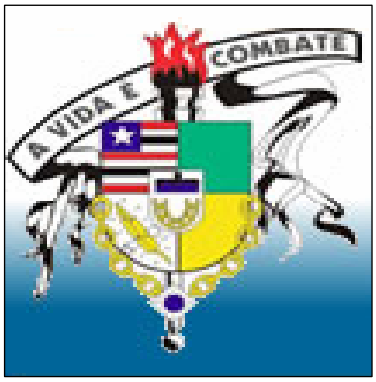

UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

REINALDO DE JESUS DA SILVA

SIMCQC: SISTEMA INTELIGENTE PARA MONITORAMENTO E CONTROLE DA QUALIDADE DE COMBUSTÍVEL

São Luís
2008

UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

REINALDO DE JESUS DA SILVA

SIMCQC: SISTEMA INTELIGENTE PARA MONITORAMENTO E CONTROLE DA QUALIDADE DE COMBUSTÍVEL

São Luís
2008

REINALDO DE JESUS DA SILVA

SIMCQC: SISTEMA INTELIGENTE PARA MONITORAMENTO E CONTROLE DA QUALIDADE DE COMBUSTÍVEL

Dissertação apresentada ao Programa de Pós-graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão, para obtenção do grau de Mestre em Engenharia de Eletricidade, na área de Ciência da Computação.

Orientador: Prof. Dr. Sofiane Labidi.

São Luís
2008

Silva, Reinaldo de Jesus da
SIMCQC: sistema inteligente para monitoramento e controle da qualidade de combustível/:Reinaldo de Jesus da Silva – São Luís, 2008.
110 f.

Dissertação (Pós-Graduação em Engenharia de Eletricidade) – Universidade Federal do Maranhão – UFMA, 2008.

1. Inteligência Artificial. 2. Agentes Inteligentes. 3. Data Warehouse. 4. JESS. 5. PASSI. I.Título: Sistema inteligente para monitoramento e controle de combustível. II. Título.

CDU 004.8:662.6/9

SIMCQC: SISTEMA INTELIGENTE PARA MONITORAMENTO E CONTROLE DA QUALIDADE DE COMBUSTÍVEL

Reinaldo de Jesus da Silva

Dissertação aprovada em 28 de junho de 2008.

Prof. Sofiane Labidi, Dr.
(Orientador)

Prof. Pedro Porfírio Muniz Farias, Dr.
(Membro da Banca Examinadora)

Prof. Zair Abdelouahab, Ph.D.
(Membro da Banca Examinadora)

A Deus, apoio espiritual constante.

A Bianca e Renan, filhos amados, alegrias de minha vida, razões de minha luta, sempre.

A minha Esposa, Rita de Cássia Montenegro, e aos meus irmãos, presença e apoio incondicional.

AGRADECIMENTOS

A minha família, apoio constante em tudo que faço.

A minha mãe e meu pai pelo amor e dedicação recebidos durante toda a minha vida, pois tenho certeza de que o sonho, não só deles, mas de todos os pais, é de ver os seus filhos conquistando mais vitórias.

A minha esposa Rita de Cássia Montenegro pela compreensão, carinho, apoio e incentivo.

Ao Prof. Dr. Sofiane Labidi pela orientação e, sobretudo, por sua capacidade de dividir conhecimento e experiência, que se fez presente quando foi necessário o apoio nos momentos difíceis dessa jornada.

Ao Prof. Milson Monteiro pelas contribuições no desenvolvimento desta pesquisa.

A Ana Claudia Montenegro pela normalização desta pesquisa.

Ao Alcides, Secretário da Coordenação, por sua atenção e paciência em fornecer informação sobre o andamento do mestrado.

A Prof. Mekaele Frota pelo incentivo e pela força durante o desenvolvimento desta pesquisa.

Aos colegas do Laboratório de Sistemas Inteligentes (LSI) que contribuíram para o desenvolvimento desta pesquisa.

A todas as pessoas que contribuíram direta ou indiretamente para a realização desta pesquisa.

"A mente que se abre a uma nova idéia jamais voltará a seu tamanho original".

Albert Einstein

RESUMO

A Agência Nacional de Petróleo, Gás Natural e Biocombustíveis instituiu o Programa de Monitoramento da Qualidade de Combustíveis Automotivos, com o objetivo de avaliar permanentemente a qualidade dos combustíveis e mapear os problemas de não conformidade às normas estabelecidas. A pesquisa propõe o desenvolvimento de um sistema para o monitoramento da qualidade de combustíveis aplicando a metodologia PASSI e técnicas provenientes da Inteligência Artificial, utilizando agentes inteligentes, assim como a plataforma JADE e a linguagem JESS juntamente com Data Warehouse para criar um repositório de dados que facilite o gerenciamento e o apoio à tomada de decisão do Laboratório de Análise e Pesquisa em Química Analítica de Petróleo da Universidade Federal do Maranhão.

Palavras-chave: Inteligência Artificial. Agentes Inteligentes. Data Warehouse. JESS. PASSI.

ABSTRACT

The National Agency, Natural Gas and Biofuels, established the Program for Monitoring the Quality of Fuel Automotive, to evaluate continuously the quality of fuels and map the problems of non-compliance to standards. The research proposes the development of a system for monitoring the quality of fuel PASSI applying the methodology and techniques from artificial intelligence, using intelligent agents, and the JADE platform and language along with JESS Data Warehouse to create a repository of data that facilitates the management and support for decision-making of the Laboratory for Analysis and Research in Analytical Chemistry of Oil of the Federal University of Maranhão.

Keywords: Artificial Intelligence, Intelligent Agents, Data Warehouse, JESS, PASSI.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SIGLAS

AI	-	Agentes Inteligentes
AID	-	Agent Identifier
ACC	-	Agent Communication Channel
AMS	-	Agent Management System
ANP	-	Agência Nacional do Petróleo, Gás Natural e Biocombustíveis
CSELT	-	Centro de Estudo e Laboratório de Telecomunicação
CORBA	-	Common Object Request Broker Architecture
CNP	-	Conselho Nacional de Petróleo
DF	-	Directory Facilitator
DNC	-	Departamento Nacional de Combustível
DW	-	Data Warehouse
FIPA	-	Foundation for Intelligent Physical Agents
FIPA-ACL	-	Foundation for Intelligent Physical Agents-Agent Communication Language
HOLAP	-	Hybrid Online Analytical Processing
HTML	-	Hyper Text Markup Language
HTTP	-	Hypertext Transmission Protocol
IA	-	Inteligência Artificial
IIOP	-	Internet Inter-ORB Protocol
JADE	-	Java Agent Development Framework
JESS	-	Java Expert System Shell
KQML	-	Knowledge Query and Manipulation Language
LAPQAP	-	Laboratório de Análise e Pesquisa em Química Analítica do Petróleo
LGPL	-	Lesser General Public License
MASE	-	Multiagent Systems Engineering
MESSAGE	-	Methodology for Engineering Systems of Software Agents
MQC	-	Sistema de Monitoramento da Qualidade do Combustível
MOLAP	-	Multidimensional Online Analytical Processing
OLE	-	Object Linking and Embedding
OLAP	-	Processamento Analítico Online
PASSI	-	Process for Societies Specification and Implementation
PTK	-	PASSI Toolkit
PMQC	-	Programa de Monitoramento e Controle da Qualidade de Combustíveis
ROLAP	-	Relation Online Analytical Processing
TILAB	-	Telecomm Italia Lab
UFMA	-	Universidade Federal do Maranhão
XML	-	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

A Agência Nacional do Petróleo, Gás Natural e Biocombustíveis (ANP) instituiu o Programa de Monitoramento e Controle da Qualidade de Combustíveis (PMQC), com o objetivo de garantir padrões de qualidade nas fases de extração, refino, distribuição e monitoramento.

O processo de fiscalização do PMQC abrange diversas fases – da produção à comercialização dos produtos petrolíferos - e é composto pelas fases de coleta, tratamento dos dados e análise das amostras dos combustíveis.

Neste sentido, a ANP celebrou convênios com instituições federais, estaduais e municipais, delegando competências e responsabilidades para os respectivos órgãos.

Em 2001, através do contrato nº 4067/2001, foi assinado o convênio entre a Agência Nacional do Petróleo, Gás Natural e Biocombustíveis e a Universidade Federal do Maranhão (ANP/UFMA) para o Monitoramento da Qualidade dos Combustíveis Automotivos comercializados no Estado do Maranhão.

Neste período, as instituições conveniadas a ANP desenvolveram metodologias semi-automatizadas para desenvolver estas fases e observou-se que as fases do processo estão “inter-relacionadas”, porque uma fase fornece subsídios para as outras, mas, ao término de cada fase, precisa-se da análise realizada pela equipe técnica das fases anteriores para iniciar a realização da próxima fase.

A operacionalização do processo apresentava inconsistências que eram ocasionadas por diversos fatores, dentre os quais podemos elencar:

- Ausência de integração das bases de dados existentes;
- As fases não estavam completamente automatizadas;
- Não existia relatório gerencial por um determinado período de tempo;
- Não possuía padronização no processamento de comunicação com as bases de dados do sistema MQC da ANP.

Esta realidade motivou o desenvolvimento de um sistema inteligente que automatizasse o processo de monitoramento e controle da qualidade de combustíveis, levando em consideração os fatores elencados anteriormente. Por este motivo, a pesquisa foi direcionada inicialmente para as áreas da Inteligência Artificial (IA) e Banco de Dados (BD) para fundamentar conceitualmente a compreensão do domínio do problema.

A análise do domínio do problema possibilitou que, durante o desenvolvimento da pesquisa, esta fosse conduzida para a área de Sistemas Inteligentes (SI) e Data Warehouse (DW).

A estrutura da dissertação está organizada em cinco capítulos, que são:

– O Capítulo 2 mostra os conceitos e fundamentos dos Sistemas Inteligentes (SI);

– O Capítulo 3 contextualiza os fundamentos e modelos de Data Warehouse (DW);

– O Capítulo 4 mostra o projeto do sistema inteligente no processo de monitoramento e controle da qualidade de combustível;

– O Capítulo 5 apresenta o desenvolvimento de um protótipo de um sistema inteligente para realizar o monitoramento e controle da qualidade de combustível no estado do Maranhão;

– Finalmente, o Capítulo 6 apresenta as conclusões da pesquisa.

2 SISTEMAS INTELIGENTES

A evolução do desenvolvimento científico e tecnológico ocasionou o desenvolvimento de técnicas computacionais que podem auxiliar a resolução de problemas complexos. Na década de 1950, surgiu a Inteligência Artificial com o propósito inovador de estudar o comportamento inteligente dos seres humanos, visando compreender e simular esses comportamentos em um ambiente computacional, originando uma nova área de conhecimento na inteligência artificial denominada de “sistemas inteligentes”.

Para Russell (2004), os sistemas inteligentes têm como objetivo simular situações do mundo real, por exemplo, através da percepção, compreensão, investigação, além de buscar desenvolver áreas de aprendizagem para tomada de decisões.

Segundo Wooldridge (2002), foram apresentadas diversas técnicas para desenvolver sistemas inteligentes. As aplicações destas técnicas podem ser facilmente encontradas em: controle de tráfego aéreo, controle de processos, logística de transportes, sistema de diagnóstico, planejamento logístico, gestão da informação, educação à distância, gerenciamento de redes, entre outros.

De acordo com Rezende (2005), as características principais de um Sistema Inteligente são:

- Habilidade para usar conhecimento para desempenhar tarefa ou resolver problemas complexos;

- Capacidade para aproveitar associações e estabelecer inferências para resolução de problemas complexos;

Assim, para um sistema computacional ser “Inteligente”, ele precisa de pelo menos um subconjunto dessas habilidades e conhecer como elas modelam tarefas específicas, buscando um encadeamento de decisões, onde a escolha da decisão é seqüencialmente sincronizada e orientada ao objetivo do sistema.

Portanto, o objetivo do Sistema Inteligente é auxiliar o processo de tomada de decisão de uma organização com tarefas relacionadas aos fluxos de trabalho, buscando uma padronização, utilizando as bases de conhecimento.

2.1 Agente

Na comunidade científica, existem inúmeras definições sobre agentes, que são:

Segundo Maes (1994), agentes são sistemas computacionais que habitam um ambiente, realizando metas para alcançar objetivos para os quais foram projetados.

De acordo com Wooldridge (2002), agente é um programa de computador situado em um ambiente que é capaz de agir autonomamente em seu ambiente a fim de encontrar seus objetivos.

A FIPA (2007) define agente como uma entidade que reside em um ambiente onde interpreta dados através de sensores que refletem eventos no ambiente e executam ações que produzem efeito no ambiente.

Segundo Russell (2004), o agente é uma entidade autônoma que percebe seu ambiente através de sensores e age sobre o mesmo por intermédio dos executores.

A Figura 1 mostra as interações de um agente genérico com seu ambiente.

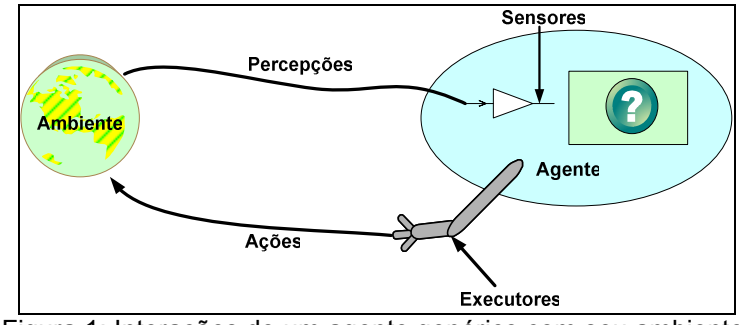

Figura 1: Interações de um agente genérico com seu ambiente.
Fonte: Russell *et al*, 2004, p. 33.

Um agente possui uma estrutura básica bem simples, contendo uma memória interna que irá atualizar-se com a chegada de novas percepções, onde a memória é utilizada nos procedimentos de tomada de decisão, gerando ações para serem executadas.

De acordo com Wooldridge (2002), o agente computacional deve ter algumas características como:

- Autonomia: permite ao agente desempenhar funções de forma autônoma, sem intervenção humana ou de outros agentes para realizar ações por iniciativa própria de acordo com o conhecimento que possui do ambiente e sua capacidade de raciocinar.

- Proatividade: o agente simplesmente não age em resposta a seu ambiente, ele possui habilidade para tomar iniciativa que é direcionada para alcançar seus objetivos.

- Reatividade: permite ao agente perceber seu ambiente e responder de forma oportuna às mudanças dele recebida.

- Sociabilidade: envolve o estabelecimento de canais de comunicação entre os agentes através de protocolos de interação. Assim, eles podem cooperar, competir, negociar etc.

2.2 Sistema Multiagente

Em um Sistema Multiagente podem ser utilizados os recursos de cooperação, negociação e interação entre os agentes e o usuário.

De acordo com Wooldridge (2002), o Sistema Multiagente tem como característica principal a comunicação entre os agentes em diferentes ambientes.

A Figura 2 mostra uma estrutura de um Sistema Multiagente.

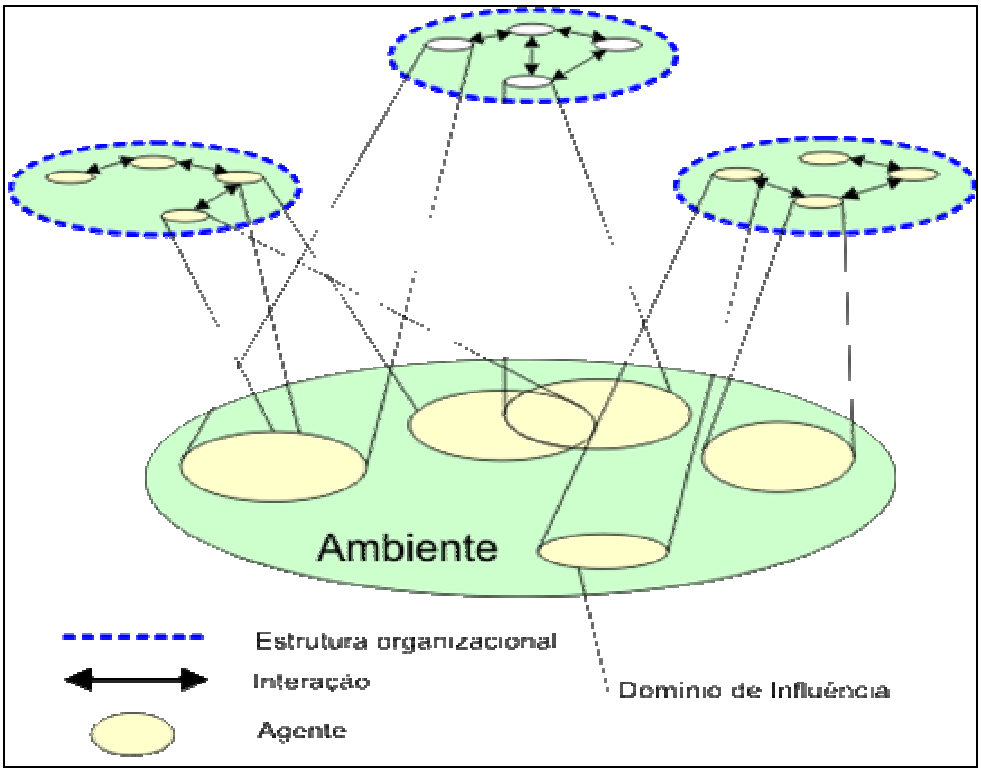

Figura 2 Estrutura de um Sistema Multiagente.
Fonte: Wooldridge, 2002, p. 106.

Na Figura 2, observa-se que os agentes estão se comunicando, negociando e interagindo uns com os outros em vários ambientes e subsistemas, a fim de executar determinada tarefa. O agente é capaz de agir em um ambiente onde vários agentes têm diferentes "domínios de influência", no sentido em que terá o controle efetivo e deverá ser capaz de influenciar outros agentes no SMA.

A estrutura de um Sistema Multiagente é uma especificação formal de um modelo abstrato que contextualiza o mundo real, possibilitando padronização, onde os termos escolhidos sejam suficientes para especificar e definir conceitos, e permitir relacionamentos adequados a partir da escolha terminológica realizada. Esta expressão inclui a representação do domínio específico do conhecimento.

Segundo Weiss (1999), o ambiente de um Sistema Multiagente deverá ter as seguintes características:

– Especificar uma infra-estrutura composta de protocolos de comunicação e interação.

– Apresentar uma estrutura aberta.

– Apresentar agentes que são autônomos e distribuídos, e podem ser competitivos ou cooperativos.

A Tabela 1 apresenta as características de um ambiente Multiagente.

PROPRIEDADES	VALORES
Autonomia de Projeto	Plataforma/ Protocolo de Interação / Linguagem / Arquitetura Interna.
Infra-estrutura de comunicação	Memória compartilhada (*blackboard*) ou baseada em mensagem, orientada à conexão ou não orientada à conexão (e-mail), ponto-a-ponto, *multicast* ou *broadcast*, síncrono ou assíncrono.
Diretório de Serviços	Página Branca, Página Amarela.
Protocolo de mensagem	KQML, FIPA-ACL, HTTP, HTML, OLE, CORBA.
Serviços Mediação	Baseado em Ontologia ou Transações.
Serviços Segurança	Autenticação.
Operações de Suporte	Armazenamento/Redundância/Restauração/ Negociação.

Tabela 1: Características de um ambiente Multiagente.
Fonte: Weiss, 1999, p. 82.

2.3 Metodologia para o Desenvolvimento de Agente Inteligente

A metodologia utilizada no processo de desenvolvimento de softwares baseados em agentes inteligentes necessita de um conjunto de técnicas e ferramentas apropriadas para a sua especificação.

Para Jennings (2000), as metodologias utilizadas no desenvolvimento de agentes inteligentes apresentam e direcionam o processo para abordar a construção de aplicações distribuídas, inteligentes e robustas.

Atualmente, existem diversas metodologias para o desenvolvimento de agentes inteligentes, que são:

a) *GAIA* – metodologia desenvolvida por Wooldridge (2000) cujo objetivo é a especificação de sistemas baseados em agentes como se fosse uma estrutura organizacional. A GAIA está dividida em duas fases, que são:

- A fase de análise que está subdivida em modelagem de interação e modelagem de papéis. O objetivo desta fase é compreender o domínio do sistema e estruturar os diversos papéis que serão desempenhados na organização, através do modelo de papéis, e definir como os mesmos interagem, através do modelo de interação;
- A fase de projeto que está subdivida em modelagem de agentes, modelagem de serviços e modelagem de comunicação. O objetivo desta fase é identificar os papéis, interações e os serviços associados a cada classe de agente.

A Figura 3 mostra a metodologia Gaia e suas relações no processo.

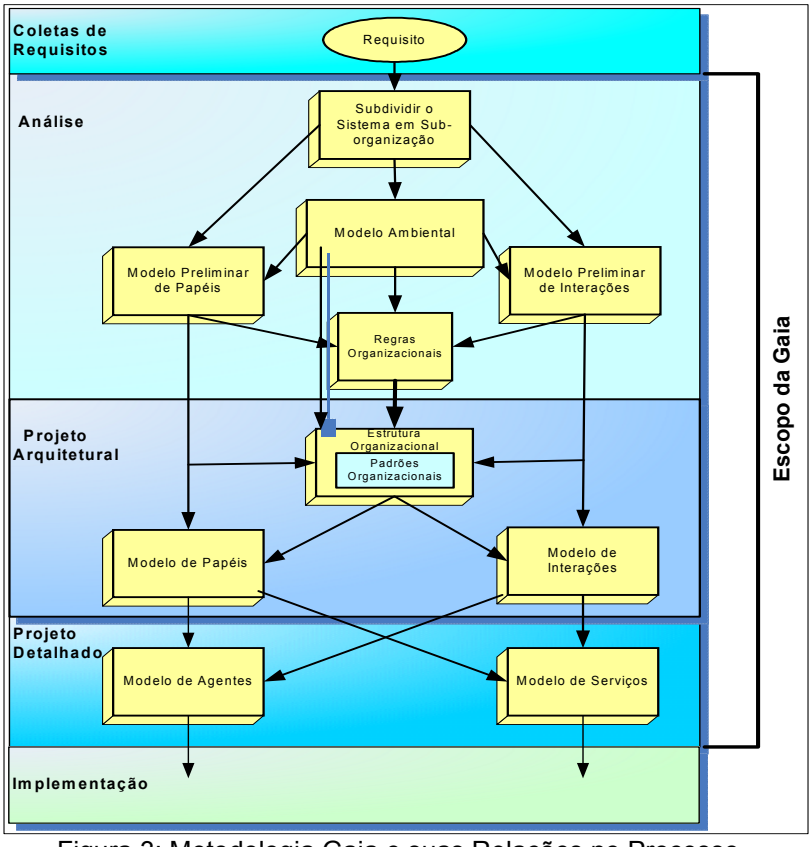

Figura 3: Metodologia Gaia e suas Relações no Processo.
Fonte: Zambonelli *et al*, 2003.

b) *MASE* (*Multiagent Systems Engineering*) – segundo Deloach (2001), esta metodologia tem como objetivo o desenvolvimento de agentes no SMA, pois possibilita a especificação inicial do sistema para produzir os seus artefatos. A MASE tem como objetivo orientar o projetista no desenvolvimento do ciclo de vida do SMA, independente da arquitetura que deverá ser selecionada. Esta metodologia é composta por duas fases, que são: análise e projeto.

- Na fase de análise são identificados os objetivos do sistema, os papéis e suas responsabilidades. Esta fase contém três subfases, que são:

a) Captura de objetivos – consiste em identificar as especificações iniciais do sistema, transformando os objetivos em conjunto estruturados de objetivos em um diagrama de hierarquia de objetivos;

b) Aplicação de caso de uso – captura os objetivos transformados em papéis e tarefas associadas, onde o analista elabora os casos de usos pelos requisitos do sistema e dos usuários e estes são demonstrados graficamente em um diagrama de seqüência. O diagrama de seqüência é composto por seqüências narrativas dos eventos que definem o comportamento desejado para o sistema;

c) Refinamento dos papéis – esta etapa define os papéis necessários que foram identificados durante as etapas anteriores. Os papéis são descrições abstratas previstas em uma entidade que encapsula os objetivos e as responsabilidades atribuídas ao sistema.

- Na fase de projeto são definidos os agentes, protocolo de comunicação, arquitetura e desenvolvimento do SMA. Esta fase contém quatro subfases, que são:

a) Criação das classes de agentes – os agentes são identificados através dos papéis que são visualizados por meio do diagrama de classes dos agentes;

b) Construção das conversões – são definidos os protocolos de comunicação entre os agentes, utilizando o diagrama de estados, assim como relacionamento das trocas de mensagens utilizadas em cada ato de comunicação;

c) Organização dos agentes – nesta subfase, é definida a arquitetura dos agentes e componentes do sistema;

d) Desenvolvimento do sistema – nesta fase, é utilizado um *framework* para implementar o sistema.

A Figura 4 mostra a metodologia *MASE*.

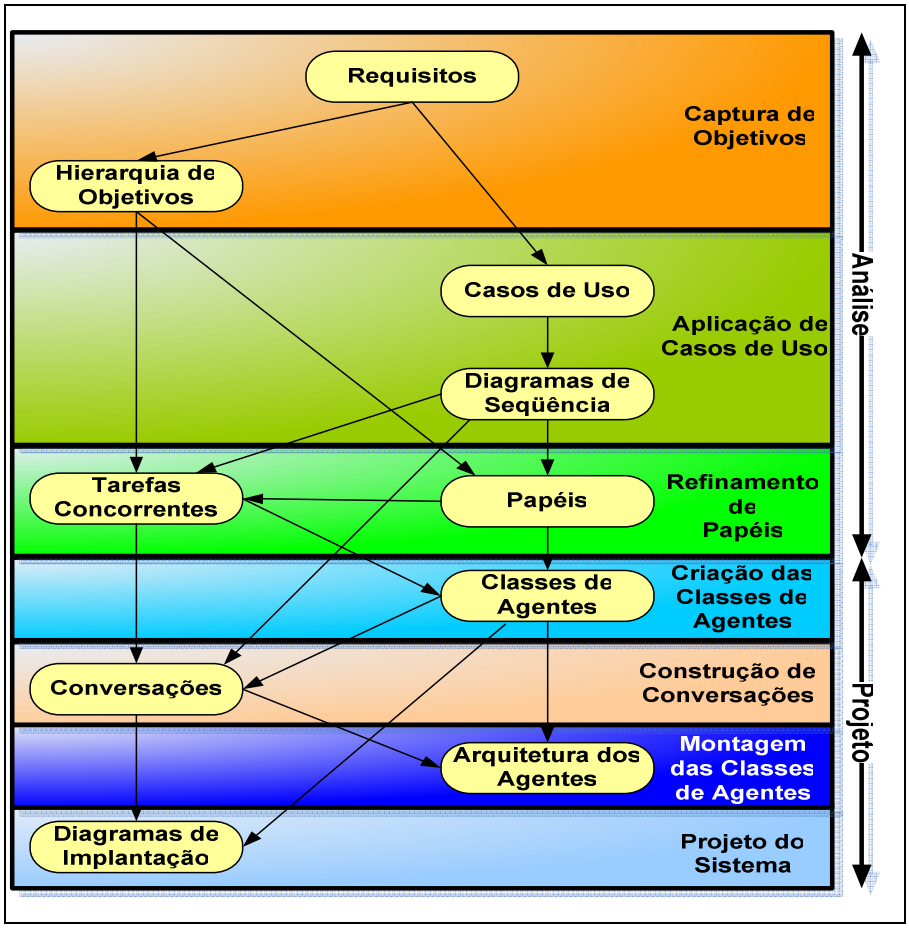

Figura 4: Metodologia MaSE.
Fonte: DeLoach (2001).

c) *PROMETHEUS*: Segundo Padgham *et al* (2004), esta metodologia está subdividida em três fases, que são:

– Especificação do sistema – esta fase especifica o ambiente e as funcionalidades básicas do sistema, juntamente com as entradas, percepções e o conjunto de ações provenientes do sistema ou dos dados externos. Nesta fase, são identificadas as funcionalidades básicas para alcançar os objetivos que serão desenvolvidos pelos agentes;

– Projeto arquitetural – esta fase utiliza as ações, percepções e funcionalidades identificadas na fase anterior para determinar as responsabilidades dos agentes no sistema e como eles irão colaborar;

– Projeto detalhado – esta fase analisa as estruturas internas de cada agente, assim como a responsabilidade das tarefas que serão desenvolvidas no sistema por cada agente.

A Figura 5 mostra as fases da metodologia *Prometheus*.

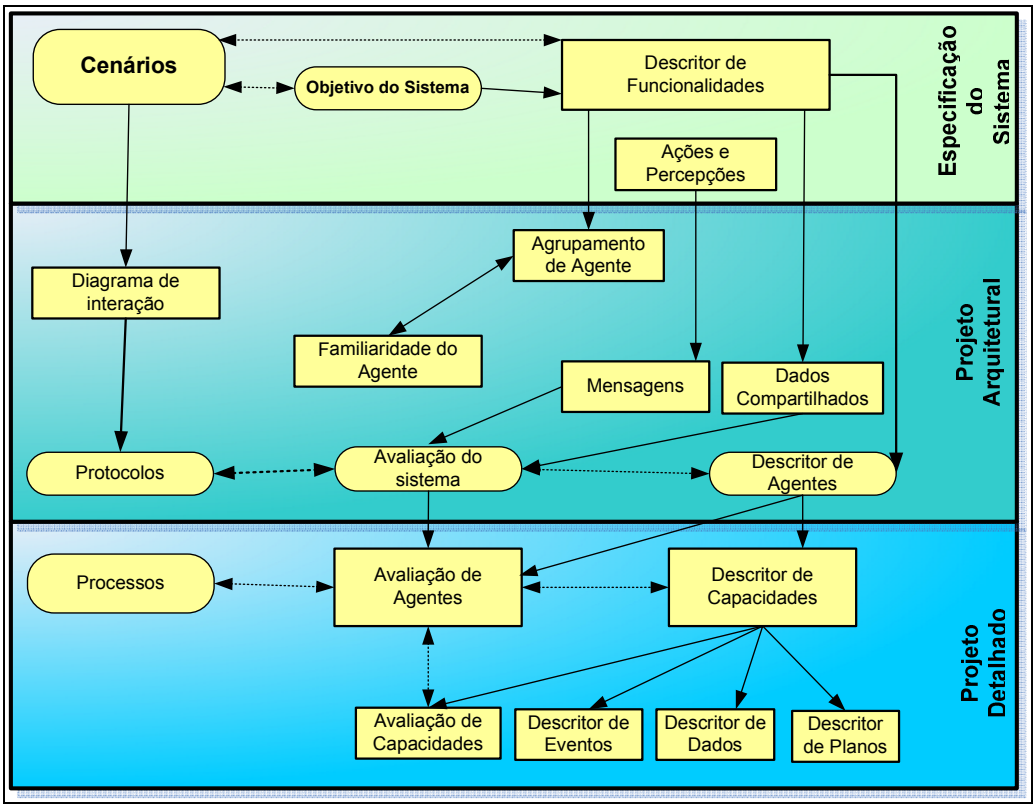

Figura 5: Fases da metodologia Prometheus.
Fonte: Padgham *et al* (2004).

As ferramentas utilizadas no desenvolvimento da metodologia Prometheus são: *JACK* e PDT (Prometheus Design Tools).

d) *MESSAGE* (*Methodology for Engineering Systems of Software Agents)*: Segundo Caire *et al*.(2001), esta metodologia é dividida em duas fases que são:

– Análise – que tem como objetivo produzir a especificação do sistema a ser desenvolvido;

– Projeto – define a solução do problema, através da identificação dos componentes computacionais do sistema, além da especificação sobre o que eles devem realizar e estabelece as interfaces entre esses componentes.

A Figura 6 mostra a arquitetura da metodologia *Message*.

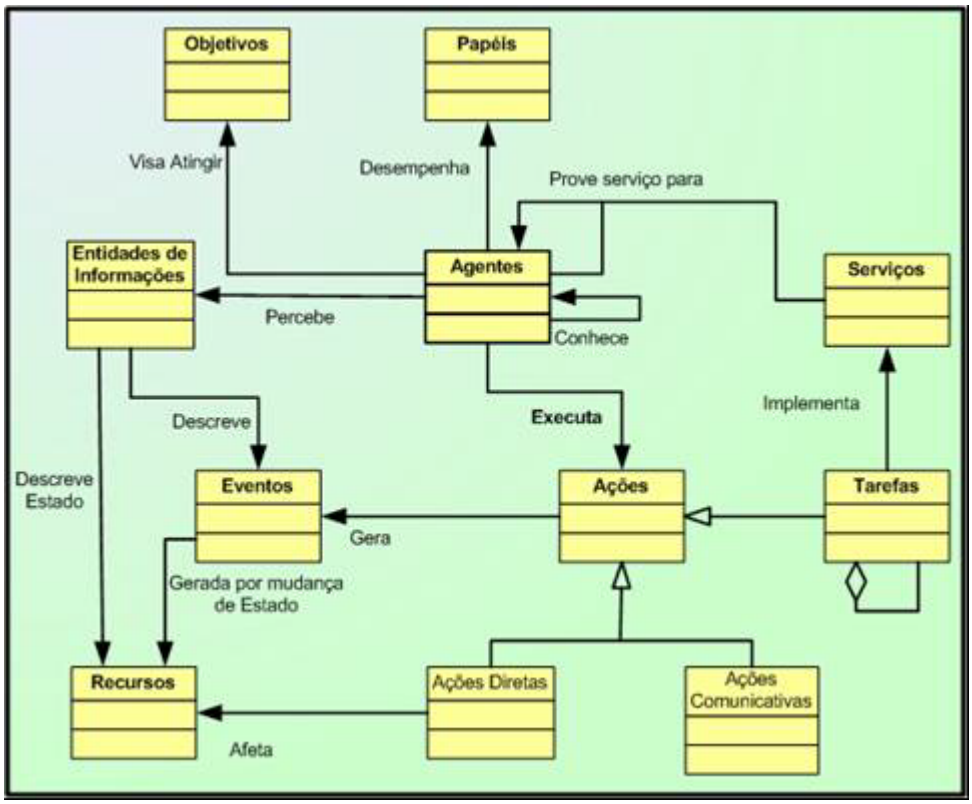

Figura 6: Arquitetura da Metodologia Message.
Fonte: Caire *et al* (2001).

A *Message* é uma metodologia genérica que pode ser aplicada a diversos domínios de conhecimento. Ela estende a UML (*Unified Modeling Language*) para incluir conceitos orientados a agentes no nível do conhecimento. Assim, os diagramas de classes e de atividades adquirem uma notação que permite representar semanticamente tais conceitos.

e) *PASSI* (*Process for Agent Societies Specification and Implementation*). Segundo Cossentino *et al* (2002), esta metodologia foi projetada para a construção de um sistema multiagente e facilita o desenvolvimento do projetista, além de aumentar e possibilitar o reuso de código.

A Figura 7 mostra os modelos e fases da metodologia PASSI.

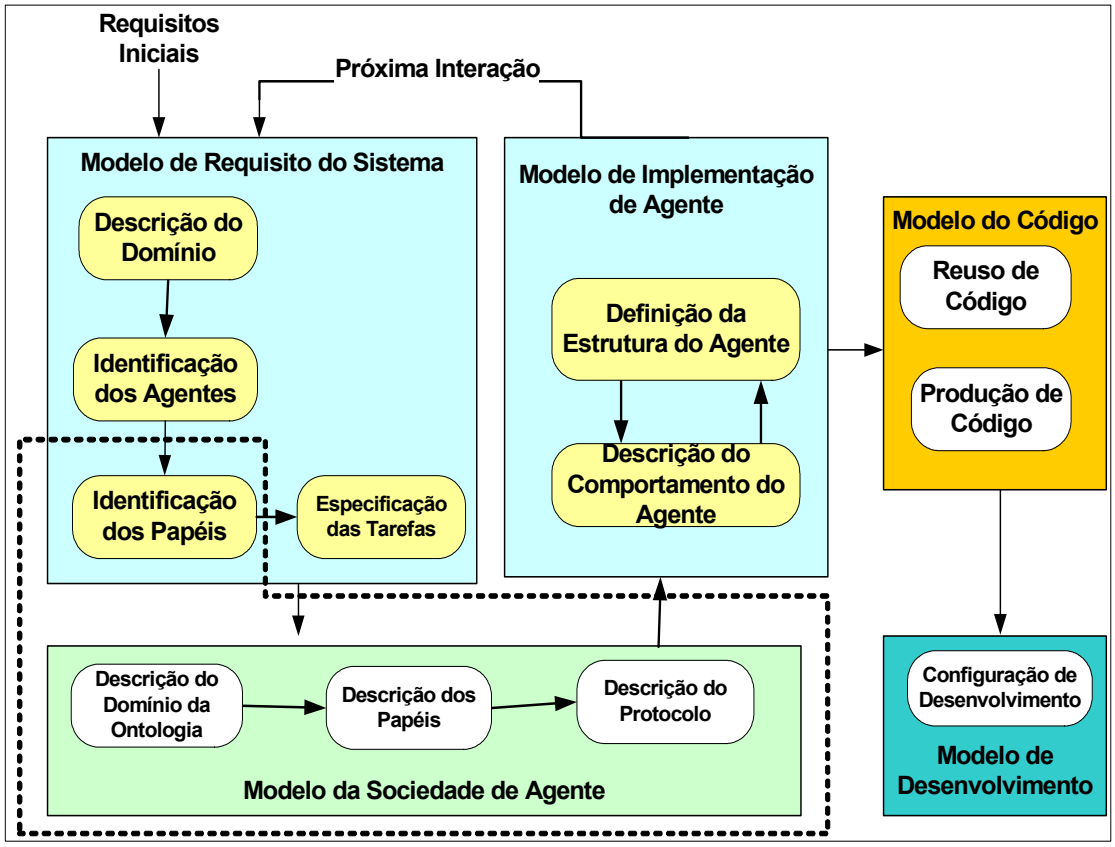

Figura 7: Modelos e fases da metodologia de PASSI.
Fonte: Burrafato, P. *et al* (2002).

A metodologia PASSI é composta de cinco modelos, que são:

– Modelo do Requisito do Sistema – este modelo é composto de quatros fases, que são:

a) Descrição do Domínio – esta fase descreve a funcionalidade do sistema, utilizando o diagrama de caso de uso;

b) Identificação dos Agentes – nesta fase, são atribuídas as responsabilidades para os agentes, representados como pacotes UML estereotipados;

c) Identificação dos Papéis – esta fase identifica os cenários de cada agente através das especificações de papéis e é representada através do diagrama de seqüência;

d) Especificação das tarefas – nesta fase, especificam-se as capacidades de cada agente através do diagrama de atividade.

– Modelo da Sociedade de Agentes – é um modelo que representa as interações e dependências sociais entre os agentes envolvidos na solução. Este modelo é composto de três fases:

a) Descrição do Domínio da Ontologia – esta fase utiliza o diagrama de classe para descrever o conhecimento relacionado aos agentes e suas interações;

b) Descrição dos Papéis – descreve os papéis representados pelos agentes, as tarefas envolvidas, capacidades de comunicação e dependências entre os agentes;

c) Descrição de Protocolo – esta fase utiliza o diagrama de seqüência para especificar a semântica de cada protocolo de comunicação em atos performativos.

– Modelo de Desenvolvimento do Agente – é o modelo clássico da arquitetura de solução em termos de classes e métodos, onde a diferença relevante consiste na abordagem orientada ao agente, para o qual existem dois níveis de abstrações, que são:

a) Definição de Estrutura do Agente – esta fase define os diagramas de classes convencionais e a estrutura da solução das classes dos agentes;

b) Descrição do Comportamento do Agente – esta fase apresenta o diagrama de atividade, descrevendo o comportamento dos agentes individualmente.

– Modelo de Código – este modelo apresenta a solução no âmbito do código. A geração de código, neste modelo, é composta das seguintes fases, que são:

a) Reuso de código – representa uma biblioteca de classe e diagramas associados que correspondem ao código reutilizável;

b) Produção de código – representa o código fonte do sistema.

– Modelo de Desenvolvimento – este modelo apresenta a distribuição das partes do sistema através das unidades de processamento do hardware, onde tal modelo envolve a fase de configuração da implementação. Geralmente, a configuração da implementação utiliza o desenvolvimento dos diagramas e descreve a distribuição dos agentes nas unidades de processos ativos, aplicando as restrições na mobilidade e relacionamentos dos respectivos agentes.

A metodologia *PASSI* utiliza o *PTK* (*PASSI Toolkit*) que é composto por uma ferramenta para reuso de padrões de agentes, chamada de *AgentFactory*.

2.4 Formas de Comunicação no Ambiente Multiagente

A comunicação é essencial em um SMA, onde vários agentes precisam trocar informações com os usuários, com recursos de sistema ou com outro agente. Para tanto, faz-se necessária a utilização de uma linguagem de comunicação de agente.

Para padronizar a forma de comunicação, é necessário compartilhar um conjunto de vocabulários que é composto por palavras e significados. Esta padronização é denominada de ontologia.

Segundo *Swartout et al* (1999), a ontologia representa um conjunto de conceitos e termos que podem ser usados para descrever alguma área do conhecimento ou construir uma representação para o conhecimento.

Existem duas formas de comunicação entre agentes em um sistema multiagente:

– Comunicação direta – cada agente comunica-se com qualquer outro agente sem qualquer intermediário ou federado, onde a troca de mensagem é realizada através de mediadores ou facilitadores.

– Comunicação indireta – um agente, para se comunicar, utiliza uma estrutura compartilhada, ou seja, um quadro negro. Nesse contexto, um agente não

é capaz de estabelecer comunicação com outro agente de maneira direta, porque o agente não conhece o endereço do destinatário.

2.5 Arquitetura de Agente

A arquitetura de agente é um mecanismo fundamental, pois reflete como um determinado agente se comportará em um sistema (ambiente) real e dinâmico.

O funcionamento interno de um agente depende de sua arquitetura. Atualmente, existem diversas arquiteturas de agentes, divididas em três grupos principais: deliberativa, reativa e híbrida.

2.5.1 Arquitetura Deliberativa ou Cognitiva

A arquitetura deliberativa tem como principal característica a representação simbólica de seu ambiente para permitir a tomada de decisões através de raciocínio lógico. São exemplos dessas arquiteturas: *PRS (GEORGEFF, M.. LANSKY*, A. 1987), *IRMA (BRATMAN, MICHAEL et al*., 1987), *GRATI (JENNINGS, N*., 1993), *METATEM (FISHER, MICHAEL*, 1994), etc.

A Figura 8 mostra o modelo da arquitetura PRS.

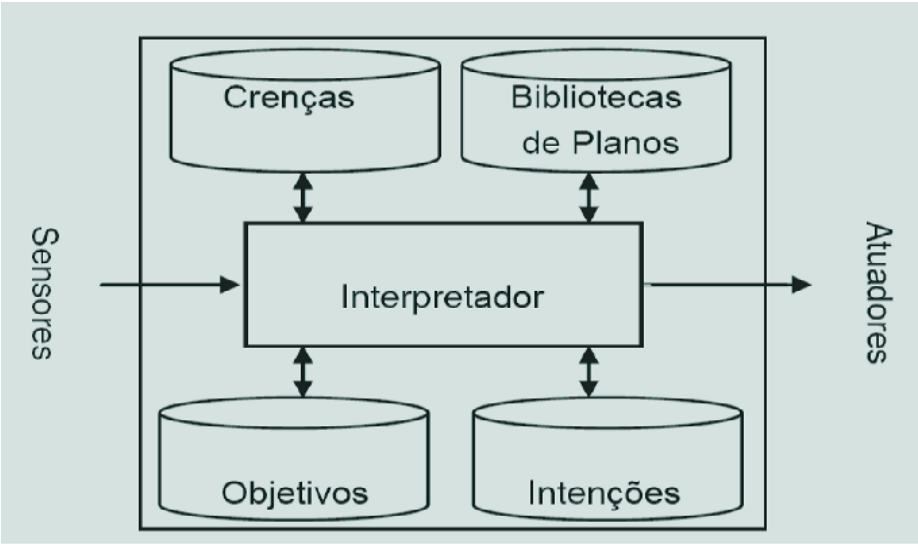

Figura 8: Modelo da arquitetura PRS.
Fonte: Bellifemine *et al* (2007, p. 5).

A arquitetura deliberativa é utilizada pelo agente baseado em lógica. Esse tipo de agente permite a representação simbólica de seu ambiente e seu comportamento, possibilitando a manipulação através de dedução lógica.

2.5.2 Arquitetura Reativa

A principal característica dessa arquitetura é que ela não representa o modelo do mundo lógico/simbólico, mas somente das reações/ações que ocorrem no ambiente.

Essa arquitetura é baseada em mecanismos de estímulos/respostas que são ativados pelas percepções do sensor. Diferentemente das arquiteturas baseadas em lógica, esta arquitetura não possui representações simbólicas do ambiente. As principais idéias de *Brooks*, percebidas nessa arquitetura, apontam que um comportamento inteligente pode ser gerado sem representação explícita do ambiente e raciocínio abstrato.

A Figura 9 mostra o modelo da arquitetura reativa *SUBSUMPTION*.

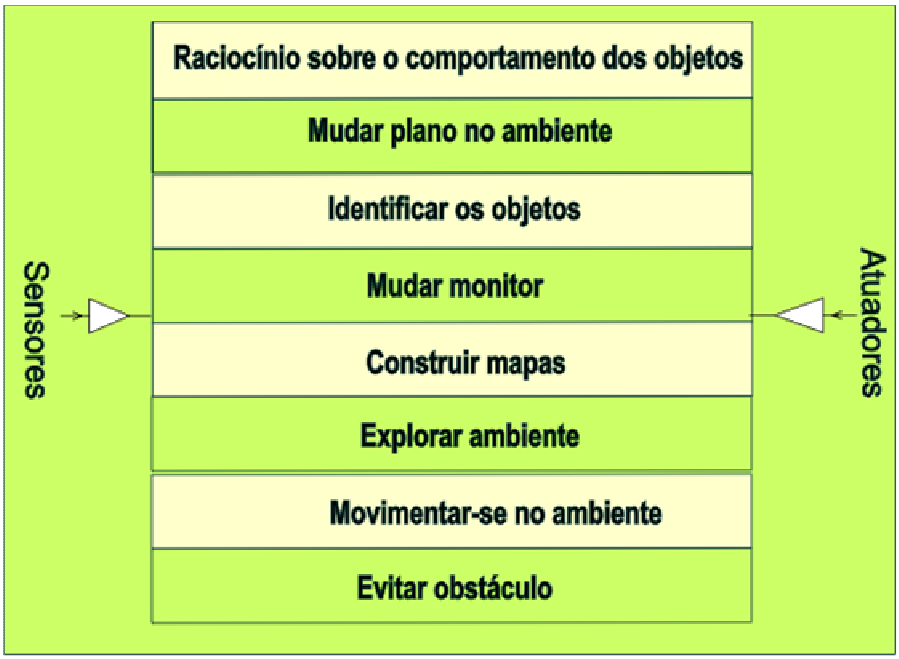

Figura 9: Modelo da arquitetura reativa SUBSUMPTION.
Fonte: Bellifemine *et al* (2007, p. 4).

As referidas camadas formam uma hierarquia de comportamentos em que os níveis mais baixos possuem menor controle que os níveis mais altos da pilha. Deste modo, a tomada de decisão é alcançada através de comportamentos meta-dirigidos.

A arquitetura reativa é utilizada pelos agentes reativos, pois esses agentes possuem comportamento inteligente, baseado em ações, gerando reações às mudanças que ocorrem no ambiente.

2.5.3 Arquitetura Híbrida

A arquitetura híbrida é composta pela combinação dos componentes das arquiteturas cognitiva e reativa.

Este modelo de arquitetura é utilizado pelos agentes baseados em lógica (cognitivos) e agentes baseados em reação (reativos), onde o primeiro agente contém o modelo lógico/simbólico e outro agente é capaz de reagir às mudanças que acontecem no ambiente.

A Figura 10 apresenta o controle e o fluxo de dados na arquitetura híbrida em Camadas.

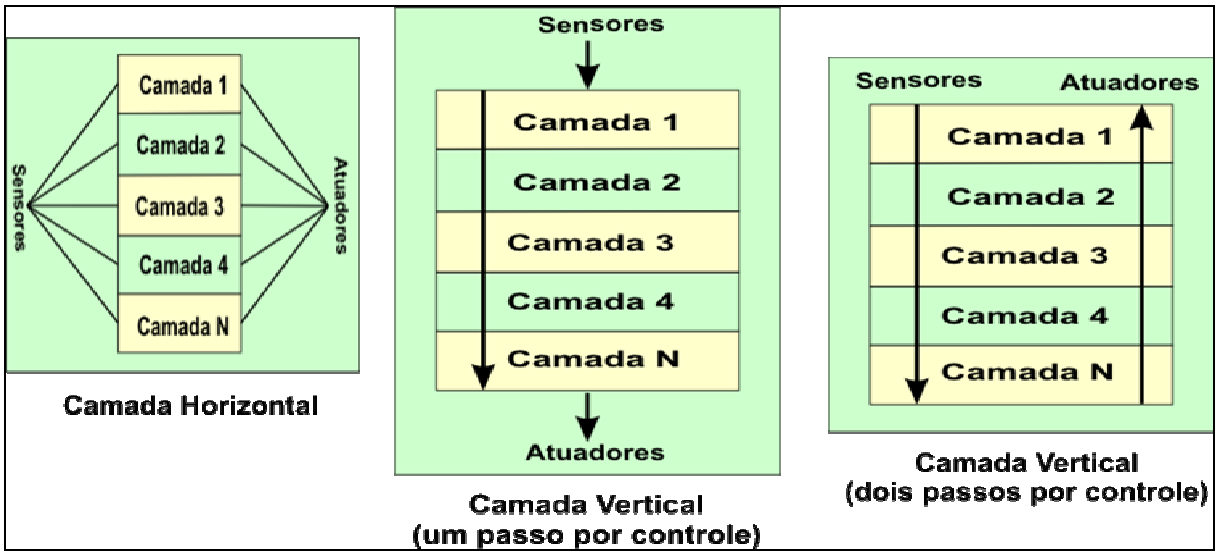

Figura 10: Controle e fluxo de dados na arquitetura híbrida em camadas.
Fonte: Wooldridge, 2002, p. 98.

Nessa arquitetura podemos identificar dois tipos de fluxo de controle, que são:

• Horizontal – cada camada de software está diretamente conectada ao sensor de entrada e a ação de saída.

• Vertical – os sensores de entrada estão ligados apenas a uma das camadas, enquanto as outras camadas são responsáveis pela execução da ação. A arquitetura vertical está subdividida em um passo por controle e dois passos de controle. Ou seja, na arquitetura vertical de um passo de controle, a informação da camada é realizada até a última camada que vai executar a ação. Enquanto que, na arquitetura vertical de dois passos de controle, a informação da camada é realizada até a última camada do primeiro fluxo de controle e retorna da camada inferior até a primeira camada que vai executar a ação.

2.6 Protocolo de Comunicação entre Agentes

O processo de comunicação entre agentes utiliza uma linguagem que descreve os aspectos sintáticos e semânticos das trocas de mensagens entre os agentes. Existem, atualmente, duas linguagens de comunicação baseadas em trocas de mensagens, que são: a KQML (*Knoweledge Query and Manipulation Language*) e a FIPA-ACL (*Foundation for Intelligent Physical Agents-Agent Communication Language*).

2.6.1 KQML

A linguagem KQML utiliza protocolos de comunicação de alto nível e foi desenvolvida no início da década de 90, pelo grupo *Knowledge Sharing Effort*, patrocinado pelo DARPA (*Defense Advanced Research Projects Agency*), com o objetivo de desenvolver sistemas baseados em conhecimentos para compartilhamento em tempo real.

O propósito da KQML é compartilhar informações e conhecimentos entre os agentes. Caracteriza-se por utilizar uma padronização nas mensagens que serão utilizadas. A KQML é estruturada em três camadas, que são:

– Camada de conteúdo: refere-se à mensagem, onde a sintaxe representa qualquer conteúdo da linguagem KQML.

– Camada de comunicação: nesta camada, os parâmetros de alto nível são codificados (identidade do agente emissor e receptor, identificação da mensagem).

– Camada da mensagem: esta camada identifica o protocolo usado para transmitir a mensagem e a performativa que o agente transmissor anexa ao conteúdo.

A linguagem KQML disponibiliza tipos de mensagens denominadas performativas (atos comunicativos), que representam os objetivos do agente ao enviar, consultar, trocar, requisitar, sincronizar e gerenciar uma mensagem. As mensagens performativas são mensagens pré-definidas que decidem as operações que podem ser executadas entre os agentes.

A Tabela 2 mostra as performativas reservadas e os seus identificadores.

PERFORMATIVAS	IDENTIFICADORES
Consultas Básicas	evaluate, reply, ask-if, ask-one, ask-about, ask-all
Troca de informações	tell, cancel, deny, untell, insert, uninsert, delete-one, delete-all, undelete
Requisição de Serviços	subscribe, achieve, unachieve, broker-one, broker-all
Sincronismo do envio de mensagens	error, sorry, standby, ready, next, forward, broadcast, rest, discard, transport-address
Gerenciamento de agentes	register, unregister, advertive, unaundvertive, recomended-all, recruit-one, ecruit-all

Tabela 2: Performativas reservadas e os seus identificadores.
Fonte: Finin *et al,* 1995.

A linguagem KQML apresenta também um conjunto de palavras reservadas que são úteis para estabelecer graus de uniformidade dos parâmetros e

suporte para que os agentes forneçam apoio para outras performativas que não estão na linguagem.

A Tabela 3 mostra palavras reservadas da KQML.

PALAVRAS RESERVADAS	SIGNIFICADOS
:content	A informação sobre qual performativa expressa uma atitude.
:in-reply-to	A identificação que está esperando uma resposta.
:language	O nome da linguagem contida no parâmetro :content.
:ontology	O nome da ontologia usado no parâmetro :content.
:receiver	O agente receptor atual da performativa.
:reply-with	O agente transmissor espera por uma resposta, caso afirmativa será enviada uma identificação na resposta.
:sender	O agente transmissor da performativa no momento.

Tabela 3: Palavras reservadas da KQML.
Fonte: Finin *et al*, 1995

O formato de uma mensagem KQML contém parâmetros de transporte com os respectivos endereços do agente transmissor e receptor envolvidos na comunicação. Estes parâmetros podem ser combinados utilizando um serviço de nomes, endereçamento e registros para realizar a conversão em endereços compatíveis com a camada de comunicação.

2.6.2 FIPA-ACL

A FIPA-ACL é uma linguagem de comunicação de agentes desenvolvida de acordo com as especificações da FIPA. A sua sintaxe é semelhante ao KQML, mas o conjunto de performativas (atos comunicativos) é diferente.

A sintaxe da linguagem de comunicação FIPA-ACL contém tipos de mensagens e descrições das mensagens sobre os agentes transmissores e receptores. Os parâmetros inerentes da FIPA-ACL são divididos em categorias, que são:

- Ato comunicativo: refere-se ao tipo de ato comunicativo da mensagem.

– Participantes da comunicação: refere-se à transmissão, recepção e identificação das mensagens que deverão ser enviadas para os agentes.

– Conteúdo de mensagem: indica qual o conteúdo da mensagem deve ser transportado.

– Descrição do conteúdo: indica a semântica e a codificação da linguagem utilizada.

– Controle da conversão: representa o protocolo de comunicação entre o agente e a mensagem.

A Tabela 4 mostra as categorias com os parâmetros das mensagens FIPA-ACL.

CATEGORIA	PARÂMETRO
Tipo de ato comunicativo	performative
Participantes da comunicação	sender, receiver, reply-to
Conteúdo da Mensagem	Content
Descrição do Conteúdo	language, encoding, ontology
Controle de Comunicação	protocol, conversation-id, reply-With, in-reply-to, reply-by

Tabela 4: Categorias com os parâmetros das mensagens FIPA-ACL.
Fonte: Bellifemine *et al* (2007, p. 18).

O conteúdo de uma mensagem, que é indicado pelo parâmetro *content*, referencia a informação sobre qual ato comunicativo será aplicado. Observa-se, com freqüência, que o conteúdo pode ser expresso em qualquer linguagem declarada no parâmetro *language*. Cada mensagem FIPA-ACL transporta um ato comunicativo, que representa a vontade de um agente sobre determinada informação transportada na mensagem. O ato comunicativo estabelecido pelo padrão FIPA pode ser: *Accept Proposal, Agree, Cancel, Confirm*, etc.

A FIPA possui um modelo de referência da plataforma de agente padronizado. Esse padrão possui um conjunto de serviços que representam as regras normativas para uma sociedade de agente existir, funcionar e ser gerenciada.

2.7 Plataforma de Agente JADE

O *Java Agent Development Framework* (JADE) é uma plataforma para desenvolvimento de sistemas multiagentes e foi desenvolvido pelo CSELT (Centro de Estudo e Laboratório de Telecomunicação). Atualmente, é conhecido como TILAB (*Telecomm Italia Lab*), juntamente com o Grupo de Engenharia de Computação da Universidade de Parma. Esta plataforma está de acordo com as especificações FIPA.

O JADE é atualmente utilizado pelos pesquisadores e as justificativas para sua expansão são, principalmente:

– JADE é um software "*open source*" sob os termos da licença LGPL (*Lesser General Public License*).

– Facilidade no desenvolvimento de sistemas multiagentes através de um conjunto de ferramentas gráficas que suporta a depuração e a fase de desenvolvimento.

– O *middleware* da plataforma JADE possui bibliotecas de alto nível e apresenta facilidade para desenvolver e fornecer serviços genéricos que são úteis, não apenas por aplicação simples, mas, preferencialmente, para uma variedade de aplicações.

– A plataforma JADE inclui um sistema gerenciador de agentes (AMS – *Agent Management System*), um diretório facilitador (DF – *Directory Facilitator*) e um canal de comunicação de agentes (ACC – *Agent Communication Channel*) de acordo com a especificação FIPA. Estes três componentes são automaticamente iniciados quando a plataforma é instanciada.

– A plataforma JADE inclui as bibliotecas de classes "jade.core, jade.content, jade.domain, jade.gui etc. " que os programadores reutilizam (herança, agregação ou composição) no desenvolvimento de agentes. Além disso, a plataforma JADE fornece os serviços básicos necessários para a execução de aplicações ponto-a-ponto e permite que cada agente descubra dinamicamente outros agentes para realizar o processo de comunicação.

A simplicidade no desenvolvimento de agentes fornece serviços relacionados com o ciclo de vida, monitoramento e execução de serviços para garantir um padrão de interoperabilidade entre os diversos sistemas multiagentes.

2.7.1 Modelo Padrão da Plataforma JADE

O modelo padrão da plataforma JADE utiliza o modelo de especificação da FIPA.

A Figura 11 mostra o modelo padrão da plataforma JADE.

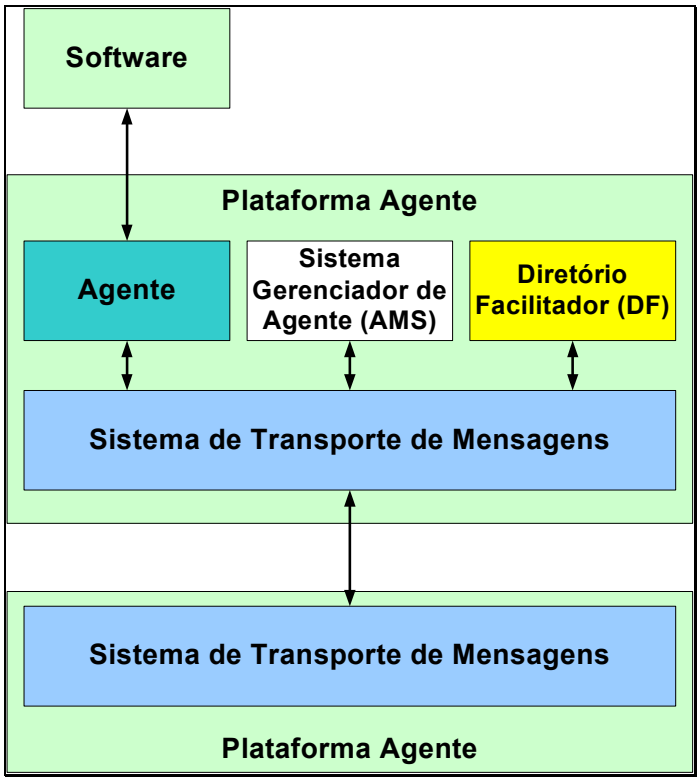

Figura 11: Modelo padrão da plataforma JADE.
Fonte: FIPA, 2007.

Neste modelo, a FIPA identificou os agentes necessários e os seus papéis para a administração da plataforma, que são:

– Agente: conjunto de atividades onde são definidos os papéis e os objetivos do sistema. É o agente que realiza a comunicação com os outros agentes, utilizando o conceito de troca de mensagens.

– Sistema gerenciador de agentes (AMS): este sistema gerencia o acesso e o uso da plataforma de agentes e fornece serviços de páginas brancas e controle do ciclo de vida, mantendo um diretório de identificadores de agentes (*Agent Identifier* – AID) com os respectivos estados dos agentes. Portanto, o AMS é responsável pela autenticação dos agentes e pelo controle dos seus registros. Todo agente necessita se registrar no AMS para obter um AID válido.

– Diretório Facilitador (*Directory Facilitador* – DF): provê serviços de páginas amarelas (*Yellow Pages* – YP) na plataforma JADE.

– Sistema de transporte de mensagens: é o sistema que possui canais que facilitam a comunicação entre os agentes (*Agent Communication Channel* – ACC).

A infraestrutura necessária para a execução da plataforma JADE geralmente pode ser implementada em um sistema distribuído. Segundo Tanenbaum *et al* (2000), um sistema distribuído é um conjunto de computadores independentes que se apresenta ao usuário como um sistema único. Para Coulouris *et al* (2007), um sistema distribuído é um conjunto de computadores autônomos interligados através de uma rede de computadores e equipados com software que permita o compartilhamento dos recursos do sistema.

A Figura 12 mostra um modelo de sistema distribuído.

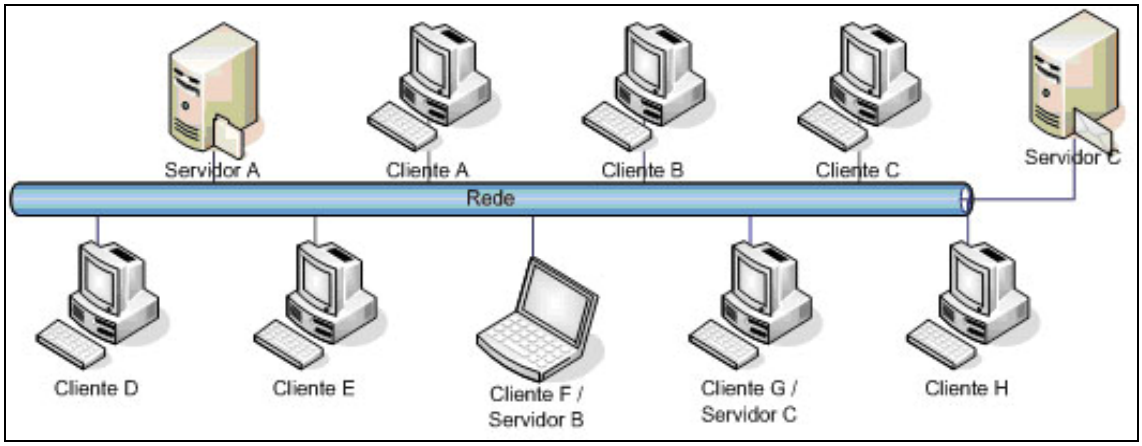

Figura 12: Modelo de sistema distribuído.

Assim, a computação distribuída consiste em adicionar o poder computacional de diversos computadores interligados por uma rede de computadores ou mais de um processador, trabalhando em conjunto no mesmo computador, para processar colaborativamente determinada tarefa de forma coerente e transparente, como se apenas um único e centralizado computador estivesse executando a tarefa. A união desses diversos computadores com o objetivo de compartilhar a execução de tarefas é conhecida como sistema distribuído.

No projeto de um sistema distribuído, algumas características desejáveis deverão ser implementadas, tais como:

a) Transparência: é uma característica dos sistemas distribuídos que permite esconder das aplicações o uso de sistemas, abstraindo do desenvolvedor a complexidade das funcionalidades. Existem características de transparências de localização, migração, replicação, concorrência e paralelismo.

– Transparência de localização: os recursos são acessados sem que sua localização seja determinada.

– Transparência de migração: os recursos podem ser movidos de lugar sem que seja necessário mudar seus nomes.

– Transparência de replicação: podem existir múltiplas cópias de um recurso para aumentar a performance e disponibilidade dos seus serviços, sem o conhecimento das réplicas por usuários e programadores.

– Transparência de concorrência: os processos executam concorrentemente, utilizando recursos compartilhados, sem interferirem na execução dos outros.

– Transparência a falhas: é possível ocultar e tratar as falhas de hardware ou software, permitindo que as aplicações ou usuários completem suas tarefas sem interrupção.

b) Flexibilidade: os sistemas distribuídos devem ser desenvolvidos de forma flexível para que se possa configurá-los novamente ao longo do tempo, caso algo aconteça de errado. Os requisitos mudam e nada no mundo é estático;

c) Desempenho: podemos medir as seguintes características de velocidade: tempo de resposta, velocidade de processamento (*throughput*), utilização dos processadores, parcela da rede utilizada;

d) Tolerância a Falhas: os sistemas distribuídos devem estar em alta disponibilidade, evitar perda de dados em caso de falha de uma das máquinas da rede e permitir um mecanismo de replicação;

e) Escalabilidade: o sistema distribuído deve funcionar com um maior número de computadores à medida que se acrescenta maior poder de processamento ou novas unidades de processamento ao conjunto de servidores. Para implementar esta característica de escalabilidade nos algoritmos dos sistemas distribuídos, são necessários algumas medidas, que são:

– Nenhuma máquina deve conter toda a informação sobre o sistema.
– As máquinas devem tomar decisões baseadas em informações locais.
– A falha de uma máquina não inviabiliza todo o algoritmo.
– Não se deve assumir a existência de um relógio global.

O projeto da arquitetura JADE implementa as características desejáveis dos sistemas distribuídos.

A Figura 13 mostra o modelo da Arquitetura JADE.

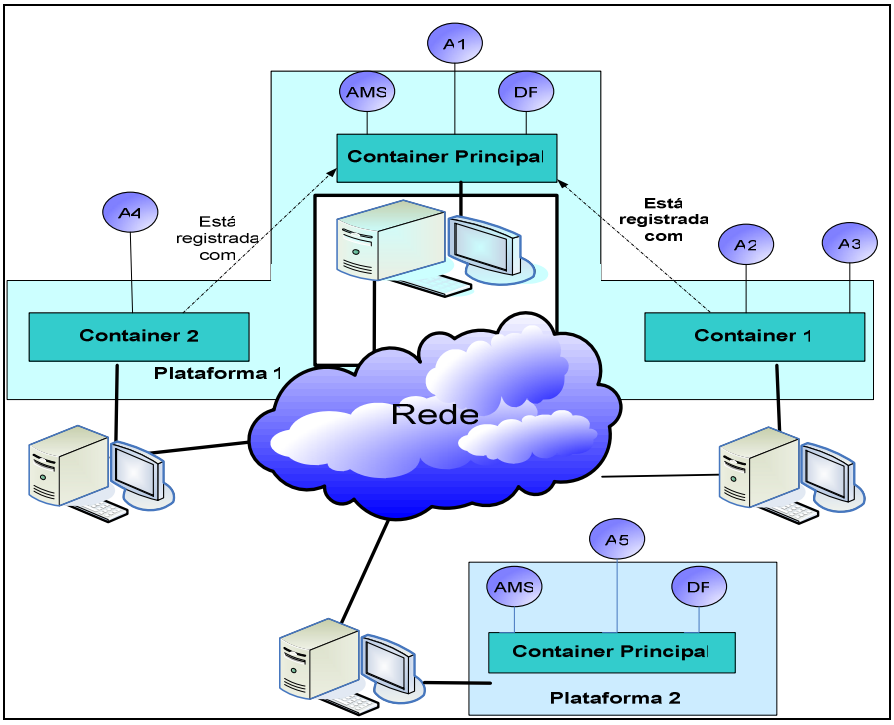

Figura 13: Modelo da Arquitetura JADE.
Fonte: JADE, 2007.

Neste modelo de arquitetura JADE, a sua instância é denominada *container.* Cada *container* pode abrigar vários agentes, que provêem mecanismos para assegurar que os agentes AMS e DF realizem serviços de página branca e página amarela respectivamente, onde o *container* principal é o responsável pelos registros de outros *containers* na arquitetura.

A função da arquitetura é disponibilizar um sistema distribuído que oculta dos agentes a complexidade relacionada à topologia, hardware, sistemas operacionais etc.

No modelo de arquitetura da Figura 13, observa-se que a Plataforma 1 contém três *containers* e a Plataforma 2 apenas um *container*, ressalta-se que neste modelo freqüentemente existe somente um *container* principal. Se for observada a execução de diferentes *containers,* então existem diversas instâncias de outras plataformas, onde cada instância possui seu respectivo *container* denominado secundário.

Os agentes instanciados no JADE são identificados por um nome único. A utilização deste mecanismo permite que cada agente conheça os nomes dos demais agentes e estabeleça o processo de comunicação.

A Figura 14 mostra a plataforma de agente distribuída JADE.

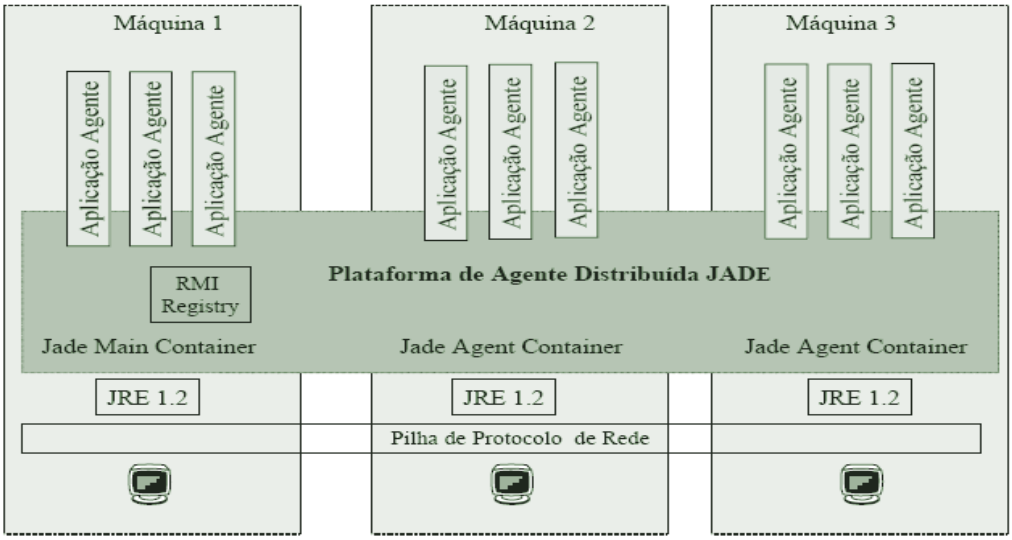

Figura 14: Plataforma de agente distribuída JADE.
Fonte: JADE, 2007.

Esta arquitetura é baseada na distribuição de vários JRE'S (*Java Runtime Environment*) que podem estar distribuídas por diversas máquinas para execução de agentes na plataforma JADE.

2.7.2 Agentes na Plataforma JADE

Os agentes desenvolvidos na plataforma JADE são uma subclasse da superclasse Agent do pacote "jade.core.Agent". Esse agente definido herda todos os métodos de sua superclasse. Portanto, para criar um agente reativo na plataforma JADE é preciso instanciar o objeto da classe "jade.core.Agent". O método setup() herdado realiza as inicializações necessárias para que o agente criado desempenhe o seu objetivo. Portanto, um agente na plataforma JADE é simplesmente uma instância da superclasse Agent, na qual os programadores escrevem seus agentes como subclasses e adicionam comportamentos específicos aos objetivos da aplicação e utilizam as capacidades herdadas da classe Agent.

A Figura 15 mostra a estrutura da classe jade.core.Agent.

```
public class AgentGestor extends jade.core.Agent {
  protected void setup() {
    addBehaviour(new jade.core.behaviours.TickerBehaviour(this, 10000) {
    protected void onTick() {
        //Realiza uma tarefa específica
      }
    } );
  }
}
```

Figura 15: Estrutura da classe *jade.core.Agent*.

A classe *Agent* provê métodos para executar tarefas básicas, tais como:

– Passagens de mensagens usando objetos *ACLMessage*;

– Suporte completo ao ciclo de vida dos agentes;

– Escalonamento e execução de múltiplas atividades concorrentes;

– Interação simplificada com sistemas de agentes FIPA para a automação de tarefas comuns aos agentes.

É importante ressaltar que as especificações FIPA estabelecem padrões comportamentais relacionados aos agentes externos ao sistema. Observa-se também que a plataforma JADE não determina o tipo de agente que pode ser desenvolvido.

Os agentes na plataforma JADE podem estar em diversos estados, de acordo com o ciclo de vida especificado pela FIPA.

A Figura 16 mostra o modelo do ciclo de vida de um agente definido pela FIPA.

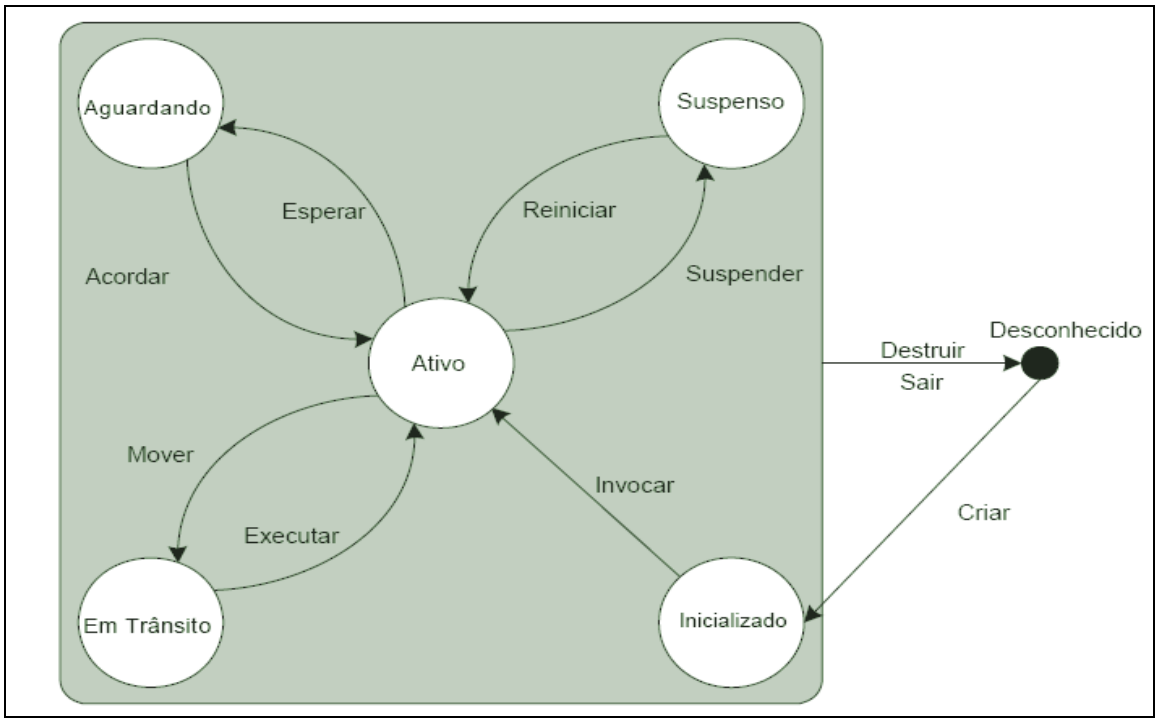

Figura 16: Modelo do ciclo de vida de um agente definido pela FIPA.
Fonte: FIPA, 2007.

As transições de estados são feitas por meio de métodos públicos da superclasse Agent como, por exemplo:

– Método "*doWait()*" – que faz o agente passar do estado ativo para o estado em espera.

– Método "*doSuspend()*" – a transição do estado ativo ou em espera para o estado suspenso.

– Método "*doDelete()*" – por sua vez, faz a transição dos estados ativo, suspenso ou em espera para o estado desconhecido, que causa a destruição do agente.

Um agente somente pode executar seus comportamentos no estado ativo. Assim, quando algum comportamento do agente chama o método "*doWait()*", ocasionará a interrupção de todas as suas atividades, inclusive, outros comportamentos e não só o comportamento que chamou o método. O método "*block()*" da classe *Behaviour*, ao contrário do método *doWait*, suspende apenas um comportamento do agente.

2.7.3 Comportamentos

Um agente na plataforma JADE desempenha tarefas e ações através de comportamentos com a finalidade de cumprir os objetivos da sociedade de agentes, sendo estas tarefas implementadas como comportamentos específicos de um agente.

Desta forma, um agente para realizar um objetivo deve implementar comportamentos específicos, instanciá-los e adicioná-los ao método *addBehaviour*(*Behaviour*) herdado da superclasse *Agent*. As tarefas são codificadas como objetos da classe *Behaviour*. Cada agente na plataforma JADE é representado por uma linha (*thread*) de execução. A plataforma JADE utiliza uma *thread* por agente ao invés de usar uma *thread* para cada comportamento para não afetar o desempenho da aplicação.

2.7.4 Comunicação entre Agentes na Plataforma JADE

Os modelos de comunicação na plataforma JADE são realizados através de:

– Cooperação: compartilhando conhecimentos.

– Colaboração: coordenações de ações entre os agentes.

– Negociação: processo para a tomada de decisões conjunta entre os agentes visando um acordo para benefício mútuo.

A troca de mensagens no ambiente JADE é realizada através de métodos próprios e com o uso de instâncias da classe *ACLMessage*, que possui um conjunto de atributos que estão em conformidade com as especificações da FIPA, implementada na linguagem FIPA-ACL. Em uma mensagem do JADE, o remetente é sempre um agente e o receptor pode ser um agente ou um conjunto de agentes.

Para um agente enviar uma mensagem deve, primeiro, instanciar um objeto da classe *ACLMessage*, preencher os atributos com as informações necessárias e chamar o método "*send()*" da classe *Agent*. Por outro lado, para

receber uma mensagem, o agente deve fazer referência ao método "*receive()*" ou "*blockingReceive()*". Os métodos, para enviar e receber mensagens, estão implementados na classe *Agent*.

O envio e o recebimento de mensagens podem ser escalonados como atividades independentes do agente, através da adição dos comportamentos "*SenderBehaviour* " e "*ReceiverBehaviour* " na fila de comportamentos do agente.

O padrão FIPA define que somente as mensagens ACL são transportadas na plataforma, porém não especifica nenhum padrão para o conteúdo da mensagem. Cada agente possui uma fila privativa de mensagens ACL que é processada de forma que o agente achar conveniente.

Quando um agente envia uma mensagem para outro agente, essa mensagem vai para o DF (Diretório Facilitador), entra em uma fila e fica acessível a todos os agentes que consultam o DF.

A plataforma JADE utiliza um mecanismo de transporte adaptável, ou seja, um mecanismo de transporte com o objetivo único de atingir o menor custo possível de passagens de mensagens. O processo de passagem de mensagem ocasiona algumas situações possíveis, que são:

– Se o agente receptor reside no mesmo container de agentes, então, o objeto JAVA, representante da mensagem ACL, é passado para o receptor via eventos.

– Se o agente receptor reside na mesma plataforma JADE, mas em containeres diferentes, então, a mensagem ACL, é enviada usando Java RMI.

– Se o agente receptor reside em uma plataforma de agente diferente, então, o protocolo IIOP (*Internet Inter ORB Protocol*) é usado de acordo com o padrão FIPA.

2.8 Conclusões

Este capítulo apresentou os conceitos sobre Sistemas Inteligentes (SI´s). e Sistemas Multiagentes (SMA´s).

Em seguida, foram apresentadas as principais metodologias para o desenvolvimento de softwares baseados em sistemas multiagentes, que são: *GAIA, MASE, PROMETHEUS* e *PAS*SI, mostrando as arquiteturas, formas de comunicação e protocolo de comunicação dos agentes.

Finalmente, apresentamos a plataforma de agente JADE que foi utilizada para desenvolver os agentes, seus comportamentos e a sua forma de comunicação.

3 DATA WAREHOUSE

A tecnologia Data Warehouse (DW) tem o objetivo de transformar os dados existentes nas bases de dados operacionais das organizações em informações, onde os gestores possam visualizar suas tendências, sazonalidades e realizar as análises gerenciais para obter informações necessárias aos seus processos decisórios.

Segundo Ramalho (1999, p. 183), o DW é um “banco de dados orientado para consultas, resultado de uma extensa análise para transformação de dados da empresa”.

Para *Boghi et al* (2002), as aplicações que utilizam DW objetivam realizar a conversão dos dados armazenados nas bases de dados dos ambientes operacionais em informações utilizáveis, além de permitir o uso estratégico destes dados para a tomada de decisões.

Assim, o Data Warehouse é uma combinação de várias tecnologias, tendo como primeiro objetivo a integração efetiva de bases de dados operacionais em ambiente DW para o uso estratégico dos dados.

Segundo Passos *et al* (2005), o DW permite reunir os dados corporativos localizados em diferentes bases de dados operacionais em um único repositório, apresentando informações específicas nos níveis gerenciais e estratégicos das organizações.

O DW é composto por um conjunto de dados baseados em assuntos, integrados, não voláteis, variáveis em relação ao tempo e destinado a auxiliar a tomada de decisões (MACHADO, 2000. INMON, 2002. GONÇALVES, 2003). A tecnologia DW possui um conjunto de características que a diferencia das bases de dados dos ambientes operacionais, que são:

- Orientado por assunto: o DW é projetado para agrupar informações e analisar os seus conjuntos de dados por assunto de interesse, possibilitando a tomada de decisão.

A Figura 17 mostra a organização dos dados orientados por assunto no ambiente operacional e no DW.

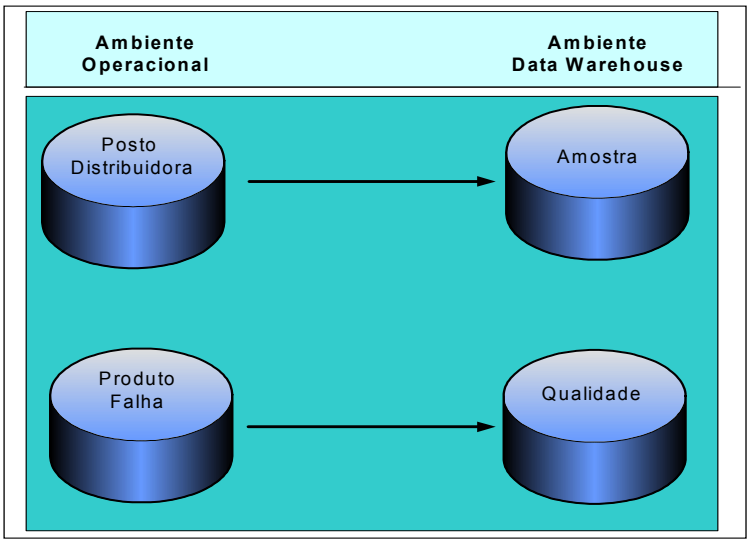

Figura 17: Organização dos dados orientados por assuntos no ambiente operacional e no DW.
Fonte adaptada: Inmon, 2002.

As bases de dados do ambiente operacional são utilizadas para controle das aplicações, enquanto no ambiente DW são utilizadas para a tomada de decisões.

– Variável no Tempo: no ambiente operacional, o usuário visualiza somente a última atualização, enquanto no DW, os dados, uma vez incluídos, não podem ser alterados, ou seja, eles somente são atualizados na sua carga.

A Figura 18 mostra os dados existentes no ambiente operacional e no DW.

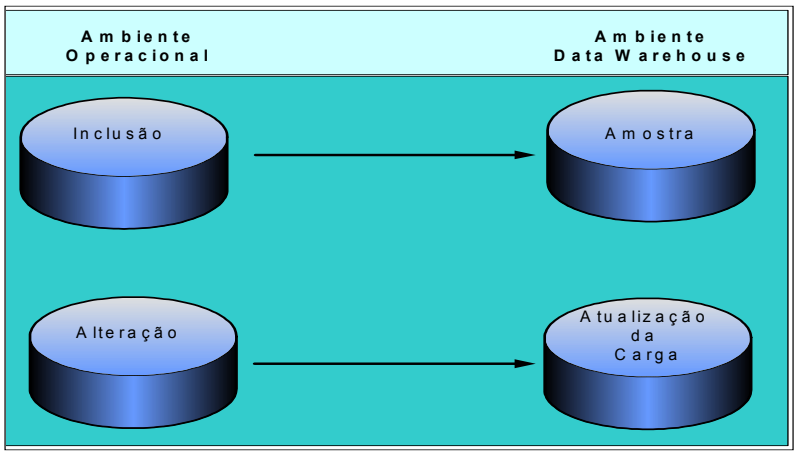

Figura 18: Dados existentes no ambiente operacional e no DW.
Fonte adaptada: Inmon, 2002.

Podemos observar que uma atualização no ambiente operacional modifica o registro atual, enquanto no DW uma alteração gera um novo registro permitindo a visualização do histórico das alterações.

– Não Volatilidade: uma vez que os dados são armazenados, o DW não permite mais a sua atualização por parte do usuário, ou seja, é possível somente a alteração no armazenamento de um novo conjunto de dados associados a um determinado período de tempo.

A Figura 19 mostra a não volatilidade no ambiente operacional e no DW.

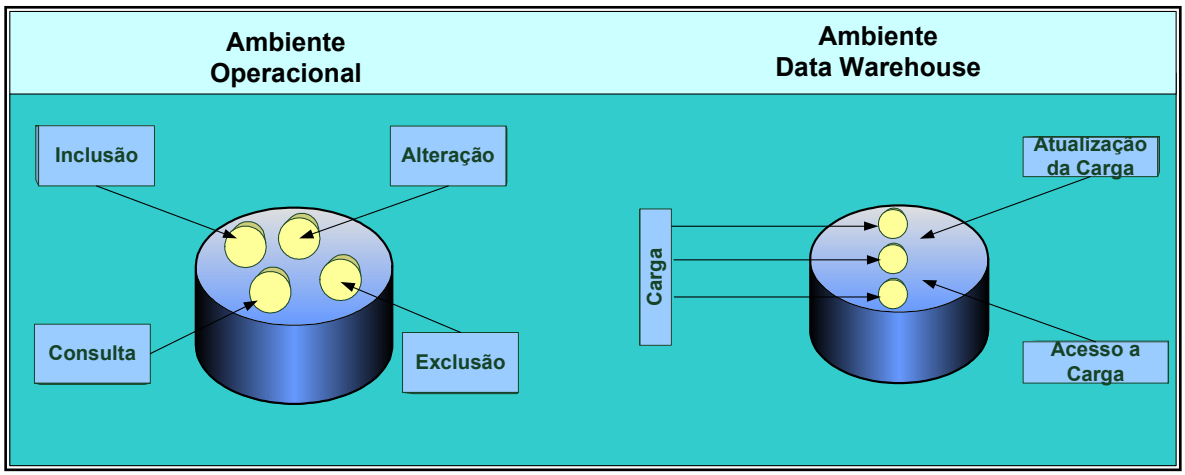

Figura 19: Não volatilidade no ambiente operacional e no DW.
Fonte adaptada: Inmon, 2002.

– Integração: a integração das diversas bases de dados é um conceito fundamental no DW, pois possuem diversos objetivos, que são:

a) Padronização nos dados: a padronização é realizada para evitar duplicidade na geração de relatório.

b) Convenções de nomes: refere-se à integração dos atributos da base de dados em um único formato.

c) Eliminações de inconsistências: tem como objetivo evitar duplicidade de dados para gerar o formato único.

A Figura 20 mostra a integração das bases de dados heterogêneas existentes no ambiente operacional para a base de dados do DW.

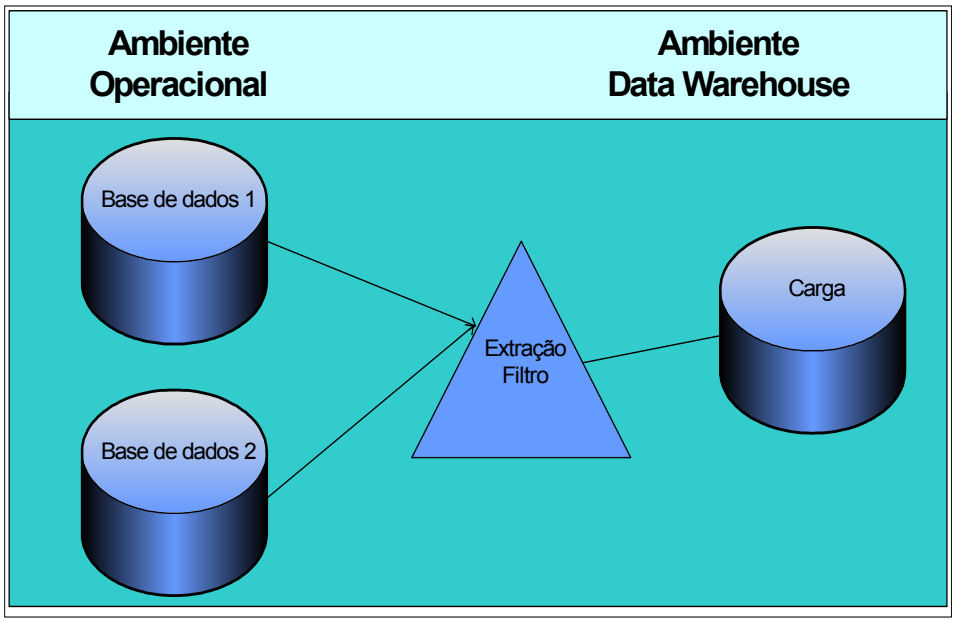

Figura 20: Integração das bases de dados heterogêneas existentes no ambiente operacional para a base de dados do DW.
Fonte: Machado, 2000.

As inconsistências existentes e a ausência de padronização nas bases de dados operacionais recomendam que o DW seja utilizado visando eliminar as redundâncias dos dados, além de uniformizar a padronização destes para garantir uma visão analítica dos dados contidos nas bases de dados do ambiente DW.

Estes conjuntos de características proporcionam desenvolver o projeto arquitetural do DW para garantir a sua construção e manutenção. Estas características são:

- Agrupar as informações por assunto.
- Garantir a periodicidade da volatilidade dos dados armazenados.
- Possibilitar a integração das diversas bases de dados.

3.1 Granularidade

Segundo Machado (2000), a granularidade no DW está associada ao nível de detalhes, observando que, quanto maior o nível de detalhes, menor a granularidade e, quanto menor o nível de detalhes, maior a granularidade.

A Figura 21 mostra a granularidade das bases de dados no ambiente DW.

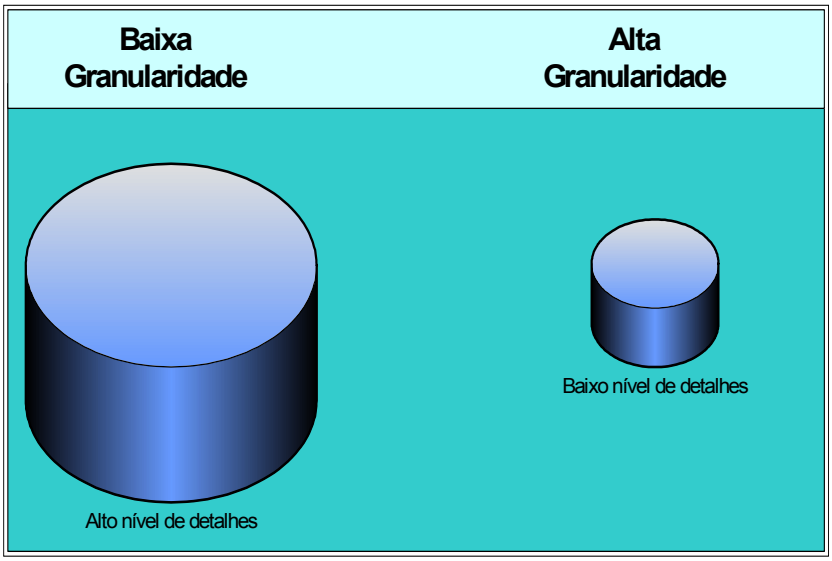

Figura 21: Granularidade das bases de dados no ambiente DW.
Fonte: Inmon, 2002.

A granularidade é fundamental no projeto do DW e pode interferir nas análises de desempenho que são condicionadas pelo aumento no volume de dados e no tempo de resposta das consultas. Quando se tem um nível de granularidade muito alto, o espaço em disco e o número de índices necessários diminuem, ou seja, existe uma correspondente redução nas possibilidades de utilização dos dados para atender a consultas detalhadas.

Assim, o nível de detalhes indica qual é o nível de granularidade de um DW. Portanto, quando se tem um nível de granularidade alto, significa que o detalhe sobre o assunto é relativamente baixo. Por outro lado, com um nível baixo de granularidade, pode-se responder a qualquer consulta. Entretanto, em um ambiente DW dificilmente um evento isolado é examinado, ou seja, é comum ocorrer sempre a utilização das visões dos conjuntos de dados (INMON, 2002).

3.2 Data Marts

Para Machado (2000), a utilização de Data Mart (DM) no DW é representada por conjuntos de dados específicos. A DM permite acessar aleatoriamente os conjuntos de dados e serve como referência para formar a composição das bases de dados individualmente.

De acordo com Passos (2005), o DM representa um subconjunto do DW para atender uma área específica da empresa, oferecendo diversas vantagens, que são: simplicidade, menor custo e agilidade durante o processo de construção e manutenção do DW.

Segundo Date (2003, p. 604), o DM é o "*depósito especializado, orientado a assunto, integrado, volátil e variável no tempo que fornece apoio a um subconjunto específico de apoios gerenciais*".

Portanto, o DM é projetado para fornecer informações específicas sobre um determinado departamento ou setor de uma organização, tais como: financeiro, administrativo, estoque, entre outros.

3.3 Modelagem Multidimensional

Para Passos (2005), a modelagem multidimensional é uma técnica computacional que foi desenvolvida para projetar os conjuntos de dados existentes no DW, além de descrever os aspectos comuns sobre determinados assuntos. Esta técnica também é utilizada para sumarizar e organizar dados apresentados por visões que suportam a análise dos valores envolvidos.

Segundo Kimball (2002), a modelagem multidimensional possui três componentes básicos, que são:

- Medidas: são atributos numéricos que representam um fato.
- Tabela de Fatos: é uma coleção de itens de dados que é composta por dados de medidas e dados de contexto, ou seja, possui como característica básica a sua representação através de valores numéricos.
- Tabela de Dimensão: é um tipo de informação que participa de um fato, ou seja, ela representa e armazena as descrições textuais.

O projeto físico da modelagem multidimensional do DW pode ser realizado utilizando duas metodologias, que são:

a) Modelo Estrela (*Star*) – o modelo estrela tem uma estrutura básica multidimensional que simplifica a sua implementação e é composta pela tabela de fato e por várias tabelas de dimensões que são agrupadas.

A Figura 22 mostra o modelo estrela no ambiente DW.

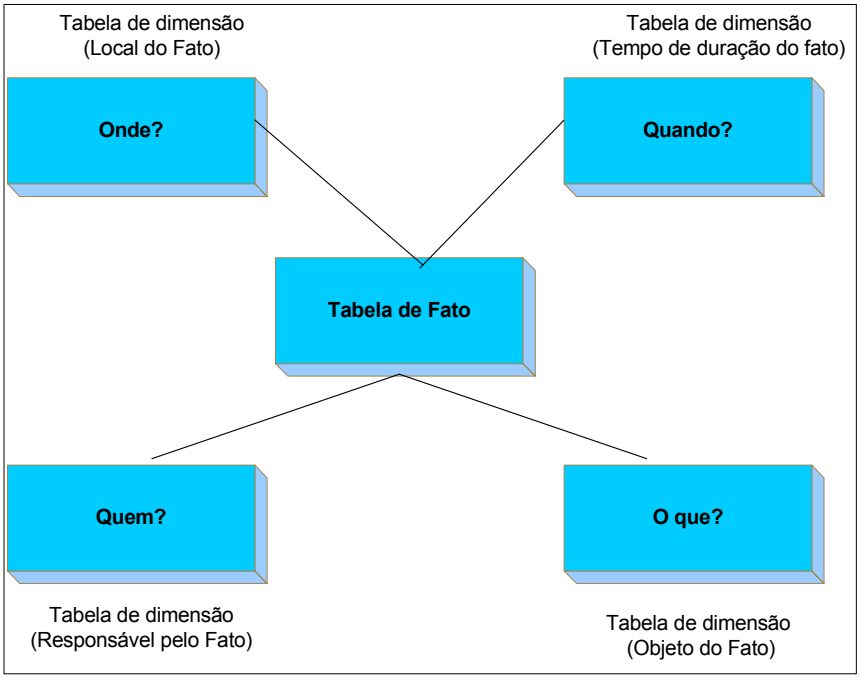

Figura 22: Modelo estrela no ambiente DW.
Fonte: Machado, 2000.

Este modelo é formado pela composição da tabela de fato e por tabelas de dimensões.

As funcionalidades de cada tabela de dimensão existente no modelo estrela são:

- Tabela de dimensão local: identifica o local onde aconteceu o fato;
- Tabela de dimensão temporal: registra quando aconteceu o fato;
- Tabela de dimensão responsável pelo fato: associa quem executou o fato;
- Tabela de dimensão objeto do fato: identifica qual é o objeto do fato.

b) Modelo Floco de Neve (*SnowFlake*) – este modelo apresenta estrutura semelhante ao modelo estrela, porém, a única diferença é que as

tabelas de dimensões podem ser subdivididas, originando novas tabelas de dimensões específicas.

A Figura 23 mostra o modelo floco de neve no ambiente DW.

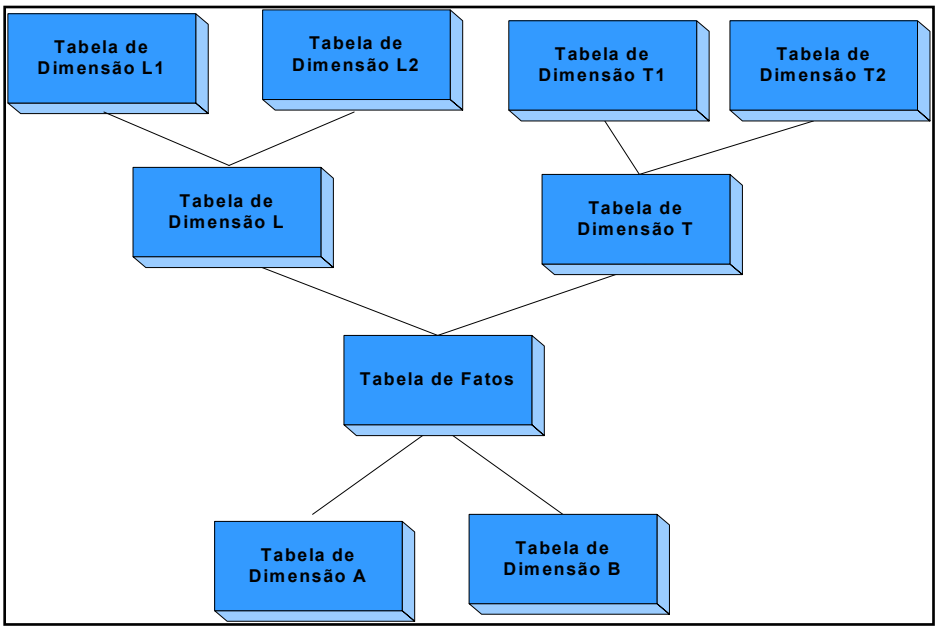

Figura 23: Modelo floco de neve no ambiente DW.
Fonte: Machado, 2000.

Este modelo possui como principal característica a subdivisão das tabelas de dimensões, originando novas tabelas de dimensões e os respectivos relacionamentos entre elas. Assim, o modelo floco de neve é uma extensão dos relacionamentos anteriores com os relacionamentos de uma ou mais tabelas de dimensões.

3.3.1 Processamento Analítico Online (OLAP)

Para Date (2003), este é um processo interativo para criar, gerenciar, analisar e gerar relatórios sobre os dados.

Assim, o OLAP representa uma tecnologia que permite aos gestores a visualização de dados corporativos de forma rápida, além de possibilitar uma visão multidimensional dos dados da organização.

De acordo Date (2003), as arquiteturas que se destacam atualmente são:

– ROLAP (*Relational Online Analytical Processing*), baseada em base de dados relacionais;

– MOLAP (*Multidimensional Online Analytical Processing*), baseada em bancos multidimensionais;

– HOLAP (*Hybrid Online Analytical Processing*), baseada na combinação das arquiteturas ROLAP e MOLAP, onde se pode selecionar os benefícios de cada arquitetura, ou seja, a escabilidade na ROLAP e o desempenho da MOLAP.

Para manipular o modelo multidimensional, faz-se necessário conhecer algumas funcionalidades básicas do OLAP (SELL, 2006).

A Tabela 5 mostra as funcionalidades básicas do OLAP.

FUNCIONALIDADE	DESCRIÇÃO
Drill Down	Aumenta o nível de detalhe da informação e, conseqüentemente, diminui o grau de granularidade.
Roll Down	Reduz o nível de detalhes de uma informação até o nível de sumarização.
Drill Up	Diminui o nível de detalhe é aumenta o grau de granularidade.
Roll Up	Aumenta o nível de detalhes de uma informação até o nível mais elevado de sumarização.
Slice	Corta o cubo, mas mantém a mesma perspectiva de visualização dos dados.
Dice	Muda a perspectiva da visão multidimensional como se o cubo fosse girado. Permite descobrir comportamentos e tendências entre os valores das medidas analisadas em diversas perspectivas.
Pivoting	Inverte as dimensões entre linhas e colunas (Passos, 2005).
Data Surfing	Executa uma mesma análise em outro conjunto de dados (Passos, 2005).

Tabela 5: Funcionalidades básicas do OLAP.
Fonte adaptada: Sell, 2006.

Estas funcionalidades permitem ao sistema OLAP controlar efetivamente a criação e a geração dos relatórios analíticos, de acordo com as necessidades do usuário.

3.4 Conclusões

A utilização do DW, nesta pesquisa, tem como objetivo transformar os dados existentes nas bases de dados operacionais das organizações em informações.

Assim, foram apresentadas as principais características do DW e as suas divergências relacionadas ao ambiente operacional para a base de dados do DW. Finalmente mostramos o DM e a Modelagem Multidimensional, destacando o OLAP e as suas funcionalidades.

4 PROJETO DO SISTEMA INTELIGENTE DO PROCESSO DE MONITORAMENTO E CONTROLE DA QUALIDADE DE COMBUSTÍVEL

A pesquisa desenvolvida teve como objetivo o desenvolvimento de um sistema inteligente a ser utilizado no Monitoramento da Qualidade de Combustíveis (MQC) do Laboratório de Análise e Pesquisa em Química Analítica do Petróleo da Universidade Federal do Maranhão (LAPQAP/UFMA). Este sistema automatizará o processo de análise e tratamento de dados, além de fornecer suporte para o gerenciamento e o apoio à tomada de decisões. Por isso, foi concebido usando a tecnologia de agentes inteligentes.

Apresenta-se, a seguir, as fases utilizadas no projeto do sistema SIMCQC.

4.1 Análise do Processo de Negócio do SIMCQC

Atualmente, o processo de monitoramento da qualidade de combustíveis consiste em diversas fases, que são:

- Coleta das amostras dos combustíveis;
- Realização das análises químicas;
- Análise e tratamento de dados no MQC;
- Envio das informações para ANP.

Para executar essas fases, a ANP mantém acordos com 23 (vinte e três) instituições conveniadas (universidades e centros tecnológicos) que atualmente monitoram a qualidade dos combustíveis brasileiros. Estas instituições possuem metodologias disponíveis não automatizadas para realizar as atividades de monitoramento da qualidade do combustível.

Cada amostra é submetida a diferentes métodos de análises para avaliar se o combustível apresenta ou não algum tipo de não conformidade. Na fase da

análise química, podem-se utilizar as técnicas de ponto de fulgor, massa específica, destilação, infravermelho e teor de álcool etílico anidro combustível, entre outras.

A pesquisa desenvolvida utilizou, na fase de análise química, a técnica de destilação de gasolina que é composta pelos componentes de hardware (equipamento de destilação) e software (*Herzog Labor Informations System 32 Bit – HLIS 32*). O componente de software realiza a análise química selecionada, registra o comportamento da curva de destilação da amostra, além de gerar dados para posterior análise. Então, uma determinada equipe do laboratório realiza a análise dos dados gerados e depois são digitadas no MQC.

Observou-se, portanto, que a fase da análise química está automatizada, mas a integração da base de dados resultantes da análise química com o software MQC (padrão ANP) estava semi-automatizada. Além disso, não realizava nenhum tipo de análise gerencial e apoio à tomada de decisões, ou seja, os dados gerados pelas análises químicas eram simplesmente digitados no MQC para posterior envio à ANP.

A Figura 24 mostra o processo de monitoramento da qualidade de combustível encontrado no LAPQAP /UFMA.

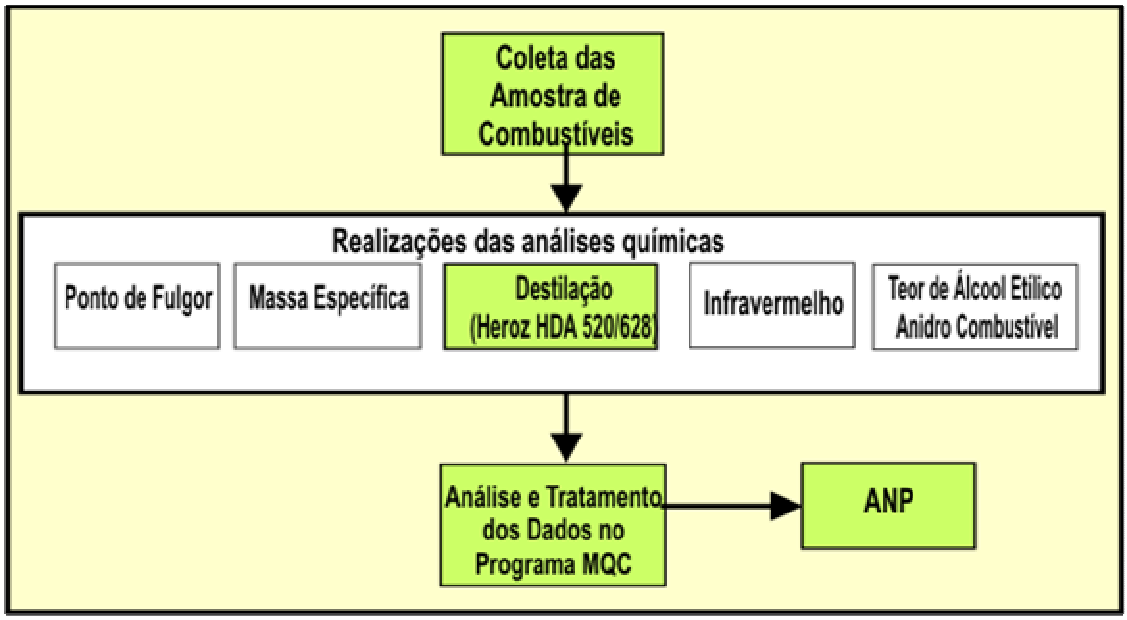

Figura 24: Processo de monitoramento de combustível encontrado no LAPQAP/UFMA.

Apresenta-se, a seguir, detalhadamente, as fases que compreendem o processo de monitoramento de combustível. Estas são:

– **Fase de Coleta da Amostra**: uma equipe do laboratório coleta as amostras dos combustíveis nos postos de combustíveis, conforme as normas/critérios da ANP. Essas amostras são levadas ao laboratório, onde há uma equipe técnica preparada para iniciar a segunda etapa do processo.

– **Fase de Realização da Análise Química**: consiste em submeter a amostra ao destilador. Ao término desta etapa, o software do destilador mostra o resultado com todos os dados provenientes da destilação.

– **Fase de Análise e Tratamento de Dados**: inicia-se quando a equipe do laboratório analisa e organiza os dados obtidos na segunda fase, anotando as informações em um formulário. As informações desse formulário são enviadas para a ANP através do software MQC (Monitoramento da Qualidade do Combustível).

A ausência de análise gerencial e, conseqüentemente, o apoio à tomada de decisões motivou o desenvolvimento de um sistema inteligente que automatizasse as fases da análise e tratamento de dados.

A Figura 25 mostra o modelo da tecnologia de agente inteligente proposto no processo de monitoramento de combustível.

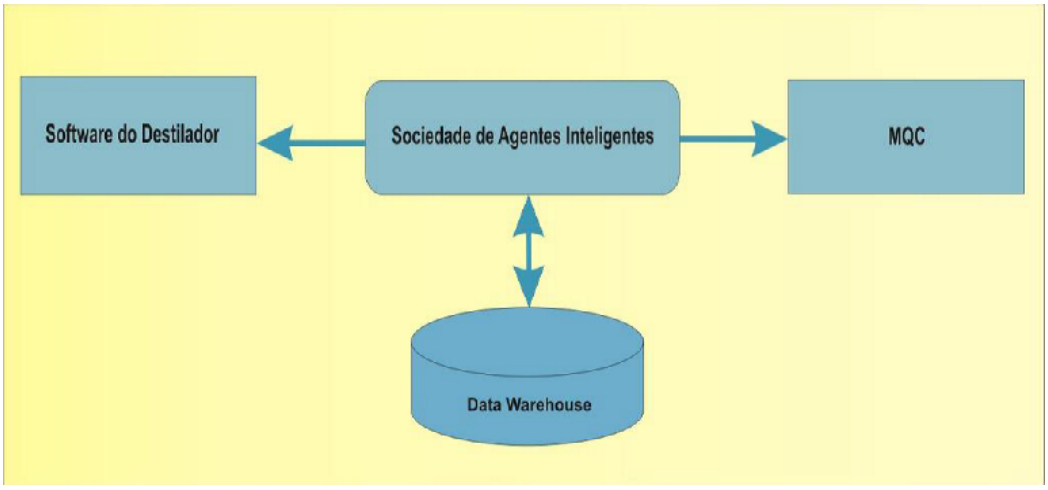

Figura 25: Modelo da tecnologia de agente inteligente proposto no processo de monitoramento de combustível.

Deste modo, o modelo da tecnologia de agente inteligente proposto no processo de monitoramento de combustível está subdivido em:

– Software destilador – tem como objetivo gerar os dados provenientes das amostras analisadas;

– Sociedade de agente inteligente – esta sociedade tem como objetivo criação dos agentes necessário para o funcionamento do sistema inteligente (monitor,gestor,conversor e dw).

– MQC – tem como objetivo o envio dos dados analisados para ANP;

– Data Warehouse – esta tecnologia tem como objetivo a criação de um repositório de dados, visando obtenção de informação estratégica para tomada de decisões.

Este modelo automatiza as diversas atividades (obtenção, organização e análise de dados) que são realizadas pelos pesquisadores.

A Figura 26 mostra o desenvolvimento do DW para o modelo proposto na arquitetura.

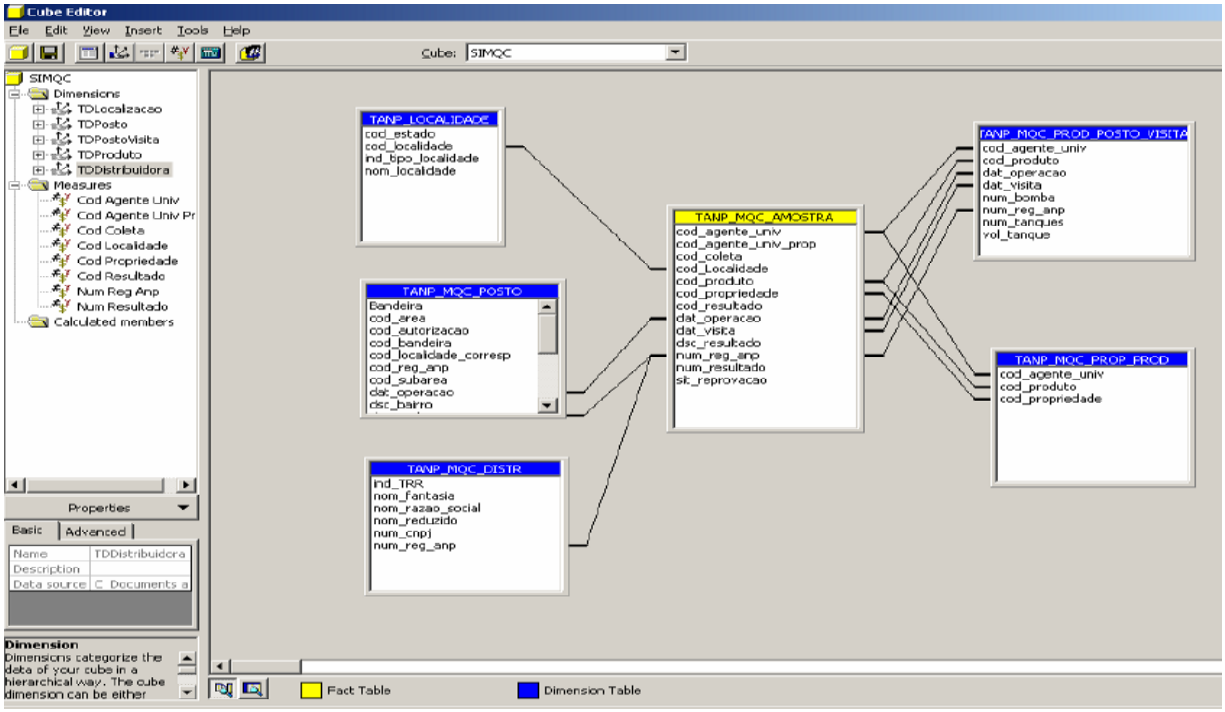

Figura 26: Desenvolvimento do DW para o modelo proposto na arquitetura.

O desenvolvimento do DW para o modelo proposto é representado pelo modelo estrela (star), composta pela seguinte tabela, que são:

– Tabela de fatos Amostra: contém os registros das amostras coletadas e a situação de cada amostra, se ela encontra-se conforme ou não conforme de acordo com o padrão ANP;

– Dimensão Localidade: refere-se à localização dos postos e nomes dos municípios onde se encontram os postos;

– Dimensão Posto Visitados: referem-se aos dados das propriedades analisadas e data da visita;

– Dimensão Distribuidora: referem-se aos dados dos distribuidores de combustíveis;

– Dimensão Posto: razão social do posto e seu registro junto a ANP;

– Dimensão Produto: tipo de combustível sendo analisado e comercializado no estado do Maranhão.

4.2 A Sociedade de Agentes Inteligentes no Processo de Monitoramento

O diagrama de caso de uso representa a interação entre um sistema e os agentes externos que utilizam esse sistema (BEZERRA, 2002. SOMMERVILLE, 2003).

A Figura 27 mostra o diagrama de caso de uso do sistema no monitoramento da qualidade de combustíveis.

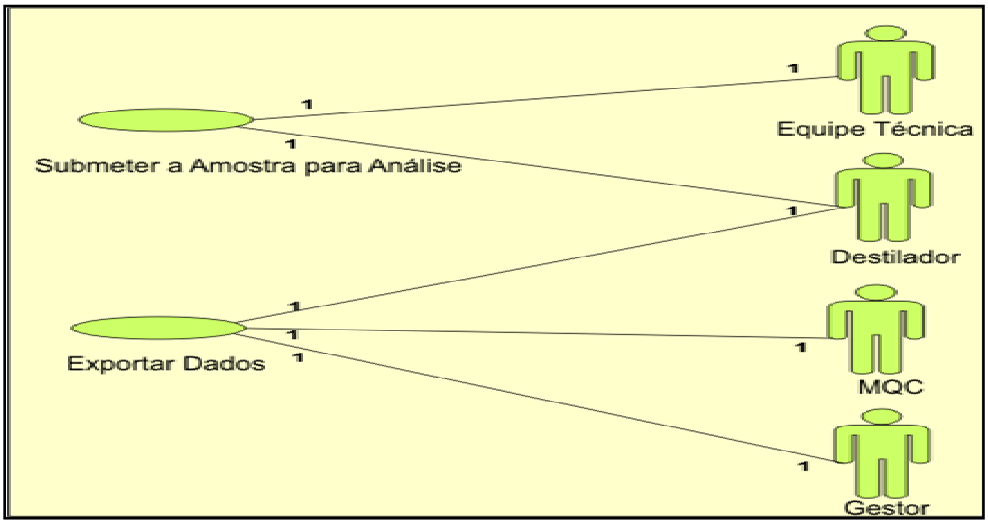

Figura 27: Diagrama de caso de uso do sistema no monitoramento da qualidade de combustíveis.

As funcionalidade do diagrama de casos de uso do sistema no monitoramento da qualidade de combustíveis estão representadas nas Tabelas 6 e 7 as quais apresentam os casos de usos Submeter Amostra para Análise e Exportar Dados.

SUBMETER AMOSTRA PARA ANÁLISE
Sumário: A equipe de amostragem envia a amostra coletada para equipe técnica submeter a amostra à análise química no destilador, obedecendo aos critérios estabelecidos pela ANP. **Ator Principal:** Destilador (Responsável pela análise química das amostras coletadas). **Ator Secundário**: Equipe Técnica (Responsável por submeter a amostra coletada à análise química). **Pré-condições:** 1. A equipe técnica realiza a identificação da amostra coletada para análise química. **Fluxo Principal:** – A equipe técnica verifica o Lacre. – A equipe técnica coloca a amostra coletada no ensaio de destilação. – A equipe técnica cadastra a data de entrega da amostra. – A equipe técnica cadastra a data da análise. – A equipe técnica cadastra o técnico responsável pela coleta. – A equipe técnica cadastra o técnico responsável pela análise da amostra coletada. – O destilador realiza a análise química. – O destilador gera o resultado da amostra analisada em tempo real ou em arquivo. **Exceções:** Se a amostra estiver com o lacre rompido, não poderá ser analisada. **Fluxo Alternativo:** Se a análise da amostra coletada estiver muito adulterada, o destilador não gera a curva de destilação e, por conseguinte, todo o processo será realizado manualmente, conforme apresentamos a seguir: – A equipe técnica realiza a análise manualmente no equipamento. – A equipe técnica verifica a temperatura manualmente fazendo uso de termômetro. – A equipe técnica realiza o controle do tempo manualmente. **Pós-condições:** A equipe técnica deverá analisar a curva de destilação, a fim de verificar a conformidade ou não conformidade da amostra analisada.

Tabela 6: Caso de uso Submeter Amostra para Análise.

EXPORTAR DADOS
Sumário: A equipe envia os dados analisados para o MQC e para o Gestor, possibilitando o gerenciamento das análises para tomada de decisão. **Ator Principal:** Destilador (Responsável pela exportação de dados resultantes da análise). **Atores Secundários:** MQC (Responsável pelo envio dos resultados da análise para ANP), **Gestor** (Responsável pela análise dos dados para tomada de decisão no LAPQAP). **Pré-condição:** 1. Gera o arquivo com os dados analisados. **Fluxo Principal:** – Mostra o resultado da análise. – Exporta o arquivo da análise. **Exceção:** Se o arquivo a ser exportado estiver vazio, não será exportado. **Fluxo Alternativo:** Se o local de armazenamento estiver cheio, é necessário selecionar outro dispositivo de armazenamento. **Pós-condições:** Exporta o arquivo resultante das análises.

Tabela 7: Caso de uso Exportar Dados.

A Tabela 8 apresenta a identificação dos agentes inteligentes e seus objetivos.

	ETAPAS	FUNCIONALIDADE DO AGENTE	TIPO DE AGENTE	INTERAÇÃO	OBJETIVO DO AGENTE
1	Consultar a base de dados.	Extrair informações de novo resultado e enviar para o agente Gestor.	Reativo	Base de Dados Textual	Consultar permanentemente a base de dados textual para extrair informações de novas amostras.
2	Analisar e tratar as informações.	Realizar a análise e tratamento das informações de novos resultados.	Cognitivo	Agentes Conversor e DW	Analisar e tratar as informações do resultado de amostra.
3	Converter o resultado da amostra tratadas em formato XML.	Aguardar mensagens do agente Gestor para disponibilizar as informações do resultado da amostra no formato XML.	Reativo	MQC	Esperar mensagens e convertê-las no formato XML e disponibilizar as informações para o Software MQC.
4	Manter um repositório do DW.	Montar um repositório do Data Warehouse com as informações do resultado da amostra advindas do agente Gestor, a fim de dar suporte para a equipe.	Reativo	Repositório Data Warehouse	Esperar mensagens contendo informações sobre as amostras do agente Gestor para serem armazenadas em um repositório do Data Warehouse.

Tabela 8: Identificação dos agentes inteligentes e seus objetivos

Esta tabela tem como finalidade identificar as etapas, funcionalidades, tipos de agentes, interações do agente no sistema e os objetivos de cada agente.

A Figura 28 mostra o modelo da arquitetura proposta para o sistema inteligente de monitoramento e controle da qualidade de combustíveis.

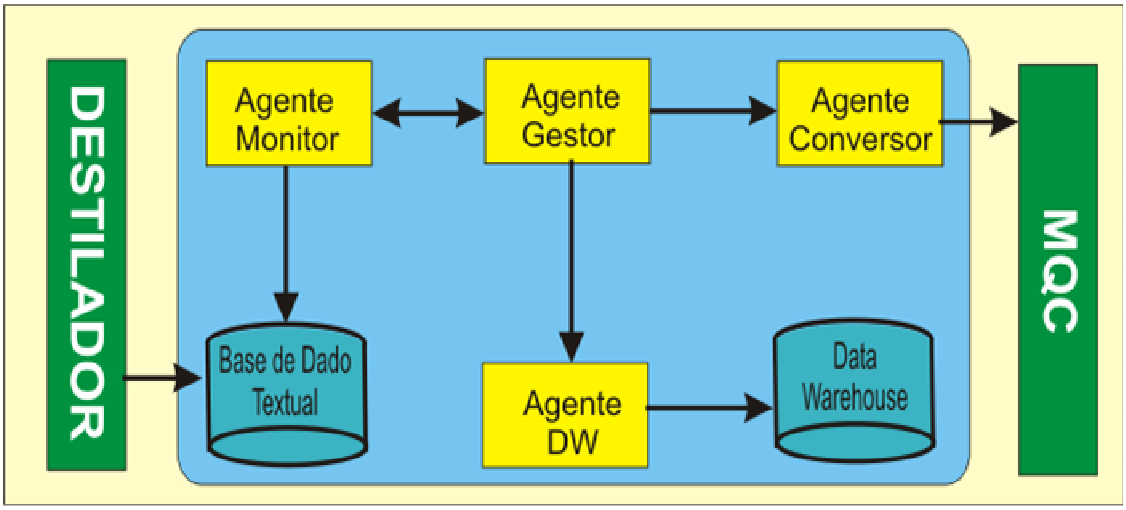

Figura 28: Modelo da arquitetura proposta para o sistema inteligente de monitoramento e controle da qualidade de combustíveis.

Nesta arquitetura, os agentes trocam mensagens para alcançar seus objetivos. Para tanto, utilizou-se a plataforma JADE e a metodologia PASSI para a modelagem do sistema de monitoramento de combustíveis.

Descreve-se, a seguir, os papéis de cada agente identificado no sistema, que são:

a) Agente Monitor

Este agente reativo desempenha o papel de consultar permanentemente a base de dados textual para extrair informações dos resultados de novas amostras provenientes do destilador. Caso existam novas amostras na base de dados textuais, este agente deverá enviar uma mensagem ACL contendo as informações do resultado da nova amostra para o Agente Gestor.

A base de dados textual é um arquivo gerado pelo software do destilador ao término de cada amostra. Assim, quando o destilador finaliza uma amostra, ele gera um arquivo com as informações relacionadas à amostra, tais como: identificação da amostra, data, hora, operador, ponto de ebulição Inicial (IBP), ponto

de ebulição 10%, ponto de ebulição 50%, ponto de ebulição 85% e ponto de ebulição 90%, ponto de ebulição final da amostra (IBP).

O Quadro 1 mostra a funcionalidade do agente monitor na sociedade de agentes.

AGENTE MONITOR
O agente monitor tem como objetivo identificar as amostras na base de dado textual de acordo com as instruções apresentadas abaixo: – Identifica a amostra. – Verifica data da análise. – Identifica técnico responsável pela análise da amostra. **Percepção**: consulta periodicamente a base de dado textual. **Objetivo**: verifica as amostras coletadas. **Ação**: identifica a amostra. **Tipo de Agente**: Agente Reativo.

Quadro 1: Funcionalidade do agente monitor na sociedade de agentes.

b) Agente Gestor

O agente gestor é um agente cognitivo que possui algumas funções no sistema, tais como:

- Realizar a análise e o tratamento das informações da amostra que foram enviadas pelo agente monitor;
- Automatizar as atividades desenvolvidas anteriormente pela equipe do laboratório;
- Enviar duas mensagens para os agentes conversores e DW com informações da amostra.

O Quadro 2 mostra a funcionalidade do agente gestor na sociedade de agentes.

AGENTE GESTOR
O agente gestor analisa e trata as informações enviadas pelo agente monitor. **Percepção**: recebe a mensagem do agente monitor. **Objetivo**: analisa e trata a informação recebida. **Ação**: realiza a análise dos dados e envia duas mensagens para os Agentes Conversores e DW. **Tipo de Agente**: Agente Cognitivo.

Quadro 2: Funcionalidade do agente gestor na sociedade de agentes.

c) Agente Conversor

O agente conversor é um agente reativo que desempenha o papel de monitorar as mensagens enviadas pelo agente Gestor. Ao receber uma mensagem ACL, contendo as informações da amostra, o agente Conversor transforma essas informações em um arquivo no formato padrão XML.

O Quadro 3 mostra a funcionalidade do agente conversor na sociedade de agentes.

AGENTE CONVERSOR
Converte os dados analisados pelo Agente Gestor e envia para a ANP os dados através do Software MQC. **Percepção**: recebe mensagens do agente gestor sobre a disponibilidade dos dados. **Objetivo**: converte os dados recebidos pelo Agente Gestor no formato XML e envia para o software MQC. **Ação**: envia os dados para ANP. **Tipo de Agente**: Agente Reativo.

Quadro 3: Funcionalidade do agente conversor na sociedade de agentes.

d) Agente DW

O Agente DW é o agente reativo que espera permanentemente por mensagens contendo as informações da amostra do agente Gestor para armazená-las em um repositório do Data Warehouse.

O Quadro 4 mostra a funcionalidade do agente DW na sociedade de agentes.

AGENTE DW
O Agente DW recebe os dados analisados pelo Agente Gestor e gera dados técnicos e analíticos para a tomada de decisões pelo LAPQAP. **Percepção**: recebe os dados analisados pelo Agente Gestor. **Objetivo**: gera dados técnicos e analíticos. **Ação**: analisa os dados para tomadas de decisões. **Tipo de Agente**: Agente Reativo.

Quadro 4: Funcionalidade do agente DW na sociedade de agentes.

Este agente interage com o Data Warehouse e, dependendo dos critérios estabelecidos, poderá gerar dados técnicos e analíticos para a tomada de decisões no LAPQAP/UFMA.

4.3 Conclusões

Neste capítulo apresentamos o projeto do Sistema Inteligente do processo de monitoramento e controle da qualidade de combustível. Inicialmente, foi realizada a análise do processo de negócio do SIMCQC na sociedade de agentes, ou seja, foram identificados os agentes inteligentes, seus objetivos e a arquitetura proposta.

O objetivo da proposta para o projeto da arquitetura do sistema inteligente foi facilitar a compreensão das trocas de mensagens entre os agentes para alcançar os seus objetivos.

5 PROTOTIPAGEM DO SISTEMA INTELIGENTE NO MONITORAMENTO E CONTROLE DA QUALIDADE DE COMBUSTÍVEIS

A prototipagem do sistema de monitoramento e controle da qualidade de combustíveis foi desenvolvida utilizando a metodologia PASSI (COSSENTINO *et al* 2002), juntamente com a plataforma JADE e JESS (FRIEDMAN, 2007) para desenvolver o protótipo do sistema inteligente SIMCQC.

A metodologia PASSI *Toolkit* aplica padrões de reuso que possibilitam a prototipagem rápida de um projeto. Além disso, a PASSI possui as seguintes características:

– Compilação automática dos diagramas - permite que os diagramas sejam salvos em tempo de análise e projeto possibilitando ao projetista aplicar as doze fases da Passi. A Passi *Toolkit* apresenta também outras sub-características, que são:

a) Diagrama da especificação de tarefa: é atribuído quando o agente é instanciado;

b) Modelo da estrutura do agente: é descrito na definição da estrutura do sistema multiagente;

c) Diagrama de descrição do comportamento do agente: é atribuído ao novo agente.

– Suporte automático para execução de operações recorrentes - a ferramenta Passi *Toolkit* permite que os desenvolvedores modifiquem o modelo e obtenham a atualização automática de todos os diagramas que dependem da modificação;

– Consistência de projeto - consiste em verificar a consistência do sistema multiagente. Além disso, uma operação de verificação é executada automaticamente sempre que o usuário termina alguma fase, ou seja, a verificação informará ao usuário sobre algum erro ou inconsistência;

– A compilação automática dos relatórios e documentos do projeto permite gerar artefatos no formato Microsoft Word;

– Acesso a um repositório de padrões comuns da engenharia de software e agentes (AUML, FIPA, XML, RDF);

– Geração de Código e Engenharia Reversa - utiliza a ferramenta PTK para gerar o código dos diagramas do modelo da implementação. O código é produzido formando uma estrutura interna do agente escrito na linguagem JAVA, incluindo as subclasses das tarefas. Além disso, PTK permite a engenharia reversa pela criação de um diagrama de definição da estrutura do agente no modelo *Rational Rose* a partir de um arquivo fonte em Java. Durante este procedimento, a ferramenta atualiza todos os diagramas relacionados.

5.1 Fases da metodologia PASSI para o desenvolvimento do sistema de monitoramento e controle da qualidade de combustíveis

A primeira fase do modelo de requisito da metodologia PASSI tem como objetivo a construção de uma série hierárquica do diagrama de caso de uso para descrever a funcionalidade do sistema. Os pacotes desta fase são compostos pelo diagrama de contexto e descrição de domínio que permitem obter uma visão analítica de abstração e identificação dos agentes no sistema inteligente.

A Figura 29 mostra a descrição de domínio da PASSI no sistema do monitoramento e controle da qualidade de combustíveis.

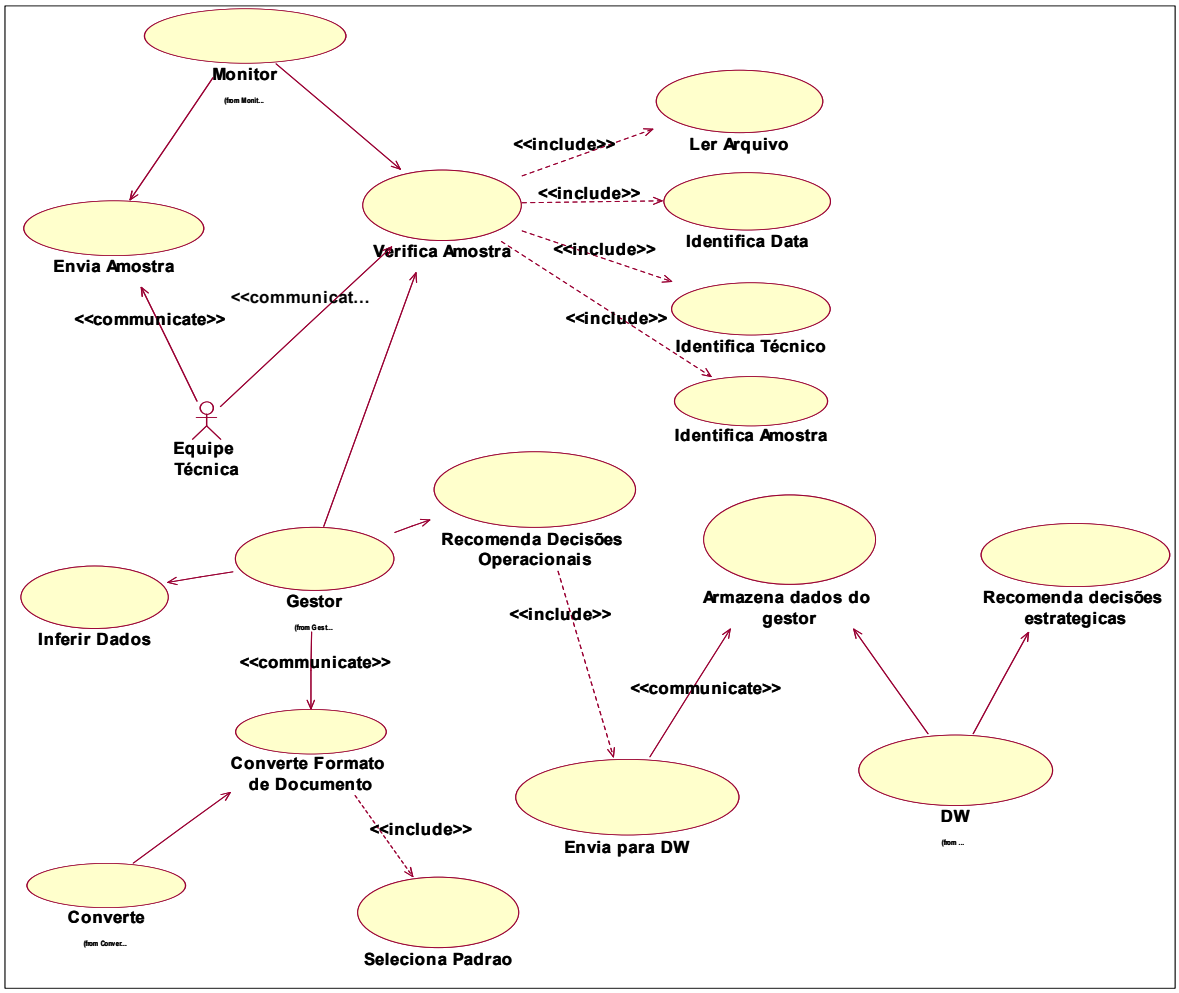

Figura 29: Descrição de domínio da PASSI no sistema do monitoramento e controle da qualidade de combustíveis.

Assim, na descrição do domínio da Passi no sistema do monitoramento e controle da qualidade de combustíveis identificamos os seguintes casos de uso, que são:

– Monitor - tem como objetivo enviar e verificar amostra;

– Gestor - tem como objetivo inferir dados, recomenda decisões operacionais, envia para o repositório de dados e converte formato de documento;

– DW - tem como objetivo armazenar dados do gestor e recomendações decisões estratégicas;

– Conversor - tem como objetivo converter dados no formato XML e selecionar padrão no formato XML.

5.1.1 Modelo de Requisitos do Sistema

A segunda fase da metodologia PASSI, no modelo de requisitos do sistema, realiza o mapeamento dos agentes mostrados na Figura 30 na descrição de domínio.

A Figura 30 mostra a identificação dos agentes no sistema proposto para o monitoramento e controle da qualidade de combustíveis.

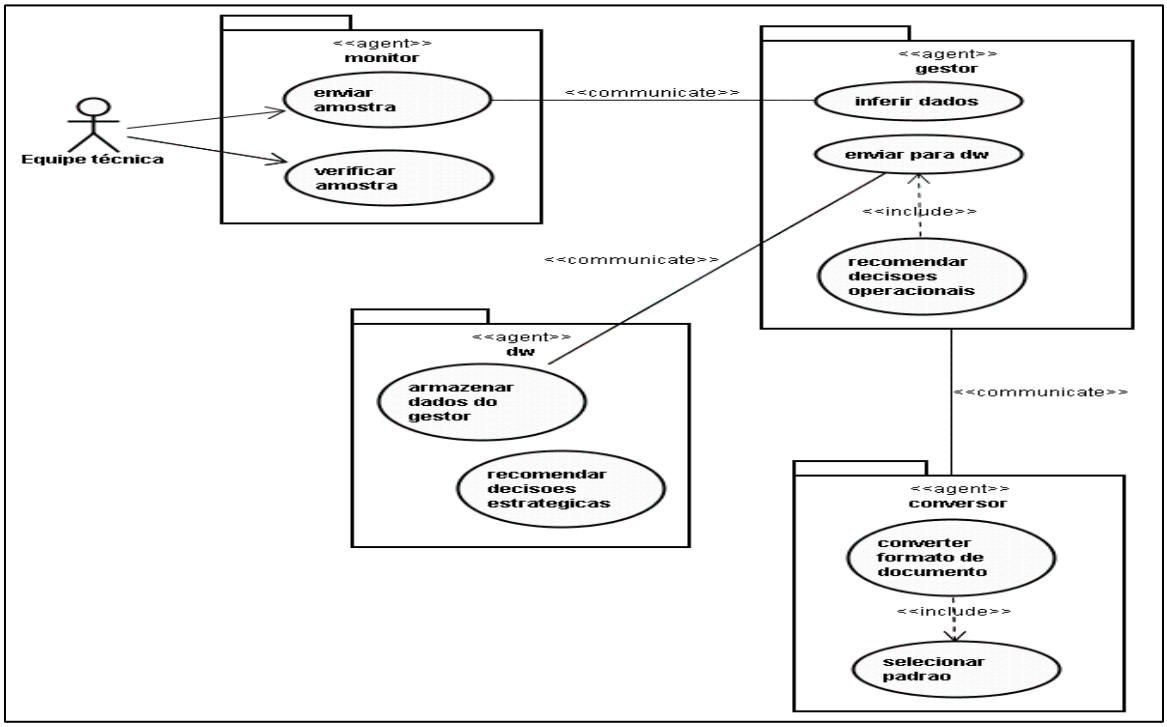

Figura 30: Identificação dos agentes no sistema proposto para o monitoramento e controle da qualidade de combustíveis

Neste diagrama, as funcionalidades dos artefatos estão agrupadas em pacotes, representando os agentes para que seja possível abstrair as suas atribuições e responsabilidades.

De acordo com Cossentino *et al* (2002), os relacionamentos identificados nos diagramas de caso de uso, classe e pacote são traduzidos para o desenvolvimento do modelo de comunicação.

Assim, as responsabilidades relacionadas com os processos decisórios estão presentes nos agentes “gestor” e “dw”. A única diferença entre os dois agentes está relacionada ao conceito de temporalidade, ou seja, enquanto o agente Gestor

solicita a tomada de decisões em tempo de execução, o agente DW utiliza o histórico do *Data Warehouse* para realizar as projeções e as estratégias para analisar os dados em intervalo de tempo.

Para o entendimento da Figura 30, destaca-se o diagrama de pacote proposto utilizado na prototipagem.

A Figura 31 apresenta diagrama de pacotes para o sistema de monitoramento e controle da qualidade de combustíveis e seus respectivos relacionamentos de dependências.

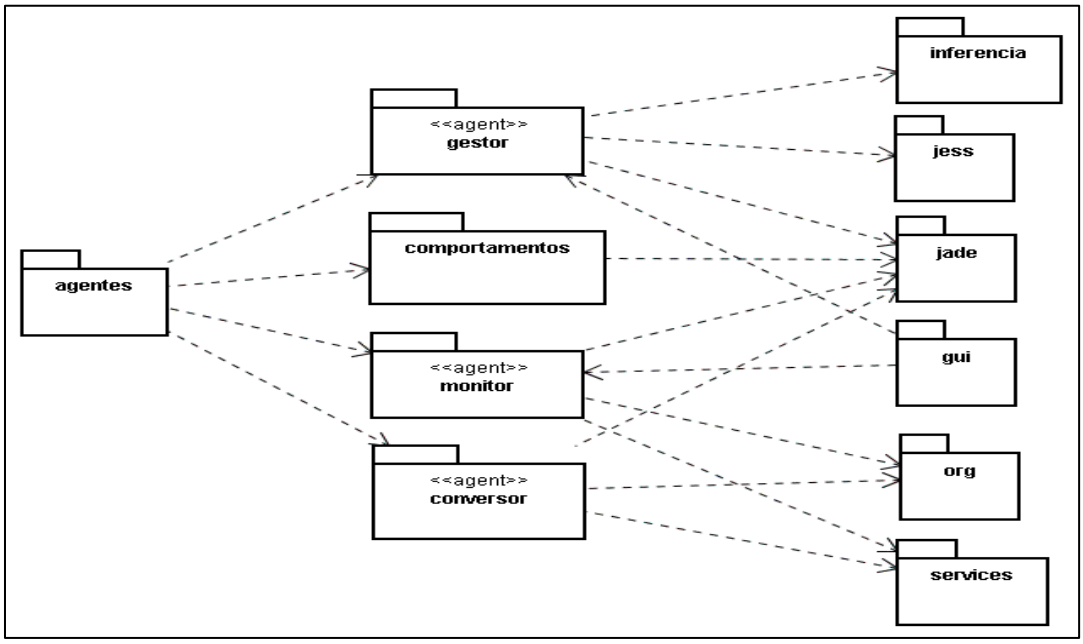

Figura 31: Diagrama de pacotes do sistema proposto para o monitoramento e controle da qualidade de combustíveis

As principais funcionalidades, em termos de dependências entre os pacotes da Figura 32, referem-se primeiramente aos pacotes “gestor” e “monitor”, onde cada um revela a presença das classes dos agentes. Além disso, são os mesmos pacotes que devem ser representados nos diagramas de domínio da metodologia PASSI, utilizados para a identificação dos agentes do sistema.

O pacote “comportamentos” agrupa as responsabilidades que cada agente assume durante suas tarefas. O pacote “inferencia” reúne rotinas necessárias para as trocas de informações entre os objetos JAVA e JESS para o mecanismo decisório do agente gestor.

Os pacotes “*jess*”, “org” e “jade” agrupam, respectivamente, as API’s (*Aplication Programming Interface*) que são operações de inferência, leitura e escrita de documentos XML (*eXtensible Marckup Language*) e criação de agentes de software. O pacote “*services*” organiza as classes de serviço como conexão a banco de dados e outras bibliotecas de código, úteis para geração de gráficos de relatórios.

5.1.2 Identificação dos Papéis

A identificação dos papéis dos agentes do sistema inteligente foi especificada utilizando o diagrama de seqüência para apresentar as interações entre os agentes e os seus papéis.

A Figura 32 mostra o diagrama de seqüência do sistema para o monitoramento e controle da qualidade de combustíveis.

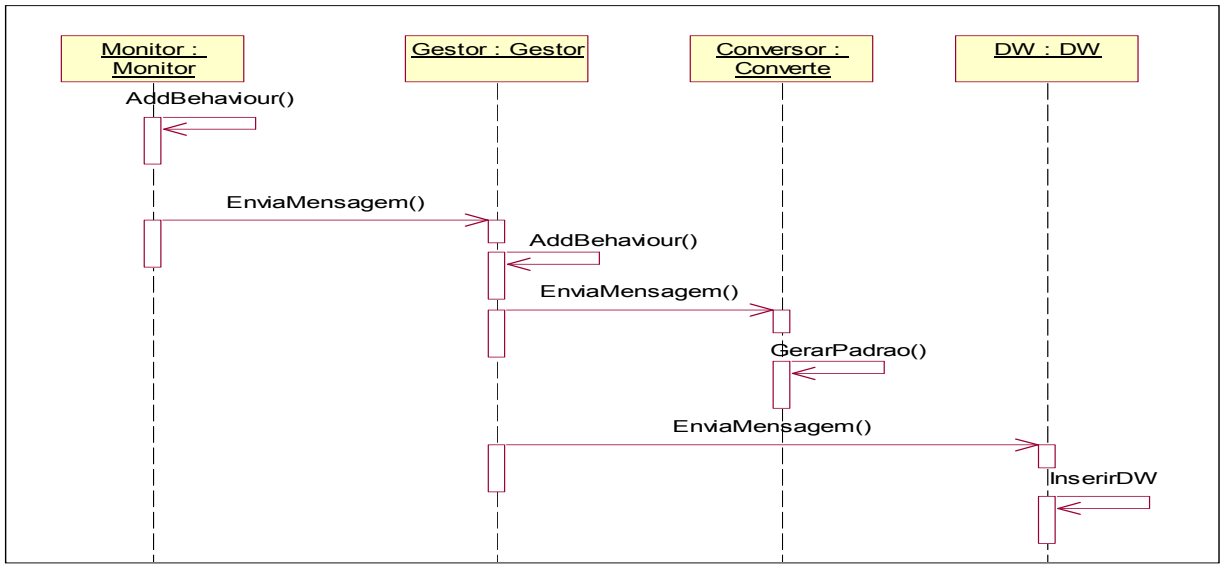

Figura 32: Diagrama de seqüência do sistema para o monitoramento e controle da qualidade de combustíveis.

Neste diagrama foram identificados os seguintes papéis do agente, que são:

– Monitor - tem como objetivo envia mensagem para o papel do agente gestor contendo dados da amostra;

– Gestor – analisa os dados da amostra e envia duas mensagens uma para papel do agente conversor e outra para o papel agente DW;

– Conversor – tem como objetivo gerar padrão no formato XML;

– DW – tem como objetivo inserir dados da amostra no repositório de dados.

5.2 Especificação das Tarefas

Nesta fase é realizada a especificação das tarefas, utilizando o diagrama de atividade para descrever como os agentes podem realizar as tarefas no sistema para atingir seus planos e objetivos.

A Figura 33 apresenta o digrama de atividade para especificar as tarefas dos agentes no modelo de requisitos do sistema.

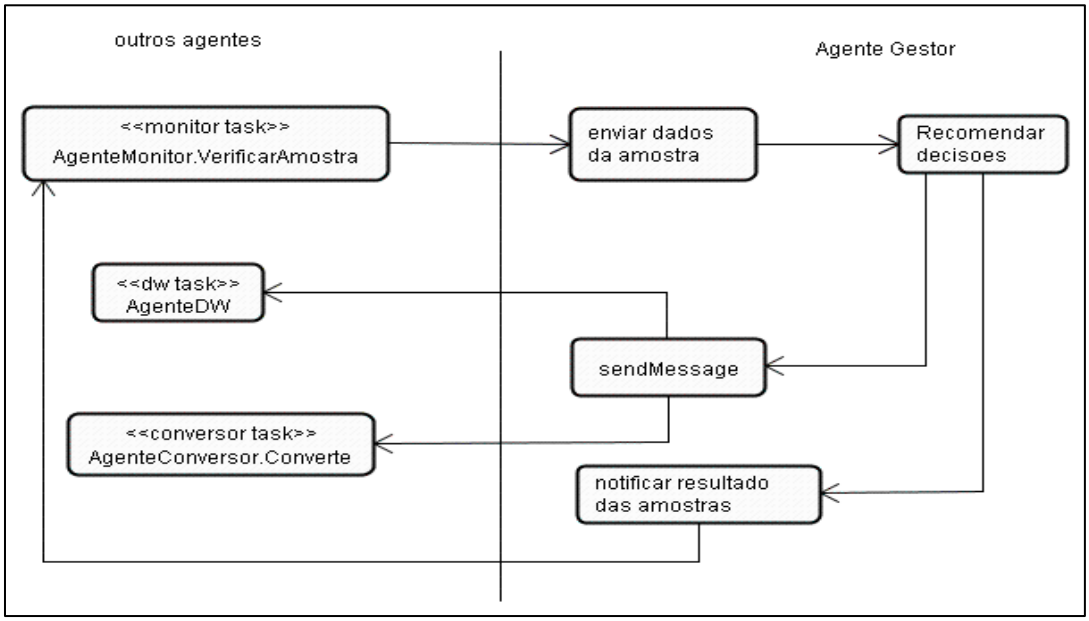

Figura 33: Diagrama de atividade para especificar as tarefas dos agentes no modelo de requisitos do sistema

As tarefas dos agentes foram estabelecidas com o propósito de identificar as atividades necessárias para que cada agente possa alcançar seus objetivos seguindo um conjunto de planos.

Neste cenário, as tarefas do “AgenteGestor” foram representadas através de um conjunto de interações com os outros agentes participantes, onde estas

interações começam quando o “AgenteMonitor” envia uma mensagem para o “AgenteGestor” contendo os dados da amostra a ser analisada. Neste instante é realizada a verificação das conformidades pelo “AgenteGestor” que examina os marcadores de temperatura e ponto de ebulição do tipo de combustível.

Após verificar os dados da amostra, o “AgenteGestor” notifica o resultado do processo para o “AgenteMonitor” que envia os dados examinados naquele ciclo para o “AgenteDW”, com o propósito de incrementar a base de dados histórica. Além disso, o “AgenteGestor” também envia os dados inferidos, ou seja, faz a notificação dos resultados para o “AgenteConversor” para padronizar o documento XML, visando a exportação em outros formatos.

Segundo Burrafalo *et al* (2002), a finalidade do modelo da sociedade de agentes é descrever as soluções envolvidas com cada agente em termos de interações sociais, papéis, dependências e o conhecimento necessário para o compartilhamento de informações e reuso.

Na fase de projeto foi elaborado o modelo da sociedade de agentes para facilitar o entendimento das interações e potenciais dependências dos agentes “*AgenteMonitor*”, “*AgenteGestor*”, “*AgenteDW*” e “*AgenteConversor*”.

A Figura 34 apresenta a descrição dos papéis dos agentes no modelo da sociedade de agentes.

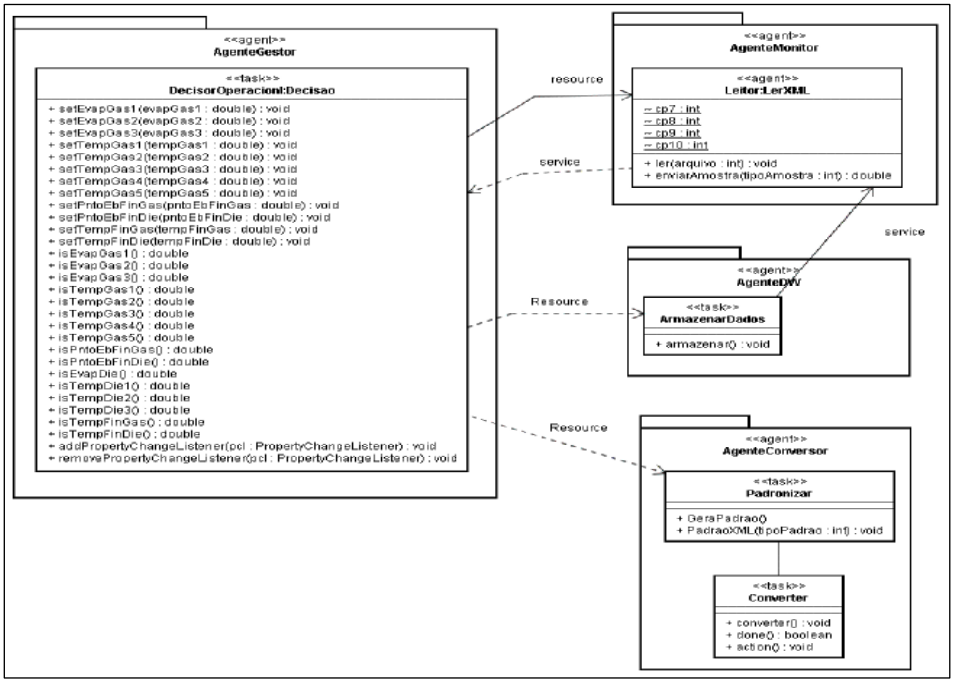

Figura 34: Descrição dos papéis dos agentes no modelo da sociedade de agentes.

Cada classe presente nos pacotes representa os papéis que os agentes assumem. O “AgenteGestor” e o “AgenteDW” representam papéis decisórios no sistema. O “AgenteMonitor” atua como um leitor de documentos XML, provenientes do destilador.

O “*AgenteConversor*” possui dois papéis, que são: recebe os dados da amostra examinados pelo “*AgenteGestor*” e transformar estes resultados em um documento XML.

O Quadro 5 mostra o código do documento XML gerado pelo AgenteConversor.

```
1. <LML>
2. −<amostra id="12345">
3. <resultado codimp="ETANOL" valor="26.7"/>
4. ...
5. </amostra>
6. </LML>
```

Quadro 5: Código do documento XML gerado pelo AgenteConversor.

O documento convertido contém o identificador da amostra, o resultado e os atributos que identificam o combustível e o seu respectivo valor. Esse é o formato padrão do documento XML que o “*AgenteConversor*” cria, mas, além dele, um outro formato pode ser gerado e, por esse motivo, selecionou-se o papel de padronizador ao “AgenteConversor”.

O Quadro 6 mostra código documento XML gerado pelo *AgenteConversor* em outro formato.

```
1. <LML>
2. −<amostra id="12345">
3. <resultado>
4. <codimp>
5. ETANOL
6. </codimp>
7. <valor>
8. 26.7
9. <valor/>
10. <codimp>
11. ...
12. </resultado>
13. </amostra>
14. </LML>
```

Quadro 6: Código do documento XML gerado pelo AgenteConversor em outro formato.

A diferença entre os documentos está na disposição dos elementos pais, elementos filhos e atributos de cada tag. Por exemplo, no primeiro trecho de código, a tag resultado é estruturada com a inclusão de vários atributos. No segundo trecho de código, os atributos constituem elementos filhos da tag resultado que, por sua vez, é elemento filho da tag amostra.

5.3 Modelo de Implementação de Agentes

Para Cossetino (2002), as técnicas da metodologia PASSI utilizadas no modelo de implementação do sistema tratarão as classes dos agentes do sistema. O objetivo desta fase é transformar os conceitos abstratos em conceitos concretos.

A Figura 35 apresenta o diagrama de classe do sistema no modelo de implementação.

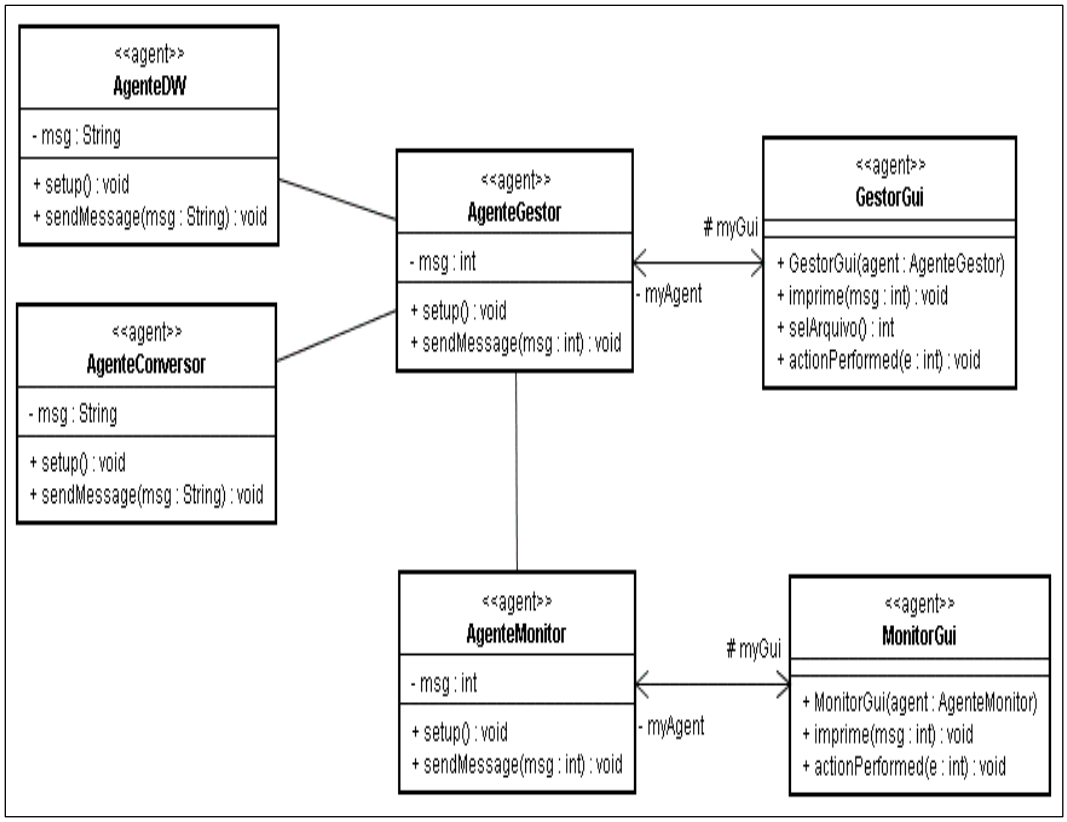

Figura 35: Diagrama de classe do sistema no modelo de implementação.

Após a identificação das classes dos agentes, as tarefas puderam ser elaboradas para o desenvolvimento da definição da estrutura dos principais agentes.

A Figura 36 apresenta o diagrama de tarefas dos agentes no modelo de implementação.

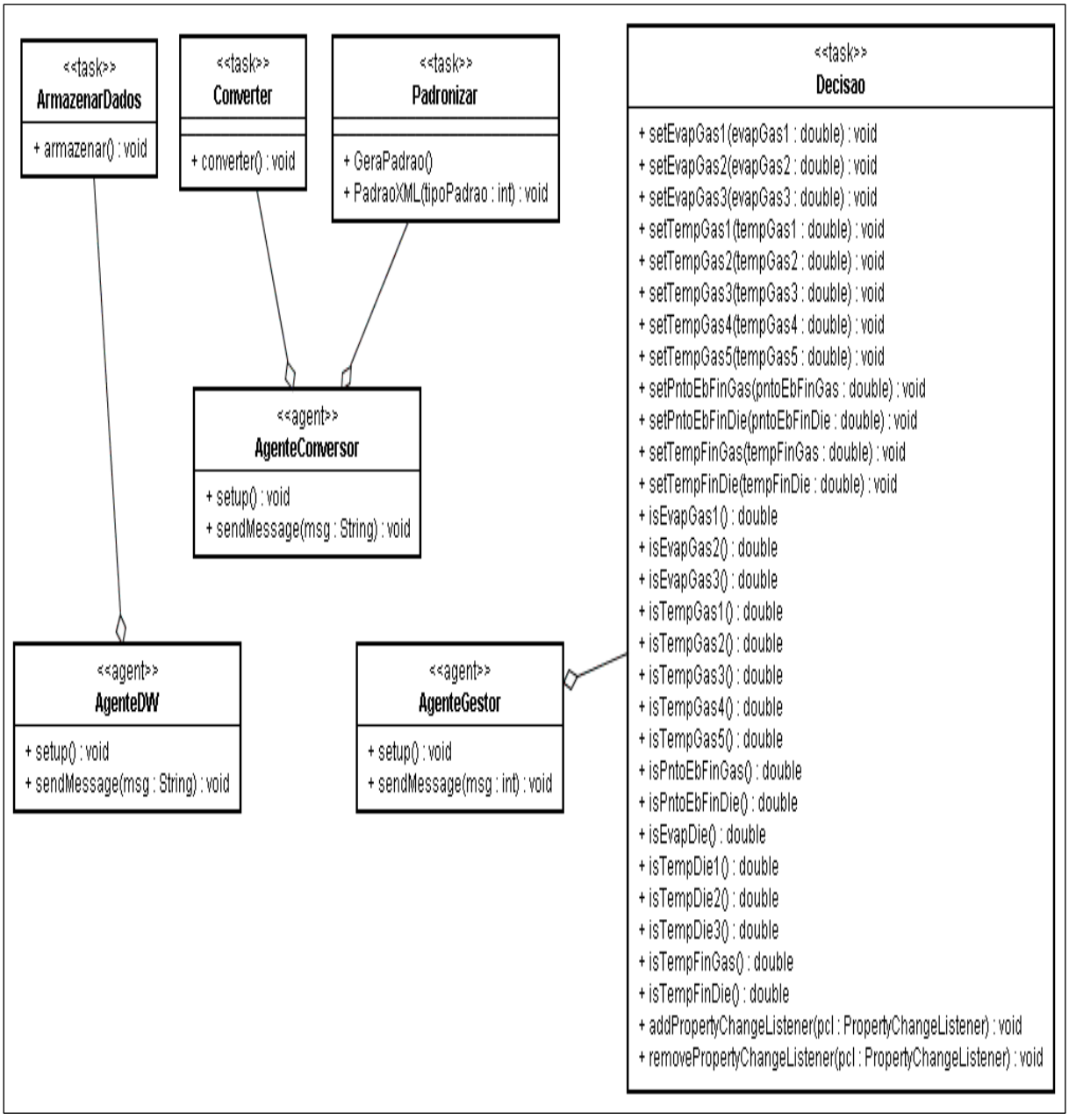

Figura 36: Diagrama de tarefas dos agentes no modelo de implementação.

Os agentes “AgenteGestor”, “AgenteDW” e “AgenteConversor” possuem tarefas agregadas, que são mostradas como relacionamentos de agregação. Essa decisão de projeto foi tomada levando-se em consideração a relevância deste tipo de associação para alcançar os objetivos do sistema. Assim, outros artefatos relevantes estão associados aos comportamentos dos agentes do sistema. Estes se

apresentam como resultantes dos mapeamentos das responsabilidades para as classes de comportamentos do *framework* JADE.

A Figura 37 apresenta o diagrama de comportamento dos *AgenteGestor* e *AgenteMonitor* no modelo de implementação.

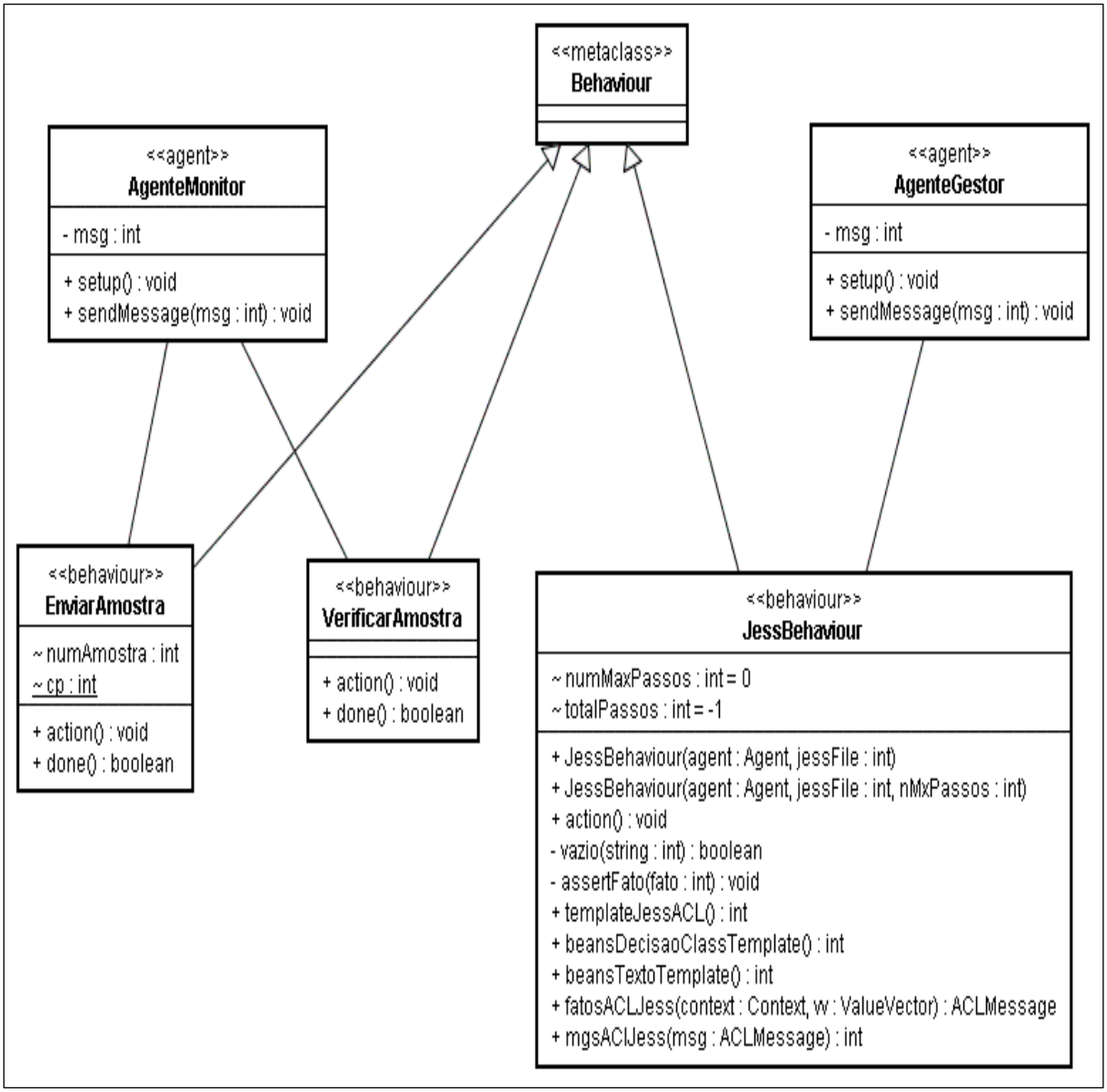

Figura 37: Diagrama de comportamento dos AgenteGestor e AgenteMonitor no modelo de implementação.

Em seguida, as operações fundamentais de cada classe da Figura 38 foram analisadas em unidades, visando atender as especificações de projeto detalhado dos principais artefatos dos agentes “AgenteMonitor” e “AgenteGestor”.

O Quadro 7 mostra a estrutura do código do AgenteMonitor.

```
1.  package agentes.monitor.
2.  import agentes.comportamentos.*.
3.  import jade.core.*.
4.  import jade.lang.acl.ACLMessage.
5.  import gui.*.
6.
7.  public class AgenteMonitor extends Agent{
8.
9.          private String msg.
10.         protected MonitorGui myGui.
11.         /** inicializa o comportamento do agente*/
12.         public void setup(){
13.                 /* configura a interface gráfica (i.e. gui) */
14.          myGui = new MonitorGui(this).
15.                 /* reponsabilidade/comportamento do agente */
16.                 addBehaviour(new VerificarAmostra()).
17.                 addBehaviour(new EnviarAmostra()).
18.          //argumentos passados via linha de comando
19.                 Object[] args = getArguments().
20.                 if (args != null && args.length>0){
21.                         msg = (String) args[0].
22.                 }else{
23.                         System.out.println("Nenhuma mensagem foi especificada
    para o agente "+this.getLocalName()).
24.                         doDelete().
25.                 }
26.                 //getLocalName() retornará o nome do agente
27.                 System.out.println("Agente:"+this.getLocalName()).
28.                 sendMessage(msg).
29.         }
30.         /**Envia as mensagens
31.          * @param
32.          * String msg mensagem a ser enviada para um agente*/
33.         public void sendMessage(String msg) {
34.
35.          ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST).
36.          aclMessage.addReceiver(new AID("gestor",AID.ISLOCALNAME)).
37.          aclMessage.setContent(msg).
38.          this.send(aclMessage). }}
```

Quadro 7: Estrutura do código do AgenteMonitor

A estrutura do código do “AgenteMonitor” e os comportamentos “VerificarAmostra” e “EnviarAmostra” estão implementados no (anexo A) para a especificação de suas capacidades durante o ciclo de execução do sistema. Estes comportamentos fornecem características reativas ao AgenteMonitor.

Para Russel (2004), a característica reativa não infere nenhum resultado ou passa por processos de deliberação, diferente do que ocorre com o AgenteGestor.

O Quadro 8 mostra a estrutura do código do AgenteGestor.

```
1.  package agentes.gestor.
2.  import jade.core.*.
3.  import jade.lang.acl.ACLMessage.
4.  import gui.*.
5.  import agentes.comportamentos.*.
6.  public class AgenteGestor extends Agent{
7.  private String msg.
8.  protected GestorGui myGui.
9.  /** inicializa o comportamento do agente*/
10. public void setup(){/* configura a interface gráfica (i.e. gui) */
11. myGui = new GestorGui(this).  /* reponsabilidade do agente */
12. addBehaviour(new JessBehaviour(this,myGui.selArquivo(),1)).
13. myGui.setVisible(true).
14. //argumentos passados via linha de comando
15. Object[] args = getArguments().
16. if (args != null && args.length>0){
17. msg = (String) args[0].
18. }else{
19. System.out.println("Nenhuma mensagem foi especificada para o agente
    "+this.getLocalName()).
20. doDelete().}
21. //getLocalName() retornará o nome do agente
22. System.out.println("Agente:"+this.getLocalName()).
23. sendMessage(msg).}
24. /**Envia as mensagens
25. @param
26. String msg mensagem a ser enviada para um agente*/
27. public void sendMessage(String msg) {
28. ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST).
29. aclMessage.addReceiver(new AID("monitor",AID.ISLOCALNAME)).
30. aclMessage.setContent(msg).
31. this.send(aclMessage).}}
```

Quadro 8: Estrutura do código do AgenteGestor

No Quadro 8 observa-se que o “AgenteGestor” atua como um agente cognitivo, ou seja, no seu comportamento, implementou-se um motor de inferência que é responsável pela dedução das amostras que chegam até ele por meio do “AgenteMonitor”. Para atingir esta meta, o “AgenteGestor” faz uma chamada ao motor de inferência do JESS, denominado JessBehaviour, que contém o código JESS conforme o (anexo B) para a inicialização e execução da máquina de inferência.

Segundo Friedman (2007), o JESS permite a criação de inferências para atualizar os comportamentos do AgenteGestor. O Quadro 9 mostra a decodificação das regras para o módulo decisório do AgenteGestor.

Linguagem Natural		Linguagem JESS	
Premissas	**Conclusões**	**Premissas**	**Conclusões**
Linguagem natural	**Linguagem natural**	**Linguagem JESS**	**Linguagem JESS**
1. A amostra apresenta evaporação (i.e. destilação) de **10**% e temperatura acima de 65ºC (Gasolina).	Esta amostra apresenta indício de contaminação por material **pesado** (ex. solvente com ponto de ebulição alto, óleo diesel ou teor de álcool alto).	(defrule regra1 (test (< ?rmt (amostra isEvapGas1))) (test (< ?rmt (amostra isTempGas1)))	(enviaMSG "Amostra com indício de contaminação por material pesado (ex. solvente com ponto de ebulição alto, óleo diesel ou teor de álcool alto).") (store RESULT1 "Amostra com indício de contaminação por material pesado (ex. solvente com ponto de ebulição alto, óleo diesel ou teor de álcool alto).")
2. A amostra apresenta evaporação de **50**% e temperatura acima de 80ºC (Gasolina).	Neste caso, a amostra apresenta indício de contaminação por material **pesado** (ex. solvente com ponto ebulição alto, óleo diesel ou teor de álcool alto).	(defrule regra2 (test (< ?rmt (amostra isEvapGas2))) (test (< ?rmt (amostra isTempGas2)))	(enviaMSG "Amostra com indício de contaminação por material pesado (ex. Solvente com ponto ebulição alto, óleo diesel ou teor de álcool alto).") (store RESULT2 "Amostra com indício de contaminação por material pesado (ex. Solvente com ponto ebulição alto, óleo diesel ou teor de álcool alto).")
3. A amostra apresenta evaporação de **90**% e temperatura abaixo de 145ºC (Gasolina).	A amostra apresenta indício de contaminação por material **leve**.	(defrule regra3 (test (< ?rmt (amostra isEvapGas3))) (test (< ?rmt (amostra isTempGas3)))	(enviaMSG "Amostra com indício de contaminação por material leve.")(store RESULT3 "A amostra apresenta indício de contaminação por material leve.")
4. A amostra apresenta evaporação de 90% e temperatura acima de 190ºC (Gasolina).	Caracteriza-se pelo indício de contaminação por material **pesado**, ou seja, substância com elevado ponto de ebulição.	(defrule regra4 (test (< ?rmt (amostra isEvapGas4))) (test (< ?rmt (amostra isTempGas4)))	(enviaMSG "Amostra com indício de contaminação por material pesado. Substância com elevado ponto de ebulição.") (store RESULT4 "Amostra com indício de contaminação por material pesado. Substância com elevado ponto de ebulição.")
5. A amostra apresenta evaporação de **Ponto de Ebulição Final** e temperatura acima de 220ºC. (Gasolina).	Caracteriza-se pelo indício de contaminação por material **pesado**, ou seja, substância com elevado ponto de ebulição.	(defrule regra5 (test (> ?rmt (amostra isPntoEbFinGas))) (test (> ?rmt (amostra isTempFinGas)))	(enviaMSG "Amostra com indício de contaminação por material pesado. Substância com elevado ponto de ebulição.") (store RESULT5 "Amostra com indício de contaminação por material pesado. Substância com elevado ponto de ebulição.")
6. A amostra apresenta evaporação de 50% e temperatura abaixo de 245ºC. (Óleo Diesel)	Caracteriza-se pela presença de contaminante leve (ex. querosene).	(defrule regra6 (test (> ?rmt (amostra isTempDie1))) (test (> ?rmt (amostra isEvapDie1)))	(enviaMSG "Amostra com indício de contaminante leve (ex. querosene)") (store RESULT6 "Amostra com indício de contaminante leve (ex. querosene)")
7. A amostra apresenta evaporação de 50% e temperatura acima de 310ºC. (Óleo Diesel).	Caracteriza-se pela presença de contaminante **leve** (ex. solvente em suspensão ou solvente de borracha).	(defrule regra7(test (> ?rmt (amostra isTempDie2))) (test (> ?rmt (amostra isEvapDie2)))	(enviaMSG "Amostra com indício de contaminante leve (ex. solvente em suspensão ou solvente de borracha)") (store RESULT7 "Amostra com indício de contaminante leve (ex. solvente em suspensão ou solvente de borracha)").

Quadro 9: Decodificação das regras para o módulo decisório do AgenteGestor.

No Quadro 9, observa-se que a decodificação das regras de linguagem natural para a linguagem JESS é realizada através do mapeamento das regras descritas no (anexo B) para a identificação de não conformidade dos combustíveis, bem como apresenta os componentes que contribuem para o módulo decisório do AgenteGestor.

As principais instruções presentes na linguagem JESS são: *test*, enviaMSG, amostra e *store*. A instrução *test* verifica o resultado da função amostra e compara o conteúdo da mensagem enviada por um agente monitor armazenado na variável '?rmt'– com o valor de retorno dos métodos 'set' e 'is' implementados na classe 'Decisao'.

Caso a comparação atenda aos padrões, ou seja, às premissas do JESS, a regra é disparada e a função enviaMSG transmite a mensagem para o agente monitor, no mesmo instante que, por meio do comando *store*, o conteúdo desta recomendação pode ser armazenado e recuperado no ambiente DW.

Os métodos amostra e enviaMSG foram desenvolvidos para solucionar os problemas de mensagens dos agentes. Os métodos *test* e *store* são nativos da linguagem JESS.

5.4 Comunicação entre o agente Gestor e Monitor

Os agentes gestor e monitor foram instanciados na plataforma JADE utilizando o RMA para estabelecer a comunicação entre esses agentes.

A Figura 38 apresenta os agentes gestor e monitor na plataforma JADE.

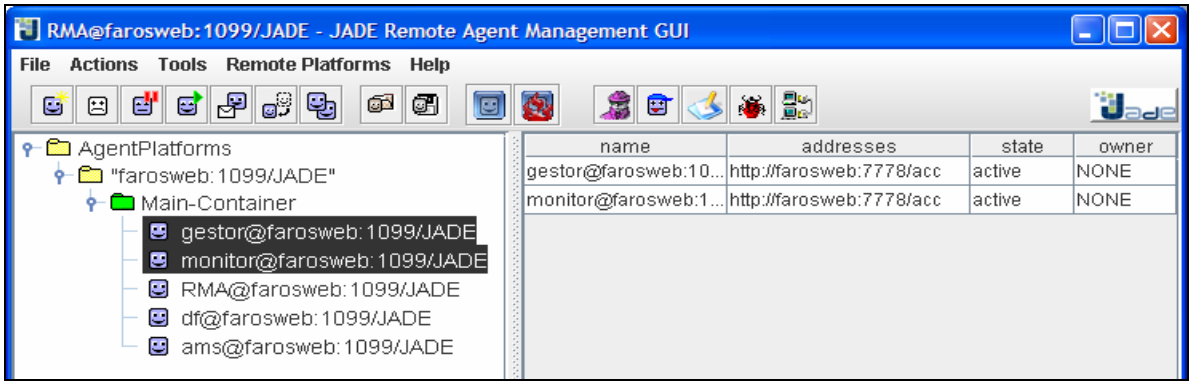

Figura 38: Agente gestor e monitor na plataforma JADE

A plataforma JADE apresenta a performativa de comunicação, visando o monitoramento das requisições das informações que os agentes gestor e monitor necessitam durante a execução de suas atividades.

A Figura 39 mostra o processo de trocas de mensagens e intenções entre estes agentes.

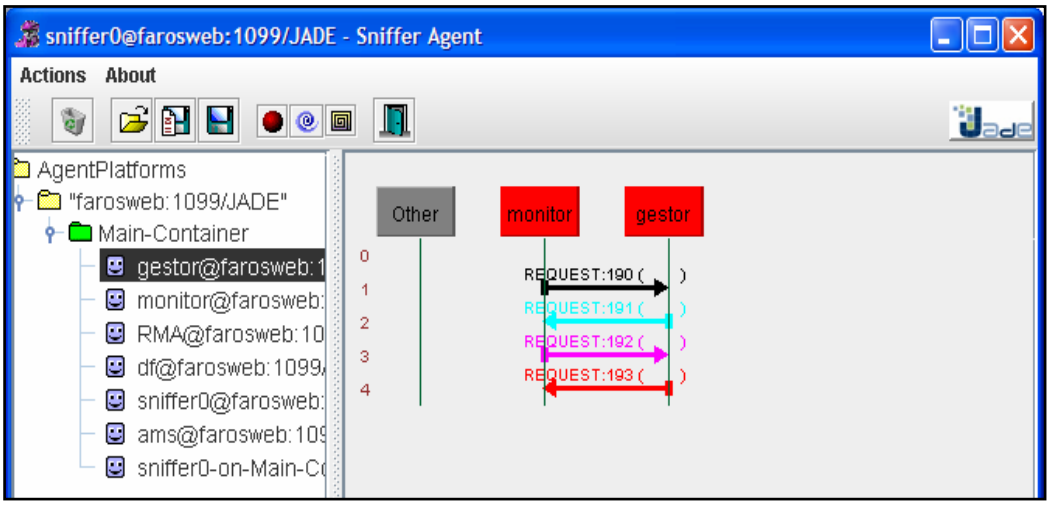

Figura 39: Trocas de mensagens entre os agentes ‘gestor’ e ‘monitor’ no ambiente JADE.

O Quadro 10 mostra o conteúdo das mensagens resultante das interações entre os agentes de acordo com padrão FIPA.

1. 15/04/08 13:52
2. (REQUEST :sender (agent-identifier :name gestor@farosweb:1099/JADE :addresses (sequence http://farosweb:7778/acc))
3. :receiver (set (agent-identifier :name monitor@farosweb:1099/JADE))
4. :content "Amostra com indicio de contaminacao por material leve"
5. :protocol fipa-request)
6. 15/04/08 13:52
7. (REQUEST :sender (agent-identifier :name monitor@farosweb:1099/JADE :addresses (sequence http://farosweb:7778/acc))
8. :receiver (set (agent-identifier :name gestor@farosweb:1099/JADE))
9. :content "90-145"
10. :protocol fipa-request)
11. 15/04/08 13:51
12. (REQUEST :sender (agent-identifier :name gestor@farosweb:1099/JADE :addresses (sequence http://farosweb:7778/acc))
13. :receiver (set (agent-identifier :name monitor@farosweb:1099/JADE))
14. :content "Amostra com indicio de contaminacao por material pesado (ex. solvente com ponto de ebulicao alto, oleo diesel ou teor de alcool alto)."
15. :protocol fipa-request)
16. 15/04/08 13:49
17. (REQUEST :sender (agent-identifier :name monitor@farosweb:1099/JADE :addresses (sequence http://farosweb:7778/acc))
18. :receiver (set (agent-identifier :name gestor@farosweb:1099/JADE))
19. :content "10-65"
20. :protocol fipa-request)

Quadro 10: Interações entre os agentes de acordo com padrão FIPA

No quadro 10, observa-se que o conteúdo da mensagem do campo *content* está presente também em cada mensagem de *request*. Como se trata de uma pilha, a última mensagem fica no topo. O quadro 10 mostra que o agente monitor envia a mensagem com o conteúdo igual a 10-65, que indica amostra com evaporação de 10% e temperatura de 65ºC, para o agente 'gestor' e este, por sua vez, infere os resultados recebidos como fatos na sua memória cognitiva, ou seja, a memória de trabalho, e deduz o resultado, retornando-o para o agente 'monitor' com a seguinte recomendação: "Amostra com indício de contaminação por material pesado (ex. solvente com ponto de ebulição alto, óleo diesel ou teor de álcool alto)".

5.5 Conclusões

Neste capítulo foi desenvolvida a prototipagem do sistema inteligente no monitoramento e controle da qualidade de combustíveis, utilizando a metodologia PASSI , a máquina de inferência JESS e a plataforma JADE.

A metodologia PASSI foi utilizada para modelar os requisitos do sistema e a identificar os papéis dos agentes.

O motor de inferência JESS auxiliou a tomada de deicisões pelo agente inteligente gestor.

A plataforma JADE foi utilizada para desenvolver o sistema inteligente para o monitoramento e controle da qualidade de combustíveis.

6 CONCLUSÃO

A necessidade para automatizar as fases de tratamento e análise química do Laboratório de Análise e Pesquisa em Química Analítica do Petróleo da Universidade Federal do Maranhão, além de padronizar o armazenamento e recuperação dos dados e a geração de relatórios gerenciais para tomada de decisões motivaram a realização desta pesquisa. Nesta perspectiva, a utilização de sistemas inteligentes para criar mecanismos de inferência e suporte à tomada de decisões e a tecnologia Data Warehouse apresentaram-se como elementos potencializadores para desenvolver o sistema inteligente com o intuito de realizar o monitoramento e controle da qualidade de combustíveis no estado do Maranhão.

Neste trabalho, foi necessário estruturar a realização da pesquisa em diversas fases que precisavam ser executadas para realizar a fundamentação teórica e implementar o protótipo do sistema. Apresenta-se, a seguir, os conceitos que foram utilizados:

- Elaboração da modelagem do sistema utilizando a metodologia PASSI, juntamente com a plataforma JADE e a linguagem JESS;
- Integração das bases de dados e suporte para relacionar dado ou conjunto de dados não estruturados utilizando Data Warehouse (DW).

A justificativa para a escolha da plataforma de agente JADE para implementar o sistema foi:

- Disponibilização de uma Interface Gráfica;
- Utilização do Padrão FIPA-ACL;
- Padronização nos formatos das mensagens;
- Arquitetura Distribuída;
- Padronização dos protocolos de comunicação.

A utilização da plataforma JADE para o desenvolvimento de agentes provê um ambiente para agilizar a elaboração dos códigos dos agentes. Além disso, na pesquisa, foi necessário implementar um motor de inferência, utilizando a tecnologia JESS para desenvolver o comportamento deliberativo do agente ‘gestor’.

As contribuições para o modelo proposto do Sistema Inteligente de Monitoramento e Controle da Qualidade de Combustível foram:

– Redução do tempo para a tomada de decisões por parte dos técnicos do laboratório da UFMA com a inclusão de um motor de inferência no sistema inteligente proposto;

– Ampliação das potencialidades apresentadas pelo sistema, através da redução do processamento das operações que possibilitaram armazenar o conhecimento dos especialistas do domínio no Agente Gestor, além de manter este conhecimento de forma independente;

– Armazenamento do conhecimento em uma entidade de software de forma inteligente, ajudando a reduzir os impactos gerados quando um técnico ou funcionário ausenta-se do laboratório ou empresa, levando consigo conhecimentos e experiências acumuladas;

– Utilização de fundamentos de Sistema Multiagente visando à aplicação de técnicas para a construção de sistemas orientados a agentes, pois a metodologia PASSI não reflete, na sua totalidade, os aspectos da Engenharia do Conhecimento, unificando conceitos de orientação a objetos com o vocabulário de agentes e componentes notacionais baseados em UML.

Neste sentido, as tecnologias utilizadas no desenvolvimento do sistema inteligente projetam, na prática, os seguintes conceitos:

- A interoperabilidade foi contemplada com o uso do padrão XML na leitura e integração com o sistema legado destilador;
- Na integração, aproveitou-se uma interface para gerar os arquivos em XML que serve de entrada para o agente ‘monitor’. A leitura é rápida e não influenciou em tempo de resposta, mesmo considerando o sistema *multiagente* como uma aplicação *multithreading;*

- A portabilidade do sistema foi viabilizada utilizando a linguagem JAVA, pois apresenta o sistema *multiagente* para ser adaptado a outras aplicações para análise das amostras. Assim, o sistema inteligente pode ser conectado a uma base

de dados para coletar as informações para serem enviadas ao agente ‘monitor’, utilizando o formato XML.

O desenvolvimento do sistema inteligente proposto automatizou as fases de tratamento e análise química, proporcionando a padronização dos dados e geração de relatórios gerenciais para a tomada de decisões.

Além disso, convém ressaltar as dimensões que esta pesquisa alcançou, uma vez que os resultados, nela obtidos, foram aceitos nas seguintes conferências:

- *IADIS Multi Conference on Computer Science and Information Systems* 2008 *Amsterdam, Netherlands* 22 – 27 *July* 2008.
- *Eighth International Conference on Hybrid Intelligent Systems– HIS2008*, *Barcelona, Spain September* 10-12th, 2008

6.1 Trabalhos Futuros

Como perspectivas futuras e complementares ao desenvolvimento desta pesquisa, propõe-se que as seguintes extensões possam surgir:

– Desenvolvimento de algoritmos de roteamento no sistema para auxiliar o processo de logística, a fim de determinar a melhor rota para coletar as amostras dos combustíveis;

– Implementação do Data Mining;

– Integração do sistema de monitoramento de combustível com a Internet.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 14724:** informação e documentação; citações em documentos; apresentação. Rio de Janeiro, 2005.

_____. **NBR 10520:** informação e documentação; citações em documentos; apresentação. Rio de Janeiro, 2002.

_____. **NBR 6023:** informação e documentação; referências; elaboração. Rio de Janeiro, 2002.

BELLIFEMINE, Fabio; et al. **JADE Administrator's Guide**. Disponível em: <http://jade.tilab.com/doc/administratorsguide.pdf>. Acesso em: 01 dez. 2007.

BELLIFEMINE, Fabio; et al. **Developing multi-agent systems with JADE**. John Wiley & Sons Ltd., 2007.

BEZERRA, Eduardo. **Princípios de Análise e Projeto de Sistemas com UML**. Rio de Janeiro: Elsevier, 2002.

BOGHI, Cláudio; SHITSUKA, Ricardo. **Sistemas de Informação:** um enfoque dinâmico. São Paulo: Érica, 2002.

BRATMAN, M. **Intention, Plans and Practical Reason**. Harvard University Press, Cambridge, 1987.

BRASIL. **Ministério das Minas e Energia. Agência Nacional do Petróleo. Gás Natural e Biocombustíveis**. Disponível em: <http://www.anp.gov.br>. Acesso em: 10 dez. 2007.

BURRAFATO, P;COSSENTINO, M. **Designing a multi-agent solution for a bookstore with the PASSI methodology**. Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002). Toronto, Ontario Canada at CAiSE'02. May 2002, Disponível em: <http://citeseer.ist.psu.edu/burrafato02designing.html>. Acesso em: 15 set. 2007.

CAIRE, Giovanni; et al. **Agent Oriented Analysis using MESSAGE/UML**. Proceedings of Second International Workshop on Agent-oriented Software Engineering (AOSE 2001). Lecture Notes in Computer Science, Springer-Verlag GmbH, ISSN 0302-9743, V. 2222, p. 119. Montreal, Canada. May, 2001. Disponível em: <http://www.springerlink.com/content/txtcll5fl4aypmn58>. Acesso em: 15 set. 2007.

COSSENTINO,M; POTTS,C. **PASSI:** a process for specifying and implementing multi-agent systems using UML, 2002. Disponível em: <citeseer.ist.psu.edu/668818.html >. Acesso em: 13 jun. 2007.

COULOURIS, George; et al. **Sistemas Distribuídos:** conceitos e projeto. Porto Alegre: Bookman, 2007.

DAM, K.; WINIKOFF, M. **Comparing Agent-Oriented Methodologie**s, In: Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS), Melbourne, Australia, july, 2003. Disponível em: < http://citeseer.ist.psu.edu/dam03comparing.html >. Acesso em: 13 jun. 2007.

DATE, C.J. **Introdução a Sistemas de Banco de Dados**. ed. Rio de Janeiro: Campus, 2003.

DELOACH, Scott A. **Analysis and Design using MaSE and agentTool**. 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001) Miami University, Oxford, Ohio. Air Force Institute of Technology, 2001. Disponível em: < http:// citeseer.ist.psu.edu/deloach01analysis.html>. Acesso em: 10 jan. 2007.

FININ, Tim; LABROU, Yannis; MAYFIELD, James. **KQML as an agent communication language**. Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM}'94, p. 456-463, 1994. Disponível em: < http://citeseer.ist.psu.edu/article/finin95kqml.html>. Acesso em 18 jun. 2007.

FIPA. Foundation for Intelligent Physical Agents. Disponível em: <http://www.fipa.org/>. Acesso em: 12 jan. 2007.

FIPA. **FIPA Agent Management Specification**. Disponível em: <http://www.fipa.org/specs/fipa00023/SC00023J.html>. Acesso em: 03 jan. 2007.

FISHER, Michael. **A Survey of Concurrent METATEM The Language and its Applications**. Temporal Logic - Proceedings of the First International Conference (LNAI Volume 827), Springer-Verlag: Heidelberg, Germany. P. 480-505, 1994. Disponível em: < http://citeseer.ist.psu.edu/fisher94survey.html" >. Acesso em: 03 fev. 2007

FRIEDMAN, Hill. **JESS The Rule Engine for the Java Plataform - Language Reference – version 7.0p1** DRAFT, Sandia National Laboratories. Livermore, CA, USA. Disponível em: < http://herzberg.ca.sandia.gov/>. Acesso em: 03 marc. 2007.

GEORGEFF, M.; LANSKY, A. **Reactive Reasoning and Planning: an Experiment with a Mobile Robot. In Proceedings of the 7th National Conference on Artificial Intelligence,** p. 677–682, Seattle, WA, 1987.

GONÇALVES, Marcio. **Extração de Dados para Data Warehouse**. São Paulo: Axcel Books, 2003.

INMON, W.H., **Building the Data Warehouse**. 4ª ed. Indianapolis: Wiley Publishing Inc., 2002.

JADE. **Java Agent Development Framework**. Disponível em: <http://jade.cselt.it/doc/programmersguide.pdf/>. Acesso em: 29 jan. 2007.

JADE. **Java Agent Development Framework**. Disponível em: <http://jade.cselt.it/>. Acesso em: 29 jan. 2007.

JENNINGS, N. R. **Specifications and implementation of a belief desire joint-intention architecture for collaborative problem solving.** Journal of Intelligent and Cooperative Information System. 2(3), p. 289-318, 1993. Disponível em: <http://citesser.ist.psu.edu/jennings93specification.html>. Acesso em: 15 março. 2007.

JENNINGS, N. R.. **On Agent-based Software Engineering**. Artificial Intelligence: 117, p. 277-296, Elsevier Press, April, 2000. Disponível em: <http://citesser.ist.psu.edu/jennings00agentbased.html>. Acesso em: 05 maio. 2007.

KIMBALL, Ralph;Ross, Margy. **The Data Warehouse Toolkit - Guia Completo para Modelagem Dimensional**. 2. ed. Rio de Janeiro: Campus, 2002.

MAES, P. **Agents that Reduce Work and Information Overload**. Commun. ACM V.37, p.30-40, jul 1994. Disponível em: <http://portal.acm.org.citation.cfm?id=176792>. Acesso em: 05 maio. 2007.

MACHADO, Felipe Nery Rodrigues. **Projeto de Data Warehouse:** uma visão multidimensional. São Paulo: Érica, 2000.

PADGHAM. L.;WINIKOFF. M. **Developing Intelligent Agent Systems:** a pratical guide.Wiley Serie in Agent Tecnology, 2004.

PASSOS, Emmanuel; GOLDSCHMIDT, Ronaldo. **Data Mining:** um guia prático. Editora Campus, Rio de Janeiro: Elsevier, 2005.

RAMALHO, José Antonio. **SQL Server 7:**Iniciação e referência. São Paulo:Makron Books, 1999.

REZENDE, Solange Oliveira. **Sistemas Inteligentes:** fundamentos e aplicações. São Paulo: Manole, 2005.

RUSSELL, Stuart; NORVIG Peter. **Inteligência Artificial.** 2. ed. Edição. Rio de Janeiro: Campus, 2004.

SELL, Denílson. **Uma arquitetura para business intelligence baseada em tecnologias semânticas para suporte a aplicações analíticas**. Tese (doutorado) - Universidade Federal de Santa Catarina. Programa de Pós-graduação em

Engenharia de Produção. Florianópolis, 2006. Disponível em <http://teses.eps.ufsc.br/defesa/pdf/13113.pdf > . Acesso em: 10 jul. 2007.

SHEHORY O.; STURM, A. **Evaluation of modeling techniques for agent-based systems**. Proceedings of the Fifth International Conference on Autonomous Agents. Montreal. Canada pp. 624-631 ACM Press.2001. Disponível em <http://portal.acm.org/citation.cfm?id=375735.376473 > . Acesso em 20 jul. 2007.

SWARTOUT, W.R.; et al. **Representing Capabilities of Problem Solving Methods**. Em Proceeding of IJCAI Workshop on Ontologies and Problem-Solving Methods,1999. <http://citeseer.ist.psu.edu/swartout99representing.html>. Acesso em: 15 jan. 2007.

SOMMERVILLE, Ian. **Engenharia de Software**. 6. ed. São Paulo: Pearson Addison Wesley, 2003.

TANENBAUM, Andrew S.; WOODHULL, Albert S. **Sistemas Operacionais:** projeto e implementação. Porto Alegre: Bookman, 2000.

WEISS G. **Multiagent systems :** a modern approach to distributed artificial intelligence. Massachusetts: MIT Press, 1999.

WOOLDRIDGE M. et al. **The Gaia methodology for agent-oriented analysis and design**. Autonomous Agents and Multi-Agent Systems, Springer Netherlands V. 3(3): p.285–312, September 2000. Disponível em <http://www.springerlink.com/content/t1p651134r076800/> . Acesso em 20 agosto. 2007.

WOOLDRIDGE, Michael. **In An Introduction to MultiAgent Svstems**. John Wiley & Sons Ltd., 2002.

ZAMBONELLI, Franco; *et a*l. **Developing Multiagent systems**: The Gaia methodology. ACM Transactions on Software Engineering and Methodology, v.12, n.3, p.317-370, 2003. Disponível em <http://citeseer.ist.psu.edu/zambonelli03developing.html> . Acesso em: 22 jul. 2007.

ANEXOS

ANEXO A: Código Fonte dos agentes

```
package agentes.gestor;
import jade.core.*;
import jade.lang.acl.ACLMessage;
import gui.*;
import agentes.comportamentos.*;

public class AgenteGestor extends Agent{
      private String msg;
      protected GestorGui myGui;
      /** inicializa o comportamento do agente*/
      public void setup(){
            /* configura a interface gráfica (i.e. gui) */
          myGui = new GestorGui(this);
            /* reponsabilidade do agente */
          addBehaviour(new JessBehaviour(this,myGui.selArquivo(),1));
            myGui.setVisible(true);

          //argumentos passados via linha de comando
            Object[] args = getArguments();
            if (args != null && args.length>0){
                  msg = (String) args[0];
            }else{
                  System.out.println("Nenhuma  mensagem  foi  especificada
para o agente "+this.getLocalName());
                  doDelete();
            }
            //getLocalName() retornará o nome do agente
            System.out.println("Agente:"+this.getLocalName());
            sendMessage(msg);
      }
      /**Envia as mensagens
       * @param
       * String msg mensagem a ser enviada para um agente*/
      public void sendMessage(String msg) {
          ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST);
          aclMessage.addReceiver(new AID("monitor",AID.ISLOCALNAME));
          aclMessage.setContent(msg);
          this.send(aclMessage);
        }
}
```

```
package agentes.monitor;
import agentes.comportamentos.*;
import jade.core.*;
import jade.lang.acl.ACLMessage;
import gui.*;

public class AgenteMonitor extends Agent{

      private String msg;
      protected MonitorGui myGui;
      /** inicializa o comportamento do agente*/
      public void setup(){
            /* configura a interface gráfica (i.e. gui) */
          myGui = new MonitorGui(this);
          myGui.setVisible(true);
            /* reponsabilidade/comportamento do agente */
            addBehaviour(new VerificarAmostra());
            addBehaviour(new EnviarAmostra());
          //argumentos passados via linha de comando
            Object[] args = getArguments();
            if (args != null && args.length>0){
                  msg = (String) args[0];
            }else{
                  System.out.println("Nenhuma   mensagem   foi   especificada
para o agente "+this.getLocalName());
                  doDelete();
            }
            //getLocalName() retornará o nome do agente
            System.out.println("Agente:"+this.getLocalName());
            sendMessage(msg);
      }
      /**Envia as mensagens
       * @param
       * String msg mensagem a ser enviada para um agente*/
      public void sendMessage(String msg) {

          ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST);
          aclMessage.addReceiver(new AID("gestor",AID.ISLOCALNAME));
          aclMessage.setContent(msg);
          this.send(aclMessage);
        }
}
```

```
package gui;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import javax.swing.*;
import java.io.Serializable;
import java.beans.*;
import agentes.monitor.AgenteMonitor;

public class MonitorGui extends JFrame implements ActionListener{
      private JFileChooser fc = new JFileChooser();
      private AgenteMonitor myAgent;
      private JButton btnEnviar = new JButton("Enviar");
      public JTextField tfMensagemA = new JTextField(20);

      public MonitorGui (AgenteMonitor agent){
            myAgent = agent;
            setTitle(myAgent.getLocalName());
        JPanel base = new JPanel();
          //registra os Listeners
        btnEnviar.addActionListener(this);
        base.add(btnEnviar);
        base.add(tfMensagemA);
        getContentPane().add(base);
          setSize(470, 350);

      }
      public void imprime(String msg){
            System.out.println(msg);
      }

      public void actionPerformed(ActionEvent e) {
          if(e.getSource() == btnEnviar){
            myAgent.sendMessage(tfMensagemA.getText());
            imprime(tfMensagemA.getText());
          } }     }

package gui;
import java.awt.event.*;
import javax.swing.*;
import agentes.gestor.AgenteGestor;

public class GestorGui extends JFrame implements ActionListener{

      private AgenteGestor myAgent;
      private JButton btnEnviar = new JButton("Enviar");
      private JTextField tfMensagem = new JTextField(20);

      public GestorGui (AgenteGestor agent){
            myAgent = agent;
            setTitle(myAgent.getLocalName());
        JPanel base = new JPanel();
          //registra os Listeners
        btnEnviar.addActionListener(this);
        base.add(btnEnviar);
        base.add(tfMensagem);
        getContentPane().add(base);
          setSize(470, 350);

      }
      public void imprime(String msg){
```

```
            System.out.println(msg);
      }
      public String selArquivo(){
            String fileName = "";
            fileName = "regras.clp";
        return fileName;
      }
      public void actionPerformed(ActionEvent e) {
          if(e.getSource() == btnEnviar){
            myAgent.sendMessage(tfMensagem.getText());
            imprime(tfMensagem.getText());
          }
      }
}


package agentes.conversor;

import gui.DWGui;
import agentes.comportamentos.*;
import jade.core.AID;
import jade.core.Agent;
import jade.lang.acl.ACLMessage;

public class AgenteConversor extends Agent{

      private String msg;

      public void setup(){
            //conversor do padrão dos documentos XML
            addBehaviour(new Converte());
          //argumentos passados via linha de comando
            Object[] args = getArguments();
            if (args != null && args.length>0){
                  msg = (String) args[0];
            }else{
                  System.out.println("Nenhuma   mensagem   foi   especificada
para o agente "+this.getLocalName());
                  doDelete();
            }
            //getLocalName() retornará o nome do agente
            System.out.println("Agente:"+this.getLocalName());
            sendMessage(msg);
      }
      /**Envia as mensagens
       * @param
       * String msg mensagem a ser enviada para um agente*/
      public void sendMessage(String msg) {
          ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST);
          aclMessage.addReceiver(new AID("monitor",AID.ISLOCALNAME));
          aclMessage.setContent(msg);
          this.send(aclMessage);
        }

}
```

```
package agentes.dw;

import gui.DWGui;
import gui.GestorGui;
import jade.core.AID;
import jade.core.Agent;
import jade.lang.acl.ACLMessage;
import agentes.comportamentos.GerarInformacao;
import agentes.comportamentos.JessBehaviour;
import agentes.comportamentos.ReceberDado;

public class AgenteDW extends Agent{

private String msg;
protected DWGui myGui;
/** inicializa o comportamento do agente*/
public void setup(){
      /* configura a interface gráfica (i.e. gui) */
    myGui = new DWGui(this);
      /* reponsabilidade do agente */
    //Recebe os dados operativos
    addBehaviour(new ReceberDado());
    //Analisa os dados para gerar informação de suporte
    addBehaviour(new GerarInformacao());
      myGui.setVisible(true);

    //argumentos passados via linha de comando
      Object[] args = getArguments();
      if (args != null && args.length>0){
            msg = (String) args[0];
      }else{
            System.out.println("Nenhuma mensagem foi especificada para o
agente "+this.getLocalName());
            doDelete();
      }
      //getLocalName() retornará o nome do agente
      System.out.println("Agente:"+this.getLocalName());
      sendMessage(msg);
}
/**Envia as mensagens
 * @param
 * String msg mensagem a ser enviada para um agente*/
public void sendMessage(String msg) {
    ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST);
    aclMessage.addReceiver(new AID("monitor",AID.ISLOCALNAME));
    aclMessage.setContent(msg);
    this.send(aclMessage);
  }
}
```

```
package agentes.comportamentos;
import jade.core.behaviours.CyclicBehaviour;
import jess.*;
import jade.core.*;
import jade.lang.acl.ACLMessage;
import java.io.*;
import java.util.*;
import agentes.messages.*;


public class JessBehaviour extends CyclicBehaviour{

      //instancias para tratamento do conteúdo das mensagens
    TrataMensagens tratMsg = new TrataMensagens();
      TrataAID tratAID      = new TrataAID();


      /** JessSend é a classe interna que implementa
       * a interface Userfunction do jess para extender
       * linguagem com comandos definidos pelo usuário
       * */
      public class JessSend implements Userfunction {

            Agent ag;
            JessBehaviour jb;

            public JessSend(Agent a, JessBehaviour jessbehaviour){
                  ag = a;
                  jb = jessbehaviour;

            }

            /* nome do metodo pelo qual a função aparece no Jess (i.e.
userFunction)*/
            public String getName() {

                  return("enviaMSG");

            }


    /**Este método só é chamado quando se invoca o comando enviaMSG no
código Jess*/
    public Value call(ValueVector vv, Context context) throws JessException
{
        // se no código jess for invocado o comando (send ?m)
      if(vv.get(1).type() == RU.VARIABLE)
            vv                                                              =
context.getEngine().findFactByID(vv.get(1).factValue(context).getFactId());
      //senão se no código jess for invocado o comando (send (assert
(mensagemACL ...)))
      //else if(vv.get(1).type() == RU.FUNCALL){
      else if(vv.get(1).type() == RU.FUNCALL){
            Funcall fc = vv.get(1).funcallValue(context);
            vv =  fc.get(1).factValue(context);
      }
      ACLMessage msg = jb.fatosACLJess(context, vv);
      ag.send(msg);
        return Funcall.TRUE;
    }
```

```
  } // fim classe JessSend



      Rete  rete;
      //instância da classe agente para referenciar os agentes
      Agent myAgent;
      //mantém o controle de passos a serem alcancados cada vez que o Jess
for rodado
      int   numMaxPassos =  0;
      //conta o número de passos dado pelo jess na execução anterior
      int   totalPassos = -1;


    public JessBehaviour(Agent agent, String jessFile){
      jessFile = "regras.clp";
      myAgent = agent;
      tratAID.hashTabAID = new Hashtable<String, AID>();
      // cria o motor jess
      rete = new Rete();
      try {
                  //  definição  do  template  que  trata  as  amostra  com
propriedades Beans
          rete.eval(beansDecisaoClassTemplate());
          rete.eval(beansTextoTemplate());
          //definição do template que trata as mensagens ACL
                  rete.eval(templateJessACL());
                // definição do template templateAgente
                  rete.eval("(deftemplate        templateAgente        (slot
nomeAgente))");
                  rete.addUserfunction(new JessSend(myAgent,this));
                  // inserção do fato (templateAgente (nomeAgente ...))
                  rete.eval("(deffacts    templateAgente    (templateAgente
(nomeAgente " + myAgent.getName() + ")))");
                  // chamada ao arquivo .clp (arquivo das regras)
                  FileReader fr = new FileReader(jessFile);
                  Jesp j = new Jesp(fr, rete);
                  j.parse(false);
                } catch (JessException re){
                  System.out.println(re);
                } catch (FileNotFoundException e) {
            System.out.println(e);
                }
      }

     public JessBehaviour(Agent agent, String jessFile, int nMxPassos){
       this(agent,jessFile);
       numMaxPassos = nMxPassos;
     }


     /*Executa o comportamento do agente */
     public void action() {

       ACLMessage msg;
       // espera por mensagens
       if (totalPassos < numMaxPassos) {
             System.out.println(myAgent.getName()+  "  esta  bloqueado  pra
esperar mensagens...");
            msg = myAgent.blockingReceive();
            // insere o fato no Jess
```

```
            assertFato(mgsAClJess(msg));
      } else {
            System.out.println(myAgent.getName()+ " verificando se existem
mensagens...");
            msg = myAgent.receive();
            if (msg != null)
                  assertFato(mgsAClJess(msg));
      }
      // inicia a execução do Jess
      try {
            if (numMaxPassos > 0) {
                  totalPassos = rete.run(numMaxPassos);
                  System.out.println("Realizado(s)          "+totalPassos+"
passo(s)");
            }
            else
                  rete.run();

      } catch (JessException re) {
            re.printStackTrace(System.err);
      }
    }
    //verifica a consistência das mensagens
    private boolean vazio(String string) {

      return string == null || string.equals("");

    }

 /* insere um fato representando uma mensagem ACL no Jess*/
       private void assertFato(String fato) {

             try{
                 rete.eval(fato);
               }
             catch (JessException re) {
                 re.printStackTrace(System.err);
             }
       }
       /**Cria o template de comunicação no jess**/
       public String templateJessACL() {

             String cmd = "(deftemplate mensagemACL " +
                                 "(slot performativa) " +
                                 "(slot remetente (type FLOAT)) " +
                                 "(multislot destinatario) " +
                                 "(slot conteudo))";

             return cmd;
       }

       /*Cria a instancia da classe Decisao para acesso às propriedades
set, get ou is
        * utilizadas para as consultas dos valores das amostras*/
       public String beansDecisaoClassTemplate() {

             String cmd = "(import inferencia.*)" +
                           "(defclass decisao Decisao)"+
                           "(ppdeftemplate decisao)";

             return cmd;
```

```
        }

/*Cria o template para a recuperação do texto enviado pelos
       * agentes*/

        public String beansTextoTemplate() {

              String cmd = "(import services.*)" +
                             "(defclass myContAgent ConteudoAgentes)"+
                             "(ppdeftemplate myContAgent)";

              return cmd;
        }

        /*Mensagem enviada pelo remetente */
        public  ACLMessage  fatosACLJess(Context  context,  jess.ValueVector
vv) throws jess.JessException {

              int                              perf                          =
ACLMessage.getInteger(vv.get(0).stringValue(context));
              ACLMessage msg = new ACLMessage(perf);

              System.out.println("********   Remetente   *********   "   +
vv.get(1).toString());

              if (vv.get(1).stringValue(context) != "nil")

msg.setSender(tratAID.obtemAIDAgentes(vv.get(1).stringValue(context)));

              if (vv.get(2).toString() != "nil") {
                    List     l     =     tratAID.obtemListaAgentes(context,
vv.get(2).listValue(context));
                    for (int i=0; i<l.size(); i++)
                          msg.addReceiver((AID)l.get(i));
              }
              if (vv.get(3).stringValue(context) != "nil") {

msg.setContent(tratMsg.semCote(vv.get(3).stringValue(context)));

              }
        return msg;

        }



  /** Manipula o template mensagemACL retornando o
   * comando da lingugem Jess*/
   public String mgsAClJess(ACLMessage msg){
         /* fat inicia a inserção das linhas dos comando da linguagem Jess
a serem codificados
          * no arquivo .clp*/
         String fat;

         if (msg == null)
               return "";

         fat     =     "(assert     (mensagemACL     (performativa     "     +
ACLMessage.getPerformative(msg.getPerformative());

         if (msg.getSender() != null) {
```

```
                fat = fat + ") (remetente " + msg.getSender().getName();
                tratAID.adicionaAID(msg.getSender());
          }
          Iterator i = msg.getAllReceiver();
          if (i.hasNext()) {
                fat = fat + ") (destinatario ";
                while (i.hasNext()) {
                      AID aid = (AID)i.next();
                      tratAID.adicionaAID(aid);
                      fat = fat + aid.getName();
                }
          }
          if (msg.getContent() != null)
                fat      =      fat      +      ")      (conteudo      "      +
tratMsg.cotaStr(msg.getContent());

          if (msg.getReplyByDate()!=null)
                fat=fat+") (reply-by " + msg.getReplyByDate().getTime();

          fat=fat+")))";
          return fat;
    }

} // fim da classe JessBehaviour
```

```
package agentes.comportamentos;
import jade.core.behaviours.Behaviour;
import services.*;

public class VerificarAmostra extends Behaviour{

	LerXML lerXML = new LerXML();

	/* comportamento do agente */
	public void action(){
		while (true){
			try{
				System.out.println("Verificando dados da
amostra...");
				//leitura do arquivo XML e envio dos dados
				lerXML.ler("mqc-padrao.xml");

			}catch (Exception e){
				System.out.println("Erro ao enviar arquivo.
"+e.getMessage());
			}

		}
	}
	/* método obrigatório para o encerramento do loop*/
	public boolean done(){
		return true;
	}}

package agentes.comportamentos;
import jade.core.behaviours.Behaviour;
import services.*;

public class EnviarAmostra extends Behaviour{

		LerXML lerXML = new LerXML();
		int numAmostra;
		static String cp;
		/* comportamento do agente */
		public void action(){
			while (true){
				try{
					System.out.println("Verificando dados da
amostra...");
					//envio dos dados do arquivo XML
					for(int i=7;i<=10;i++){
						System.out.println("Enviando dados da
amostra...");
						//captura o a variável amostra e envia
						lerXML.enviarAmostra(cp+numAmostra);
					}
				}catch (Exception e){
					System.out.println("Erro ao enviar arquivo.
"+e.getMessage());
				}

			}
		}
		/* método obrigatório para o encerramento do loop*/
		public boolean done(){
			return true;}}
```

```
;##########################################################################;
;       Módulo de decisão para o comportamento do agente determinador     ;
;##########################################################################;

;*************************************************************************;
;     Função de tratamento e consulta de dados                            ;
;*************************************************************************;
(deffunction amostra (?composto)
    (bind ?d (new Decisao))
    (definstance decisao ?d)
    (bind ?res (call ?d ?composto))
    ;;imprime o valor da propriedade por meio da variavel ?d
    return ?res
)
;*************************************************************************
;                     Regras de tratamento de mensagens                   ;
;*************************************************************************;
;variáveis globais para armazenar o nome dos agentes
(defglobal ?*agRem* ="")
(defglobal ?*agDes* ="")

;enviaMSG é a userFunction definida na classe JessBehaviour

(defrule proposalMensagem
 ?m <- (mensagemACL (performativa REQUEST) (remetente ?s) (conteudo ?c)
(destinatario ?r))
 =>
 (store AgRem ?s)
 (store AgDes ?r)
 (bind ?*agRem*(fetch AgRem))
 (bind ?*agDes*(fetch AgDes))
 (enviaMSG (assert (mensagemACL (performativa RESQUEST) (destinatario ?s)
(conteudo ?c) )))
 (assert (mensagemACL (performativa REQUEST) (remetente ?r) (destinatario
?s) (conteudo ?c) ))
 (retract ?m)
)
```

ANEXO B: Regras do JESS

```
;*************************************************************************
;                                   Regras                                ;
;*************************************************************************
(defrule regra1
(test (< ?rmt (amostra isEvapGas1)))
(test (< ?rmt (amostra isTempGas1)))
=>
(enviaMSG "Amostra com indício de contaminação por material pesado (ex.
solvente com ponto de ebulição alto, óleo diesel ou teor de álcool alto).")
(store RESULT1 "Amostra com indício de contaminação por material pesado
(ex. solvente com ponto de ebulição alto, óleo diesel ou teor de álcool
alto).")

(defrule regra2
(test (< ?rmt (amostra isEvapGas2)))
(test (< ?rmt (amostra isTempGas2)))
=>
(enviaMSG "Amostra com indício de contaminação por material pesado (ex.
Solvente com ponto ebulição alto óleo diesel ou teor de álcool alto).")
(store RESULT2 "Amostra com indício de contaminação por material pesado
(ex. Solvente com ponto ebulição alto, óleo diesel ou teor de álcool
alto).")

(defrule regra3
(test (< ?rmt (amostra isEvapGas3)))
(test (< ?rmt (amostra isTempGas3)))
=>
(enviaMSG "Amostra com indício de contaminação por material leve.")
(store RESULT3 "A amostra apresenta indício de contaminação por material
leve.")

(defrule regra4
(test (< ?rmt (amostra isEvapGas4)))
(test (< ?rmt (amostra isTempGas4)))
=>
(enviaMSG "Amostra com indício de contaminação por material pesado.
Substância com elevado ponto de ebulição.")
(store RESULT4 "Amostra com indício de contaminação por material pesado.
Substância com elevado ponto de ebulição.")

(defrule regra5
(test (> ?rmt (amostra isPntoEbFinGas)))
(test (> ?rmt (amostra isTempFinGas)))
=>
(enviaMSG "Amostra com indício de contaminação por material pesado.
Substância com elevado ponto de ebulição.")
(store RESULT5 "Amostra com indício de contaminação por material pesado.
Substância com elevado ponto de ebulição.")

(defrule regra6
(test (> ?rmt (amostra isTempDie1)))
(test (> ?rmt (amostra isEvapDie1)))
=>
(enviaMSG "Amostra com indício de contaminante leve (ex. querosene)")
(store RESULT6 "Amostra com indício de contaminante leve (ex. querosene)")

(defrule regra7
(test (> ?rmt (amostra isTempDie2)))
```

```
(test (> ?rmt (amostra isEvapDie2)))
=>
(enviaMSG "Amostra com indício de contaminante leve (ex. solvente em
suspensão ou solvente de borracha)")
(store RESULT7 "Amostra com indício de contaminante leve (ex. solvente em
suspensão ou solvente de borracha)")

(defrule regra8
(test (> ?rmt (amostra isTempDie3)))
(test (> ?rmt (amostra isEvapDie3)))
=>
(enviaMSG "Amostra com indício de contaminante leve (ex. solvente em
suspensão ou solvente de borracha)")
(store RESULT8 "Amostra com indício de contaminante leve (ex. solvente em
suspensão ou solvente de borracha)")
```