do agente centralizador, o mesmo procedimento será adotado, porém a comunicação deverá ser feita com cada um dos membros da sociedade, o que não é ideal, pois poderá causar uma sobrecarga na rede.

6.3 A Arquitetura proposta

Considerando os problemas analisados na seção anterior, propõe-se uma arquitetura para coordenar a interação de agentes, baseada no uso de um agente central por dominio da Internet.

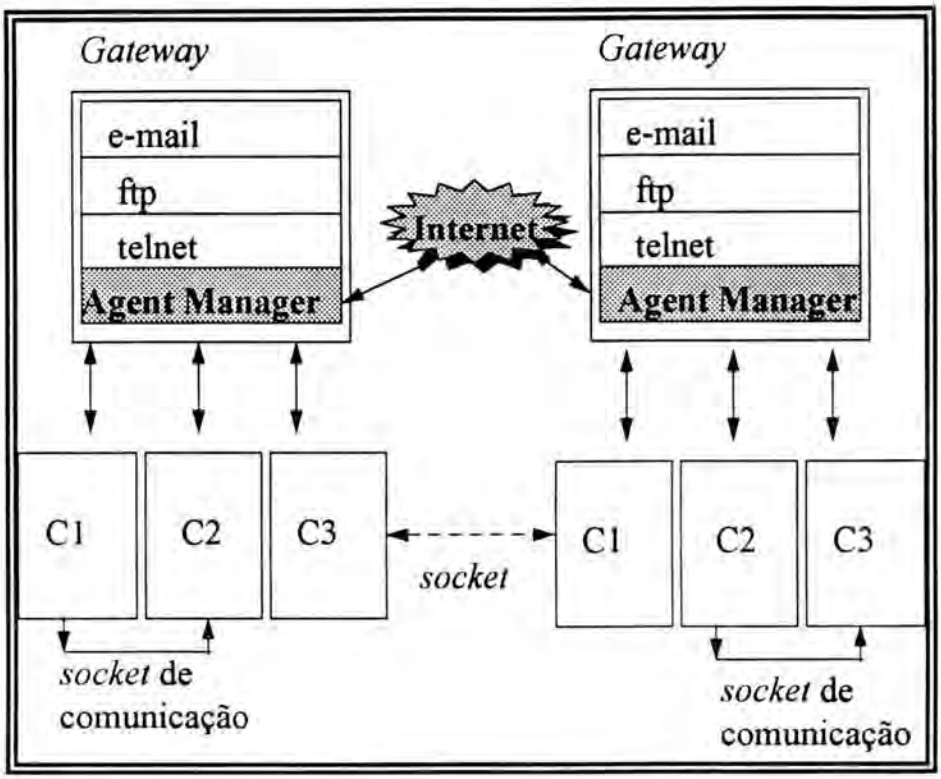

FIGURA 6.6 - Visão geral da Arquitetura de Agentes para a INTERNET

A figura 6.6 apresenta uma visão geral desta arquitetura. Na figura pode-se identificar duas sub-redes (com seus computadores locais designados por c1, c2 e c3) conectados aos seus respectivos *gateways*, comunicando-se através da Internet .

As principais características desta arquitetura são:

1) Na arquitetura existe um agente central, por domínio, que tem como função armazenar (em um arquivo local) e fornecer (quando solicitado) os endereços dos agentes que compõem a sociedade e suas descrições. Este agente é denominado de "***Agent Manager***", constitui-se em um processo que está sempre em execução e em estado de *listening*.

2) Para haver uma maior qualidade no serviço prestado por este agente central, ele é fisicamente colocado na máquina *gateway* da rede, ou seja, ele passa a figurar como mais um serviço (ou aplicativo) fornecido por esta máquina.

Cada um dos serviços suportado por um *gateway* (e-mail, FTP, Telnet, HTTP) faz uso de uma porta lógica específica. No caso, este agente (*Agent Manager*) utiliza uma porta lógica própria (dedicada) para que os agentes façam conexão (via *socket*) com ele.

Colocar este *Agent Manager* na máquina *gateway* traz facilidades várias do tipo:

- se um agente desejar encontrar o agente central basta reportar-se a máquina *gateway* da rede e conectar-se a porta correta;

- para um agente de uma sociedade conectar-se com o agente de outra sociedade basta ser feita conexão de *gateway* para *gateway*;

- para haver conexão entre agentes da mesma sociedade, basta o agente pedir o endereço do outro agente desejado para o *Agent Manager* e realizar a conexão via *socket*, não sendo necessária a participação do *Agent Manager* nesta comunicação;

- uma vez fornecido este endereço, o agente local o guarda em um arquivo local em sua máquina, para caso, haja nova necessidade de conexão, ele não precise mais pedir o endereço para o *Agente Manager*.

6.3.1 Estrutura dos Agentes

Na figura do item anterior (figura 6.6) verificou-se como seria a integração entre os agentes da sociedade e o agente central. Agora será apresentada uma visão dos agentes dentro de uma das máquinas que compõem a rede de comunicação (conforme figura 6.7).

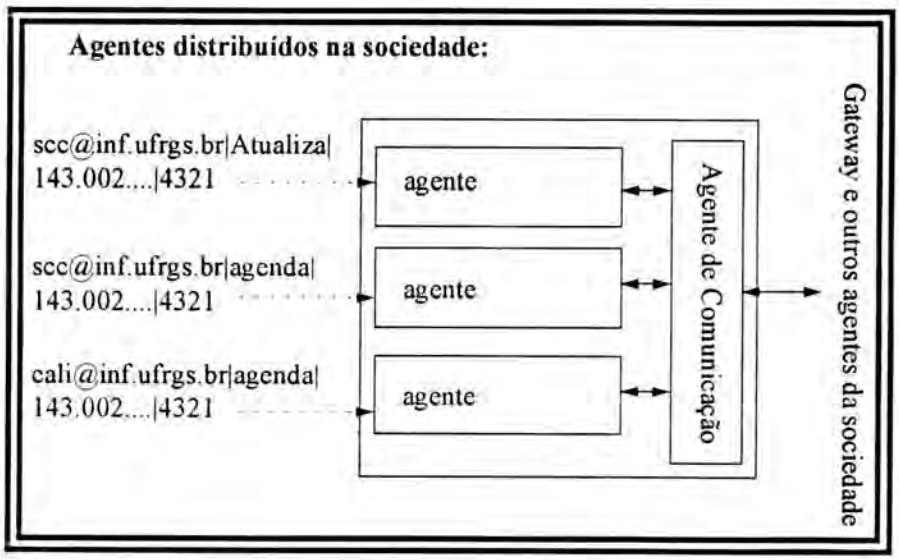

FIGURA 6.7 - Identificação do Agente na máquina local

Nesta arquitetura, cada máquina pode alocar um ou mais agentes específicos, independente de função[15] ou origem deste. Em cada máquina da rede teremos uma entidade específica denominada de **Agente de Comunicação,** que é um processo que está sempre em execução e em estado de *listening*, em uma porta lógica específica e desempenha duas funções de fundamental importância:

1. Este Agente de Comunicação é responsável por distinguir por mecanismos internos, para qual agente alocado na sua máquina é destinada a informação que está chegando via *socket*, ou seja, ao receber uma *string* externa ou interna ele deve fazer uma triagem para saber para que agente é destinada a informação e depois entregá-la (todo o processo de comunicação será feito utilizando conexão por *socket*). Para levar a

[15] Nesta arquitetura está sendo abordado o aspecto de integração e comunicação entre os agentes que compõem uma sociedade de agentes, não é relevante neste momento a função desempenhada pelo agente como entidade funcional.

cabo está missão, ele possui localmente um arquivo com todos os dados de identificação dos agentes que estão alocados na sua máquina.

2. Como já foi citado, este agente constitui um processo que está constantemente sendo executado, "escutando" através de uma porta lógica, ou seja, na verdade ele é a porta de comunicação do mundo dos agentes de uma determinada máquina com o mundo externo e interno.

Como, na realidade, somente o Agente de Comunicação terá acesso a uma porta lógica, é ele que deve receber o que vem de fora (mundo externo) e entregar ao agente correto dentro de sua máquina. O mesmo deve ocorrer com a comunicação de dentro para fora, para um agente interno à máquina mandar alguma informação para o mundo externo, ele tem de fazer uso dos recursos de comunicação oferecidos pelo Agente de Comunicação, ou seja, pedir para este entregar a informação.

No caso de necessidade de comunicação entre os agentes de uma mesma máquina, também será necessária a participação do Agente de Comunicação, ou seja, o agente interessado em enviar alguma informação para outro agente dentro da máquina, deverá solicitar os serviços do Agente de Comunicação. Pode-se pensar no Agente de Comunicação como sendo uma Classe responsável por toda triagem da comunicação realizada entre a máquina e a rede, e dentro da própria máquina.

Não se deve pensar que esta centralização da comunicação no Agente Comunicação seja uma forma de tirar a flexibilidade da arquitetura proposta, ela é necessária devido a alguns aspectos:

- Toda a segurança para garantir a integridade da máquina, será feita pelo Agente de Comunicação, ou seja, ele será uma espécie de gargalo, onde passa toda a informação, e cabe a ele verificar se a informação não está burlando a segurança da máquina.

- Seria inviável propor uma arquitetura onde todos os agentes ficassem em execução constante na máquina, apenas aguardando o momento de agirem, pois desta forma a *performance* da CPU, estaria altamente comprometida. A arquitetura propõe que apenas o Agente de Comunicação fique em execução, e no momento em que for necessária a ativação de um determinado agente, este deverá ser instanciado pelo Agente de Comunicação para que possa agir.

- Devido ao Agente de Comunicação ser uma classe do tipo *multithread*, ele poderá abrir uma linha de execução para cada um dos agente que estão sendo ativados.

6.3.2 Criação de agentes

Para criar um agente o usuário deve cadastrá-lo junto ao *Agent Manager* e ao Agente de Comunicação de sua máquina. Para tanto, deve fazer uso de um programa em Java (um *frame*[16]), que deve existir em cada máquina da rede. Ao disparar este programa

[16] Os programas para cadastrar agentes e retirar agentes da sociedade serão feitos utilizando a linguagem Java. Utiliza-se o termo *frame*, pois estes programas serão construídos utilizando uma classe denominada em Java de *frame*.

serão solicitadas as informações sobre o novo agente que está sendo cadastrado. Ao preencher todos os campos (sem exceção), este programa irá inscrever os dados deste novo agente, dentro do arquivo de agentes da máquina local, e depois estabelecerá uma comunicação via *socket* com o *Agent Manager* da rede e informará os dados deste novo agente. Feito isto o agente estará pronto para fazer parte da sociedade. A figura 6.8 demonstra a interface do *frame* para cadastro de agentes:

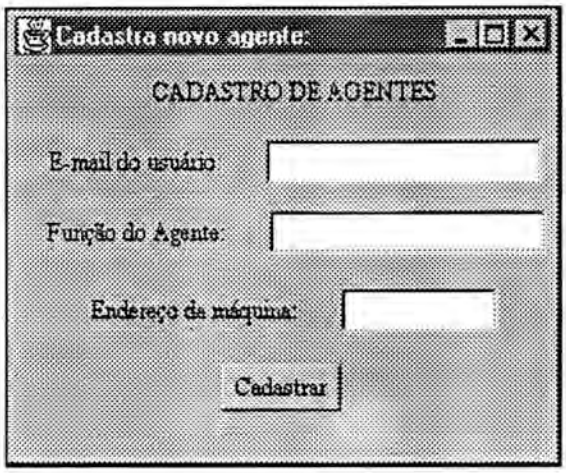

FIGURA 6.8 - Cadastro de agentes novos

6.3.3 Indentificação do agente

A identificação de um agente deve ser formada por duas *strings* básicas:

1) Nome do Agente = E-mail do usuário + Função do agente

Neste campo deve aparecer o e-mail do construtor do agente concatenado com a funcionalidade do mesmo, como por exemplo: scc@inf.ufrgs.br|agenda.

O e-mail do construtor do agente é necessário, por três motivos:

- para que no caso de termos mais de um agente de funções semelhantes funcionando na mesma máquina (por exemplo, várias agendas de diferentes usuários), seja possível identificar para qual destes agentes a informação é destinada;

- este nome é importante, para que os roteadores saibam em qual domínio de rede encontram-se os agentes, ou seja para onde deve-se encaminhar o fluxo de dados;

- caso um determinado agente1 tente estabelecer conexão com um outro agente2, e após um certo número de tentativas esta conexão não seja possível, fica definido que o agente1 poderá mandar um e-mail padronizado informando ao responsável pelo agente2, que algum tipo de irregularidade está ocorrendo com o agente.

2) Número da máquina hospedeira e porta de comunicação ativa:

Neste campo deve aparecer o endereço IP da máquina na Internet que está hospedando o agente em questão, e também qual o número da porta de comunicação disponível para o estabelecimento de *sockets* de comunicação, como por exemplo: 143.002.345.122|4321. Pode-se observar que para os agentes que estiverem na mesma máquina, o campo referente a porta de comunicação será o mesmo, no caso o número de porta lógica alocado para o Agente de Comunicação.

A importância deste campo é exatamente para o estabelecimento de um *socket* de comunicação.

Um exemplo completo de denominação de um agente poderia ser:

scc@inf.ufrgs.br|agenda|143.002.345.122|4321

Interpretando este nome de agente temos:

- o usuário que projetou o agente foi o dono do e-mail: scc@inf.ufrgs.br;

- este agente é responsável por desempenhar a função de agendamento dos compromissos do seu usuário;

- a máquina em que o agente está localizado fisicamente, é a máquina designada pelo endereço IP = 143.002.345.122, dentro da rede;

- a porta lógica designada para ser feita a comunicação entre agentes é a de número 4321. Neste caso, se um agente desejar manter comunicação direta com este agente, deve simplesmente estabelecer um *socket* de comunicação com a máquina 143.002.345.122, através da porta lógica 4321 (lembrando que a comunicação é via Agente de Comunicação).

6.3.4 Comunicação entre diversos agentes

A comunicação entre os agentes desta sociedade dar-se-á da seguinte maneira: um agente1 deseja estabelecer comunicação com o agente2 de uma determinada máquina, mas desconhece o endereço atual do mesmo. O agente1 entra em contato com o *Agent Manager* e pede o endereço atual do agente2.

De posse desta informação é firmada a comunicação via *socket*, sendo que esta informação sobre o endereço do agente2 é armazenada em um arquivo local de endereços de agentes na máquina do agente1. Este arquivo armazena informações sobre (por exemplo) os últimos dez endereços de agentes acessados e pode ser consultado por todos os agentes que estão localizados na máquina do agente1. A idéia é a de evitar que um agente que está constantemente comunicando-se com um outro, tenha de a cada interação com este, ter de solicitar ao *Agent Manager* o endereço do agente.

Quando for ser firmada a comunicação com o agente de outra máquina, primeiramente o endereço deste agente é buscado neste arquivo comum, caso não seja encontrado, então é estabelecida a comunicação com o *Agent Manager* para obter esta informação.

Quando o agente for estabelecer comunicação, deve ser fornecido, por parâmetros de protocolo para o Agente de Comunicação, o nome, a função, o número da máquina e número da porta do agente desejado.

Caso um agente tente entrar em contato com outro agente, e o endereço deste já encontre-se gravado no arquivo comum, a comunicação é estabelecida de forma direta (sem consulta ao *Agent Manager).* Caso o estabelecimento da comunicação demore algum tempo, o agente deve parar de tentar a conexão e verificar com o *Agent Manager* se não houve alterações de dados daquele agente. Caso haja alterações, os dados devem ser atualizados no arquivo comum. Se mesmo de posse de todos os dados não for possível firmar a comunicação, o agente deve mandar um e-mail informando o problema para o responsável pelo agente.

No momento da comunicação entre agentes, para evitar a ocorrência de situações de *deadloock*, propõe-se que os agentes controlem o tempo de espera de resposta, e caso este tempo ultrapasse um limite pré-determinado, a conexão deve ser abortada.

6.3.5 Saída de agente da sociedade

Caso um agente queira retirar-se da sociedade por vontade própria (vontade própria, refere-se a saídas não provocadas pelo usuário ou por problemas na rede) este deve entrar em contato com o *Agent Manager* e informar sobre sua saída.

Caso o usuário queira retirar o seu agente de *software* da sociedade (desativá-lo), este deverá executar um programa em java (*frame*), para pedir que este agente seja retirado da sociedade. Este programa pedirá que seja fornecido o e-mail do usuário, a função desempenhada pelo agente e o número da máquina. De posse destas informações o programa irá eliminar o nome do agente do arquivo de agentes da máquina em questão, e depois fará uma conexão com o *Agent Manager* e pedirá que seja retirado do arquivo de agentes da sociedade o nome deste agente. A figura 6.9 mostra a interface do *frame* para retirada de agentes da sociedade.

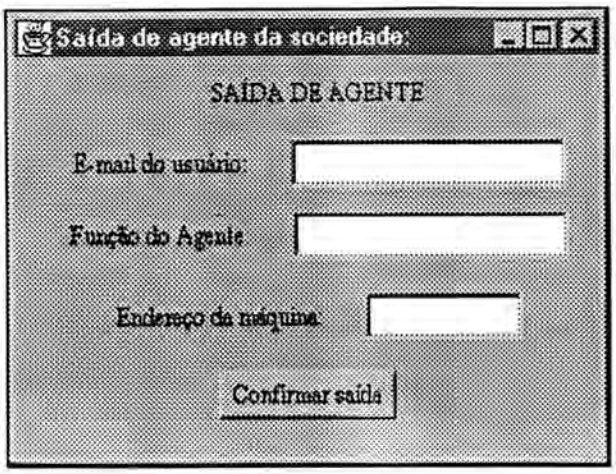

FIGURA 6.9 - Retirada de Agentes da Sociedade

No caso daqueles agentes que trabalham com um IP dinâmico (como já foi devidamente explicado no início deste capítulo), a saída destes agentes será tratada de duas maneiras:

1) Caso o agente apenas seja desconectado da rede, mas irá retornar na próxima vez que a sua máquina for conectada a Internet, o agente irá informar este fato para o *Agent Manager*. O *Agent Manager* estando a par deste fato, irá verificar em seu arquivo local de cadastro de agentes, qual o registro deste arquivo que está ocupado pelos dados deste agente. Feito isto, os dados do agente não serão deletados, mas será colocado um *flag* junto ao registro, informando que está temporariamente indisponível. O agente que informou sua saída da rede, receberá do *Agent* Manager, o número correspondente ao seu registro.

No momento de retorno deste agente para a sociedade, ele não necessitará ser recadastrado, apenas fará uma conexão via *socket* com seu *Agent Manager*, e informará seu número de registro dentro do arquivo local de endereços de agentes e o seu novo IP. O *Agent Manager*, de posse destas informações, irá reativar o registro do agente.

2) Caso o agente seja desativado pelo usuário, o procedimento será feito de maneira idêntica ao de qualquer outro agente.

6.3.6 Segurança das informações da sociedade

A segurança, será feita através da verificação do nome dos agentes que estão sendo cadastrados, verificação esta feita pelo próprio programa de cadastramento e pelo *Agent Manager*:

1) Verifica se a estrutura do nome do agente está OK;

2) Verifica se o novo nome que está sendo cadastrado não está repetido (igual a outro).

3) Para verificar se o novo agente está pronto para trabalhar, o *Agent Manager* estabelece um *socket* de comunicação com o novo agente e aguarda uma resposta padrão:

Agent Manager: "Bem vindo!"
Novo agente: "OK!"

A segurança também estará presente na arquitetura através das verificações feitas pelo Agente de Comunicação (através de métodos específicos) no momento em que este receber alguma *string* de comunicação, e através das funcões de *exception* que integram a própria linguagem Java.

6.3.7 Comunicação Agent Manager e sociedade de agentes

A comunicação entre o *Agente Manager* e a sociedade, quando algum membro desta estiver procurando o endereço de outro agente, será feita de maneira direta, onde o agente informa, por exemplo, que deseja o endereço do agente "scc@inf.ufrgs.br|agenda", e recebe em seguida o número da máquina e da porta.

Esta consulta também poderá ser feita utilizado palavras chaves, por exemplo, um agente está precisando saber o endereço de todos os agentes do usuário "scc@inf.ufrgs.br" que estão na máquina "x", neste caso o *Agente Manager* irá varrer o arquivo local de endereços de agentes atrás destas informações, retornando uma lista de endereços para o agente.

Outro aspecto importante de ser ressaltado, na comunicação entre o *Agent Manager* e a sociedade de agentes, é o de que a cada saída ou entrada de agentes da sociedade em questão, o *Agent Manager* não informará para todos os agentes da sociedade o ocorrido, pois isto, poderia acarretar uma sobrecarga na comunicação da rede, caso a sociedade fosse extremamente dinâmica, e para muitos agentes, a informação de que um agente que desempenha uma função "x" entrou na sociedade, seria algo irrelevante.

6.3.8 Estrutura básica dos agentes

Nesta seção apresentam-se as estruturas básicas de agentes, procurando apresentar quais os métodos[17] serão importantes para a implementação futura das classes genéricas *Agent Manager*, *Agente de Comunicação* e *agente*. Esta primeira classe descrita refere-se a estrutura do *Agent Manager*:

[17] Nesta definição da estrutura básica do Agent Manager, Agente de Comunicação, e o próprio agente, ainda não há a preocupação quanto ao aspecto de implementação dos métodos que compõem as classes, mas sim quanto a definição de quais métodos irão compor as classes. Deve-se observar que aqui são propostos os métodos básicos; novos métodos deverão surgir durante uma futura fase de implementação da arquitetura.

Agent Manager:

```
class Manager
{
  public Comunica()
  {
   //Permite que haja a comunicação com os outros agentes.
  }
  public  Escreve_Informação_do_Agente()
  {
  //Fornece a função de escrita no arquivo de dados de agente.
  }
  public  Deleta_Informação_do_Agente()
  {
  //Retira do arquivo de informações de agente, as informações de um agente que saiu da sociedade de agentes.
  }
  public  Altera_Informação_do_Agente()
  {
  //Altera informações de agentes no arquivo de registro de agentes. Será utilizado principalmente com os agentes de IP dinâmicos.
  }
  public  Protocolo()
  {
  //Permite que o Agent Manager possa interpretar os formatos de mensagens provenientes de outros agentes.
  }
  public Informa()
  {
  //Responsável por varrer o arquivo de registro de dados de agentes, e informar ao agente que consultou, o endereço do agente desejado. Dentro deste método estará incluído o código para realização de consultas por palavras chaves.
  }
  public  Segurança()
  {
  //Este método deverá conter os aspectos de controle de segurança de rede, pois o Agent Manager estará alocado na máquina gateway da rede e com uma porta lógica aberta, portanto deverá restringir as formas de acesso.
  }
```

```
    public Verifica_Nome()
    {
    // verifica se o nome do agente que vai ser cadastrado está especificado de maneira correta.
    }
}
```

A classe abaixo descrita, refere-se a estrutura da classe Agente de Comunicação:

Agente de Comunicação:

```
class Comunicação
{
  public Comunica()
  {
   //Permite que haja a comunicação com os outros agentes.
  }
  public  Escreve_Informação_do_Agente()
  {
  //Fornece a função de escrita no arquivo de dados local, de dados referentes ao último agente consultado.
  }
  public  Protocolo()
  {
  //Permite que o Agente de Comunicação possa interpretar os formatos de mensagens provenientes de outros agentes.
  }
  public Informa()
  {
  //Responsável por varrer o arquivo de registro de dados de agentes, e informar ao agente que consultou, o endereço do agente desejado. Aqui está-se referindo ao arquivo local da máquina onde estão cadastrados, os últimos agentes que foram consultados.
  }
  public  Segurança()
  {
  //Este método deverá conter os aspectos de controle de segurança de rede, pois o Agente de Comunicação estará alocado em  uma máquina da rede e com uma porta lógica aberta, portanto deverá restringir as formas de acesso.
  }
```

```
        public Entrega_interna()
        {
        //Função responsável por determinar para qual agente da máquina é destinada a mensagem recebida por socket, e posteriormente entregar a mensagem.
        }
        public Entrega_Externa()
        {
        //Função responsável pela criação de socket de comunicação com outras máquinas.
        }
    }
```

A classe abaixo descrita refere-se a estrutura da classe de agentes funcionais[18]:

Agente:

```
class AgenteNOME
{
    public Estabelece_Comunicação()
    {
    //Este método tem a função de estabelecer comunicação com o Agente de Comunicação da própria máquina.
    }
    public Time_out()
    {
  //Este método deve verificar o tempo de demora e o número de tentativas feitas para tentar acessar um determinado agente, para então determinar a ação a ser tomada: pedir confirmação dos dados para o Agent Manager ou mandar um e-mail padrão para o usuário responsável pelo agente.
    }
    public Responde_Comunicação()
    {
    //Método reponsável por elaborar a resposta a uma mensagem, após esta ter sido devidamente interpretada.
    }
```

[18] Agente funcional é apenas uma denominação genérica para os agentes implementados pelo usuário e que desempenham alguma função específica.

```
        public  Protocolo()
        {
        //Permite que o Agente possa interpretar os formatos de mensagens provenientes
de outros agentes.
        }
        public Função()
        {
        //Neste método estará determinada a função específica de cada agente.
        }
        public Mail()
        {
        //Método responsável por elaborar e-mail para destinos específicos.
        }
    }
```

6.4 Protocolo de comunicação (KQML)

Nesta dissertação não é de fundamental importância o aspecto de como será a comunicação real entre os agentes, mas sim os aspectos referentes a arquitetura, ou seja, como seria estabelecida esta comunicação. Devido a este fato, apenas será citado como pretende-se fazer esta integração para comunicação dos agentes, e serão comentados alguns aspectos sobre a linguagem que se deseja utilizar para esta integração.

A comunicação entre agentes será definida mediante o uso da linguagem de comunicação KQML. As linguagens de comunicação de agentes devem definir como: **[AMA 96]**

- Transferir conhecimento;
- Realizar solicitações;
- Aceitar ou rejeitar uma solicitação;
- Negociar introduzindo outras propostas;
- Informar falhas;
- Informar se está ou não satisfeito com a colaboração.

O que é KQML?

KQML ou Linguagem de Consulta e Manipulação do Conhecimento (*Knowledge Query and Manipulation Language*) é uma linguagem para troca de informações e

conhecimento. KQML é tanto um formato de mensagem, como um protocolo para manipulação de mensagens que suporta, em tempo de execução, o compartilhamento de conhecimento entre agentes **[KQM 96]**.

KQML é uma linguagem para os programas utilizarem para comunicar atitudes sobre informações, como questionamento, afirmações, crenças, pedidos, concordância, e oferta.

KQML pode ser usada como uma linguagem para um programa de aplicação interagir com um sistema inteligente ou com dois ou mais sistemas inteligentes, que compartilham conhecimento na solução cooperativa de um problema.

Enfoca um amplo conjunto de *performatives*, as quais definem as operações possíveis de serem realizadas pelos agentes, sobre o conhecimento e metas armazenadas de outros agentes. As atuações compreendem o desenvolvimento de modelos em alto-nível de interação entre agentes, como redes de contrato e negociações. Além disto, o KQML fornece um arquitetura básica para compartilhamento de conhecimento, através de uma classe especial de agente, chamada facilitador de comunicação, o qual coordena a interação com outros agentes.

Como resumo, pode se dizer que o KQML, é uma linguagem de comunicação que foi projetada para facilitar a cooperação e interoperação de alto-nível, entre agentes.

Atualmente a linguagem KQML está sendo utilizada em projetos onde é necessário o estabelecimento de comunicação entre agentes de *software*, como por exemplo: Java Agent Template (JAT), desenvolvido por Rob Frost da Universidade de Stanford e descrito no capítulo anterior.

Dentro da arquitetura proposta, pretende-se que os agentes possam trocar mensagens no formato KQML, para estabelecerem comunicação entre si.

No Anexo 5 é apresentada uma tabela contendo um conjunto de *Performatives* do KQML e uma tabela apresentando as palavras reservadas da linguagem.

6.5 Exemplo de utilização da arquitetura

Neste tópico, será apresentado um exemplo do uso da arquitetura proposta. Como exemplo, será utilizado o sistema de agendamento, apresentado na seção 4.2.2.

Para a exemplificação será utilizado o esquema apresentado na figura 6.10 abaixo:

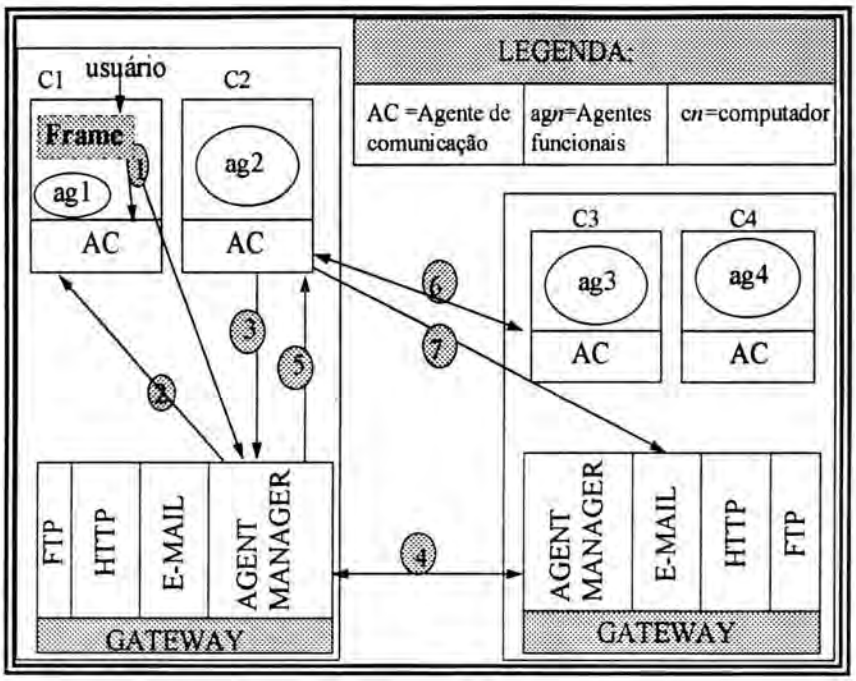

FIGURA 6.10 - Exemplo de utilização da Arquitetura proposta

1) Na primeira etapa do esquema, será suposto que o agente1 (ag1) ainda não é parte integrante da sociedade. Sendo assim, este deve apresentar-se para o *Agent Manager* da sua rede, e para o Agente de Comunicação de sua máquina, para ser cadastrado.

Nesta etapa cabe ao proprietário executar o *frame* de cadastramento e passar as informações sobre o novo agente. No esquema, esta etapa está sendo representada pelo item número 1.

Depois do programa de cadastramento ter informado ao Agente de Comunicação e ao *Agent Manager* da rede, da presença deste novo agente, o próprio *Agent Manager* cria um *socket* de comunicação com o novo agente para verificar se ele está devidamente cadastrado e respondendo (representado no esquema pelo item 2).

2) Na segunda etapa do exemplo, supõe-se que o agente2 (ag2) e o agente3 (ag3) desempenham a função de agentes agenda. O usuário do agente2 pediu para que fosse marcada uma reunião com o agente3.

Para poder marcar a reunião com o agente desejado, primeiramente o agente2 procura o endereço deste agente no arquivo local de endereços de agentes de sua máquina (função realizada na verdade pelo Agente de Comunicação). Não o encontrando, entra em contato, via *socket*, com o *Agent Manager* da rede e solicita o endereço do agente desejado (item 3 do esquema). Neste exemplo, apresenta-se a comunicação entre os agentes como se fossem diretas, pois fica transparente a participação do Agente de Comunicação nesta atividade.

Como neste caso o agente desejado não faz parte da sub-rede do *Agent Manager*, este terá de estabelecer um *socket* de comunicação com a máquina que possui o domínio referenciado no nome do agente e entrar em contado com o *Agent Manager*

desta outra rede (item 4 do esquema), e finalmente solicitar o endereço desejado. De posse deste endereço, ele é passado para o agente2 (item 5), e este agora pode entrar em contato com o agente3.

Deve-se observar que este acesso a outro *Agent Manager* ocorreu pelo fato do agente em questão não estar na mesma rede do agente2.

3) Na terceira etapa (item 6) o agente2 estabelece um *socket* de comunicação, com o agente3 (no esquema a conexão é apresentada de maneira direta ag2 -> ag3, os detalhes de conexão através das camadas da rede, não são abordados, pois fica transparente ao usuário), sendo feita a partir deste momento toda a interação exigida por um agente agenda. Após o estabelecimento da conexão, e depois de toda a negociação ter terminado, a conexão é desfeita e o endereço do agente3 é armazenado no arquivo local de endereços da máquina do agente2.

Supondo que mesmo de posse do endereço completo do agente3, o agente2 não tenha conseguido estabelecer conexão. Depois de algumas tentativas frustradas, o agente2 mandará um e-mail informando o ocorrido para o usuário responsável por este agente (item 7).

6.6 Considerações Finais

Nestas considerações finais pretende-se expor alguns detalhes não apresentados sobre a arquitetura, bem como críticas a esta, e alguns aspectos referentes a pretensões futuras.

Respostas aos problemas apresentados

No início deste capítulo, no item 6.2, foram apresentados três problemas principais, para os quais a arquitetura deveria apresentar respostas para tornar-se robusta. Abaixo serão comentadas as respostas dadas a estes problemas:

- **Problema número 1:**

Este problema enfocava o fato de que a arquitetura proposta necessitava ser independente do fato da sociedade de agentes ser aberta ou fechada. Na arquitetura proposta não existe a preocupação com este fator, pois não importa se os agentes são sempre os mesmos (sociedade fechada) ou se estes migram (sociedade aberta), pois para um agente entrar ou sair da sociedade basta ele dirigir-se ao *Agent Manager*, não interferindo assim no resto da sociedade. Caso a sociedade seja aberta, o número de acessos ao *Agent Manager*, poderá ser muito grande; no caso da sociedade ser fechada o número de acessos para registrar a entrada de agentes será mínimo.

- **Problema número 2:**

Neste problema foi abordado o fato da conveniência ou não da existência de um agente centralizador de endereços de agentes. Depois de alguma experiência com programação de agentes e com o estudo de outros sistemas, optou-se pela necessidade da existência de um agente centralizador, pois este tornaria mais cômoda a integração da sociedade.

Mas deve-se notar que tentou-se tornar a dependência, da sociedade em relação a este agente centralizador, a menor possível. Isto pode ser vislumbrado pelo fato da existência do arquivo local de endereços de agentes nas máquinas da rede, o qual tem por objetivo evitar a consulta constante e repetitiva ao *Agent Manager*.

- **Problema número 3:**

Este problema enfocava o fato da existência de máquinas que recebem endereços IP dinâmicos, que alteram-se a cada conexão máquina-provedor de acesso. Para evitar o cadastramento constante deste agente que apesar de fazer parte da sociedade, não possui um endereço fixo, foi proposta a solução apresentada no item 6.3.5.

Denominação dos agentes

A estrutura proposta para denominação dos agentes, é relativamente complexa (e-mail do usuário, função do agente, número da máquina e número da porta de comunicação), para permitir, que um usuário possa possuir mais de um agente por máquina. Por exemplo, no caso o usuário "scc@inf.ufrgs.br" poderia dispor na mesma máquina de agentes como: "scc@inf.ufrgs.br|agenda|....|....", "scc@inf.ufrgs.br|atualiza|...|....", etc.

Problema previsto

Um problema que vem a tona junto com a proposta da arquitetura no formato apresentado, mas não é um problema provocado pela arquitetura em si, é o da dependência da sociedade de agentes do bom funcionamento do *Agent Manager*, ou melhor dizendo, a dependência da sociedade de agentes do bom funcionamento da máquina *gateway* da rede.

Caso venha a ocorrer alguma pane na máquina que desempenha o papel de *gateway* e aloca o *Agent Manager*, toda a comunicação da sociedade torna-se inviável devido a utilização dos *sockets*, que dependem da integridade desta máquina. Mas este problema é próprio de qualquer programa que utiliza uma rede de comunicação para acessar outras máquinas, constituindo-se mais em um problema do sistema utilizado. Como por exemplo, é o caso do sistema operacional SUN, onde a comunicação da rede depende da máquina central estar em condições de funcionamento, caso contrário toda a comunicação da rede é comprometida.

Arquitetura proposta x SodaBot

Uma semelhança que existe entre o SodaBot e a arquitetura proposta é a existência de um processo que está em todas as máquinas que executam agentes, que está sempre em estado de *listening* e é responsável pela comunicação entre os agentes e ativação dos mesmos. No caso da arquitetura proposta neste trabalho, este papel é desempenhado pelo Agente de Comunicação, no caso do SodaBot este é desempenhado pelo BSA.

Uma diferença no aspecto destes processos, é que no SodaBot existe em cada máquina um BSA em execução por usuário, no caso da arquitetura proposta existe um

Agente de Comunicação responsável por todos os agentes de sua máquina, independente de quem é o usuário "proprietário".

O SodaBot implementa seu próprio esquema de nomes. Cada BSA tem um nome que parece com um endereço de e-mail, mas os nomes de usuário e *host*, não necessitam corresponder a entidades endereçáveis da Internet. Na arquitetura proposta procura-se aproveitar a designação de entidades já endereçáveis, ou seja, utilizar o e-mail como uma forma do *Agent Manager* saber a qual domínio de rede pertence um dado agente.

Os BSAs são interconectados via três camadas de servidores dispostos de maneira hierárquica, para que os agentes implementados possam encontrar-se. Na arquitetura proposta caso um agente queira saber como encontrar um outro, basta questionar o *Agent Manager* que encontra-se na máquina *gateway* da sua rede.

Arquitetura x JAT

Comparando a arquitetura proposta neste trabalho com a do JAT, podemos ter como um ponto a favor a não necessidade de a cada interação, os agentes terem de entrar em contato com o centralizador, pois o agente poderá adquirir o endereço do agente com o centralizador (*Agent Manager)* ou com o arquivo local de endereços de agentes, e dai para frente a comunicação passa a ser direta com o agente desejado (via *socket*), liberando desta maneira o centralizador.

Pretensões futuras

Assim como existem os serviços padrões de HTTP, E-MAIL, FTP, pretende-se futuramente propor um serviço para coordenar a interação entre agentes (*Agent Protocol*), de acordo com a arquitetura proposta e com a definição de uma porta lógica dedicada.

Assim cada vez que um agente desejasse entrar em contato com um agente de outro computador, bastaria ele ter o nome do agente e o número da máquina.

A denominação do agente teria o formato mascarado, por exemplo, no caso de um agente referenciado como "scc@inf.ufrgs.br|agenda|143.002.345.122|4321", esta *string* passaria a ter a seguinte estrutura: "AP://scc@inf.ufrgs.br/Agenda/tucano", "AP" significa que esta *string* refere-se a um endereço de agente (*Agent Protocol*), "scc@inf.ufrgs.br", como já foi dito, refere-se ao e-mail do usuário que projetou o agente, e a partir dele pode-se obter o domínio de rede, "Agenda" permanece referindo a função desempenhada pelo agente, e por fim "tucano" refere-se ao nome da máquina dentro da rede que aloca o agente (utiliza-se o nome da máquina pois este é de mais fácil visualização e assimilação pelo o usuário, e menos sujeito a erros na hora do cadastro, do que um número)

7 Conclusões

O objetivo geral desta dissertação de mestrado, foi o de definir uma arquitetura genérica que permitisse a implementação de agentes no ambiente da Internet.

O resultado de todo o estudo realizado, foi a definição de uma arquitetura, que é robusta, no aspecto de facilitar o estabelecimento de comunicação entre agentes que compõem uma mesma sociedade, como entre agentes de sociedades distintas. Tentou-se aproveitar todos os mecanismos que facilitam a comunicação em rede.

A proposta de arquitetura descrita nesta dissertação, surgiu do estudo de agentes já existentes, de implementações realizadas e através do estudo de dois sistemas que foram tomados como exemplos base, para auxiliar na definição da arquitetura.

A dissertação, como foi dito na introdução, foi organizada em cinco capítulos básicos, os quais tentaram demonstrar todo o estudo relevante realizado até chegar-se a uma proposta satisfatória de arquitetura.

No capítulo 2, foram apresentadas as definições básicas de Inteligência Artificial Distribuída e Sistemas Multiagentes. Este capítulo constitui-se em um apanhado geral sobre a área de IAD, onde são apresentados os agentes, definições possíveis e suas características; os tipos de sistemas existentes, e aspectos sobre o que são Sistemas Multiagentes e como é feita a integração entre estas entidades que os compõe.

No capítulo 3, procurou-se apresentar uma visão rápida do que existe disponível em agentes de *software*, tanto como sistemas simples de agentes que são utilizados como *spiders* na Internet, como através de outros sistemas mais complexos. Este capítulo tenta, acima de tudo, demonstrar a utilidade destes agentes, e demonstrar que eles estão sendo implementados e amplamente utilizados, inclusive na própria Internet, onde parecem ter alta aplicabilidade.

No capítulo 4, foi feito um apanhado geral sobre a linguagem Java, e suas características. Neste capítulo foram apresentadas duas propostas de implementação de agentes utilizando a linguagem. Esta experiência de implementar agentes foi de extrema importância no momento de definir alguns aspectos fundamentais da arquitetura, pois toda a proposta baseou-se na observação da viabilidade de implementação futura usando Java.

No capítulo 5, são apresentados dois sistemas que foram utilizados como modelos a serem analisados para propor a arquitetura: Sistema SodaBot e o Sistema JAT. O primeiro foi escolhido devido a quantidade farta de informações que estavam disponíveis para seu estudo, e devido a clareza das informações fornecidas; o segundo foi escolhido por quatro motivos: este sistema foi projetado utilizando a linguagem Java, o que era muito interessante, já que se pretendia implementar os agentes na

arquitetura utilizando Java; era possível disponibilizar pela rede o pacote referente ao JAT_3.0 para testes; os agente utilizavam a linguagem KQML para firmar comunicação, e por último, também existia uma documentação muito rica sobre este sistema.

No capítulo 6, finalmente foi apresentada a arquitetura, proposta procurando detalhar todas as suas peculiaridades estruturais. Junto com a apresentação da arquitetura foi feito um resumo sobre o funcionamento do protocolo TCP/IP, pois este seria muito comentado durante a apresentação da arquitetura.

Portanto com estes capítulos, tentou-se criar uma seqüência lógica para que o futuro leitor tivesse uma visão ampla de todo o estudo envolvido na definição de uma arquitetura para coordenar a interação de agentes na Internet:

1. Estudo da teoria sobre IAD e Sistemas Multiagentes;

2. Apresentação de sistemas que utilizam agentes;

3. Estudo de uma linguagem para implementação de agentes, e protótipos implementados;

4. Estudo da arquitetura de dois sistemas que se destinam a implementação de agentes;

5. Apresentação da Arquitetura final.

Todos os detalhes propostos nesta arquitetura, foram propostos após a implementação de agentes utilizando a Linguagem Java. A partir destas implementações foi possível ter-se uma ampla e clara visão das vantagens do uso da linguagem e sobre as possibilidades de programar-se a arquitetura proposta. Deve-se salientar que tudo o que foi proposto, seguiu nos mínimos detalhes as possibilidades reais de implementação, apresentadas durante o período de programação utilizando a linguagem em questão.

Talvez alguns aspectos referentes a arquitetura, pudessem ter sido propostos de uma maneira mais simples, ou mais prática, mas se não o foram, foi devido a uma preocupação constante com o aspecto de máquina e rede de comunicação. Tentou-se ao máximo, impedir a criação de problemas do tipo: sobrecarga de CPU, sobrecarga da comunicação na rede e criação de pontos de entrada que viessem a impor aspectos de vulnerabilidade nas máquinas que compõem a rede.

Deve-se ter em mente que a proposta inicial de criar-se uma arquitetura que possibilitasse a implementação de Sistemas Multiagentes, tinha por proposta permitir a criação de agentes de *software* que pudessem garantir todos os aspectos que caracterizam estes agentes, mas sem criar aspectos que viessem a comprometer o bom desempenho de máquina e rede de comunicação.

7.1 Contribuições

Esta arquitetura certamente não é única e possivelmente não seja a melhor para algumas classes de aplicações, mas apresenta um ponto que é de grande peso a seu favor, que é o da possibilidade de utilizar toda a estrutura já definida para a Internet, como é o caso de utilizar-se das máquinas *gateways* das redes e dos domínios já existentes. Esta arquitetura apresenta um ponto vulnerável, o qual já foi apresentado em momento oportuno, mas que vale ser repetido: por depender em parte da máquina *gateway* da rede para toda a comunicação com outros domínios na Internet, no momento que esta máquina sofrer alguma espécie de pane, a comunicação ficará prejudicada.

Apesar da proposta ter se baseado nas possibilidades reais de implementação da arquitetura, e do perfil da estrutura básica dos agentes já ter sido traçado, com a definição dos métodos que serão necessários em cada classe, certamente novos métodos surgirão, no momento da implementação desta arquitetura.

A maior contribuição que foi obtida com a definição desta Arquitetura de Agentes para Internet, acredita-se ter sido principalmente para a área de Sistemas Multiagentes (SMA), pois com esta arquitetura torna-se relativamente fácil a implementação de futuros Sistemas Multiagentes.

7.2 Perspectivas para o Futuro

O presente trabalho pode ser complementado e aperfeiçoado com novos estudos. Dentre estes, pode-se salientar três, muito importantes, e que podem ser realizados imediatamente:

1. Como já foi dito anteriormente, espera-se que desta arquitetura possa surgir um protocolo comum para endereçamento de agentes, com a definição de uma porta lógica privativa para comunicação de entidades agentes (assim como existem portas dedicadas para HTTP, E-MAIL), e que o *Agent Manager* possa tornar-se mais um serviço oferecido pela Internet.

2. Espera-se que em momento oportuno esta arquitetura seja implementada, e passe a ser amplamente utilizada para criação de Sistemas Multiagentes, e que esta também sirva como uma etapa inicial para a implementação do projeto ***MASWEB - Um ambiente distribuído baseado em agentes para suporte ao trabalho em grupo***, que propõe o ensino à distância utilizando agentes no ambiente da Internet.

3. Como o *Agent Manager* estará em uma máquina *gateway*, deverão ser estudados os procedimentos cabíveis para garantir todos os aspectos de segurança da máquina.

Anexo 1

Arquivo: AtualizaTutorial.java
Autor: Sílvio César Cazella
Data: 23/05/96
Descrição:
Este programa possibilita a implementação de um agente que tem como função manter atualizado um tutorial na conta do usuário. Para realizar esta tarefa, o agente deve ser mantido em execução constante na máquina, e o usuário deve informar a URL onde encontra-se o tutorial. Diariamente o agente deve se conectar a esta URL e verificar o tamanho do tutorial que esta neste *site*, e comparar com o tamanho do tutorial na conta do usuário. Caso o tutorial do *site* seja maior o agente deve copiá-lo para a conta do usuário, e informá-lo da atualização por e-mail.

```
import java.net.*;
import java.io.*;
import java.util.*;
import java.lang.*;
import SendMail;
```

Aqui começa a declaração da classe de Atualização de Tutorial.

```
class AtualizaTutorial
{
 FileOutputStream                f1;
 PrintStream                     o1;
 static Socket                   socket;
 static InputStream              in;
 static OutputStream             out;
 static DataInputStream din;
 static PrintStream              prout;
 int                             SMTPport = 25;
 String                          incoming = new String();
 String                          MailHost="tucano.inf.ufrgs.br";
 String                          HELO="HELO onca.inf.ufgrs.br";
 String                          MAILFROM="MAIL
                                   FROM:<AgenteAtualiza@inf.ufrgs.br>";
 String                          RCPTTO="RCPTTO:<scc@inf.ufrgs.br>";
 String                          DATA= "DATA";
```

As mensagens devem ser terminadas com "\r\n.\r\n".

```
 String                          Msg= "Atualizei o arquivo.\r\n.\r\n";
```

O *mailhost* retorna "220" ou "250" para indicar que tudo ocorreu OK.

```
String                          OKCmd   = "220|250";
```

Início da declaração do método de atualização do Tutorial.

```
public AtualizaTutorial(){
    try {
FileOutputStream                fl;
PrintStream                     ol;
```

Declaração da URL para conexão com o tutorial.

```
URL sun = new URL("http://www.javasoft.com/tutorial");
URLConnection url = sun.openConnection();
url.connect();
```

Especificação do arquivo onde será gravada a cópia do tutorial.

```
File arq = new File("/home/coruja/scc/fontjava/tutorial.txt");
sun.openConnection();
```

Teste de verificação de qual dos tutoriais é mais moderno. Se o tutorial do *site* for mais moderno, é iniciada a cópia para a máquina do usuário e é estabelecido um *socket* para mandar um e-mail para o usuário.

```
if (arq.length()< url.getContentLength())
  {
                DataInputStream dis;
                String inputLine;
```

Conexão com o servidor de e-mail da rede.

```
                System.out.println("Connecting to " + MailHost + "...");
                System.out.flush();
                try {
                socket = new Socket(MailHost, SMTPport);
                } catch (IOException e) {
                System.out.println("Error opening socket.");
                return;
  }
  try {
        in    = socket.getInputStream();
        din   = new DataInputStream(in);
        out   = socket.getOutputStream();
        prout = new PrintStream(out);
        }
  catch (IOException e) {
```

```
 System.out.println("Error opening inputstream.");
 return;
}
      prout.println(HELO);
      prout.flush();
      System.out.println("Sent: " + HELO);
try {
      incoming = din.readLine();
      }
catch (IOException e) {
      System.out.println("Error reading from socket.");
      return;
}
      System.out.println("Received: " + incoming);
```

Informando ao servidor que se deseja mandar um e-mail.

```
      prout.println(MAILFROM);
      prout.flush();
      System.out.println("Sent: " + MAILFROM);
try {
      incoming = din.readLine();
      }
catch (IOException e) {
      System.out.println("Error reading from socket.");
      return;
      }
      System.out.println("Received: " + incoming);
```

Informando ao servidor para quem é o e-mail.

```
      prout.println(RCPTTO);
      prout.flush();
      System.out.println("Sent: " + RCPTTO);
try {
      incoming = din.readLine();
      }
catch (IOException e) {
      System.out.println("Error reading from socket.");
      return;
      }
      System.out.println("Received: " + incoming);
```

Informando ao servidor que vai ser mandada uma mensagem.

```
      prout.println(DATA);
      prout.flush();
      System.out.println("Sent: " + DATA);
try {
      incoming = din.readLine();
      }
```

```
catch (IOException e) {
        System.out.println("Error reading from socket.");
        return;
        }
        System.out.println("Received: " + incoming);
```

Mandando a mensagem.

```
        prout.println(Msg);
        prout.flush();
        System.out.println("Sent: " + Msg);
try {
        incoming = din.readLine();
        }
catch (IOException e) {
        System.out.println("Error reading from socket.");
        return;
        }
        System.out.println("Received: " + incoming);
```

Mandada a mensagem, é desfeita a conexão.

```
        System.out.print("Disconnecting...");
try {
        socket.close();
        }
catch (IOException e) {
        System.out.println("Error closing socket.");
        }
        System.out.println("done.");
```

Início do processo de cópia do tutorial do *site* para a máquina do usuário.

```
        fl= new FileOutputStream("tutorial.txt");
        o1 = new PrintStream(f1);
        dis = new DataInputStream(sun.openStream());
        while ((inputLine = dis.readLine()) != null)
        {
        o1.println(inputLine);
        }
        dis.close();
 }
else
 {
 System.out.println("O original ainda nao foi atualizado..");
 }
}
catch (MalformedURLException me)
 {
 System.out.println("MalformedURLException: " + me);
 }
```

```
    catch (IOException ioe)
     {
      System.out.println("IOException: " + ioe);
     }
}
```

Método principal para executar o método AtualizaTutorial(), pois o programa em questão é *standalone*.

```
public static void main(String args[])
 {
   new AtualizaTutorial();
 }
}
```

Anexo 2

Arquivo: AgenteCliente.java
Autor: Sílvio César Cazella
Data: 30/07/96
Descrição:
Este programa na implementação dos agentes de agendamento, tem a função de estabelecer a comunicação entre o usuário que deseja marcar uma reunião com o resto do grupo. O programa trabalha como um programa cliente que conecta cada um dos Agentes Servidores.

```
import java.net.*;
import java.io.*;
import java.lang.*;
```

Início da declaração da classe AgenteCliente.

```
class AgenteCliente
{
 DataInputStream i1;
 DataInputStream cinput;
 PrintStream coutput;
 Socket clisoc;

 public AgenteCliente(){
try{

      i1 = new DataInputStream(System.in);
      int soc[ ] = new int[ 3];
      int port[ ] = new int[ 3];
      int desocupado=0;
```

Declaração dos números das máquinas e das portas onde estão os demais agentes. Neste caso a consulta inclui os agentes servidores da sociedade e o agente servidor do próprio usuário. Esta consulta ao agente servidor da própria máquina, visa verificar se o usuário não está solicitando um dia indisponivel para ele próprio. A consulta a própria máquina é feita com o endereço IP de máquina = 127.0.0.1. Quando é utilizado o endereço 127.0.0.1 a máquina entende que a comunicação por *socket* deve ser estabelecida entre processos na mesma máquina.

```
      int soc[ ] = {127.0.0.1,143.32..... ,143.32....}; //Os demais números de máquinas foram representados por 143.32....., pois são importantes, somente, no momento da implementação.
      int port[ ] = {4321, 4231, 4565};
```

Após estabelecida a comunicação, o usuário determina se a reunião deve ser segunda, terça ou quarta. O AgenteCliente remete para os Servidores o pedido, e pede que estes verifiquem se a data solicitada está desocupada.

```
System.out.println("Entre com a data desejada:(seg/ter/quar)");
String str = i1.readLine();
String str2 = "Verificar";
for (int x=0; x<=2; x++)
{
 clisoc = new Socket(soc[ x], port[x]);
 cinput = new DataInputStream(clisoc.getInputStream());
 coutput = new PrintStream(clisoc.getOutputStream());
 coutput.println(str);
 coutput.println(str2);
 String str3 = cinput.readLine();
 if (str3.equals("Desocupado"))
 {
  int desocupado=desocupado+1;
 }
 System.out.println("Disconectado:"+ clisoc.getInetAddress()+";"+clisoc.getPort());
 clisoc.close();
}
if (desocupado == 3)
{
String str = "Marcar";
for (int x=0; x<=2; x++)
 {
  clisoc = new Socket(soc[ x], port[x]);
  cinput = new DataInputStream(clisoc.getInputStream());
  coutput = new PrintStream(clisoc.getOutputStream());
  coutput.println(str);
  System.out.println("Disconectado:"+ clisoc.getInetAddress()+";"+clisoc.getPort());
  clisoc.close();
  }
 }
else
 {
  String str = "NMarcar";
  for (int x=0; x<=2; x++)
  {
   clisoc = new Socket(soc[ x], port[x]);
   cinput = new DataInputStream(clisoc.getInputStream());
   coutput = new PrintStream(clisoc.getOutputStream());
   coutput.println(str);
   System.out.println("Disconectado:"+ clisoc.getInetAddress()+";"+clisoc.getPort());
   clisoc.close();
  }
 }
}
System.exit(0);
}
```

```
 catch(Exception e)
 {
  System.err.println(e);
  System.exit(1);
 }
}
```

Método principal para executar o método AgenteCliente(), pois o programa em questão é *standalone*.

```
public static void main(String args[])
{
 new AgenteCliente();
}
}
```

Anexo 3

Arquivo: AgenteServidor.java
Autor: Sílvio César Cazella
Data: 10/07/96
Descrição:

Este programa funciona como um servidor. O programa principal denominado AgenteServidor, pode dar ***start*** em várias *threads* (definidas em uma classe denominada ConexBasica) cada uma criando um *socket* de servidor em uma dada porta. Cada *thread* criada fica sendo executada em *loop* até que o cliente envie uma mensagem padrão de conexão. Enviada esta mensagem padrão de conexão é criada uma *socket* cliente e estabelecida a conexão. Essa conexão opera em uma nova *thread* (denominada de ConexSec). Isto irá possibilitar que vários clientes entrem em conexão com o servidor (no caso a porta monitorada por este), pois a cada conexão será criada uma nova *thread*.

Este agente servidor está sempre em execução nas máquinas, e é ele que sempre está em *listening* em uma dada porta, aguardando por consultas. Este agente também tem a função de controlar a agenda do usuário.

```
import java.net.*;
import java.io.*;
import java.lang.*;
```

Aqui começa a declaração da classe Servidor.

```
class AgenteServidor
{
 ConexBasica cb;
```

Declaração do método servidor.

```
 public AgenteServidor(int port)
 {
  System.out.println("Servidor listening na porta: "+ port);
  cb = new ConexBasica(port);
  cb.start();
  int agenda[ ] = {0,0,0};
 }

 public static void main(String args[])
 {
  new AgenteServidor(4321);
 }
```

```
}
```

Declaração da classe ConexBasica.

```
class ConexBasica extends Thread
{
 int pt;
 ServerSocket ssocket;
 Conexsec csec;
```

Declaração do método ConexBasica.

```
 public ConexBasica(int port)
  {
  pt = port;
  }

 public void run()
  {
  try
   {
   ssocket = new ServerSocket(pt);
   }
  catch (Exception e)
   {
   System.err.println(e);
   System.exit(1);
   }
  while (true)
   {
   try
   {
   Socket clisocket = ssocket.accept();
   csec = new Conexsec(clisocket);
   csec.start();
   }
   catch(Exception e)
   {
   System.err.println(e);
   }
  }
 }
}
```

Declaração da classe Conexsec.

```
class Conexsec extends Thread
 {
 Socket csk;
 PrintStream soutput;
```

```
DataInputStream sinput;

public Conexsec(Socket s)
 {
  csk = s;
 }

public void run()
 {
     System.out.println("Criada conexao: "+csk.getInetAddress()+":"+csk.getPort());
  try
   {
   soutput = new PrintStream(csk.getOutputStream());
   sinput =  new DataInputStream(csk.getInputStream());
```

Verifica se a agenda está disponível e informa ao solicitante.

```
    String str = sinput.readLine();
    soutput.println("eco do servidor:"+str);
    String str2 = sinput.readLine();
    if (str2.equals("Verificar"))
     {
     String str = sinput.readLine();
     if (str.equals("Seg"))
      {
       if ((agenda[0] == 0)
       {
        soutput.println("Desocupado");
        int y = 0;
        }
       else
        {
         soutput.println("Ocupado");
        }
       }
      else
       {
        if (str.equals("Ter"))
        {
         if ((agenda[1] == 0)
          {
           soutput.println("Desocupado");
           int y = 1;
          }
         else
          {
           soutput.println("Ocupado");
          }
        }
        else
         {
          if (str.equals("Quar"))
```

```
        {
         if ((agenda[2] == 0)
         {
          soutput.println("Desocupado");
          int y = 2;
         }
        else
        {
         soutput.println("Ocupado");
         }

        }
      }
```

Marcando o compromisso.

```
    String str2 = sinput.readLine();
    if (str2.equals("Marcar")) agenda [y] = 1;
     }
      csk.close();
 System.out.println("Disconectado:  "+csk.getInetAddress()+  ":"  +csk.getPort()+"
Listening...");
      stop();
    }
   }
  }
  catch(Exception e)
  {
   System.err.println(e);
  }
 }
}
```

Anexo 4

Especificação da linguagem SodaBotL:

Declarações Funcionais:

<Program>	::= <Mail filter>[<Program>] | <Agent>[<Program>]
<Mail filter>	::= Mail filter:[<Declaration>*]<Maildescription>*
<Mail description>	::== Received Mail[{<Mail specification>}]:<Statements>*
<Agent>	::= Agent <Agent name>: [<Input>] <Statements>* <Agent requests>* <Subroutine>*
<Agent request>	::=Request<RequestString>:[{group<String>}]<Statements>*
<Subroutine>	::= Subroutine<Sub name>([<Arg list>]):[{group<String>}] {Statements>*}
<Declaration>	::= Library<Library name>|<Declaration>

Declarações:

<Statement>	::= <Assignment>|<System>| <Save>|<Load>|<Reply>|<mail>|<Contact Agent> |<Sub Call>|<If>|<GUI call>
<Assignment>	::=<Variable>=<String value>;
<System>	::= System<String value>;
<Save>	::= Save[{Append}]<Filename> <String value>;
<Load>	::=Load<Filename><Variable>;
<Reply>	::= Reply with<String value>;
<Mail>	::=Mail to<Address>:<String value>;

<Contact agent> ::= Contact agent<<Agent name>;<request *string*>>
{users:<address list>}[<String value>];

<Sub call> ::= &<Sub name>([<Arg List>]);

<If> ::= If (<Condition>){<Statement>}
[elseif(<Condition>){<Statement>}]*[else{<Statement>}]

Declarações GUI:

<GUI call> ::= <Display>|<Get response>|<Query>

<Display> ::=[{<Display options>*}]<String value>;

<Get response> ::=[{<Query options>*}]<String value>;

<Query> ::=[{<Query options>*}]<String value>;

Condições:

<Condition> ::= <Boolean>
::= <String value>eq<String value>
::= <String value>neq<String value>
::= <Reg exp>in<String value>|<String value>=~<Reg exp>
::= <Reg exp>nin<String value>|<String value>!~<Reg exp>
::=(<Condition>)or(<Condition>)|(<Condition>)||(<Condition>)
::=(<Condition>)and(<Condition>)|(<Condition>)&&(<Condition>)

Tipos de Dados:

<String Value> ::= <String>|<Variable>

<String> ::= <Multiline *string*>|<Simple *string*>

<Variable> ::= $<Simple *string*>

<Filename> ::= [/] <Simple *string*>[/][<Filename>]|<variable>

<Sub name> ::=<Simple *string*>

<Agent name> ::= <Simple *string*>

<Library name> ::=<Simple *string*>

<Arg list> ::= <String value>[,<Arg list>]

<Multline *string*> ::= <Rich *string*>\n[<Multline *string*>]

<Rich *string*> ::=All characteres except \n

<Simple *string*> ::=[a-z,A-Z,0-9,_]*

Filtros de Mail:

<Mail specification> ::= <Field>:/<Reg exp>/[;<Mail Specification>]

<Field> ::= to|cc|bcc|from|sender|replay-to|return-receipt-to|errors-to|date|return-header|message-id|subject\status|newsgroups|followup-to

Especificação de entrada do agente:

<Input> ::= Required input{<Input spec>}

<Input spec> ::= <Rich *string*>:<Input type>

<Input type> :=<Rich*string*>|*name|*username|*filename|*date|*time|*address|*host|*number

Detalhando a linguagem:

Os usuários podem criar novos agentes e filtros de mail através do Agente Editor do SodaBot, ou eles podem escrever e compilar arquivos em SodaBotL diretamente.

Variáveis:

Todas as variáveis do SodaBot são precedidas do símbolo $. As variáveis são tipadas de acordo com seu contexto, sendo assim, não é necessário declarar explicitamente o tipo; por exemplo, "10" pode ser tanto um número ou uma *string*, dependendo de como está será utilizada. Abaixo temos alguns exemplos de variáveis:

```
$name = $username;

$address = "545 Technology Square...\n";

Mail to $host:

"My snail - mail address is:\n$name\n$address";
```

Variáveis e seus significados:

TABELA 5.1 - Variáveis da linguagem SodaBotL

Nome da variável	Descrição
$home	Diretório *home* do usuário
$user[_]name	Nome completo do usuário
$user_login, $me	Nome de *login* do usuário
$message	O texto completo da mensagem corrente
$body	O corpo da mensagem
$name	Nome completo do remetente se especificado
$reply-to	Se este não for especificado, o valor que está em $from é utilizado
$address	O endereço de e-mail do remetente

Condicionais:

TABELA 5.2 - Condicionais da linguagem SodaBotL

Expressão	Verdadeiro se
($a eq $b)	$a e $b são equivalentes
($a eq "moo")	$a é igual ao *string* "moo"
($a neq $b)	$a e $b não são equivalentes
($a=~/$b/)	$a contém $b
($a=~/moo/)	$a contém "moo"
($a=~/\d+/)	$a contém um número
($a!~/$b/)	$a não contém $b
($a==$b)	$a e $b são numericamente iguais
($a<=$b)	$a<=$b

Ainda podem ser utilizadas as seguintes estruturas condicionais: && para AND, || para OR e ! para NOT. Exemplo:

if (($from eq "las") || ($from eq "gjs")) $$ ($subject =~/6\.001/))

Escrevendo filtros de e-mail:

O BSA pode executar uma série de expressões para verificar a chegada de um e-mail especifico. O usuário especifica qual o e-mail que desencadeará o funcionamento do BSA, fornecendo um número de uma expressão regular que deve estar no

cabeçalho ou corpo do e-mail. Para cada uma destas *triggers*, o usuário deve informar qual a ação mais adequada para o BSA. O usuário pode criar múltiplos filtros de e-mail, os quais são examinados pelo BSA na chegada dos e-mail.

Um exemplo de filtro de e-mail em SodaBotL:

```
Mail Filter:

Received mail {from:/fax notifier/;to:/$me/;subject:/arrival/}:

  Display "An incoming fax has arrived.";

Received mail {from:/$me/;}:

  Save {append} "~/Mail/outogoing" $message;
```

Escrevendo um agente:

Como já foi apresentado em momento oportuno na figura 5.5, as aplicações de agentes são divididas em 04 partes: declarações globais, agente principal, agente de pedidos e agente de subrotinas .

1) ***Declarações Globais:***

São variáveis que podem ser referenciadas por todos os agentes de pedido e subrotinas.

2) ***Nome do agente:***

```
[required input {String_1: *type_1
                 String_2: *type_2...}]

[SodaBoltL expressions]
```

A declaração do agente inicia com a definição de seu nome. O *Required input* especifica qual será o formato da entrada que deve ser fornecida para o agente. As expressões SodaBotL são executadas pelo agente no momento que este é invocado.

A especificação *required input* é um esquema para descrever o formato da estrutura do e-mail que invocará o agente. Por exemplo o e-mail abaixo, corresponde a seguinte estrutura programada:

```
To: authorize@ai.mit.edu                Agent authorize:
.....                                   Required input{
Person1: las          <------->           person1: *username
Person2:brooks                          person2:*username
pathname:~mhcoen/tr.ps                  pathname:*pathname}
```

Primitivas SodaBotL:

➢ Contact Agent <Agentname; Requestname>{user[s]:user1, user2,...}[string];

Agentes fazem pedidos um para os outros via *Contact Agent*. O campo *users* especifica quais BSAs devem receber o pedido.

➢ Load filename variable;

Carrega o conteúdo de um arquivo na variável denominada em *variable*.

Exemplo:

```
Load $filename $contents;

Load "/hoem/c2/mhcoen/.schedule" $schedule;
```

➢ Mail [to] address:string

Simplesmente remete um mail especificado em *string* para o endereço dado. Exemplo:

```
Load "~/letter"$letter;

Mail to mhcoinai.mit.edu: $letter;

Mailto $user: ".....\n";
```

➢ Reply with string

Responde o mail atual com a seguinte *string*. Exemplo:

```
Reply with "My final offer is $USDollars.\n";
```

➢ Save [{append}] filename string;

Salva (ou append) a *string* especificada dentro do arquivo. Exemplo:

```
Save "/usr/tmp/current" "$from, $subject";

Save {append} "/user/games/XChess/$book_openings" "It was a dark and stormy night.\n";
```

Primitivas GUI :

➢ Display [{choices(cstring_1, cstring_2);...; no delay; time-out in interval; for interval}] String;

O comando *display* cria uma *window* mostrando uma *string* específica na tela do usuário. Pode-se observar que no próprio comando *display* está colocado o tempo de aguardo até que a pessoa venha a responder.

Exemplo:

```
Display {choices(yes, no); for one hour} "Mail from Lynn! Do you want to red it now?";
```

➢ Get Response [{prompt = string; no delay; time-out in interval; for interval}] string;

Este comando apresenta uma janela na tela com a *string* especificada, para permitir que a pessoa forneça sua própria resposta. Esta janela contém um mini-editor que permite que o usuário forneça seu texto. Exemplo:

```
Get Response {prompt = "What is your answer?"}

"$quiz_question";
```

Anexo 5

O KQML possui um conjunto de *performatives* e um conjunto de parâmetros. As palavras reservadas para estas *performatives* e seus parâmetros são apresentadas nas tabelas 1 e 2. Na tabela 1, o símbolo E significa o emissor, R um receptor, e BCV é a base de conhecimento virtual. **[AMA 96]**

TABELA 1 - Anexo 5: Performatives reservadas

Nome	Significado
achieve	E quer R para produzir alguma sentença verdadeira no ambiente.
advertise	E é próprio para processar uma *performative*.
ask-about	E quer toda sentença relevante na BCV de R.
ask-all	Dada uma consulta, E quer todas as respostas de R.
ask-if	E quer conhecer se uma sentença está na BCV de R.
ask-one	E quer uma das respostas de R a uma consulta.
break	E quer quebrar uma comunicação estabelecida.
broadcast	E quer que R envie uma *performative* a todas as conexões.
broker-all	E quer que R colete todas as respostas a uma *performative*.
broker-one	E quer que R ajude na resposta a uma *performative*.
deny	A *performative* não aplicável.
delete	E quer que R elimine uma sentença *ground* de sua BCV.
delete-all	E quer que R elimine todas as sentenças que fazem *matching* com uma sentença na sua BCV.
delete-one	E quer que R elimine uma sentença que faz *matching* de sua BCV.
discard	E não vai querer as respostas que restem de R de uma *performative* prévia.
eos	Fim de uma cadeia de respostas a uma consulta.
error	E considera uma mensagem de R como mal formada.
evaluate	E quer que R simplifique a sentença.
forward	E quer que R redirecione uma *performative*.
insert	E solicita a R que agregue algum conteúdo a sua BCV.
monitor	E quer modificações à resposta de R a um *stream-all*.
next	E quer a resposta seguinte de R a uma *performative* prévia.
pipe	E quer que R direcione todas as novas *performatives* a um outro agente.
ready	E está pronto para responder a uma *performative* prévia de R.
recommend-all	E quer todos os nomes de agentes que possam responder a uma *performative*.
recommend-one	E quer o nome de um agente que possa responder a uma *performative*.
recruit-one	E quer que R averigüe quais são os agentes adequados para responder uma *performative*.

Nome	Significado
recruit-all	E quer que R encontre um outro agente para responder a uma *performative*.
register	E pode transmitir *performatives* a algum agente específico.
reply	Comunica-se uma resposta esperada
rest	E quer as respostas restantes de R a uma *performative* prévia.
sorry	E não pode prover uma resposta com mais informação.
standby	E quer que R esteja pronto para responder uma *performative*.
stream-about	Versão de respostas múltiplas de *ask-about*.
stream-all	Versão de respostas múltiplas de *ask-all*.
subscribe	E quer modificações à resposta de R a uma *performative*.
tell	Uma sentença na BCV de E.
transport-address	E associa um nome simbólico a um endereço.
unregister	Uma rejeição a um *register*.
untell	Uma sentença não está na BCV de E.

TABELA 2 - Anexo 5: Palavras reservadas para parâmetros

Palavra chave	Significado
:content	Informação envolvida na atitude que implica a *performative*.
:force	Se o emissor vai desmentir o significado da *performative*.
:in-reply-to	Nome esperado para uma resposta.
:language	Nome da linguagem utilizada no parâmetro :content.
:ontology	Nome da ontologia (por exemplo, um conjunto de definições de termos) utilizado no parâmetro: content.
:receiver	Receptor atual da *performative*.
:reply-with	Se o emissor espera uma resposta, e se fora isso, um nome para a resposta.
:sender	Emissor atual da *performative*.

Um dos esquemas das *performatives* mencionadas nas tabelas com seus parâmetros, é apresentado a seguir:**[AMA 96]**

tell
:content <expressão>
:language<palavra>
:ontology<palavra>
:in-reply-to<expressão>
:force<palavra>
:sender<palavra>
:receiver<palavra>

deny
:content<*performative*>
:language KQML
:ontology<palavra>
:in-reply-to<expressão>
:sender<palavra>
:receiver<palavra>

Bibliografia

[AEK 95] AEKEN, F.Van et al. A Multi-Agent Approach for Mediation Support on the net. In: DIMAS, 1., 1995, Cracóvia. **Proceedings...** [S.l.:s.n.], 1995.

[AMA 96] AMANDI, Analía A. **Programação Orientada a agentes.** Porto Alegre: CPGCC da UFRGS, 1996. Exame de qualificação.

[BER 92] BERTHET, S. et al. Knowing each other better. In: International Workshop on Distributed Artificial Intelligence, 11., 1992, Glen Arbor, **Proceedings...** [S.l.:s.n.], 1992.

[BIT 96] BITTENCOURT, Guilherme. **Inteligência Artificial-Ferramentas e Teorias.** Campinas: Instituto de Computação, UNICAMP, 1996. Trabalho apresentado na Escola de Computação, 1996, Campinas-SP.

[BON 88] BOND, A.H. ; GASSER, Les (Eds.). **Readings in Distributed Artificial Intelligence.** San Mateo, California: Morgan Kaufmann, 1988.

[BOR 94] BORDINI, Rafael Heitor. **Suporte Lingüístico para Migração de Agentes**. Porto Alegre: CPGCC da UFRGS, 1994. Dissertação de mestrado.

[BOR 95] BORGES, M. et al. Suporte por Computador ao Trabalho Cooperativo. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 15., 1995, Canela, BRS. **Anais...** Porto Alegre: Instituto de Informática da UFRGS, 1995.

[BRO 95] BROWN, Carol et al. AI on the WWW. **IEEE Expert,** New York, v. 10, n. 4, p. 50-55, Aug. 1995.

[CES 96] CESTA, André Augusto. **Tutorial: A Linguagem de programação Java ™.** Campinas: Unicamp, Instituto de Computação, 1996.

[CHE 95] FAH-CHUN, Cheong. **Internet Agents:** Spiders, Wanderers, Brokers, and Bots. Octuber 95. New Riders. Publishing Professional. Disponível por WWW em http://www.mcp.com/306472151969881/cgi-bin/bag?isbn=1-56205-4. (29 jul. 1996).

[CIA 96] CIANCARINI, Paolo et al. Page Space: An Architecture to coordinate distributed applications on the Web. **Computer Networks and ISDN systems**, Washington, v. 28, n. 7-11, p. 941-952, May 1996.

[COE 94] COEN, Michel H. **SodaBot: A *software* Agent Environment and Construction System.** Disponível por FTP anonymous em publications.ai.mit.edu. Arquivo aitr-1493.ps.Z. Massachusetts Institute of Technology, 1994. (23 jul. 1996).

[DAM 96] DAMASCENO Jr., Américo. **Java programação na INTERNET.** São Paulo: Érica, 1996

[DEM 90] DECENTRALIZED Artificial Intelligence. In: DEMAZEAU, Yves; MÜLLER , Jean Pierre (Eds.). **Decentralized A.I.** Amsterdam:Elsevier, 1990.

[DEM 93] DEMAZEAU, Yves. Distributed Artificial Intelligence & Mult-Agent Systems. In: SIMPÓSIO BRASILEIRO DE INTELIGÊNCIA ARTIFICIAL, 10., Porto Alegre. **Anais...** Porto Alegre : SBC, 1993.

[DEM 95] DEMAZEAU, Yves. From Interactions to Collective Behaviour in Agent-Based Systems. **European Conference on Cognitive Science**, Saint Malo, 1995.

[DUM 95] DUMAS, Arthur. **Programando Winsock.** Rio de Janeiro : Axcel Books, 1995.

[EDD 94] EDDINGS, Joshua. **Como funciona a Internet.** São Paulo, SP: Ed. Quark, 1994.

[ETZ 95] ETZIONI, Oren ; WELD, Daniel S. Intelligent Agents on the Internet: Fact, Fiction, and Forecast. **IEEE Expert,** New York, v. 10, n. 4, p. 44-49, Aug. 1995.

[FER 91] FERBER, Jacques ; GASSER, Les. **Intelligence Artificiclle Distribuée. -EC2**, Avignon, 1991. Tutorial Notes of 11 th Conference on Expert Systems and their applications (Avignon '91).

[FRO 95] FROZZA, Rejane. **Resolução de Problemas utilizando Sistemas Multiagentes.** Porto Alegre: CPGCC da UFRGS, 1995.

[GAS 92] GASSER, Les. **Boudaries, identity and aggregation:** Plurality issues in multi-agent systems. In: Decentralized A.I., 3., WERNER, Eric;DEMAZEAU, Yves (Eds), pp. 199-212. - Amsterdam, Elsevier Science Publishers, 1992.

[HED 95] HEDBERG, Sara Reese. Intelligent Agents: The First Harvest of softbots looks promising. **IEEE Expert,** New York, v. 10, n. 4, p. 06-09, Aug 1995.

[HOF 96] HOFF, Arthur Van; SHAIO, Sami; STARBUCK, Orca. **Ligado em Java**: Criando Sites Web com Applets Java. São Paulo : Makron Books, 1996.

[INF 96] SILVA, Hélio. Agora você também pode falar Javanês. **Informática Exame**, São Paulo, n. 144, p. 86- 88, Maio 1996.

[JAN 96] Notas de aula da Universidade Federal de Santa Catarina. Disponivel por WWW em http://janus.inf.ufsc.br:1082. (05/08/96)

[JAT 96] **Java Agent Tamplate.** Disponível em http: http:/cdr.stanford.edu/ABE/JavaAgent.html (18/08/96).

[JEN 93] JENNINGS, N.R.; VARGA, L. Z. Transforming Standalone Expert Systems into a comunnity of cooperating agents. **Engineering Aplications of Artificial Intelligence**, [S.l.], v. 6, n. 4, p. 317 - 331, 1993.

[KID 96] KIDSIM. Disponível por WWW em http://cocoa.apple.com/cocoa/. (05/08/96)

[KQM 96] FININ, Tim. **Specification of the KQML.** Disponível por WWW em http://www.cs.umbc.edu/kqml/kqmlspec.ps. (24/06/96)

[NAG 96] NAGARATNAM, Nataraj; MASO, Brian; SRINIVASAN, Arvind. **Java™ Networking and AWT API.** Corte Madera, CA: Waite Group Press, 1996.

[PET 96] PETRIE, Charles J. **Agent-Based Engineering, the Web, and Intelligence**. Disponível em http: http://cdr.stanford.edu/NetLink/Expert.html (18/08/96).

[RIC 93] RICH, Elaine ; KNIGHT, Kevin. **Inteligência Artificial.** São Paulo: Makron Books, 1993.

[RIT 96] RITCHEY, Tim. **JAVA!** Indianapolis, USA : New Riders Publishing, 1995.

[SBA 96] SBARDELOTTO, Cristiane Maria. **Um estudo de Caso na Linguagem Java:** Trabalho Individual. Porto Alegre: CPGCC da UFRGS, 1996. (TI - 535).

[SEA 95] SEARS, Jay Allen. Harnessing the World Wide Web. **IEEE Expert,** New York, v. 10, n. 4, p. 42-43, Aug 1995.

[SIC 92] SICHMAN, Jaime ; DEMAZEAU, Yves; BOISSIER, Oliver. When can Knowledge-based systens be called agents? In: SIMPÓSIO BRASILEIRO DE INTELIGÊNCIA ARTIFICIAL, 9., Rio de Janeiro. **Anais...** Rio de Janeiro : SBC, 1992.

[SOD 94] **SodaBot: A *software* agent environment and construction system.** Disponível por ftp: ftp://ftp.ai.mit.edu/pub/users/mchoen/aitr-1493.ps.Z (18/04/96)

[STE 95] STEFANINI, M.-Hélène; DEMAZEAU, Yves. TALISMAN: A Multi-Agent System for Natural Language Processing. In: BRAZILIAN SYMPOSIUM ON ARTIFICIAL INTELLIGENCE, 12., 1995. **Proceedings...** Berlim, Springer, Veilag, 1995. p. 312 - 322 (Lecture Notes in Artificial Intelligence,v. 991).

[TIT 96] TITTEL, Ed ; GAITHER, Mark. **60 minute guide to Java.** Foster City, CA: IDG Books, 1995.

[VEN 96] VENETIER, Tomas. **HTML Desmistificando a Linguagem da Internet.** São Paulo : Makron Books, 1996.

[WAY 95] WAYER, Peter. **Agents Unleashed**. A public Domain Look at Agent Technology. USA. AP Professional, 1995.

[WER 91] WERNER, E ; DEMAZEAU, Y. **Decentralized A.I. 3. In: EUROPEAN** Workshop on Modelling Autonomous Agents in a Multi-Agent World, 3., 1991, Kaiserslautern. **Proceedings...** Amsterdam: North-Holland, 1991.

Informática  UFRGS

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Uma Arquitetura para Coordenar a Interação de Agentes na INTERNET

por

Sílvio César Cazella

Dissertação apresentada aos Senhores:

Prof. Dr. Jaime Simão Sichman (USP)

Prof. Dr. José Valdeni de Lima

Prof. Dr. Antônio Carlos da Rocha Costa (PUC-RS)

Vista e permitida a impressão.
Porto Alegre, 20/03/97.

Prof. Dr. Luis Otavio Campos Alvares,
Orientador.