

Centro Federal de Educação Tecnológica do Paraná  
Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial

---

**DISSERTAÇÃO**  
apresentada ao CEFET-PR  
para obtenção do título de

**Mestre em Ciências**  
**por**

Emerson Cabrera Paraiso

---

**Concepção e Implementação de um Sistema Multi-Agentes  
Para Monitoração e Controle de Processos Industriais**

---

Banca Examinadora:

Presidente e Orientador:

Prof. Dr. Celso A. A. Kaestner CEFET-PR

Examinadores:

Prof. Dr. Edson Scalabrin PUC-PR

Prof. Dr. Júlio C. Nievola CEFET-PR

Prof. Dr. Bráulio Ávila (Suplente) PUC-PR

**Curitiba, 09 de Janeiro de 1997**

**Emerson Cabrera Paraiso**

**Concepção e Implementação de um Sistema Multi-Agentes  
Para Monitoração e Controle de Processos Industriais**

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial do Centro Federal de Educação Tecnológica do Paraná como requisito parcial para obtenção do título de “Mestre em Ciências” - Área de Concentração: Informática Industrial

Desenvolvido em colaboração com o Instituto de Tecnologia do Paraná e apoiado pelo CNPq.

Orientador: Prof. Dr. Celso A. A. Kaestner

Curitiba  
1997

## **Agradecimentos**

O autor gostaria de agradecer ao Prof. Dr. Feliz Ribeiro Gouveia da Universidade Fernando Pessoa no Porto, Portugal, que muito auxiliou no início deste trabalho.

Também deve-se destacar a ajuda prestada pelo Engenheiro Milton P. Ramos, gerente do Núcleo de Tecnologia em Informática e Automação - NTIA do Instituto de Tecnologia do Paraná - TECPAR, onde este projeto foi parcialmente desenvolvido.

## Sumário

<b>1. INTRODUÇÃO</b>	<b>1</b>
<b>2. INTELIGÊNCIA ARTIFICIAL DISTRIBUÍDA</b>	<b>5</b>
<b>2.1. ESTRUTURA DE UM SISTEMA MULTI-AGENTES</b>	<b>9</b>
<b>2.2. PROTOCOLO E LINGUAGEM DE COMUNICAÇÃO</b>	<b>10</b>
2.2.1. O PROTOCOLO KQML (KNOWLEDGE QUERY AND MANIPULATION LANGUAGE)	11
2.2.2. O KIF (KNOWLEDGE INTERCHANGE FORMAT)	12
<b>2.3. POLÍTICA DE COOPERAÇÃO E ORGANIZAÇÃO DOS AGENTES</b>	<b>14</b>
2.3.1. COOPERAÇÃO ENTRE AGENTES	14
<b>2.4. MECANISMOS DE COORDENAÇÃO</b>	<b>19</b>
2.4.1. MECANISMO MESTRE/ESCRAVO (MME)	20
2.4.2. MECANISMO DE MERCADO (MM)	20
<b>3. EXEMPLOS DE APLICAÇÕES</b>	<b>23</b>
<b>3.1. ARCHON: AN ARCHITECTURE FOR COOPERATIVE HETEROGENEOUS ON-LINE SYSTEMS</b>	<b>23</b>
<b>3.2. TALISMAN: A MULTI-AGENT SYSTEM FOR NATURAL LANGUAGE PROCESSING</b>	<b>24</b>
<b>3.3. AGENT-BASED SYSTEM ENGINEERING (ABSE)</b>	<b>24</b>
<b>3.4. MATHEMA: A LEARNING ENVIRONMENT BASED ON A MULTI-AGENT ARCHITECTURE</b>	<b>24</b>
<b>3.5. QUADRO RESUMO</b>	<b>25</b>
<b>4. UM SISTEMA MULTI-AGENTE PARA MONITORAÇÃO E CONTROLE DE PROCESSOS INDUSTRIAIS</b>	<b>27</b>
<b>4.1. CONCEPÇÃO DE UM SMA</b>	<b>27</b>
<b>4.2. OS OBJETIVOS DE UM AGENTE</b>	<b>29</b>
<b>4.3. A ARQUITETURA DE UM AGENTE</b>	<b>31</b>
4.3.1. COMPETÊNCIA	31
4.3.2. CONTROLE	32
4.3.3. MEMÓRIA DE TRABALHO LOCAL	32
4.3.4. MEMÓRIA DE TRABALHO SOCIAL	32

4.3.5. CONTROLE DE MEMÓRIAS	33
4.3.6. COMUNICAÇÃO	33
<b>4.4. OS TIPOS DE AGENTES DO SISTEMA</b>	<b>33</b>
4.4.1. AGENTES GERADORES DE DADOS	34
4.4.2. AGENTES PROCESSADORES DE DADOS	35
4.4.3. AGENTES ATUADORES	35
4.4.4. AGENTES DE APRESENTAÇÃO DE SITUAÇÃO	35
4.4.5. AGENTES REPOSITÓRIOS DOS GERADORES DE DADOS	35
<b>4.5. MECANISMOS DE COORDENAÇÃO</b>	<b>36</b>
<b>4.6. PROTOCOLO DE COMUNICAÇÃO</b>	<b>37</b>
4.6.1. PROTOCOLO PREDICATIVO	37
4.6.2. EXEMPLO DE UTILIZAÇÃO DO CONJUNTO DE MENSAGENS	40
<b>5. IMPLEMENTAÇÃO DO SMA</b>	<b>43</b>
<b>5.1. PLATAFORMA DE DESENVOLVIMENTO</b>	<b>43</b>
<b>5.2. OS AGENTES VISTOS COMO PROCESSOS</b>	<b>43</b>
<b>5.3. CLASSE DE BAIXO NÍVEL PARA COMUNICAÇÃO</b>	<b>43</b>
<b>5.4. FLUXO DE RECEPÇÃO DAS MENSAGENS</b>	<b>44</b>
5.4.1. MENSAGEM <i>AGENTE</i>	46
5.4.2. MENSAGEM <i>SITUAÇÃO</i>	47
5.4.3. MENSAGEM <i>SABE</i>	48
5.4.4. MENSAGEM <i>INFORME</i>	49
5.4.5. MENSAGEM <i>INFORMANDO</i>	50
<b>5.5. FLUXO DE ENVIO DAS MENSAGENS</b>	<b>50</b>
5.5.1. MENSAGEM <i>AGENTE</i>	50
5.5.2. MENSAGEM <i>SITUAÇÃO</i>	52
5.5.3. MENSAGEM <i>SABE</i>	52
5.5.4. MENSAGEM <i>INFORME</i>	53
5.5.5. MENSAGEM <i>INFORMANDO</i>	54
<b>5.6. ROTINAS DE COMPILAÇÃO DE ENDEREÇO</b>	<b>54</b>
<b>5.7. ROTINAS DE CADASTRAMENTO DE AGENTES</b>	<b>55</b>
<b>5.8. ROTINA DE DESCADASTRAMENTO DE AGENTES</b>	<b>55</b>
<b>5.9. MEMÓRIAS DE TRABALHO</b>	<b>55</b>
<b>5.10. MÁQUINA DE INFERÊNCIA</b>	<b>56</b>

<b>5.11. ARMAZENAMENTO DE INFORMAÇÃO NA MTS</b>	<b>56</b>
<b>5.12. ATUALIZAÇÃO DA MTS E MTL</b>	<b>56</b>
<b><u>6. DEMONSTRAÇÃO DE UTILIZAÇÃO</u></b>	<b><u>57</u></b>
<b><u>7. CONCLUSÃO</u></b>	<b><u>63</u></b>
<u>Referências Bibliográficas</u>	<u>65</u>
<u>Anexo I - O Protocolo TCP/IP</u>	<u>69</u>
<u>Anexo II - Código dos Agentes Implementados</u>	<u>77</u>

## Índice de Figuras

FIGURA 1 - ESTRUTURA DE UM AGENTE	10
FIGURA 2 - COMUNICAÇÃO UTILIZANDO QUADRO-NEGRO	16
FIGURA 3 - COMUNICAÇÃO UTILIZANDO TROCA DE MENSAGENS	17
FIGURA 4 - MODELO DE ARMAZENAMENTO DE INFORMAÇÃO DE AGENTES	18
FIGURA 5 - COMUNICAÇÃO UTILIZANDO SISTEMA FEDERATIVO	19
FIGURA 6 - AGENTE LIGADO A UM PROCESSO	30
FIGURA 7 - ESTRUTURA DE UM AGENTE	31
FIGURA 8 - VISÃO DA PLANTA	34

## **Índice de Tabelas**

Tabela 1 - Quadro Resumo de Aplicações	26
Tabela 2 - Quadro de Relação de Envio de Mensagens por Tipo Agente	51



## Lista de Abreviaturas

- AAS - Agentes de Apresentação de Situação
- AGD - Agentes Geradores de Dados
- APD - Agentes Processadores de Dados
- ARE - Agentes *Repositório* dos Geradores de Dados
- AT - Agentes Atuadores
- BB - *BlackBoard* (Quadro Negro)
- FS - *Federal System* (Sistema Federativo)
- IA - Inteligência Artificial
- IAD - Inteligência Artificial Distribuída
- KIF - *Knowledge Interchange Format* (Formato de Troca de Conhecimento)
- KQML - *Knowledge Query and Manipulation Language* (Linguagem de Manipulação e Solicitação de Conhecimento)
- MI - Máquina de Inferência
- MM - Mecanismo de Mercado
- MME - Mecanismo Mestre/Escravo
- MOP - *Market-Oriented Programming* (Programação Orientada ao Mercado)
- MP - Modelo Próprio
- MPS - *Message Passing* (Troca de Mensagem)
- MS - Modelo Social
- MTL - Memória de Trabalho Local
- MTS - Memória de Trabalho Social
- PLN - Processamento de Linguagem Natural
- RPD - Resolução de Problemas Distribuída
- RN - Rede Neural

SE - Sistemas Especialistas

SMA - Sistemas Multi-Agentes

STR - Sistemas Tempo Real

TCP/IP - *Transmission Control Protocol /Internet Protocol*

UDP - *User Datagram Protocol*

## **Abstract**

Nowadays a great area of research in Computing is the development of toolkits that allow the computational systems to interact among them to allow the improvement of their results. It is a consensus that two or more systems working together can cooperate and obtain better results than working apart.

Agent-Based Software Engineering proposes to facilitate the software creation enabling them to interoperate. Artificial Intelligence (AI) join with this approach to propose solutions. Classically, AI provides solutions to complex problems that “conventional” systems cannot do themselves. If the several parts of a complex system become intelligent they will perform new tasks that a centralized one could not perform. The great challenge in systems like that is to manage a large system with several processes in action and to allow them to interchange their experiences to improve the quality.

This document is divided in two main parts. The first one presents a study in Distributed Artificial Intelligence and Multi-Agent Systems emergent theories to help cognitive systems to interact with the wide world. All elements to construct such systems are shown here, among them: cooperation and coordination mechanisms, communication and knowledge information protocols. Also is shown some systems that are already in operation using these approaches.

The second part of this paper describes the conception and implementation of a Multi-Agent System (MAS) for monitoring and controlling process in an industrial environment. The main goal of this system is to provide a tool which allows computational systems to interact. A prototype of this MAS is already running.

## Resumo

Atualmente, uma grande área de pesquisa em Informática, é o desenvolvimento de ferramentas que permitam sistemas computacionais interagirem entre si para a obtenção de melhores resultados. É de consenso geral que dois ou mais sistemas trabalhando cooperadamente conseguem melhores resultados do que cada um de maneira isolada.

A Engenharia de Software baseada em agentes propõe facilitar a criação de softwares, habilitando-os à interoperabilidade. Também a Inteligência Artificial vem unir-se a Engenharia de Software baseada em agentes, propondo soluções. Classicamente a Inteligência Artificial provê soluções a problemas complexos que softwares ditos “convencionais” não o fazem. Tornar os vários componentes de um sistema complexo inteligentes, fará com que eles tenham condições de realizar tarefas de forma que um sistema centralizado teria maior dificuldade em executar. O grande desafio em sistemas deste tipo é manter um ambiente onde há vários processos em ação, e permitir que estes troquem suas experiências para que o trabalho conjunto seja mais proveitoso.

Este texto é dividido em duas partes principais. Na primeira apresenta-se um estudo em Inteligência Artificial Distribuída e Sistemas Multi-Agentes, teorias emergentes para auxiliar sistemas cognitivos na busca da interação ao mundo que os rodeia. Estuda-se todos os elementos envolvidos na construção de tais sistemas, tais como mecanismos de cooperação e coordenação, protocolos de comunicação e troca de informação entre outros.

Também são apresentados alguns sistemas que já estão em operação construídos utilizando-se tais abordagens.

Na segunda parte apresenta-se a concepção e implementação de um Sistema Multi-Agentes para monitoração e controle de processos. O objetivo deste sistema é permitir que componentes computacionais de uma planta industrial troquem informações entre si, aumentando a eficiência da planta controlada. Um protótipo foi implementado baseado nesta proposta.

## 1. Introdução

Hoje em dia muitos esforços estão concentrados em tornar a informação disponível em todos os níveis. Como maior exemplo temos a revolução causada atualmente pela Internet.

Esta tarefa não é das mais triviais. Sabe-se que existem muitos grupos trabalhando sobre centenas de diferentes plataformas, desenvolvendo seus sistemas sob diferentes ambientes. Porém o que não pode ser abandonado é o poder de resolver problemas que tais sistemas podem ter. É necessário então encontrar um método de permitir que muitos tenham acesso a tais sistemas, permitindo que, de posse dos resultados gerados por eles, novos progressos sejam obtidos.

Atualmente, uma importante linha de pesquisa em Informática é o desenvolvimento de ferramentas que permitam aos sistemas computacionais interagirem entre si, para a obtenção de melhores resultados. É de consenso geral que dois ou mais sistemas trabalhando cooperadamente conseguem melhores resultados do que cada um de maneira isolada.

A construção de software baseada em agentes propõe facilitar a criação de softwares, habilitando-os à interoperabilidade. Classicamente a Inteligência Artificial provê soluções a problemas complexos que sistemas ditos “convencionais” não o fazem. Tornar os vários componentes de um sistema complexo inteligentes, fará com que eles tenham condições de realizar tarefas de forma que um sistema centralizado teria maior dificuldade em executar. O grande desafio em sistemas deste tipo é manter um ambiente onde há vários processos em ação, e permitir que estes troquem suas informações para que o trabalho conjunto dos mesmos seja de maior qualidade.

A união do estudo de técnica de Inteligência Artificial (IA) com técnicas de Sistemas Distribuídos forma a Inteligência Artificial Distribuída (IAD). A IAD é uma especialização da IA que estuda as relações sociais entre agentes de uma mesma comunidade ou sociedade. A IAD tem evoluído muito nos últimos anos. Um dos ramos de desenvolvimento da IAD são os chamados Sistemas Multi-Agentes (SMA), nos quais este trabalho baseia-se.

O objetivo principal deste trabalho é propor um SMA, destinado à monitoração e o controle de uma planta industrial, permitindo que cada um de seus componentes atue com melhores resultados, através do compartilhamento de seus conhecimentos com os demais componentes do sistema, tendo como base para implementação a planta de controle de corrosão da Refinaria Getúlio Vargas.

A motivação para desenvolvimento deste trabalho surgiu da experiência anterior do autor no desenvolvimento de Sistemas Especialistas para a área industrial e, mais especificamente, do Sistema Especialista (SE) Monitor [Ramo95].

O SE Monitor foi desenvolvido pelo TECPAR para a Refinaria Getúlio Vargas - REPAR localizada em Araucária no Paraná. O Monitor está em operação há mais de um ano na refinaria e tem como objetivo monitorar o nível de corrosão no topo das torres de destilação de petróleo. O Monitor tem acesso a todos os parâmetros do processo de corrosão vindos da rede de dados da refinaria. Diagnósticos e recomendações são periodicamente inferidos e apresentados a equipe de operação e alarme [Ramo95]. Apesar do Monitor estar operando comercialmente na refinaria, após sua instalação percebeu-se que outros SE poderiam estar trabalhando, controlando aspectos específicos em toda a planta industrial. Desta forma surgiu a idéia de criar um ambiente que pudesse permitir a interação entre SE's.

O trabalho também situa-se no contexto do projeto X-MAS, de cooperação entre as universidades e instituições do Brasil e da Europa, para a troca de conhecimentos e experiências na área dos SMA.

O capítulo 2 apresenta uma descrição teórica sobre Inteligência Artificial Distribuída e a estrutura de um Sistema Multi-Agente. Já o capítulo 3 apresenta

algumas aplicações de SMA. A partir do capítulo 4, tem-se a proposta de um SMA para monitoração e controle de processos industriais. O capítulo 5 apresenta alguns detalhes sobre a implementação realizada e o capítulo 6 apresenta a demonstração da utilização da proposta. O capítulo 7 apresenta a conclusão deste trabalho.

O anexo I apresenta algumas definições sobre o protocolo TCP/IP. No anexo II apresentam-se os aspectos mais importantes do código implementado no protótipo.





## 2. Inteligência Artificial Distribuída

Os conceitos utilizados em IAD podem ser fundamentados na seguinte frase: “dividir para conquistar”.

A IAD tornou-se um ramo de estudo recentemente. Em meados de 1980 foi realizado um workshop no *Massachusetts Institute of Technology* (MIT) nos Estados Unidos, tentando estabelecer um consenso “informal” sobre termos e seus significados [Davi81]. Muitas das idéias apresentadas neste *workshop* resultaram em artigos de grande repercussão. Um destes artigos, escrito por Nilsson em 1981, tentou posicionar este tema em termos gerais, ressaltando com alguma ênfase o planejamento de tarefas e descrevendo IAD como uma rede de sistemas livres. Outro artigo daquela época foi escrito por Smith e Davis, tratando da cooperação em Resolução de Problemas Distribuídos (RPD) e definindo a terminologia no contexto de tais sistemas [Witt92].

Cinco anos depois (1985), no quinto *workshop* em IAD, o primeiro sistema desenvolvido nesta área foi apresentado. Desde aquela época o número de protótipos, arquiteturas, linguagens, etc, têm aumentado rapidamente. Discussões sobre definições “mais formais” foram iniciadas neste percurso, tendo sido este o objetivo do *workshop* realizado em 1986.

Em 1987, Sridharam [Srid97] produziu uma primeira taxonomia da IAD. Esta divisão é mostrada a seguir, e baseia-se em 8 diferentes dimensões:

### MODELO - Nível de Distribuição

<b>Indivíduos</b> Elementos Simples Altamente Conectados <b>GRANULARIDADE</b>	<b>Grupos</b> Sistemas Especialistas de <b>Nível de Decomposição</b> Inteligência Imitada	<b>Sociedade</b> Agentes Complexos do Problema
<b>Granularidade Fina</b> Nível de Dados <b>ESCALA - Nível de Paralelismo</b>	<b>Granularidade Média</b> Nível de Tarefas	<b>Granularidade Grande</b> Nível de Cooperação
<b>Grande</b> > 10000	<b>Média</b> < 10000	<b>Pequena</b> < 16

### AGENTES - Nível de Autonomia

<b>Controlada</b>	<b>Semi-Autônomo</b>	<b>Autônomo</b>
-------------------	----------------------	-----------------

### CONSTRUÇÃO - Nível de Decomposição do Sistema

<b>Decomposição</b>	<b>Síntese</b>
---------------------	----------------

### UNIFORMIDADE - Nível de Homogeneidade dos Agentes

<b>Homogêneo</b>	<b>Heterogêneo</b>
------------------	--------------------

### RECURSOS - Nível de Restrições para Serem Consideradas

<b>Amplos</b>	<b>Restritos</b> tempo, memória...
---------------	---------------------------------------

### INTERAÇÕES - Nível de Complexidade

<b>Simples</b>	<b>Complexa</b>
----------------	-----------------

Pode-se definir simplifcadamente cada item desta taxonomia:

- 1) Modelo: indica como o nível de distribuição é aplicado ao sistema, e como são classificados os agentes em termos de sua complexidade;
- 2) Granularidade: mostra qual é o nível de interação entre os agentes;

3) Escala: é a medida do número de agentes que compõe a comunidade;

4) Agentes: indica qual é o grau de autonomia dos agentes em relação a comunidade;

5) Construção: indica como o problema foi dividido entre os agentes;

6) Uniformidade: define o grau de homogeneidade dos agentes; heterogeneidade significa ter agentes em diferentes formatos ou construídos sob diferentes metodologias (por exemplo, cognitivos ou não);

7) Recursos: define como os recursos são distribuídos pelo ambiente; pode ser considerado aqui desde equipamentos até tempo;

8) Interações: aponta o nível de interação entre os agentes; depende do mecanismo de coordenação escolhido.

Bond e Gasser [Bond88] sugerem ainda alguns benefícios para a utilização de IAD, de acordo com os seguintes critérios:

a) Adaptabilidade: sistemas de IAD são mais apropriados para lidar com problemas distribuídos em termos espaciais, lógicos, temporais ou semânticos;

b) Custo: um grande número de pequenas unidades computacionais pode ser mais interessante, quando os custos com comunicação não são relevantes;

c) Desenvolvimento e Gerenciamento: a inerente modularidade do sistema permite o desenvolvimento de partes do mesmo separadamente, garantindo a continuidade do ambiente;

d) Eficiência e Velocidade: a concorrência e a distribuição de processos em diferentes computadores pode aumentar a velocidade de computação e raciocínio, desde que haja um nível de coordenação aceitável;

e) História: a integração de recursos distribuídos, tais como redes de estações de trabalho ou diferentes domínios de especialistas;

f) Isolação/Autonomia: o controle de processos locais pode ser encarado como uma maneira de proteção ou de aumento da segurança do sistema;

g) Naturalidade: alguns problemas são melhor descritos em termos distribuídos;

h) Confiabilidade: os sistemas distribuídos podem exibir um grau maior de confiabilidade e de segurança do que sistemas centralizados, pois podem prover redundância (um agente é capaz de realizar o que outro agente faz se necessário), múltiplas verificações, “triangulação de resultados”, etc;

i) Limitação de Recursos: os agentes computacionais individuais ligados a recursos escassos podem cooperar em busca da resolução de problemas complexos e;

j) Especialização: o conhecimento e as ações podem ser escolhidas de acordo com o domínio do agente.

Em “*Readings in DAI*”, Bond e Gasser [Bond88] propuseram uma distinção entre RPD e SMA, termos que sugerem diferentes arquiteturas para especificar necessidades como, por exemplo, planejamento distribuído ou controle distribuído.

Em RPD, agentes geralmente compartilham um objetivo, uma linguagem e semântica comuns. Além disso, um único agente nunca é capaz de resolver um problema isoladamente, mas apenas a comunidade de agentes como um todo é capaz de realizar a tarefa.

Em SMA os agentes são mais independentes, compartilham um ambiente em comum, mas estão competindo pelos mesmos recursos limitados, tais como tempo, espaço, e ferramentas; têm que cooperar por razões de eficiência (beneficiando-se de resultados de outros agentes, ou ajudando-os). Mesmo que não utilizem a mesma linguagem, eles têm a capacidade de transmitir e formatar seus problemas através de uma forma de representação comum e, apenas em situações extremas, eles resolvem problemas individualmente.

Atualmente vários exemplos de sistemas bem sucedidos utilizam a abordagem dos SMA. O esforço envolvido no projeto e implementação de sistemas SMA é grande e exige uma capacidade adequada da equipe de desenvolvimento.

## 2.1. Estrutura de um Sistema Multi-Agentes

“A coletividade sabe mais que a soma dos indivíduos”.

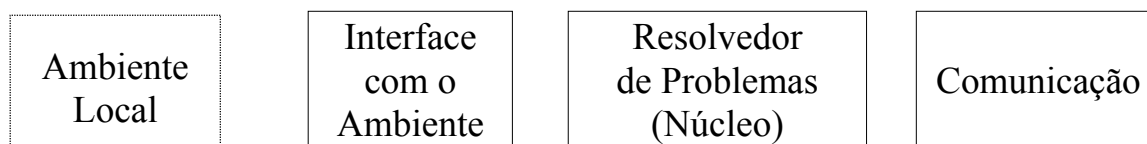
Toda teoria dos SMA está baseada na aplicação de agentes. Na definição de um agente deve-se abstrair o domínio em que o mesmo está inserido. De uma maneira geral pode-se resumir um agente como um processo composto por *software*, *hardware* ou por uma combinação de ambos, capaz de realizar uma determinada tarefa e disponibilizar seus serviços a uma coletividade de agentes, que, em conjunto, realizam um objetivo comum [Para95].

Os objetivos principais de um agente podem ser resumidos da seguinte forma:

- a) realizar as tarefas para as quais foi especificado e;
- b) cooperar com os demais agentes da rede, de forma a fornecer as informações e/ou prestar serviços para que os outros agentes possam realizar suas tarefas de forma mais eficiente.

Isto implica que um agente deve ter o conhecimento de como interagir e cooperar com os demais agentes da rede, o que exige a definição de protocolos de comunicação e de uma linguagem para a troca de informações.

Assim pode-se dizer que um sistema inteligente torna-se um agente quando este adquire a habilidade de interagir com o ambiente em que o mesmo está inserido. Na figura a seguir, pode-se inferir que esta entidade pode ser dividida em duas partes principais: um núcleo ou módulo inteligente competente na resolução de um determinado problema, e um módulo provedor de acesso ao meio físico que permite que o agente interaja com os demais agentes existentes. Além disso está representada a interface com um ambiente qualquer, responsável por alimentar o *núcleo* com alguns dados importantes ao seu processamento. Salienta-se que o papel do ambiente pode ser desempenhado por um usuário humano.





**Figura 1 - Estrutura de um Agente**

Conhecido também como cápsula, o módulo de comunicação é responsável pela troca de informação com os demais agentes da rede. Na realidade, a função deste módulo é mais ampla e exige uma análise detalhada.

As principais características deste módulo são:

- a) habilidade para interagir com a comunidade de agentes, respeitando a política de cooperação estabelecida;
- b) formatar suas questões aos demais agentes da rede em uma forma padronizada, conhecida por todos;
- c) fornecer ao *núcleo* do agente, as informações mais recentemente oferecidas pelos demais agentes.

Assim, define-se um ambiente formado por vários sistemas que atuam paralelamente, com a expectativa de interação entre si, e com o objetivo maior de obter melhores resultados. Desta forma é necessária a construção de mecanismos que permitam aos agentes realizarem suas tarefas de forma organizada, ou seja, a definição da política de cooperação entre eles.

## **2.2. Protocolo e Linguagem de Comunicação**

Para que exista a comunicação entre dois ou mais agentes devem ser estabelecidas certas regras, que garantam a ordem na comunicação e transferência de dados. O conjunto destas regras pode ser chamado de protocolo de comunicação.

Dentre os vários estudos já realizados e os em andamento sobre protocolos de comunicação em SMA, destaca-se o KQML (*Knowledge Query and Manipulation Language*). O KQML é um protocolo de comunicação baseado em performativas que possuem funções pré-determinadas [Fini94].

### **2.2.1.O Protocolo KQML (Knowledge Query and Manipulation Language)**

O protocolo de comunicação KQML é resultado do trabalho do grupo *External Interfaces Working Group* do DARPA *Knowledge Sharing Effort*. O objetivo deste grupo é oferecer ferramentas para facilitar a troca de informação entre sistemas computacionais envolvidos em ambientes heterogêneos.

KQML é um protocolo para programas que se utilizam da comunicação para troca de informação, despreocupando-se com o conteúdo desta informação. Esta linguagem está disponível para utilização e pode ser implementada em C/C++ usando-se TCP/IP para comunicação entre processos.

Assim agentes de um SMA podem utilizar-se do KQML para manterem-se trocando dados. Na prática, KQML é um protocolo para formatação de mensagens chamadas performativas. O KQML possui várias performativas com funções específicas.

#### **2.2.1.1.Algumas Performativas KQML**

Uma performativa KQML é expressa na forma de uma cadeia ASCII usando uma sintaxe bem definida. Esta sintaxe é baseada na notação polonesa pré- fixada do *Common Lisp*. Esta sintaxe foi escolhida devido a simplicidade de leitura.

Existem mais de 40 performativas. São exemplos de performativas:

**achieve**: um agente A deseja que um agente B faça algo de seu domínio;

**ask-all**: um agente A deseja todas as respostas de um agente B para uma questão;

**break**: um agente A deseja que um agente B quebre uma conexão em *pipe*;

**evaluate**: um agente A deseja que B simplifique ou calcule uma sentença;

**insert**: um agente A solicita a B para acrescentar uma sentença em sua base de conhecimento;

**tell**: informa uma sentença;

etc.

Cada performativa tem seu conjunto de parâmetros e é identificada como uma palavra reservada da linguagem.

### 2.2.2.O KIF (Knowledge Interchange Format)

O KIF é uma linguagem definida pelo grupo DARPA *Knowledge Sharing Initiative* desenvolvida com o propósito de formatar o conteúdo de uma mensagem KQML [Gene92]. KIF é uma linguagem baseada no cálculo predicativo de primeira ordem, possibilitando a comunicação e troca de conhecimento entre os agentes. O conhecimento pode ser formatado em regras, expressões quantificadas, entre outros.

Um exemplo básico do uso do KQML com o KIF é mostrado a seguir. Neste exemplo um agente A pergunta a um agente B recebendo na seqüência a resposta gerada pelo agente B, mostrando como são formatadas as mensagens com KQML e KIF.

O agente A envia para B a seguinte mensagem:

```
(evaluate :language KIF :ontology motores :reply-with q1 :content (val (torque motor1) (sim-time 5 ))).
```

O agente B responde:



(reply :language KIF :ontology motores :in-reply-to q1 :content (scalar 12 kgf)).

Nas mensagens vistas, cada primitiva possui uma combinação de parâmetros reservados [Para95b]. Cada parâmetro tem seu significado próprio.

São alguns destes parâmetros:

- : **content** - informação que expressa uma ação ou pedido;
- : **force** - significa que o agente despachante nunca irá alterar o significado da primitiva;
- : **in-reply-to** - label de uma resposta;
- : **language** - nome da linguagem de representação utilizada em : **content**;
- : **ontology** - nome de um domínio usado em : **content**;
- : **receiver** - receptor da primitiva;
- : **reply-with** - um nome para a resposta ao despachante da primitiva;
- : **sender** - despachante da mensagem.

Na performativa *ready*, por exemplo, tem-se 4 parâmetros:

ready

- :reply-with <expressão>
- :in-reply-to <expressão>
- :sender <palavra>
- :receiver <palavra>

Nesta performativa um agente indica que está pronto para responder questionamentos.

Nota-se que as performativas foram preparadas para resolver os problemas de comunicação e interação entre agentes. Sua aplicabilidade é restrita.

Outras implementações também estão sendo sugeridas. Neste trabalho será apresentada uma proposta de linguagem de comunicação para SMA no domínio da monitoração e controle de processos industriais.

### **2.3. Política de Cooperação e Organização dos Agentes**

Um sistema baseado em agentes precisa mais do que um protocolo e uma linguagem de comunicação para descrever suas “vontades” e “crenças”. Os agentes necessitam de uma motivação para realizar as comunicações e para responder a solicitações. O nível de abstração ou de considerações que devem ser utilizadas está intimamente ligado ao propósito do desenvolvimento, isto é, ao próprio objetivo do sistema. Para exemplificar, pode-se definir como princípios gerais do comportamento de um agente [Gene94]:

- a) veracidade: um agente deve dizer a verdade;
- b) autonomia: um agente não pode solicitar a outro agente a realização de uma tarefa sem o mesmo ter anunciado a vontade em realizá-la;
- c) compromisso: se um agente anuncia que pode realizar determinada tarefa, então ele é obrigado a realizá-la caso seja solicitado a fazê-lo.

Estes três comportamentos definem uma política de cooperação. A política de cooperação também está diretamente ligada a forma de troca de informação entre os agentes.

#### **2.3.1. Cooperação entre Agentes**

Agentes podem demonstrar interesses diferentes ao cooperar. Se um agente estiver conectado a tarefas prioritárias, ele pode ter menor interesse em cooperar que outro agente ligado a tarefas menos críticas.

Duas maneiras de iniciar o processo de cooperação podem ser caracterizadas: a conhecida como *Task Sharing* e a *Result Sharing*. Em *Task*

*Sharing* a conversação é iniciada quando um agente tem uma tarefa a realizar e não pode fazê-la de maneira autônoma, necessitando da ajuda de outro(s) agente(s). *Result Sharing* por sua vez, é utilizada por agentes que geram uma informação e, por livre e espontânea vontade a disponibilizam à comunidade, acreditando que outro agente possa dela necessitar em algum momento.

No processo de negociação, os agentes definem como serão formatadas as mensagens (por exemplo, utilizando-se KIF e KQML).

Quando um agente necessita de ajuda para realizar uma tarefa, ele deve ser capaz de determinar quais os outros agentes da comunidade que podem auxiliá-lo. Algumas das abordagens utilizadas para resolver este problema são analisadas a seguir.

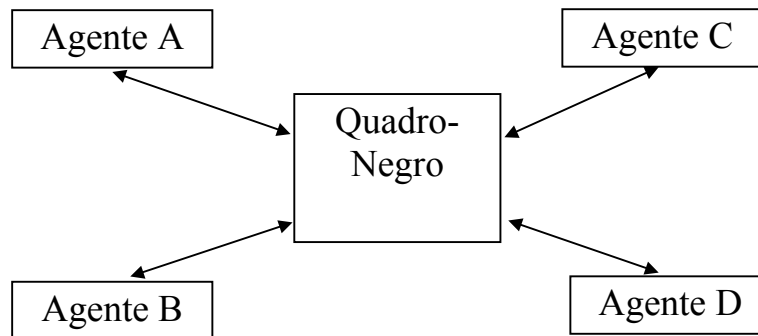
#### - Quadro-Negro ou *BlackBoard* (BB)

Um BB pode ser entendido de forma simplificada como uma memória de compartilhamento global, onde há uma concentração de conhecimento disponibilizado para leitura e escrita. Os BBs foram primeiramente utilizados na geração de Sistemas Especialistas (SE), onde estruturas de conhecimento associativas tais como regras de produção individuais foram substituídas por grandes módulos de conhecimento.

Em SMA, os BBs podem ser usados pelos agentes como repositório de perguntas e respostas. Cada agente que necessita de determinada informação envia ao BB uma mensagem. Inspeccionando o BB, um agente que souber a informação desejada, a depositará no BB, permitindo sua posterior recuperação pelo agente inicial.

Por exemplo: o agente **A** necessita do arquivo A1. Este agente constrói uma mensagem e envia ao BB seu pedido. O agente **B**, varrendo o BB descobre a necessidade do agente **A**, e como ele pode fornecer o arquivo requerido, o faz, enviando-o ao BB. Antes do envio, o agente **B** escreve no BB que irá fornecer o arquivo ao agente **A**, para que este aguarde. O agente **B** escreve o arquivo A1 no BB e só então o agente **A** pode ler o arquivo desejado.

Na figura 2, verifica-se como o BB centraliza a comunicação entre agentes.



**Figura 2 - Comunicação utilizando Quadro-Negro**

Apesar de ter a característica de reduzir o número de mensagens no sistema, a utilização de BB é limitada a sistemas não críticos em tempo, não sendo recomendada a aplicação em Sistemas Tempo Real (STR). Isto é facilmente deduzido a partir do exemplo descrito anteriormente. O agente **A** necessita do arquivo A1 com muita urgência e não pode continuar suas atividades sem o mesmo. Enquanto não houver um agente que leia o BB, verifique a necessidade do arquivo A1, esteja habilitado a resolver o problema e efetivamente realize a ação, muito tempo pode passar. A necessidade de se estar constantemente lendo o BB é um agravante sério, que inviabiliza o atendimento a agentes críticos em tempo.

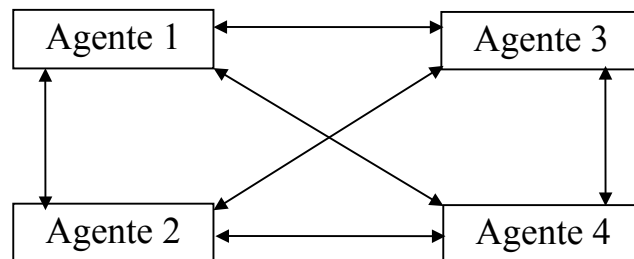
Uma alternativa seria criar um disparador (*trigger*) que enviaria uma mensagem a um determinado agente no momento em que a informação foi armazenada no BB. Esta abordagem é encontrada na literatura em sistemas que utilizam BB e possuem fontes de conhecimento com atividades dirigidas por eventos (*event-driven*) [Haye85].

### **- Troca de Mensagens (*Message Passing - MPS*)**

Neste método as mensagens são trocadas de maneira assíncrona entre agentes. Os agentes podem trocar mensagens uns com os outros, diretamente.

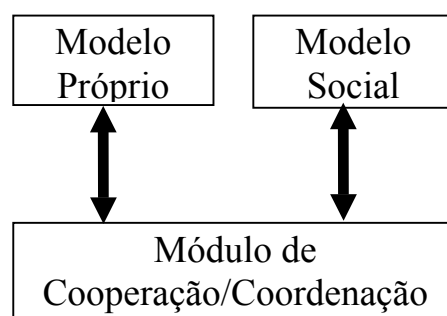
Podem haver agentes intermediários, que agem como facilitadores da comunicação. O método é mais eficiente porém, o mais custoso de ser implementado, além de apresentar um crescimento exponencial de possibilidades de comunicação entre os agentes.

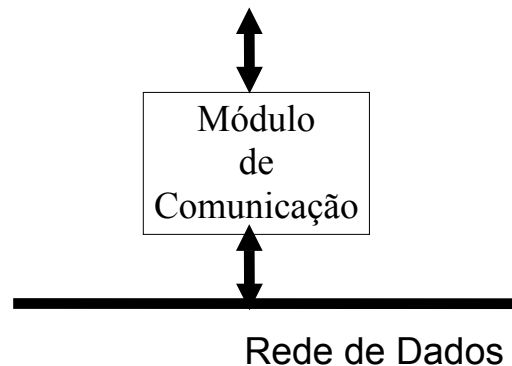
A figura 3 ilustra como é realizado o processo de troca de mensagens.



**Figura 3 - Comunicação utilizando Troca de Mensagens**

Se o agente não consegue definir, à priori, um agente para interação, ele pode enviar uma mensagem em *broadcast* para toda comunidade, de forma que os agentes que podem ajudar, indiquem esta possibilidade. Quando se deseja conexão direta agente-agente, sem *broadcast*, deve-se definir uma maneira de permitir que o agente saiba, à priori, qual agente pode lhe fornecer ajuda. Uma estratégia possível é a construção em cada agente de um módulo capaz de armazenar as informações sobre o que o próprio agente pode realizar e sobre o que os demais agentes podem fazer. Isto é conhecido como Modelo Próprio (MP) e Modelo Social (MS) [Malh94]. O MP indica o conjunto de todas as tarefas que o agente pode realizar. Já o MS, é formado por uma lista referenciando todas as tarefas que podem ser realizadas pelos outros agentes. A figura 4, apresenta uma representação básica da cápsula do agente com estes modelos.





**Figura 4 - Modelo de Armazenamento de Informação de Agentes**

Em resumo, quando um agente necessita de ajuda ele consulta seu MS verificando qual agente pode ser questionado. Sendo assim o contato é feito diretamente. No MP o agente armazena aquilo que ele pode fornecer a comunidade.

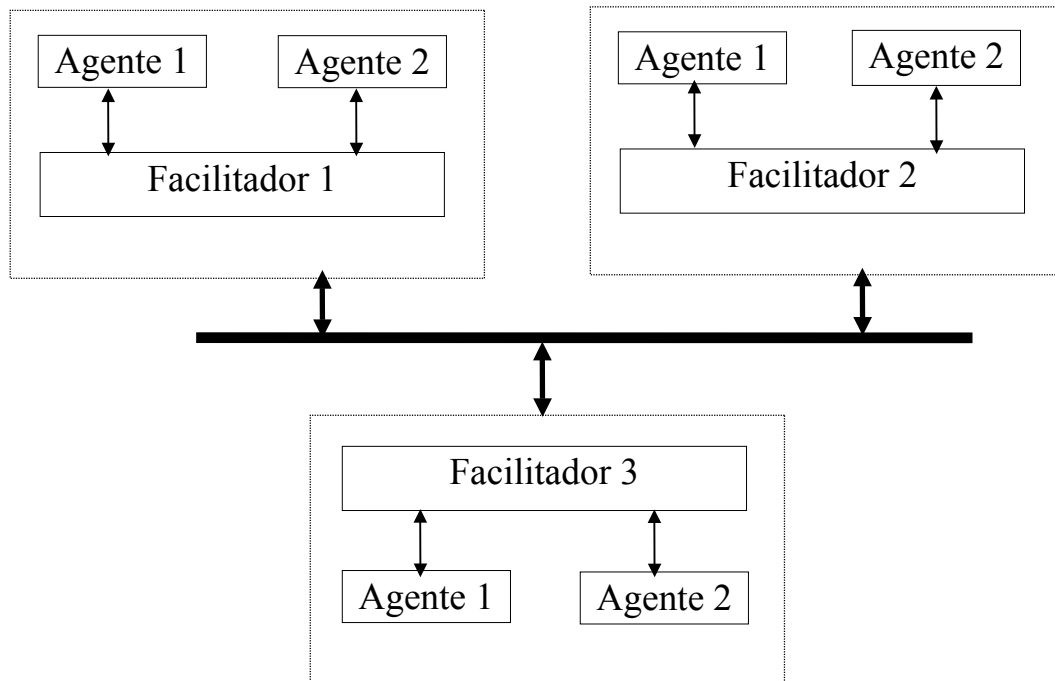
Com este modelamento, o agente necessita de um módulo de controle para seleção de agentes em seu MS. Isto pode ser resolvido por um mecanismo de seleção que, a partir da informação desejada, obtenha quais agentes podem fornecê-la.

As abordagens BB e MPS pode ser utilizadas em conjunto como ocorreu por exemplo no projeto ARCHON [Witt92].

#### **- Sistema Federativo (*Federal System* - FS)**

Em MPS, quando se utiliza *broadcasting* para envio de mensagens, se houver um grande número de agentes, este processo pode ser custoso, pois a manipulação de todas estas mensagens pode ser complicada. Se houver  $n$  agentes na comunidade, isto levará a uma interação de  $n$  elementos 2 a 2. Por exemplo: para 8 agentes, tem-se 64 possibilidades de comunicação.

Uma proposta alternativa é a utilização de *Federal System* ou Sistema Federativo (FS). Este esquema tem este nome em função da organização dos agentes em grupos semi-autônomos, numa analogia ao sistema político que divide uma comunidade em estados. Neste esquema surge a figura de agentes facilitadores. Na figura 5, este esquema é exemplificado.



**Figura 5 - Comunicação utilizando Sistema Federativo**

Num FS, os agentes ao invés de contatos diretos com demais agentes, o fazem através dos facilitadores, que por sua vez, distribuem suas necessidades aos outros facilitadores. Este esquema cria uma “federação” onde agentes são divididos em categorias e subordinados a facilitadores, que se tornam os responsáveis pela busca da informação.

## 2.4. Mecanismos de Coordenação

Como um grande número de agentes pode ser criado, deve-se especificar um mecanismo para gerenciá-los. Alguns destes mecanismos de

coordenação podem ser encontrados na literatura, entre eles destacam-se o Mestre/Escravo e o mecanismo de Mercado. Os mecanismos de coordenação determinam como a tarefa será executada.

### **2.4.1.Mecanismo Mestre/Escravo (MME)**

O MME divide os agentes em duas classes: os gerentes e os trabalhadores. Um grupo de agentes é coordenado por um gerente o qual é responsável pela distribuição de tarefas entre os trabalhadores. Neste mecanismo também podem ser utilizados facilitadores.

Este tipo de mecanismo é recomendado para ambientes onde se sabe exatamente como as tarefas podem ser resolvidas. Em ambientes pouco estruturados este mecanismo pode causar sobrecarga de trabalho para alguns agentes em relação a outros [Malo87].

Uma variação simples deste mecanismo é a configuração do tipo “exército”, onde os agentes são estruturados em vários níveis fortemente hierarquizados.

### **2.4.2.Mecanismo de Mercado (MM)**

Neste tipo de coordenação os agentes estão todos situados em um mesmo nível, e sabem o que os demais são capazes de fornecer. O mecanismo é mais geral e tem aplicações amplas, e sua característica principal é a de reduzir a troca de mensagens entre os agentes, pois cada um deles sabe o que os demais podem fornecer [Malo87].

Trabalhos recentes tem sugerido a adoção da Programação Orientada ao Mercado ou *Market-Oriented Programming* (MOP), como em [Well95]. O MOP é uma abordagem de coordenação baseada em mecanismos de “preço” (exemplo: critério de escolha de agentes) de mercado e fundamentada na computação distribuída. A idéia do MOP é utilizar a instituição de mercados e de seus modelos para construir “economias cooperativas”, para resolver



problemas de alocação de recursos distribuídos, isto é, distribuindo as tarefas de modo coerente entre todas os componentes do sistema.

Os agentes em MOP interagem em uma maneira bem restrita, oferecendo-se para comprar ou vender quantidades de uma mercadoria segundo um critério de “preços” fixos. Quando o sistema chega a um equilíbrio, isto indica, que os recursos foram alocados coerentemente pelo sistema, impondo atividades e consumo aos vários agentes. No MOP os agentes podem ser classificados em consumidores e produtores, e são direcionados para maximizar lucros e/ou utilidades.



### 3. Exemplos de Aplicações

Neste capítulo apresenta-se algumas aplicações que se utilizam da teoria exposta nos capítulos anteriores.

#### 3.1. ***ARCHON: An Architecture for Cooperative Heterogeneous On-Line Systems***

O ARCHON [Witt92] é um projeto que foi desenvolvido por um grupo de instituições europeias com início em 1989 e que teve duração de 5 anos. O domínio deste SMA é o gerenciamento da distribuição de energia elétrica. Como os próprios autores afirmam, este é um sistema balanceado em termos de complexidade, não sendo muito complexo e nem muito simples.

Neste SMA os agentes são divididos em duas camadas: o sistema inteligente ou camada de resolução de problemas e a camada de cooperação. A camada do sistema inteligente contém o conhecimento necessário sobre o domínio e é constituída, na sua maioria, por SE.

A camada de cooperação tem dupla função: coordenar as trocas de informação inter-agente e intra-agente. Nesta camada foram implementados um esquema de BB para coordenação intra-agente e MP para coordenação/comunicação inter-agente.

O conjunto de mensagens é baseado em lógica de predicados.

O sistema foi testado e aprovado em uma empresa de energia elétrica da Espanha.

### **3.2. *TALISMAN: A Multi-Agent System for Natural Language Processing***

O TALISMAN, desenvolvido na França [Stef95], implementa a análise de sentenças da língua francesa de maneira distribuída, utilizando-se de um SMA. São utilizadas as leis de lingüística para a comunicação inter-agente, e o controle é descentralizado.

Existe uma divisão da sociedade em agentes definidos de acordo com suas funções, entre elas: pré-processamento, análise morfológica, divisão em cláusulas, análise sintática, resolução estatística de ambigüidades, coordenação e negação, entre outras.

Este SMA está, na presente data, sendo implementado e algumas idéias ainda estão sendo estudadas.

### **3.3. *Agent-Based System Engineering (ABSE)***

O ABSE [McGu93] é um projeto de colaboração entre a Universidade de Stanford e o laboratório da Hewlett-Packard em Palo Alto. O ABSE é uma rede de agentes conectados através de facilitadores. Os agentes e os facilitadores estão conectados em uma configuração FS.

Os facilitadores podem receber mensagens direta ou indiretamente dos agentes. Mensagens indiretas indicam que o facilitador deve dar um destino a mensagem. Assim os facilitadores assumem tarefas tais como tradução de mensagens, decomposição de problemas em sub-problemas e escalonamento de trabalho, entre outras.

Este sistema utiliza-se do protocolo KQML para interação entre agentes.

### **3.4. *MATHEMA: A Learning Environment Based on A Multi-Agent Architecture***

O MATHEMA [Cost95] é um ambiente de aprendizado baseado em computador que se utiliza de agentes inteligentes. A idéia deste projeto é

integrar aprendizado humano em uma sociedade de agentes tutores inteligentes.

Os agentes trabalham em um domínio específico, como, por exemplo, álgebra. Os agentes cooperam entre si para promover o aprendizado de um estudante.

Cada agente tem capacidade de resolver e propor problemas para o estudante, ensinar, resolver conflitos, comunicar e cooperar com outros agentes.

### **3.5. Quadro Resumo**

A tabela a seguir resume algumas das características de cada sistema.

Projeto	Características			
	Linguagem de Comunicação	Protocolo de Comunicação	Mecanismos de Coordenação	Domínio da Aplicação
ARCHON	Própria, baseada em lógica de predicados	Próprio, baseado em lógica de predicados	Utiliza BB para comunicação intra-agente e MP para comunicação intra-agente	Distribuição de Energia Elétrica
TALISMAN	Linguagem multi-agente de aplicação	Potocolo de comunicação próprio	BB	Processamento de Linguagem Natural (PLN) (Análise de sentenças da língua francesa)
ABSE	KIF	KQML	FS (Utilização de Facilitadores)	Desenvolvimento de teoria em IAD e SMA
MATHEMA	Linguagem Própria	Protocolo Próprio	MP	Sistema de Aprendizado

**Tabela 1 - Quadro Resumo de Aplicações**

## **4. Um Sistema Multi-Agente para Monitoração e Controle de Processos Industriais**

Descreve-se a seguir um ambiente para monitoração e controle de processos industriais, utilizando a abordagem Multi-Agente.

De opinião geral, o controle de um processo industrial é uma tarefa muito difícil. A quantidade de fatores complicadores determinam a maior ou menor complexidade de um sistema deste tipo. Integrar várias plataformas, técnicas e máquinas tem se tornado um grande empecilho para uma maior automatização de plantas industriais. Existem vários tipos de dispositivos de controle, tais como sensores, atuadores, sistemas de processamento, cada qual com sua própria linguagem de programação e protocolos de comunicação nem sempre padronizados. Além disso, a inclusão de um novo componente na rede de controle exige um grande esforço, por vezes levando à parada de toda planta produtiva, com evidente desperdício de tempo e dinheiro. Para tentar auxiliar o processo de monitoração e controle, apresenta-se uma proposta de arquitetura e a implementação inicial de um ambiente multi-agentes para monitoração e controle de processos.

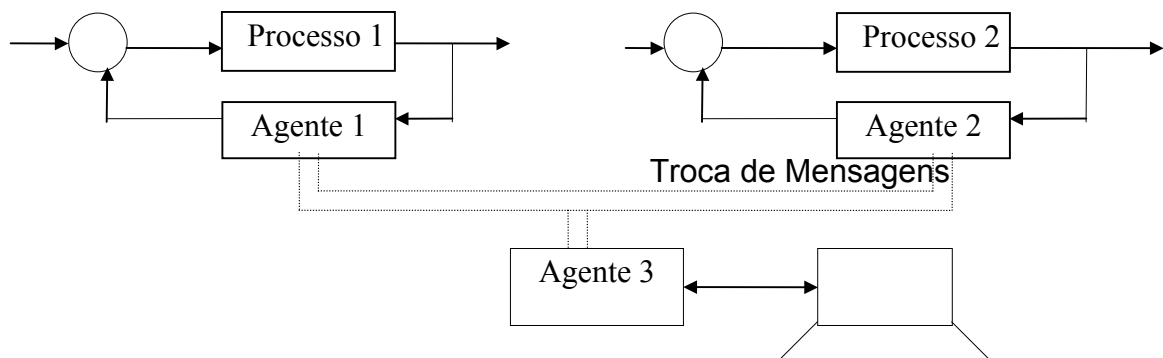
### **4.1. Concepção de um SMA**

No processo de concepção de um SMA deve-se ter em mente que será necessário definir um protocolo de comunicação, a arquitetura e o modelo dos agentes, a política de cooperação e os mecanismos de coordenação. Na seqüência apresenta-se o modelo adotado em cada um destes itens, justificando-se a escolha a cada passo.

## Os Componentes de uma Planta Industrial X O Agente

Como foi visto anteriormente, o agente é o principal componente de um SMA. Para a definição da arquitetura do agente, deve-se estudar primeiramente qual o domínio para o qual o SMA será construído. Neste caso, o domínio é uma planta industrial, com vários componentes e sub-componentes. Cada um destes componentes está relacionado a partes do processo. Todos eles tem como fim comum a perfeita operação da planta com os parâmetros desejados. Um agente é um destes componentes, capaz de realizar uma tarefa específica.

Na figura 6 apresenta-se um diagrama em blocos que ilustra o problema. Nesta figura usa-se o modelo clássico de um sistema de controle realimentado para representar como os agentes podem ser encarados como controladores de processo. Tem-se vários processos a controlar e vários sistemas ou agentes agindo de forma cooperada. A união da atuação dos processos leva ao controle da planta como um todo.



**Figura 6 - Exemplo de um SMA em ambiente Industrial.**

Após analisar-se várias plantas de controle pode-se identificar quatro subsistemas que em geral são encontrados em redes de controle [Para95a]. São eles:



- \* geradores de dados (como os conectados a sensores);
- \* processadores de dados (como um SE ou uma Rede Neural (RN));
- \* mostradores de resultados (um log ou visualizador geral); e
- \* atuadores sobre os processos.

Neste sentido, qualquer sistema de controle de processos industriais pode ser dividido em grupos de componentes segundo estes quatro tipos.

Neste trabalho os agentes foram classificados desta maneira. Esta divisão, além de facilitar a modularização geral do ambiente, permitirá a flexibilização no processo de troca de mensagens.

## **4.2. Os Objetivos de um Agente**

Durante o funcionamento do SMA os agentes estão trabalhando juntos, formando uma sociedade. Nesta sociedade cada agente tem seu papel específico, e, em geral, deve colaborar com os demais agentes, fornecendo informações disponíveis sempre que solicitado, ressaltando-se que alguns agentes poderão fornecê-las espontaneamente.

Pode-se assim sumarizar que um agente tem dois objetivos principais:

- a) realizar as tarefas a ele definidas; e
- b) cooperar com os outros agentes da comunidade, suprimindo informações e/ou serviços para ajudá-los a realizar suas tarefas.

Nesta abordagem, os agentes seguem três regras básicas:

1 - todo agente está disposto a colaborar com a sociedade, e o tempo que uma solicitação externa levará para ser respondida é apenas uma função de suas próprias atividades;

2 - o agente pode entrar ou sair da rede quando desejar, sempre informando esta situação a comunidade para que os outros agentes saibam se o agente está ou não em operação;

3 - as respostas geradas por cada um dos agentes são consideradas como verdadeiras e confiáveis, a menos que haja uma indicação em contrário.

Todos os agentes estão trabalhando com o mesmo objetivo maior: controlar uma planta industrial, o que justifica esta política de total cooperação.

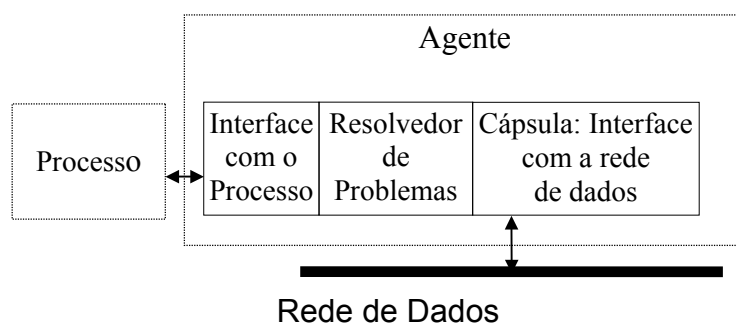
Embora existam atualmente várias caracterizações para o conceito de agente, para o propósito deste trabalho, um agente será considerado como sendo uma entidade que possui:

- conhecimento sobre como realizar suas tarefas; nesta aplicação, o domínio do conhecimento abrange os processos, ou parte dos mesmos, que o agente está monitorando;

- conhecimento sobre como interagir com a comunidade de agentes; isto leva a necessidade de definição de uma linguagem comum, um protocolo e um conjunto de regras que guiem a interação entre os processos;

- conhecimento sobre como coordenar suas ações; isto significa que algum tipo de organização é definida quando cada agente começa sua execução;

- vontade própria às suas ações; são dirigidas de acordo com seu interesse; nesta proposta, nenhum agente pode forçar outro a deixar de executar suas tarefas básicas para executar as tarefas solicitadas por outros.



**Figura 6 - Agente Ligado a um Processo**

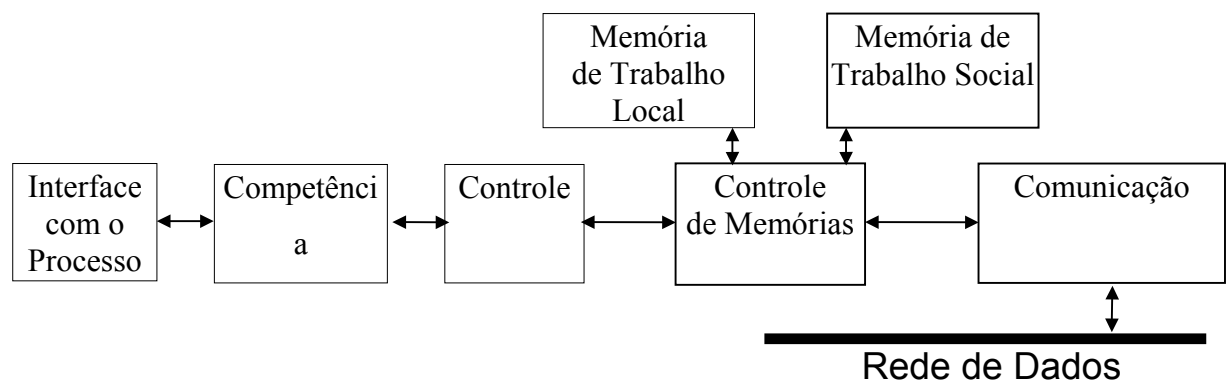
A figura 6 mostra um agente ligado a um processo. Um processo poderá ser um CNC, um sensor simples ou uma base de dados. O Resolvedor de Problemas pode ser um algoritmo, uma rede neural, um

sistema especialista, etc., que cuida de uma parte específica do processo, ao nível de tarefa. Todos terão uma cápsula que provê capacidade, ao nível de comunicação, de cooperação e coordenação, tornando-o um agente diante da comunidade.

### 4.3. A Arquitetura de Um Agente

Um agente neste sistema é particionado como mostra a figura 7. Como indicado anteriormente, um agente pode ser dividido em duas partes distintas: o resolvidor de problemas (*competência*) e a cápsula (interface com os outros agentes). Assim, um SE se tornará um agente assim que lhe for incorporado a cápsula que permite sua comunicação com os demais agentes da comunidade.

Esta adaptação deve ser suave, não exigindo alterações profundas nos algoritmos presentes em cada subsistema responsável pelo tratamento das tarefas a nível local.



**Figura 7 - Estrutura de um Agente**

#### 4.3.1. Competência

O módulo *Competência* nada mais é do que o algoritmo projetado para realizar uma tarefa específica. Em aplicações industriais serão algoritmos ligados a sensores, controladores, atuadores, SE, etc.

### **4.3.2. Controle**

O módulo de *Controle* dará condições ao mesmo de compartilhar seus resultados com outros agentes e também de formatar suas solicitações a outros agentes.

### **4.3.3. Memória de Trabalho Local**

Entende-se por memória de trabalho local uma região onde são armazenadas as informações ao nível de tarefa que o agente pode fornecer a comunidade. Nesta memória estão armazenadas as informações do MP do agente.

Neste trabalho os agentes possuem uma memória de trabalho baseada em fórmulas da lógica predicativa, como as usadas em *Prolog*. As informações estão formatadas em predicados, como será visto adiante.

### **4.3.4. Memória de Trabalho Social**

A memória de trabalho social é a região onde são armazenadas todas as informações sobre os demais agentes da comunidade. Nesta memória estão armazenadas as informações do MS do agente. Estas informações são utilizadas pelo agente quando este deseja alguma informação externa. As informações sobre todos os agentes presentes na sociedade, seu tipo (segundo um dos cinco possíveis), qual a sua situação, etc, são informações presentes na MTS. Esta região é dinâmica, mudando de conteúdo constantemente conforme a configuração da comunidade altera-se.

Quando um novo agente começa a trabalhar, ele informa em *broadcast* aos agentes com quem ele manterá negociação, a sua entrada na comunidade. Todo agente armazena na memória de trabalho social o que o novo agente pode fazer. Quando um agente necessita de alguma informação, ele consulta a memória de trabalho social para definir qual agente da comunidade pode fornecê-la.

### **4.3.5. Controle de Memórias**

Este módulo é uma máquina de inferência com encadeamento para trás (*backward chaining*) capaz de manipular informação das memórias de trabalho local e social. Um avaliador de expressões para interpretar as mensagens passadas pelos agentes, também está presente internamente a este módulo.

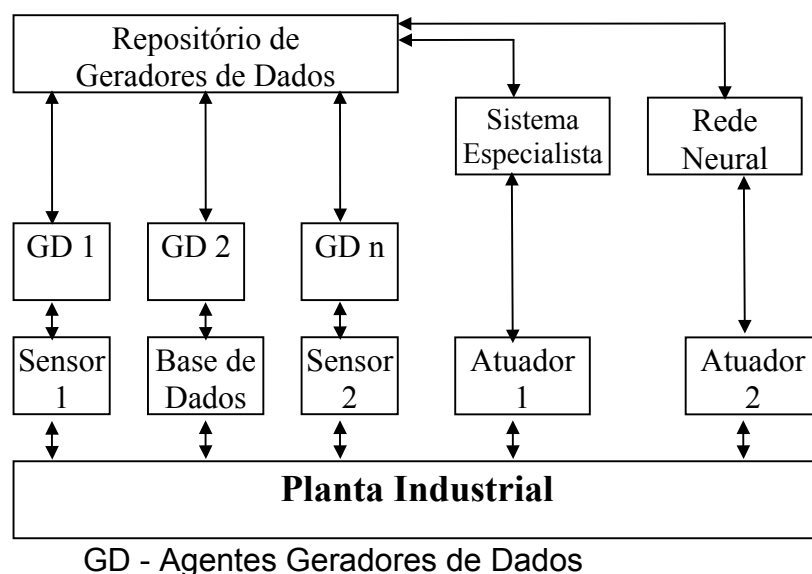
### **4.3.6. Comunicação**

O módulo de *Comunicação* é constituído por um conjunto de rotinas de acesso a rede de transporte de informação. São rotinas TCP/IP de baixo nível escritas em C++, e serão melhores descritas no capítulo que apresenta a implementação do sistema.

## **4.4. Os Tipos de Agentes do Sistema**

A comunidade de agentes deste trabalho foi dividida em tipos, de acordo com os tipos de componentes de uma planta industrial:

- Agentes Geradores de Dados (AGD);
- Agentes Processadores de Dados (APD);
- Agentes Atuadores (AT);
- Agentes de Apresentação de Situação (AAS);
- Agentes Repositórios dos Geradores de Dados (ARE).



**Figura 8 - Visão da Planta**

A figura 8 mostra como será feita a interação entre os agentes, já que cada sub-sistema destes (Senso, Atuador, etc.) será um agente.

A modularização permitirá, modelar a maior parte dos processos de monitoração e controle, e facilitar a troca de dados entre os agentes.

Por exemplo: um AGD é uma fonte de dados e não tem a necessidade de receber mensagens vindas de um AT, já que estas devem ser enviadas apenas aos APD. Um AGD por outro lado, recebe mensagens vindas de um ARE e não envia mensagens aos outros agentes, pois este centraliza suas informações, disponibilizando-as aos agentes que dela necessitem.

É importante salientar que apesar de haver tipos diferentes de agentes, todos estão num mesmo nível hierárquico, não havendo um agente que tenha prioridade sobre outro.

#### 4.4.1. Agentes Geradores de Dados

Este tipo de agente está conectado a sensores, base de dados, e

fontes de dados em geral e é capaz de suprir a comunidade com os mais recentes dados provindos do processo a que o mesmo está ligado. Estes agentes enviam informação espontaneamente aos ARE, pois os ARE armazenam as informações dos AGD, centralizando-as, e tornando-as acessíveis aos agentes APD.

#### **4.4.2. Agentes Processadores de Dados**

Estes agentes retiram informação diretamente dos ARE e as usam para controlar o processo ou os processos. Podem ser algoritmos procedurais (declarativos), SE, RN, etc., e tem o poder de decidir os parâmetros de controle da malha controlada. Quando é detectada uma situação anormal, estes agentes devem suprir a correção necessária, por intermédio dos AT.

Caso o agente do tipo ARE esteja inacessível, este poderá entrar em contato diretamente com os AGD.

#### **4.4.3. Agentes Atuadores**

São os agentes conectados a atuadores ou processos que alteram a situação do processo monitorado. Alguns deles podem receber dados quantitativos como, por exemplo, valores de variáveis para ajustes, ou dados qualitativos como procedimentos de verificação, ajustes, planos de manutenção, etc.

#### **4.4.4. Agentes de Apresentação de Situação**

São agentes que apresentam a situação total ou parcial corrente da planta monitorada. Por exemplo, eles recebem dados vindos da comunidade e apresentam ao usuário através de uma interface gráfica.

#### **4.4.5. Agentes Repositórios dos Geradores de Dados**

Este agente é definido como um repositório de informação. O ARE

concentra todas as informações vindas dos AGD, armazenando-as para serem acessadas pelos agentes do tipo APD ou AAS. Usando esta abordagem pode-se manter informação atualizada acessível por APD a qualquer momento. É importante salientar que os agentes AGD atualizam estes dados espontaneamente. Além disso a quantidade de mensagens é reduzida.

Assumindo que estes grupos de agentes trabalham juntos, a planta pode ser considerada uma malha de controle fechada.

#### **4.5. Mecanismos de Coordenação**

O ambiente de troca de informação está baseado no MPS como em [Malh94] com dois tipos básicos de mensagens: *Broadcast*, quando uma mensagem é enviada a um grupo de agentes da rede e, *Direct* quando uma mensagem é enviada a um agente especificado.

Os agentes enviam suas mensagens e esperam as respostas. Existe um mecanismo de *TimeOut* que impede que um agente fique eternamente esperando por uma resposta. Caso o agente não receba uma resposta, o módulo *competência* deve ser avisado e decidir o que deve ser feito.

Foi utilizado o mecanismo de Mercado como em [Malo87] para coordenar os agentes, sem a utilização de hierarquias.

Para facilitar o acesso aos dados gerados pelos AGD uma estratégia baseada em um agente repositório foi utilizada. O ARE centraliza as informações provenientes de um grupo de AGD. Podem haver vários ARE para vários AGD. Quando um APD ou um AAS necessita de uma informação, ele a buscará em um ARE. Esta abordagem foi utilizada para acelerar o processo de obtenção de conhecimento. Os AGD têm a obrigação de suprir os ARE com as mais recentes informações de seu processo.

Para manter um ambiente organizado, cada agente tem os seguintes princípios:

- agentes não questionam informação sem ter necessidade;



- agentes voluntariamente respondem quando inquiridos.

### **Autonomia do Agente**

Muito tem sido discutido a respeito de autonomia de agentes. Neste trabalho assumiu-se que os agentes são autônomos, atuando de acordo com sua especificação e respondendo a solicitações sempre que possível, no menor tempo possível. Isto significa que um agente responderá a uma solicitação se não tiver outra tarefa prioritária a realizar, no caso, um processamento interno, uma inferência, etc.

## **4.6. Protocolo de Comunicação**

É necessário se especificar um protocolo e uma linguagem comum entre os agentes. Os agentes usarão esta especificação para formatar suas mensagens de comunicação.

Neste projeto está sendo sugerido um conjunto de mensagens para troca de informação. Estas mensagens são utilizadas como transporte e armazenamento de informação simultaneamente. As mensagens são predicados do tipo Prolog sendo inseridas / retiradas da memória de trabalho social / local automaticamente quando necessário. Desta forma tem-se uma unificação da linguagem de Conhecimento com o Protocolo de Comunicação.

### **4.6.1. Protocolo Predicativo**

As mensagens serão formatadas através de um conjunto de predicados de aridade distinta. Quando um agente recebe uma mensagem ele a insere diretamente em sua memória social, pois sabe que ela já está num formato conhecido por ele, e também pelo mecanismo de inferência do controle de memórias, que supre o módulo *competência*.

As mensagens básicas são:

```

agente(<identificador>,'<classificação>')
situação(<identificador>,'<situação>')
sabe(<identificador>,'<expressão>')
informe(<identificador_origem>,<identificador_destino>,' <query> ')
informando(<identificador_origem>,<identificador_destino>,
           '<expressão>', '<geração> ')

```

Usando BNF pode-se definir os parâmetros das mensagens:

```

<string> ::= <ascii>
<variável> ::= <string>
<classificação> ::= 'gera_dados' | 'processa' | 'atua' | 'apresenta' |
'repositório'
<identificador> ::= <string>
<identificador_origem> ::= <string>
<identificador_destino> ::= <string>
<query> ::= <string>
<situação> ::= 'operando' | 'espera' | 'desligando'
<operadores> ::= = | < | >
<expressão> ::= <variável> | <operadores> | <variável>
<geração> ::= <string>

```

Estas mensagens são agrupadas em duas classes: mensagens do tipo *Broadcast* e do tipo *Direct*. Mensagens *Broadcast* são enviadas a um grupo de agentes e do tipo *Direct* para um agente especificado, como a mensagem *informe*.

### - Mensagem Agente

Esta mensagem é enviada quando um agente inicia sua operação na comunidade. Ela informa aos outros agentes o identificador do novo agente e sua classificação, de acordo com os 5 tipos descritos. O identificador será

utilizado pelos outros agentes para direcionar mensagens ao novo agente. Quando um agente recebe esta mensagem ele também se apresenta ao novo agente informando sua classificação e situação. É uma mensagem do tipo *Broadcast*.

#### **- Mensagem Situação**

Quando um agente altera seu estado, ele informa à comunidade sua alteração. Um agente pode estar em um dos 3 estados possíveis descritos a seguir:

- estado *operando*: informa que o agente está apto a receber solicitações;

- estado *esperando*: informa que o agente não pode ser inquirido momentaneamente, em função de um processamento interno ou de algum problema;

- estado *desligando*: informa que o agente está em processo de saída da rede.

Ao receber uma mensagem *desligando*, o agente vai até sua memória de trabalho social e retira todos os predicados referentes aquele agente que está saindo da comunidade.

Esta é uma mensagem do tipo *Broadcast*.

#### **- Mensagem Sabe**

Também é uma mensagem do tipo broadcast que identifica toda informação “compartilhável” de um agente. Podem ser enviadas várias mensagens iguais a esta para cada informação conhecida pelo agente local. No parâmetro *expressão* podem ser definidas variáveis, ou string de caracteres com informações diversas. Caberá a cada agente interpretar a informação que está expressa neste parâmetro.

#### **- Mensagem Informe**

Esta mensagem é utilizada pelo agente para formatar questionamentos a outro agente. É uma mensagem do tipo *Direct*. O agente varre sua memória de trabalho social em busca do agente que possa resolver sua solicitação. Ele encontrará esta resposta verificando todas as mensagens *sabe* disponíveis.

#### **- Mensagem Informando**

Em resposta a uma mensagem *informe*, um agente montará uma mensagem *informando*. A expressão contém a resposta a solicitação *informe*. Usando o parâmetro *identificador\_destino*, ele é capaz de enviar a mensagem diretamente ao agente que a espera. O parâmetro *geração*, informa ao agente receptor a que data e hora foi gerada a informação que está sendo recebida. Isto é importante para que um agente saiba o quão recente é a informação que está recebendo.

### **4.6.2.Exemplo de Utilização do Conjunto de Mensagens**

Para exemplificar o uso do conjunto de mensagens, tem-se um trecho de uma interação entre dois agentes. Na memória de trabalho social do agente 5 encontra-se:

*agente( 10, ' repositório ').*

*situação( 10, ' operando ').*

*sabe( 10, ' temperatura\_1 ').*

O primeiro predicado indica que o agente de número 10 é do tipo *repositório*. O segundo indica que o mesmo pode ser inquirido para uma solicitação.

Os agentes informam sua situação atual aos demais assim que esta se modifique.

Já o terceiro predicado exemplifica um dado fornecido pelo agente 10. Neste caso trata-se de uma variável chamada *temperatura\_1*.

Uma solicitação poderia ser formatada assim pelo agente 5:

```
informe(5, 10, 'temperatura_1 ').
```

Após uma pequena inferência ou consulta a sua memória de trabalho social, o agente 5 que necessita do valor da *temperatura\_1* faz uma solicitação deste dado ao agente 10. Os predicados constantes em sua memória de trabalho levam-no a acreditar que o agente 10 pode fornecer tal informação. O predicado acima possui 3 parâmetros: agente solicitante, agente destino e a informação desejada. O módulo de interface com a rede sabe como remeter esta mensagem ao agente 10.

Já o agente 10, recebendo a solicitação, pode enviar a resposta em um predicado também simples:

```
informando( 10, 5, 'temperatura_1 = 50 ','Seg 30/09/1996 16:15').
```

O agente solicitante recebe o predicado e então avalia o resultado da expressão passada como parâmetro no predicado *informando*.

Um outro exemplo de troca de informação pode ser dado avaliando o trabalho de um agente classificado como aquele que registra os resultados, um AAS. O objetivo destes agentes é permitir uma visão geral da planta controlada, podendo ser executado em uma máquina na sala central de controle. Para poder apresentar a situação dos processos controlados, é necessário que este agente tenha informações sobre cada um destes processos. Os agentes que estão monitorando cada um dos processos informarão seu estado atual ao AAS que poderá apresentá-los prontamente.

De modo geral, agentes que processam dados também terão que solicitar informação a vários agentes. Em geral, estas solicitações serão a um ARE. O interessante neste caso é que há necessidade de se processar uma

inferência para se definir qual agente deverá ser acionado. Primeiro é feita uma verificação na memória de trabalho social sobre qual agente fornece o conhecimento necessário. Depois verifica-se se o mesmo está apto a responder uma solicitação (verificando o predicado *situação*). Caso o agente remoto não esteja apto, cabe ao agente local decidir o que fazer. Havendo a possibilidade da resposta o pedido é feito. Este encadeamento é facilmente resolvido com um mecanismo de *backtracking* que deve estar presente em todos os agentes, no módulo Controle de Memórias, que como já foi descrito, é uma máquina de inferência com encadeamento para trás.

## **5. Implementação do SMA**

Apresenta-se a seguir aspectos relacionados à implementação de um protótipo baseado na proposta de SMA indicada.

### **5.1. Plataforma de Desenvolvimento**

O sistema foi desenvolvido para operar em ambiente UNIX. Como plataforma de hardware foram utilizadas máquinas SUN Sparc 1000 e o sistema operacional Solaris.

Como ferramentas de desenvolvimento foi utilizado o SparkWorks C++ e inicialmente o BinProlog para construção de um mecanismo de inferência, o qual posteriormente foi implementado em C++. Estas ferramentas foram utilizadas pois eram as que estavam disponíveis para a implementação.

### **5.2. Os Agentes Vistos como Processos**

Utilizando-se o ambiente multi-tarefa UNIX, vários processos podem ser executados paralelamente. Neste caso, cada agente é um processo distinto, que pode ou não ser executado em uma mesma máquina. A interação entre os agentes está baseada no protocolo de comunicação TCP/IP, sendo implementadas rotinas para comunicação via sockets.

### **5.3. Classe de Baixo Nível para Comunicação**

Após estudo do protocolo de comunicação TCP/IP (maiores detalhes no Anexo I), foi implementada uma classe de baixo nível, capaz de transmitir e receber mensagens. Este protocolo foi utilizado pois se tornou padrão para sistemas de comunicação abertos na indústria da computação. Sistemas computacionais do mundo inteiro usam o protocolo TCP/IP para comunicação

pois este provê um alto grau de interoperabilidade, e é compatível com a grande maioria das tecnologias de rede [Come91].

Existem dois procedimentos de conexão no protocolo TCP/IP, o formato TCP e o UDP. O formato TCP foi inicialmente implementado, mas, não é o atualmente utilizado, pois não permitiu várias conexões para um mesmo agente. A solução então foi implementar o formato UDP, que é apresentado na classe *Communication* listada no Anexo II.

Esta classe é, no modelo do agente, o módulo *Comunicação* que faz a interface com a rede de dados. A codificação desta classe pode ser encontrada no Anexo II.

Esta classe implementa todas as funções necessárias para obter uma porta de comunicação (*socket*) e rotinas de envio e recepção de mensagens (independente do formato).

Quando um agente é carregado para a memória, ou seja, inicia sua execução, uma das primeiras tarefas é criar uma porta de comunicação UDP. Ele faz isto criando um objeto do tipo *Communication*. É criada uma porta para envio de informação e outra para recepção.

Existem dois construtores nesta classe. Um deles recebe como parâmetro o endereço IP onde está sendo executado o agente. O outro construtor recebe como parâmetro dois valores inteiros. O primeiro deles informa se a porta que está sendo aberta será utilizada para envio ou recepção de mensagens. O segundo parâmetro indica qual porta será utilizada. Maiores detalhes sobre as funções utilizadas podem ser obtidos no Anexo I.

#### **5.4. Fluxo de Recepção das Mensagens**

Quando um agente recebe uma mensagem da rede de dados, esta deve ser interpretada, para que o agente tome as ações necessárias. A seguir tem-se, em forma de fluxogramas, o processo de recepção de cada uma das mensagens do conjunto de mensagens definido. Alguns módulos



aparecem em vários gráficos. Entre eles tem-se:

- *Receive*: a função *Receive*, da classe *Communication*, é responsável pela leitura da mensagem da rede de dados;

- *Send*: a função *Send*, da classe *Communication*, é responsável pelo envio de uma mensagem a comunidade através da rede de dados;

- Compilação da Mensagem: compila a mensagem indicando qual tipo de mensagem foi recebida (de acordo com as 5 possíveis);

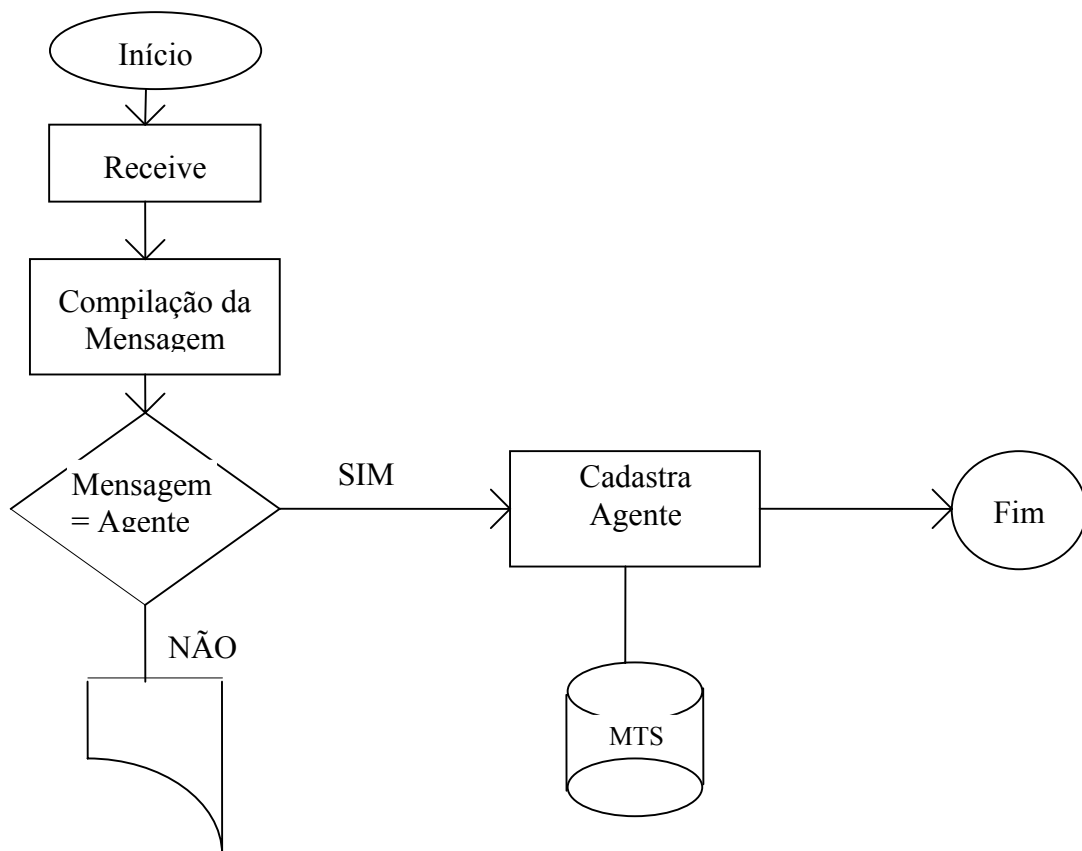
- Base de dados MTS: é a Memória de Trabalho Social;

- Base de dados MTL: é a Memória de Trabalho Local;

- *Avalia Expressão*: esta função avalia a expressão vinda em parâmetros de mensagens tais como *sabe*, *informe* e *informando*.

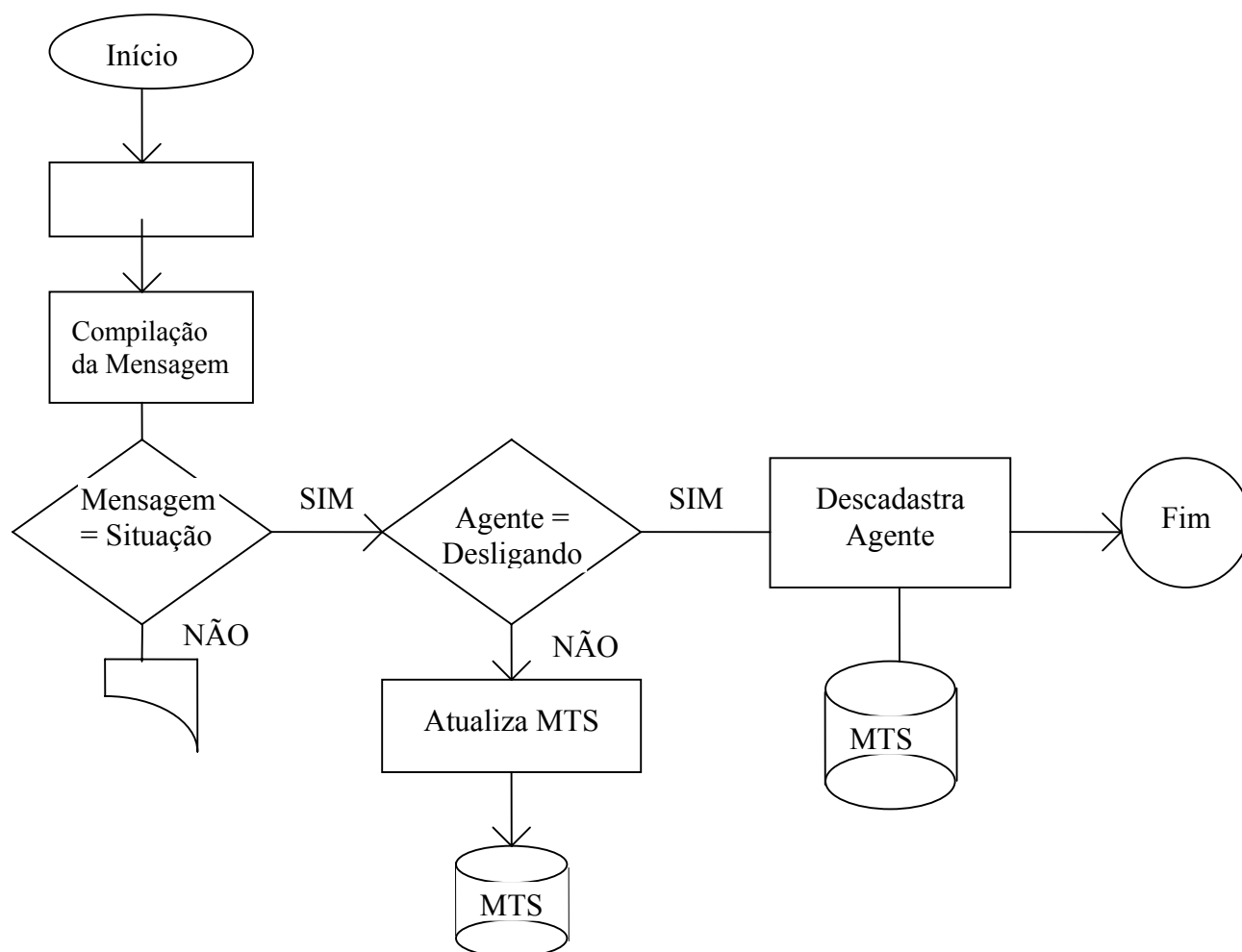
### 5.4.1. Mensagem Agente

No fluxograma a seguir tem-se o processo de interpretação da mensagem *Agente* e as ações que são tomadas na sua recepção. Após interpretada a recepção de uma mensagem do tipo *Agente*, a função *CadastaAgente* é chamada para cadastrá-lo na MTS.



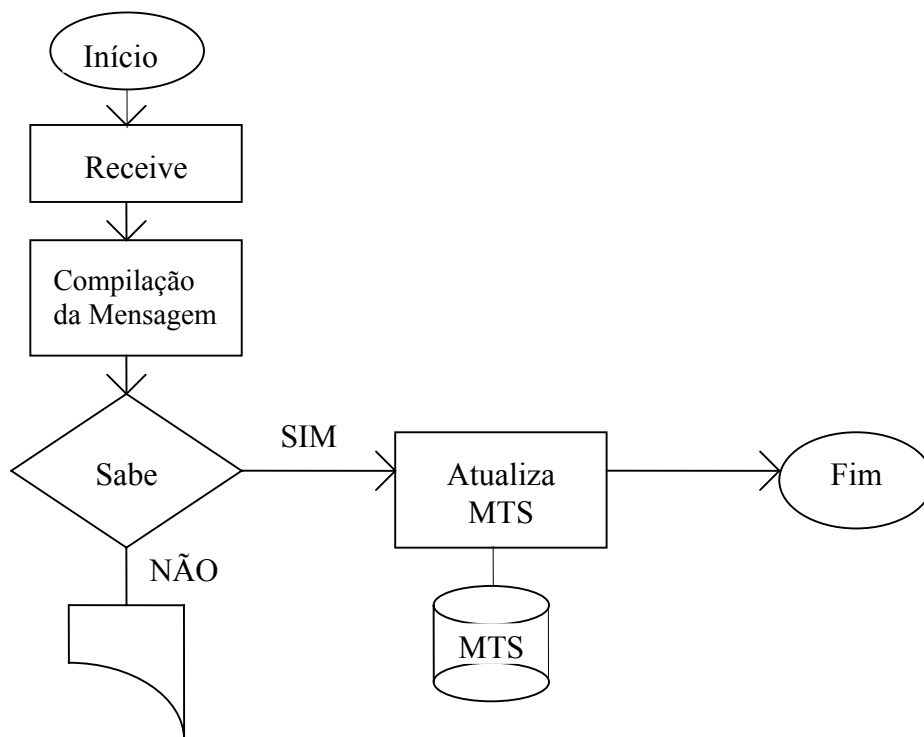
### 5.4.2. Mensagem Situação

A mensagem *Situação* informa a situação de determinado agente. A MTS é atualizada com a situação do agente emissor. Caso o parâmetro da mensagem indique que o agente emissor está se desligando, então o agente local chama a função *DescadastraAgente*, que retira todas as informações sobre aquele agente da MTS.



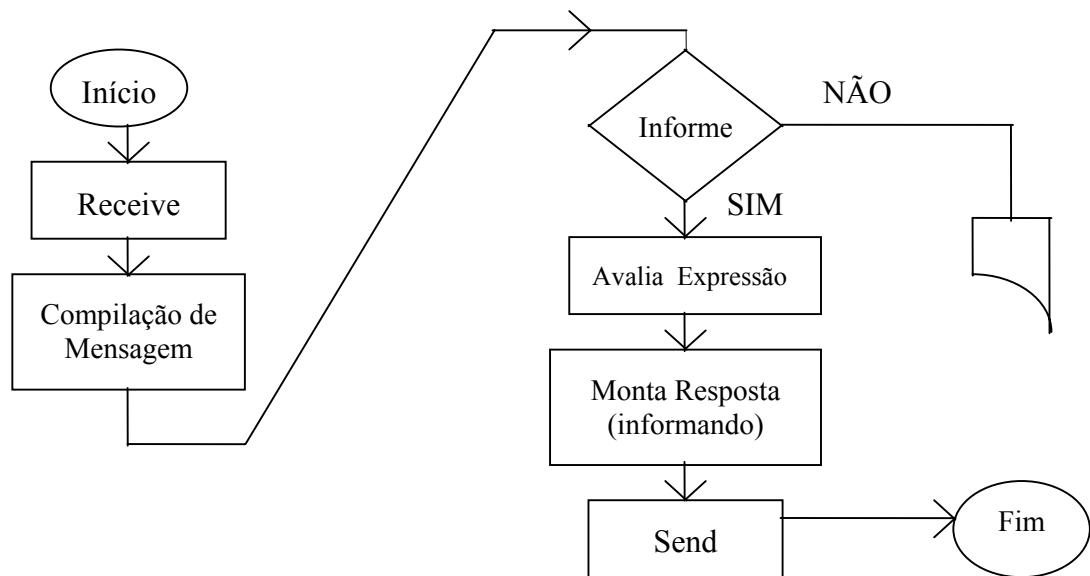
### 5.4.3. Mensagem Sabe

Na recepção da mensagem *Sabe*, a MTS é atualizada com as informações vindas no parâmetro *expressão* da mensagem.



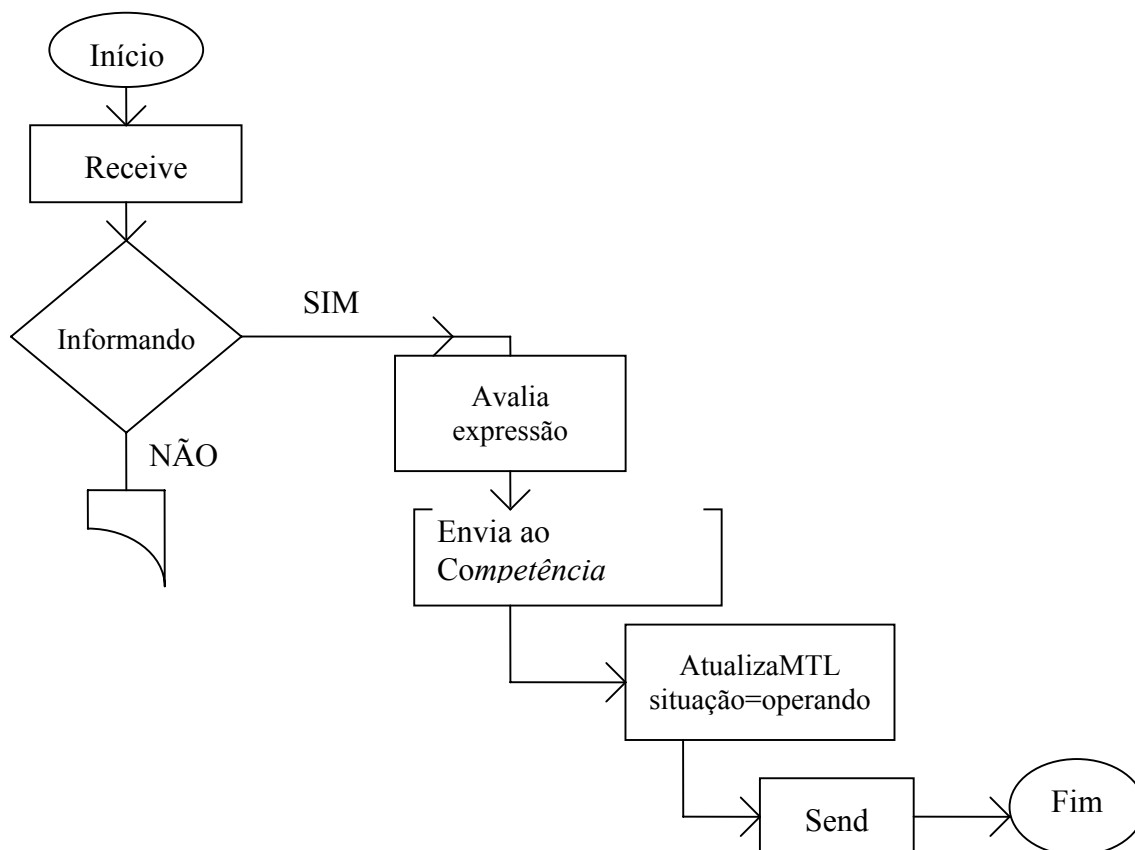
#### 5.4.4. Mensagem *Informe*

Se a mensagem recebida for *informe*, isto significa que há algum agente solicitando uma informação do agente local. O agente receptor avalia a expressão, para definir o que está sendo solicitado. Após isto o agente monta a resposta, utilizando a mensagem *informando* e a envia através de *send* ao agente solicitante.



### 5.4.5. Mensagem Informando

O agente, ao enviar uma mensagem do tipo *informe*, entra automaticamente em estado de espera. Ao receber sua resposta esta é passada ao módulo *Competência*, e a situação do agente é alterada para *operando*.



## 5.5. Fluxo de Envio das Mensagens

Nos tópicos anteriores viu-se como o agente se comporta na recepção de mensagens. A seguir apresenta-se o processo de envio de cada uma das mensagens.

### 5.5.1. Mensagem Agente

Ao entrar na rede o agente envia uma mensagem do tipo *agente* informando sua identificação e seu tipo (gera\_dados, atua, processa, apresenta ou repositório).

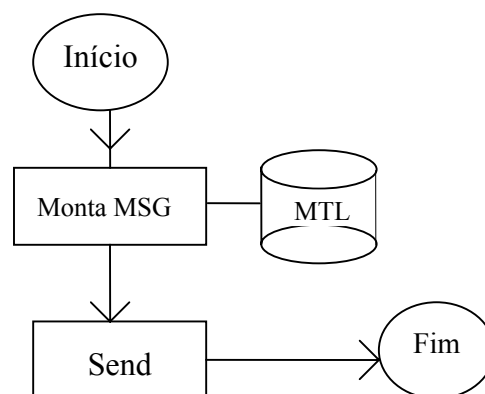
Devido a construção do ambiente, cada agente envia esta mensagem a um grupo restrito de agentes. Na tabela abaixo tem-se a indicação do envio da mensagem *agente* em função do tipo do agente.

Tipo do Agente Emissor	Tipo do Agente Receptor
Geradores de Dados	- Repositório dos Geradores de Dados
Repositório dos Geradores de Dados	- Processadores de Dados - Mostradores de Situação
Processadores de Dados	- Repositório dos Geradores de Dados - Atuadores
Atuadores	- Processadores de Dados
Mostradores de Situação	- Repositório dos Geradores de Dados

**Tabela 2 - Quadro de Relação de Envio de Mensagens por Tipo Agente**

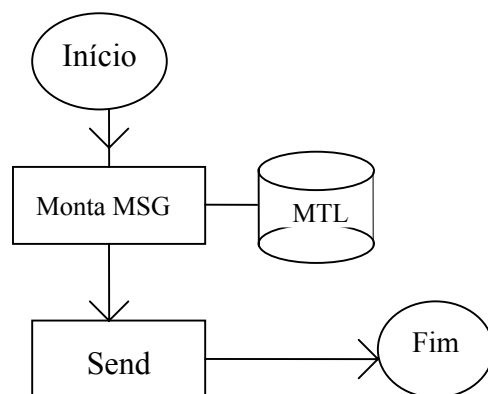
Da tabela pode-se inferir que um agente do tipo *Processa\_Dados* só troca mensagens com agentes do tipo *Repositório* e *Atuadores*. A razão desta formatação já foi vista quando da definição dos tipos de agentes.

Assim ao enviar uma mensagem do tipo *agente*, o emissor retira da MTL sua identificação e sua classificação.



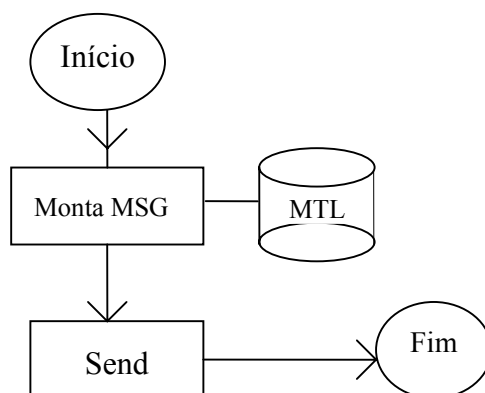
### 5.5.2. Mensagem *Situação*

O agente envia esta mensagem para informar sua situação atual. O destino desta mensagem também respeita a relação apresentada na Tabela 2.



### 5.5.3. Mensagem *Sabe*

O agente informa o que pode fornecer a comunidade através do envio desta mensagem aos agentes de seu interesse.

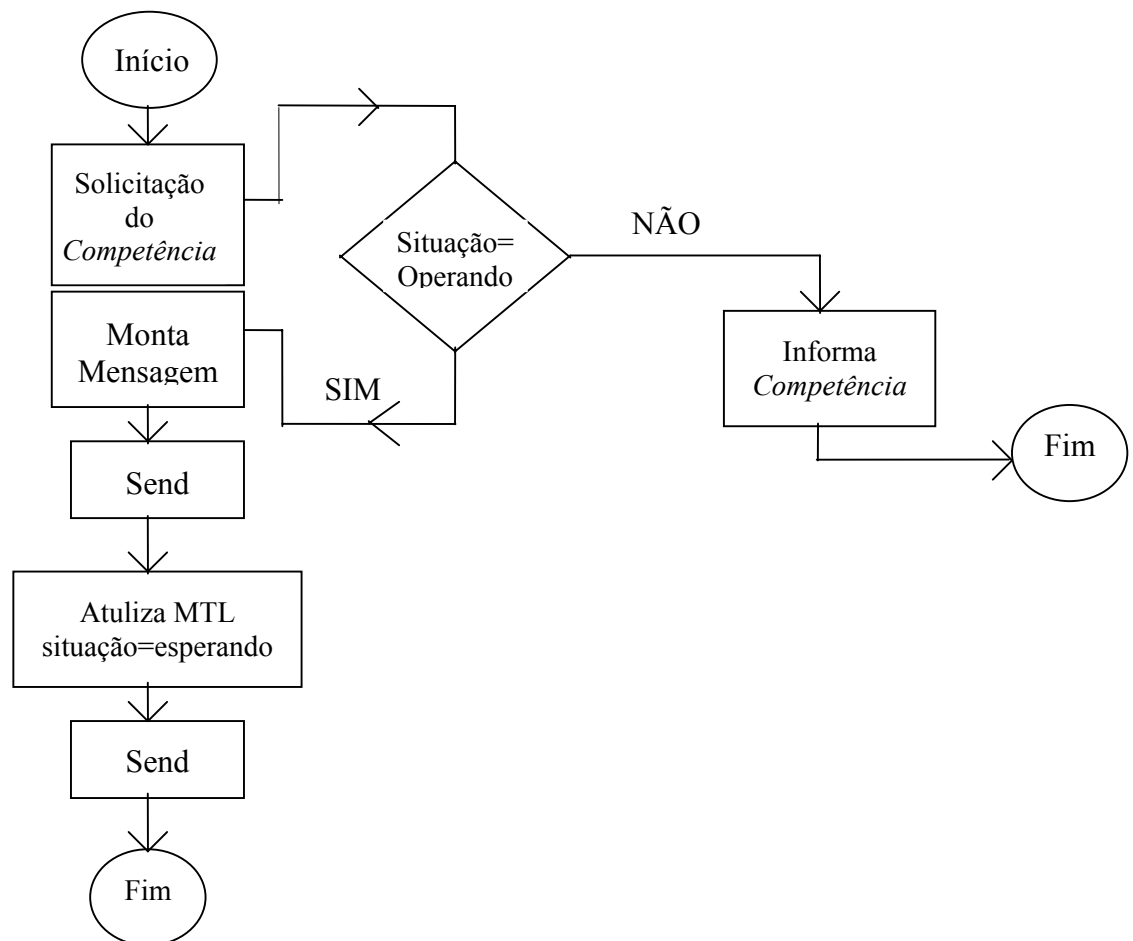




### 5.5.4. Mensagem *Informe*

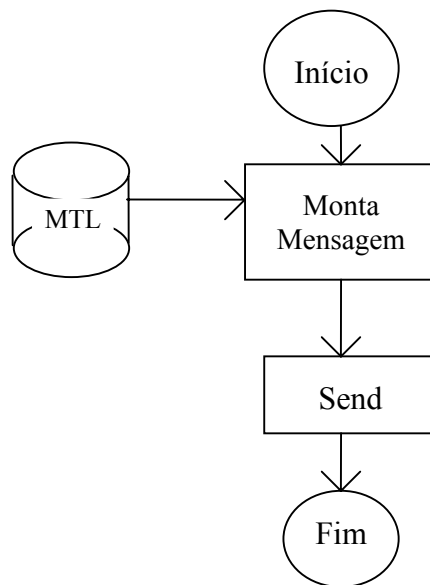
Quando o módulo *Competência* necessita de uma informação externa o agente formata uma mensagem do tipo *informe*. Após verificar qual agente pode fornecer tal informação na MTS, o agente emissor deve verificar também se o possível agente receptor não está em estado de espera. Caso isto ocorra o agente remoto está impedido de negociar informações, cabendo ao agente local decidir o que fazer.

Se a situação do agente remoto for *operando*, então a mensagem é enviada e a situação do agente emissor passa a ser *esperando*. Em seguida, uma mensagem do tipo *situação* é gerada para informar a comunidade a situação do agente.



### 5.5.5. Mensagem Informando

Recebendo uma mensagem *informe*, o agente avalia a expressão do parâmetro *expressão*, constrói uma outra expressão como resposta, e a envia ao agente solicitante.



### 5.6. Rotinas de Compilação de Endereço

Cada agente deve ser identificado para entrar na comunidade. Existe um arquivo de configuração onde estão descritos todos os agentes existentes na comunidade. O formato deste arquivo é o seguinte:

```

agente(PRESSAO, "200.19.78.3", 6005)
agente(TEMP, "200.19.78.3", 6006)
agente(BBGD, "200.19.78.3", 6007)
agente(SE, "200.19.78.3", 6008)
agente(ATUAPRESS, "200.19.78.3", 6009)
  
```

No exemplo abaixo, *BBGD* é o identificador do agente, *200.17.98.3* é o endereço *IP* do agente e *6007* é a porta de comunicação.

```
agente(BBGD, "200.17.98.3", 6007)
```

Todo agente possui uma função chamada *Le\_Address* listada no Anexo II, que faz parte de seu módulo de interface com a rede de dados. A informação compilada do arquivo agente.cfg é armazenada nas estruturas AgInfo e AgData.

### **5.7. Rotinas de Cadastramento de Agentes**

Quando um agente entra na comunidade ele deve informar esta condição. Quando uma mensagem *agente* é recebida, a rotina CadastraAgente é acionada para armazenar as informações pertinentes na estrutura AgData.

As informações passadas a esta função são a mensagem recebida e um identificador interno ao agente. Esta rotina também faz parte do módulo de interface com a rede de dados.

### **5.8. Rotina de Descadastramento de Agentes**

Assim que o agente decide sair da rede, ele deve informar à comunidade sua decisão. Um agente recebendo a mensagem *desligando* retira as informações do agente que enviou a mensagem da memória de trabalho social.

A variável *buffer* passada como parâmetro é um vetor de caracteres que contém a mensagem enviada pelo agente que está desligando.

### **5.9. Memórias de Trabalho**

As estruturas listadas no Anexo II são utilizadas pelos agentes como memória de trabalho social e local. Como as memórias tem a mesma

estrutura, o agente apenas cria duas variáveis diferentes de cada tipo.

### **5.10. Máquina de Inferência**

Em função da simplicidade das operações realizadas na MTS e MTL, a máquina de inferência (MI) implementada é simples. Cabe a MI localizar nas memórias de trabalho os predicados referentes a um determinado agente. Isto é realizado por um mecanismo de *matching* e *backtracking*.

A função *InformaValor*, informa o resultado da inferência. Verificando o campo *Variable* da estrutura *AgData*, a MI é capaz de inferir quais dados existem na MTL e MTS, e então construir o predicado desejado.

### **5.11. Armazenamento de Informação na MTS**

Ao receber uma mensagem *sabe*, o agente deve atualizar sua MTS. Para isto é chamada a função *AgenteSabe* que é encarregada de acrescentar à MTS o predicado que acaba de chegar.

### **5.12. Atualização da MTS e MTL**

Quando é recebida uma mensagem *informando*, o agente deve atualizar sua MTS. Isto é feito pela função *AtualizaValor*. Os campos *Variable* e *Value* são atualizados pela função.

## 6. Demonstração de Utilização

Para verificar o funcionamento do protótipo do sistema implementado, foi construída uma simulação de um ambiente de controle, constituído de dois agentes do tipo AGD, um agente ARE, um agente APD e dois agentes AT. O domínio dos agentes AGD é a temperatura e pressão de um determinado processo existindo também um atuador correspondente a cada sensor. Para atuar como processo foram criados dois arquivos que armazenam o valor da temperatura e da pressão. Os agentes ATs atualizam estes arquivos com valores vindos do APD. Já os agentes do tipo AGD lêem o arquivo e retiram o valor atualmente armazenado.

Para ilustrar como funciona a interação entre os agentes, cada um deles imprime na tela as principais mensagens que recebe e envia.

Quando um agente imprime na tela uma mensagem recebida ou enviada, a título de ilustração, ele imprime também uma indicação se a mensagem foi recebida (R:) ou enviada (S:).

**Agente do tipo ARE:** inicialmente o agente imprime o resultado da compilação do arquivo agente.cfg, que contém a porta de comunicação para cada agente. Na seqüência ele indica a recepção de uma mensagem *agente* vinda do agente AGD pressão. Em seguida o agente informa, através da mensagem *sabe* e da mensagem *informando*, o valor da variável pressão. O Agente AGD temperatura faz o mesmo. Após a identificação do agente APD, no caso um SE, este solicita informação referente a variável pressão. O agente ARE então, responde com a mensagem *informando*.

Toda mensagem do tipo *informe* é impressa na tela. Quando um agente AGD inicia o processo de desconexão, ele envia uma mensagem

*situação* informando que está desligando.

```

cmdtool - /bin/sh
[ eparaiso@tales ]bbgd
Agente: PRESSAO End: 192.9.200.1 Porta: 6005
Agente: TEMP End: 192.9.200.1 Porta: 6006
Agente: SE End: 192.9.200.1 Porta: 6008

R: agente(PRESSAO,GERA_DADOS)

R: sabe(PRESSAO,pressao)
Agente = PRESSAO, Sabe = pressao
AgCont = 1
R: informando(PRESSAO,BB,pressao = 0,Mon Sep 30 10:09:01 1996)
Valor: pressao = 0
Hora e Data de Recepcao: Mon Sep 30 10:09:01 1996
R: agente(TEMP,GERA_DADOS)

R: sabe(TEMP,temp)
Agente = TEMP, Sabe = temp
AgCont = 2
R: informando(TEMP,BB,temp = 0,Mon Sep 30 10:09:04 1996)
Valor: temp = 0
Hora e Data de Recepcao: Mon Sep 30 10:09:04 1996
R: agente(SE,COMPUTA_DADOS)

R: informe(SE,BB,pressao)
Buffern: informando(BB,SE,pressao = 0,Mon Sep 30 10:09:01 1996)
Porta de resp = 6008
S: informando(BB,SE,pressao = 0,Mon Sep 30 10:09:01 1996)

R: informando(PRESSAO,BB,pressao = 0,Mon Sep 30 10:09:24 1996)
Valor: pressao = 0
Hora e Data de Recepcao: Mon Sep 30 10:09:24 1996
R: informe(SE,BB,pressao)
Buffern: informando(BB,SE,pressao = 0,Mon Sep 30 10:09:24 1996)
Porta de resp = 6008
S: informando(BB,SE,pressao = 0,Mon Sep 30 10:09:24 1996)

R: informando(PRESSAO,BB,pressao = 20,Mon Sep 30 10:09:32 1996)
Valor: pressao = 20
Hora e Data de Recepcao: Mon Sep 30 10:09:32 1996
R: informe(SE,BB,temp)
Buffern: informando(BB,SE,temp = 0,Mon Sep 30 10:09:04 1996)
Porta de resp = 6008
S: informando(BB,SE,temp = 0,Mon Sep 30 10:09:04 1996)

R: situacao(PRESSAO,desligando)
Agente: PRESSAO Desconectando

R: situacao(TEMP,desligando)
Agente: TEMP Desconectando

R: situacao(SE,desligando)
Agente: SE Desconectando
[ eparaiso@tales ]_

```

### Agente do tipo ARE

**Agente AGD pressão:** o AGD pressão informa o valor da variável

pressão ao agente ARE. Esta informação é retirada de um arquivo que representa o processo monitorado.

```
cmdtool (CONSOLE) - /bin/sh
[ eparaiso@tales ]gerpres
Pressao do arq = 0
S: informando(PRESSAO,BB,pressao = 0,Mon Sep 30 10:09:01 1996)
Pressione S para sair.

Pressao do arq = 0
S: informando(PRESSAO,BB,pressao = 0,Mon Sep 30 10:09:24 1996)
Pressione S para sair.

Pressao do arq = 20
S: informando(PRESSAO,BB,pressao = 20,Mon Sep 30 10:09:32 1996)
Pressione S para sair.
S
[ eparaiso@tales ]
```

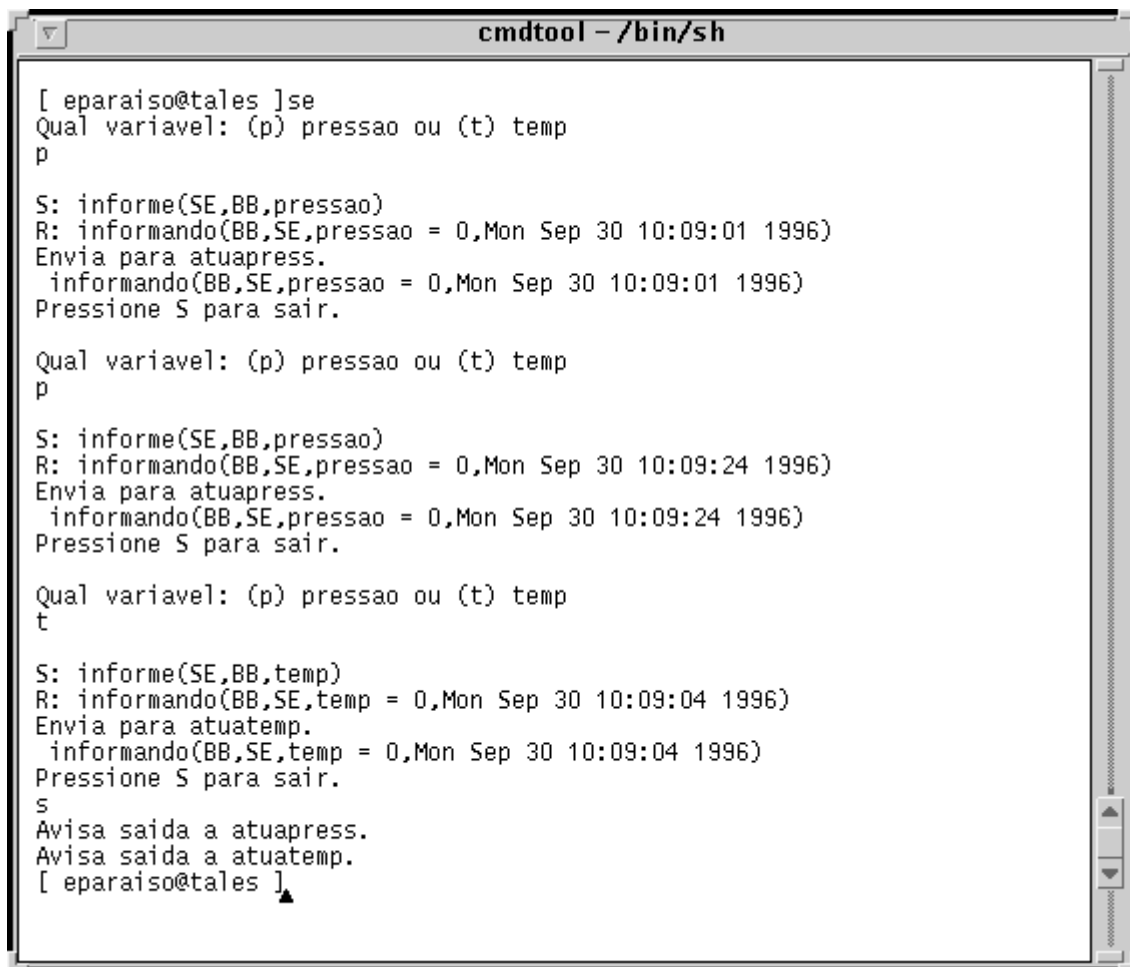
### Agente AGD Pressão

**Agente AGD temperatura:** o AGD temperatura informa o valor da variável temperatura ao agente ARE. Esta informação é retirada de um arquivo que representa o processo monitorado.

```
cmdtool - /bin/sh
[ eparaiso@tales ]gertemp
Temp. do arq = 0
S: informando(TEMP,BB,temp = 0,Mon Sep 30 10:09:04 1996)
Pressione S para sair.
S
[ eparaiso@tales ]
```

### Agente AGD Temperatura

**Agente APD:** o agente APD implementado, apenas solicita uma informação ao agente ARE e a repassa ao AT correspondente. Ao sair ele também informa sua decisão aos agentes ATs.



```
cmdtool - /bin/sh

[ eparaiso@tales ]se
Qual variavel: (p) pressao ou (t) temp
p

S: informe(SE,BB,pressao)
R: informando(BB,SE,pressao = 0,Mon Sep 30 10:09:01 1996)
Envia para atuapress.
  informando(BB,SE,pressao = 0,Mon Sep 30 10:09:01 1996)
Pressione S para sair.

Qual variavel: (p) pressao ou (t) temp
p

S: informe(SE,BB,pressao)
R: informando(BB,SE,pressao = 0,Mon Sep 30 10:09:24 1996)
Envia para atuapress.
  informando(BB,SE,pressao = 0,Mon Sep 30 10:09:24 1996)
Pressione S para sair.

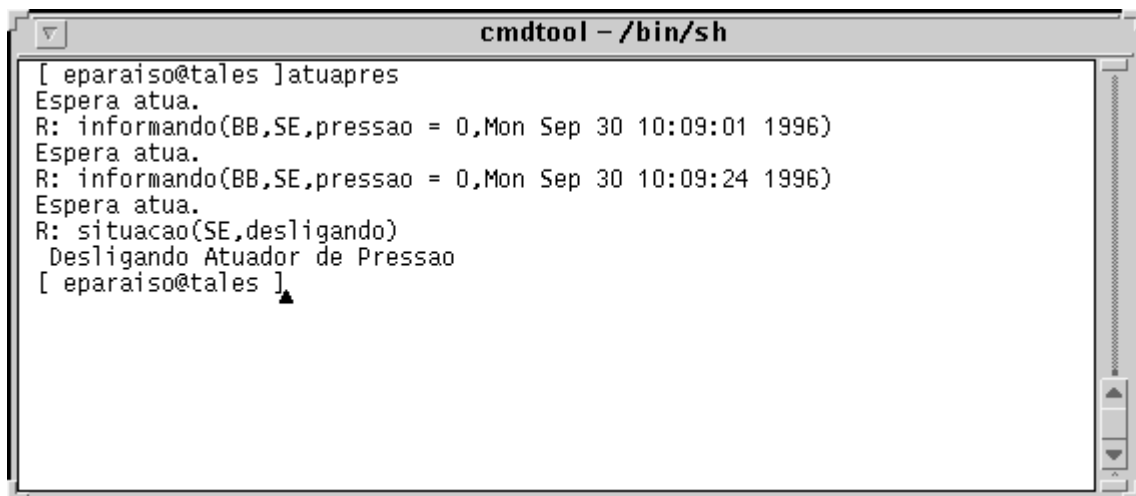
Qual variavel: (p) pressao ou (t) temp
t

S: informe(SE,BB,temp)
R: informando(BB,SE,temp = 0,Mon Sep 30 10:09:04 1996)
Envia para atuatemp.
  informando(BB,SE,temp = 0,Mon Sep 30 10:09:04 1996)
Pressione S para sair.
s
Avisa saida a atuapress.
Avisa saida a atuatemp.
[ eparaiso@tales ]
```

### Agente APD

**Agente AT pressão:** este agente recebe um dado vindo do agente APD e o escreve em um arquivo, que simboliza aqui o processo sendo controlado.






```
cmdtool - /bin/sh
[ eparaiso@tales ]atuapres
Espera atua.
R: informando(BB,SE,pressao = 0,Mon Sep 30 10:09:01 1996)
Espera atua.
R: informando(BB,SE,pressao = 0,Mon Sep 30 10:09:24 1996)
Espera atua.
R: situacao(SE,desligando)
  Desligando Atuador de Pressao
[ eparaiso@tales ]
```

### Agente AT Pressão

**Agente AT temperatura:** este agente recebe um dado vindo do agente APD e o escreve em um arquivo, que simboliza aqui o processo sendo controlado.



```
cmdtool - /bin/sh
[ eparaiso@tales ]atuatemp
Espera atua.
R: informando(BB,SE,temp = 0,Mon Sep 30 10:09:04 1996)
Espera atua.
R: situacao(SE,desligando)
  Desligando Atuador de Temepratura
[ eparaiso@tales ]
```

### Agente AT Temperatura



## 7. Conclusão

A utilização de técnicas de IAD aplicadas em sistemas no domínio do controle de processos industriais é uma realidade. Alguns exemplos de aplicações bem sucedidas foram apresentadas neste trabalho.

O objetivo maior deste trabalho foi apresentar uma proposta alternativa ao controle e monitoração de processos industriais. Desta proposta foi gerado um protótipo capaz de manipular o conjunto de mensagens proposto. Este protótipo permitiu validar o conjunto de mensagens e as rotinas de comunicação. De sua utilização pode-se concluir que:

- a aplicação dos SMA em monitoração e controle de processos é interessante quando os dados e principalmente o controle é distribuído;
- as aplicações em tempo real devem receber cuidado especial, evitando-se o excesso de trocas de mensagens;
- os agentes do SMA devem todos estar interessados em cooperar, evitando-se concorrência entre eles;
- os agentes devem ser especializados, permitindo que eles executem tarefas específicas, evitando-se agentes que executam várias tarefas importantes para o controle da planta;
- o conjunto de mensagens utilizado pelos agentes deve ser rico o suficiente para permitir a formatação de qualquer pedido pelos agentes;
- os agentes devem ser distribuídos tomando-se cuidado especial com configurações altamente dependentes que podem gerar inclusive problemas de *Deadlock*;
- os agentes devem ser capaz de gerar sinais como *Time-Out* quando

da não recepção de uma mensagem de resposta a uma solicitação.

O ambiente proposto deverá ser utilizado em um projeto conjunto envolvendo o CEFET-PR e o TECPAR. O objetivo deste projeto é gerar uma especificação de um SMA para uma aplicação no domínio do controle de processos. Como base deste estudo está a aplicação do sistema na planta de monitoração e controle da corrosão na refinaria Getúlio Vargas em Araucária no Paraná, da Petrobrás.

A proposta permite visualizar alguns trabalhos futuros que podem ser obtidos a partir desta proposta, como por exemplo:

- permitir que os agentes adquiram conhecimento durante sua execução, utilizando técnicas de aprendizado;
- implementar um *shell* de geração de agentes, baseado na arquitetura proposta;
- implementar o sistema em outras plataformas de *software* e *hardware*;
- adaptar o sistema para outros domínios de aplicação;

## Referências Bibliográficas

- [Bond88] Bond A. H.; Gasser L. "Reading in Distributed Artificial Intelligence", Morgan Kaufmann, 1988.
- [Come91] Comer, Douglas; Stevens, Daves. "Internetworking with TCP/IP Volume II", Prentice Hall International Editions, 1991.
- [Cost95] Costa, E. de B.; Lopes, M.; Ferneda, E., "MATHEMA: A Learning Environment Based on a Multi-Agent Architecture", Advances in Artificial Intelligence, 1995.
- [Cycl96] Cyclades, "Guia Internet de Conectividade", User's Guide, 1996.
- [Davi81] Davis R.; Smith, R. G., "Negotiation as a Metaphor for Distributed Problem Solving", Artificial Intelligence 20, 1981.
- [Fini94] Finin, T.; McKay, D.; Fritzson R.; McEntire R. "The KQML Knowledge Exchange Protocol. Third International Conference on Information and Knowledge Management", CIKM'94. National Institute of Standards and Technology, Gaithersburg, Maryland, 1994.
- [Gene92] Genesereth, M. R.; Fikes, R. E., "Knowledge Interchange Format, Version 3.0", Reference manual. Computer Science Department, Stanford University, Technical Report Logic-92-1, 1992.
- [Gene94] Genesereth, M. R.; Ketchpel, S. P., "Software Agents". Stanford University Logic Group, 1994.
- [Haye85] Hayes-Roth, B., "A BlackBoard Architecture for Control", Artificial Intelligence 26, 1985.

- [Malh94] Malheiro, B.; Oliveira, E., "An Intelligent Distributed System for Environmental Management", 1994.
- [Malo87] Malone, T., "Modeling Coordination in Organizations and Markets". Management Science, 1987.
- [McGu93] McGuire, J.; Pelavin, R.; Shapiro, S.; Beck, C.; Finin, T.; Weber, J.; Wiederhold, G.; Genesereth, M.; Fritzson, R.; Mckay, D., "Draft: Specification of the KQML Agent-Communication Language", 1993.
- [Para95] Paraiso, E. C.; Kaestner, C. A. A.; Ramos, M. P.; Ribeiro, F. G., "Proposta de um Ambiente Multi-Agente para Monitoração e Controle de Processos Industriais", II Simpósio Brasileiro de Automação Inteligente, 1995.
- [Para95a] Paraiso, E. C.; Ramos, M. P.; Correa, L. A., "Corrosion Control Under a Multi-Agent Approach", CORROSION 96, 1996.
- [Para95b] Paraiso, E. C.; Zanoni, J. C.; Kobren, M., "Estudo da Aplicabilidade de Sistemas Multi-Agente e da Linguagem KQML para a Monitoração e Controle de Processos Industriais", III Encontro de Estudantes de Informática do Paraná", 1995.
- [Ramo95] Ramos, M. P.; Correa, L. A., "On-Line Corrosion Control in Refinery Overhead System", CORROSION 95, 1995.
- [Stef95] Stefanini, M.; Demazeau, Y., "TALISMAN: A Multi-Agent for Natural Language Processing", Advances in Artificial Intelligence, 1995.
- [Srid87] Sridharam, N. S., "1986 Workshop on Distributed AI", AI Magazine

Fall, 1987.

[Well95] Wellman, M., "Market-Oriented Programming (Abstract)", *Advances in Artificial Intelligence*, 1995.

[Witt92] Wittig, T., "ARCHON: an Architecture for Multi-Agent Systems", Ellis Horwood, 1992.





## Anexo I - O Protocolo TCP/IP

TCP/IP (*Transmission Control Protocol /Internet Protocol*) é um conjunto de protocolos de comunicação utilizado para troca de dados entre computadores em ambientes LAN (rede local) ou WAN (rede de longa distância) [Cycl96].

Com a rede Arpanet criada em 1969 pelo ARPA (*Advanced Research Projects Agency*) do Departamento de Defesa dos EUA, iniciou-se um processo de pesquisa em software de redes de comunicação de computadores, resultando no TCP/IP. O TCP/IP realiza funções como o roteamento de informações através de redes locais (LAN) e redes de longa distância (WAN), emulação de terminal, transferência de arquivos e correio eletrônico.

O protocolo TCP/IP é adotado comercialmente por muitos ambientes como Unix, Novell, Windows NT e OS/2. O TCP/IP teve um grande reconhecimento quando a Microsoft decidiu usá-la no Windows NT e Windows 95. Atualmente a Novell suporta TCP/IP no NetWare 4.X.

Existem vários produtos que implementam o protocolo TCP/IP, entre eles tem-se:

- TCP/IP para PC's desktop (MS-Windows; OS/2 e MS-DOS)
- TCP/IP para servidores (Windows NT, Unix, OS/2 LAN, Novell NetWare)
- AIX TCP/IP para máquinas RS/6000, IBM RT PC e IBM PS/2 modelo 80
- VM TCP/IP para ambientes MVS

Os pacotes TCP/IP para PC's implementam padrões TCP/IP e NFS. Este programa faz a comunicação entre os PC's sob vários sistemas operacionais como por exemplo Windows NT, Novell, Unix, entre outros, oferecendo tipicamente os seguintes serviços:

- Telnet
- TN3270
- Serviços de impressora
- Back-up remoto
- Comandos Rlogin, Rexerc, Pcp e Rsh.
- TFTP cliente e servidor
- Ping
- Finger
- Whois
- Nickname
- Programas de impressão de spool e Imagem laser print

## **Arquitetura TCP/IP**

A arquitetura do TCP/IP é dividida em 4 camadas (veja Fig 1). A camada física do modelo OSI não está especificada pelo TCP/IP.

Normalmente utiliza-se os protocolos Ethernet, Token-Ring, PPP, X.25, Frame Relay, etc., como enlace do TCP/IP.

A próxima camada é a IP, responsável pelo roteamento e retransmissão de mensagens para a rede até a mensagem chegar ao computador destino. É o IP que “sabe” qual a melhor rota que uma dada mensagem deve seguir na rede até chegar ao destino desejado. O roteamento é possível porque cada host tem um endereço próprio na rede.

A camada TCP é responsável pela transferência segura de mensagens entre os nós finais (origem e destino). Ele tem a capacidade de estabelecer “conexões virtuais” de tal forma que as aplicações possam enviar e receber mensagens sem a preocupação de terem que gerenciar retransmissão,

controle de seqüência, perda de integridade e controle de fluxo.

APPLICATION PROTOCOLS	SERVICE PROTOCOL	ROUTING	PHSICAL NETWORK
KERB	TCP	IP	ETHERNET TOKEN RING PPP X.25 FRAME RELAY
XWIN			
REXEC			
SMTP			
TELNET			
FTP			
DNS			
TFTP			
RPC	UDP	ICMP	
NFS			
NCS			
SNMP			
TN3270			
		ARP	
		RARP	

### Arquitetura TCP/IP

A camada de aplicação contém vários níveis de protocolos de nível de aplicação como TELNET (*Terminal Emulation*), STMP (*Simple Mail*) e FTP (*File Transfer*)

### Protocolos TCP/IP

Existem muitos protocolos de aplicação, entre eles tem-se:

- TELNET

Protocolo de emulação ao terminal que permite aos usuários acessar aplicações em outros sistemas. A emulação fornecida é do tipo VT110, VT 220. Esta emulação é composta de telas não gráficas. O VT110/VT220 é bastante usado em terminais *tty* (terminais não inteligentes).

- SMTP

Sistema de correio eletrônico para transmissor e receptor.

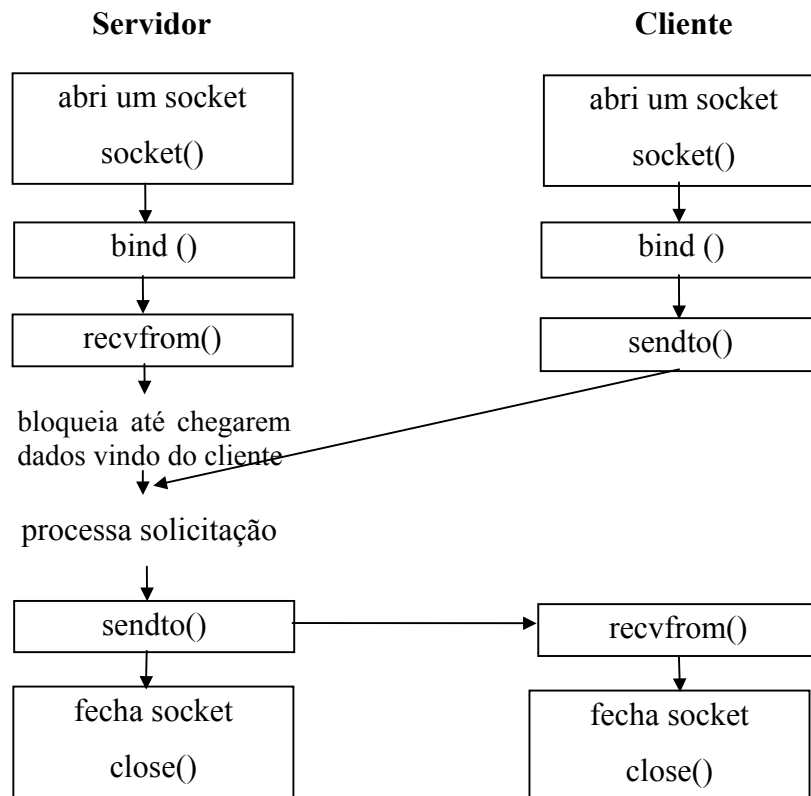
- FTP/TFTP

FTP é protocolo que tem a capacidade de conectar-se a um sistema remoto (*logon*) e fazer o acesso de diretórios e a transferência de arquivos entre estes sistemas. O TP inclui senhas de segurança. O TFTP diferencia-se do FTP por não acessar diretórios e não utilizar senhas de segurança.

- UDP

UDP (*User Datagram Protocol*) é o protocolo mais comum usado por programas “*application-to-application*”entre máquinas TCP/IP, sendo portanto um protocolo em nível de pacotes.

No sistema implementado foi utilizado o procedimento UDP na classe *Communication*. O UDP mantém uma associação sem conexão entre os processos, diferentemente do TCP. O processo de envio e recepção das mensagens é baseado no esquema cliente/servidor. A figura a seguir ilustra o processo de comunicação entre dois agentes.



### UDP - Associação sem Conexão

Um servidor abre um *socket* e fica esperando uma solicitação. O cliente também abre um *socket* e envia uma solicitação ao servidor. O servidor responde a solicitação e fecha a *socket*, o mesmo acontecendo no lado do cliente que recebe a mensagem e fecha a *socket*.

Para implementar o protocolo UDP são utilizadas as chamadas ao sistema mostradas na figura.

#### Chamada de Sistema *Socket*

Esta função retorna um inteiro semelhante a um descritor de arquivo que será usado nos outros comandos de comunicação. Ela é responsável por obter um canal de comunicação do sistema. Este número é limitado pois o número de *sockets* é limitado.

```
# include <sys/types.h>
```

```
# include <sys/socket.h>
```

```
int socket(int família, int tipo, int protocolo)
```

1° argumento: AF\_UNIX - protocolo interno do Unix

AF\_INET - protocolo Internet

AF\_NS - protocolo Xerox Ns

2° argumento: SOCK\_STREAM - TCP

SOCK\_DGRAM - UDP

3° argumento: para a maioria das aplicações do usuário é igual a 0.

### **Chamada de Sistema *Bind***

Um servidor registra o seu endereço no sistema local. Ambos os tipos de servidores (com e sem conexão) precisam executar um *bind* antes de aceitar solicitações dos clientes.

Um cliente com conexão pode registrar um endereço específico para si (opcional).

Um cliente sem conexão precisa assegurar que o sistema reserve para si algum endereço único, de modo que o servidor tenha um endereço de retorno válido para onde enviar as respostas.

```
# include <sys/types.h>
```

```
# include <sys/socket.h>
```

```
int bind (int sockfd, struct sockaddr * locaddr, int addrlen);
```

1° argumento - uma socket criada por uma chamada socket anterior

2° argumento - ponteiro para uma estrutura contendo endereço local ( end. Internet e porta)

3° argumento - tamanho do 2° argumento

### **Chamadas de Sistema *Sendto* e *Recvfrom***

Usado para transmitir e receber mensagens

```
# include <sys/types.h>
```

```
# include <sys/socket.h>
```

```
int sendto (int sockfd, char *buf, int nbyte, int flag,      struct  
sockaddr*to,  
int addrlen);
```

```
int recvfrom (int sockfd, char *but, int nbyte, int flag, struct sockaddr*to,  
int addrlen);
```

1° argumento - mesmo significado do que read/write

2° argumento - idem

3° argumento - idem

4° argumento - MSG\_OOB : dados out-of-band  
MSG\_PEEK ; examinar os dado  
MSG\_DONTROUT ; não rotear

5° argumento - ponteiro para estrutura contendo o endereço do parceiro

6° argumento - tamanho do 5° argumento

### **Chamadas de Sistema *Close***

Libera a *socket* já aberta.

```
int close (int fd);
```

## Anexo II - Código dos Agentes Implementados

### 1. Classe de Baixo Nível para Comunicação

```

class Communication
{
    private:
        struct sockaddr_in    serv_addr;
        struct sockaddr_in    cli_addr;
        int                   sockfd;

    public:
        Communication (char*);
        Communication (int, int);
        ~Communication ();
        int Socket     ();
        int Bind       ();
        int Send       (char *, int);
        int Receive    (char *, int);
        int Status     ();
};

Communication::Communication(char * serv)
{
    memset((char *)&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family        = AF_INET;
    serv_addr.sin_addr.s_addr   = inet_addr(serv);
    serv_addr.sin_port          = htons(6005);

    memset((char *)&cli_addr, 0, sizeof(cli_addr));
    cli_addr.sin_family         = AF_INET;
    cli_addr.sin_addr.s_addr    = htonl(INADDR_ANY);
    cli_addr.sin_port           = htons(0);
}

Communication::Communication(int type, int port)

```



```

{
    memset((char *)&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family      = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port       = htons(port);
    if(type == 1)
        cli_addr = serv_addr; /* Recebe informação */
}

Communication::~Communication()
{
    close(sockfd);
}

int Communication::Socket()
{
    if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        return -1;
    return sockfd;
}

int Communication::Bind()
{
    return(bind(sockfd, (struct sockaddr *) &cli_addr,
sizeof(cli_addr)));
}

int Communication::Send(char * msg, int msglen)
{
    return(sendto(sockfd, msg, msglen, 0, (struct
sockaddr *) &serv_addr, sizeof(sockaddr_in)));
}

int Communication::Receive(char * buffer, int bufferlen)
{
    int aux = sizeof(struct sockaddr);

```

```

        return(recvfrom(sockfd,    buffer,    bufferlen,    0,
(struct sockaddr *) &serv_addr, &aux));
    }

int Communication::Status()
{
    return 1;
}

```

## 2. Rotinas de Compilação de Endereço

Todo agente possui uma função chamada *Le\_Address* listada abaixo, que faz parte de seu módulo interface com a rede de dados. A informação compilada do arquivo agente.cfg é armazenada nas estruturas *AgInfo* e *AgData*.

```

struct AgInfo{
    char Name[10];
    char Address[15];
    int  Port;
};

struct AgData{
    char Id[15];
    char Classification[15];
    char Variable[20];
    char Value[30];
    char Time[100];
};

AgInfo pAgentsClient[10];
AgData pAgentsData[10];

```

```

/* Compila arquivo .CFG com enderecos de agentes */
void Le_Address(void)
{
    FILE * arq;
    int  nagent, l;
    char aux[6];
    char ch;

    nagent = 0;
    arq = fopen("agente.cfg","r+");
    while(fgetc(arq) != '(');
    do{
        l = 0;
        do{
            pAgentsClient[nagent].Name[l] =
fgetc(arq);
            l++;
        }while(pAgentsClient[nagent].Name[l-1] != ',');
        pAgentsClient[nagent].Name[l-1] = 0;

        fgetc(arq); /* consome primeiro " */

        l = 0;
        do{
            pAgentsClient[nagent].Address[l] =
fgetc(arq);
            l++;
        }while(pAgentsClient[nagent].Address[l-1] !=
'"');
        pAgentsClient[nagent].Name[l-1] = 0;

        fgetc(arq); /* consome segundo " */
        fgetc(arq); /* consome , */

        l = 0;
        do{

```

```

        aux[l] = fgetc(arq);
        l++;
    }while(aux[l-1] != ')');
    aux[l-1] = 0;
    pAgentsClient[nagent].Port = atoi(aux);
    do{
        ch = fgetc(arq);
    }while(ch != EOF || ch != '(');
    nagent++;
}while(ch == '(');
fclose(arq);
}

```

### 3. Rotinas de Cadastramento de Agentes

Quando um agente entra na comunidade ele deve informar que está entrando na rede. Quando uma mensagem *agente* é recebida, a rotina *CadastraAgente* é acionada para armazenar as informações pertinentes na estrutura *AgData*.

```

void CadastraAgente(char * buffer)
{
    int l, t;

    t = 0;
    l = 0;
    while(buffer[l] != '(') l++;
    l++; /* consome ( */
    do{
        pAgentsData[AgCont].Id[t] = buffer[l];
        l++; t++;
    }while(buffer[l] != ',');
    pAgentsData[AgCont].Id[t] = 0;
    t = 0;
    do{
        pAgentsData[AgCont].Classification[t] =
buffer[l];

```

```

        l++; t++;
    }while(buffer[l] != ' ');
    pAgentsData[AgCont].Classification[t] = 0;
}

```

As informações passadas a esta função são a mensagem recebida e um identificador interno ao agente. Esta rotina também faz parte do módulo de interface com a rede de dados.

#### 4. Rotina de Descadastramento de Agentes

Assim que o agente decide sair da rede, ele deve informar a comunidade sua decisão. Um agente recebendo a mensagem *desligando*, irá repassá-la para a função listada a seguir para retirada das informações do agente da memória de trabalho social.

```

void DesconectaAgente(char * buffer)
{
    int l, t, j;
    char agente[15];

    l = 0;
    while(buffer[l] != '(') l++;
    l++; /* consome ( */
    t = 0;
    do{
        agente[t] = buffer[l];
        l++; t++;
    }while(agente[t-1] != ',');
    agente[t-1] = 0;
    for(t = 0; t < AgCont; t++){
        if(!strcmp(agente,pAgentsData[t].Id)) break;
    }
    printf("Agente: %s Desconectando\n",agente);
    AgCont--;
}

```

```
}
```

## 5. Memórias de Trabalho

As estruturas listadas abaixo são utilizadas pelos agentes como memória de trabalho social e local. Como as memórias são idênticas, o agente apenas cria duas variáveis diferentes de cada tipo.

```
struct AgInfo{
    char Name[10];
    char Address[15];
    int Port;
};
```

Campo Name: este campo armazena o nome de cada agente sobre o qual existe informação na MTS;

Campo Address: armazena o endereço IP de cada agente sobre o qual existe informação na MTS;

Campo Port: porta de comunicação com cada agente.

```
struct AgData{
    char Id[15];
    char Classification[15];
    char Variable[20];
    char Value[30];
    char Time[100];
};
```

Campo Id: identificação dos agentes;

Campo Classification: classificação de cada agente;

Campo Variable: informações que os agentes podem fornecer;

Campo Value: campo utilizado para receber uma expressão;

Campo Time: data e hora em que foi gerada a informação.

## 6. Máquina de Inferência

```
char * InformaValor(int * port, char * buffer)
{
    int l, t, t1;
```

```

char agente[15];
char buffern[128];
char expressao[50];

l = 0;
while(buffer[l] != '(') l++;
l++; /* consome ( */
t = 0;
do{
    agente[t] = buffer[l];
    l++; t++;
}while(agente[t-1] != ',');
agente[t-1] = 0;
for(t = 0;t < AgCont; t++){
    if(!strcmp(agente,pAgentsData[t].Id)) break;
}
strcpy(buffern,"informando(BB,");
strcat(buffern,agente);
strcat(buffern,",");
while(buffer[l] != ',') l++;
l++; t1 = 0;
do{
    expressao[t1] = buffer[l];
    l++; t1++;
}while(buffer[l] != ')');
expressao[t1] = 0;
for(t1 = 0;t1 < AgCont; t1++){
    if(!strcmp(expressao,pAgentsData[t1].Variable))
break;
}
strcat(buffern,pAgentsData[t1].Value);
strcat(buffern,",");
strcat(buffern,pAgentsData[t1].Time); /* Data e Hora
de chegada do valor */
strcat(buffern,"");
for(t = 0;t < AgCont+1; t++){

```

```

        if(!strcmp(agente,pAgentsClient[t].Name))
break;
    }
    *port = pAgentsClient[t].Port;
    printf("Buffern: %s\n",buffern);
    return buffern;
}

```

## 7. Armazenamento de Informação na MTS

```

void AgenteSabe(char * buffer)
{
    int l, t, nagent;
    char agente[30];

    l = 0;
    while(buffer[l] != '(') l++;
        l++;
    t = 0;
    do{
        agente[t] = buffer[l];
        l++; t++;
    }while(buffer[l] != ',');
    agente[t] = 0;
    for(nagent = 0;nagent < AgCont; nagent++){
        if(!strcmp(agente,pAgentsData[nagent].Id))
break;
    }
    l++; t = 0;
    do{
        pAgentsData[nagent].Variable[t] = buffer[l];
        l++; t++;
    }while(buffer[l] != ')');
    pAgentsData[nagent].Variable[t] = 0;
    printf("Agente      =      %s,      Sabe      =
%s\n",pAgentsData[nagent].Id,
pAgentsData[nagent].Variable);
}

```



```

    AgCont++;
    printf("AgCont = %d",AgCont);
}

```

## 8. Atualização da MTS e MTL

```

void AtualizaValor(char * buffer)
{
    int l, t, j;
    char agente[15];

    l = 0;
    while(buffer[l] != '(') l++;
    l++; /* consome ( */
    t = 0;
    do{
        agente[t] = buffer[l];
        l++; t++;
    }while(agente[t-1] != ',');
    agente[t-1] = 0;
    for(t = 0;t < AgCont; t++){
        if(!strcmp(agente,pAgentsData[t].Id)) break;
    }
    while(buffer[l] != ',') l++;
    l++; /* consome , */
    j = 0;
    do{
        pAgentsData[t].Value[j] = buffer[l];
        l++; j++;
    }while(pAgentsData[t].Value[j-1] != ',');
    pAgentsData[t].Value[j-1] = 0;
    printf("Valor: %s\n",pAgentsData[t].Value);
    j = 0;
    do{
        pAgentsData[t].Time[j] = buffer[l];
        l++; j++;
    }while(buffer[l] != ' ');
}

```

```
    pAgentsData[t].Time[j] = 0;
    printf("Hora      e      Data      de      Recepcao:
%s",pAgentsData[t].Time);
}
```