UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE
ÁREA: CIÊNCIAS DA COMPUTAÇÃO

OSEVALDO DA SILVA FARIAS

MODELAGEM E IMPLEMENTAÇÃO DE UM SISTEMA MULTIAGENTE PARA SELEÇÃO DE FALHAS E TOMADA DE DECISÃO EM VIRADORES DE VAGÕES

São Luís
2009

OSEVALDO DA SILVA FARIAS

MODELAGEM E IMPLEMENTAÇÃO DE UM SISTEMA MULTIAGENTE PARA SELEÇÃO DE FALHAS E TOMADA DE DECISÃO EM VIRADORES DE VAGÕES

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.
Orientador. Prof. Dr. João Viana Fonseca Neto.

São Luís
2009

Farias, Osevaldo da Silva

Modelagem e Implementação de um Sistema Multiagente para Seleção de Falhas e Tomada de Decisão em Viradores de Vagões / Osevaldo da Silva Farias. – São Luís, 2009.

130f.

Orientador: Sofiane Labidi.

Co-orientador: João Viana Fonseca Neto

Impresso por computador (Fotocópia)

Dissertação (Mestrado) – Universidade Federal do Maranhão, Programa de Pós-graduação em Engenharia de Eletricidade. São Luís, 2009.

1. Sistemas Multiagentes. 2. Inteligência Artificial. 3. Viradores de Vagões. I.Labidi, Sofiane, orientador. II. Fonseca Neto, João Viana, co-orientador. III. Título

CDU 0004.891

MODELAGEM E IMPLEMENTAÇÃO DE UM SISTEMA MULTIAGENTE PARA SELEÇÃO DE FALHAS E TOMADA DE DECISÃO EM VIRADORES DE VAGÕES

Osevaldo da Silva Farias

Dissertação aprovada em 09 de fevereiro de 2009.

Prof. Sofiane Labidi, Dr.
(Orientador)

Prof. Luiz Affonso Henderson Guedes de Oliveira, Dr.
(Membro da Banca Examinadora)

Prof. João Viana da Fonseca Neto, Dr.
(Membro da Banca Examinadora)

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

A Deus, apoio espiritual constante.
À minha filha Karem Vitória, à minha mãe Antonia e à minha amada Esposa Keila.

"O conhecimento, diferentemente do dinheiro, cresce quando é compartilhado".

Prof. Hélvio Prazeres

AGRADECIMENTOS

Agradeço a Deus por permitir que a concretização desta dissertação.

A minha esposa Keila e a minha filha Karem Vitória pela paciência e apoio nos momentos em que tive que me ausentar.

Ao Professor Sofiane Labidi oportunidade de ter sido seu aluno e pesquisador do LSI. Pela orientação e pela sua presença nos momentos difíceis desta jornada, pelo seu conhecimento e experiência que me ensinaram a ter independencia sem deixar de ser dependente. Pelas recomendações sobre Inteligência Artificial e Mecanismos de Inferência que adicionaram mais contrubuições à pesquisa.

Ao Professor João Viana pelos esclarecimentos sobre automação e controle de processos durante o projeto UFMA/VALE e pelas importantes reuniões coordenadas por ele para o entendimento dos principais equipamentos e sistemas de supervisão. Pelas críticas, sugestões e correções da terminologia de trabalho descrita nos relatórios técnicos que os apresentava. Pela disponibilidade de dispositivos do Laboratório de Controle de Processos para a compreensão da programação dos CLP's e principalmente. Pela lição de vida acadêmica e profissional aprendida com ele.

Ao Professor Zair Abdelouahab pela troca de ideias durante as aulas do mestrado e encontros da ERCEMAPI. Pelas sugestões para melhoria da pesquisa com uso de metodologias orientadas a objetos e linguagem JAVA sempre que tínhamos a oportunidade de tomar um cafezinho juntos.

Ao engenheiro José Pinheiro pela articulação das entrevistas com os técnicos, Marcos, Simone, Heuser, Jarian e Falção, pela disponibilidade de mostrar a área de descarga da VALE e os centros de controle operacionais. Por possibilitar o

compartilhamento de conhecimento ao analisar as soluções candidatas que frequentemente ocorria durante almoço no restaurante da VALE. e pelas suas contribuições para concepção e refinamento do conhecimento do sistema proposto.

Aos colegas Thiagão, José Mendes, Cícero Quarto, Cláudio Sampaio, Valeska Trinta, Rosemary Midori, Luís Carlos e Reinaldo Silva. Também aos colegas Bernardo Wanghon, Rômulo Martins, Pedro Bradão e Roosewelt Lins do Laboratório de Sistemas Inteligentes (LSI) que foram os primeiros colegas a me receberem no LSI. Especialmente ao Professor Nilson Santos pelos trabalhos, métodos e modelos de programação de agentes que discutíamos e que ajudavam a refinar os primeiros estágios de desenvolvimento desta pesquisa. Também aos colegas Rafael Cunha, Rafael Cruz, Christian e Luís Oliveira.

Aos colegas Falkner, Paulo José, Pedriana, Mariano, Adriana, Lucas, Aline Coelho, Elaine, Alex Barradas, André Santos e Luciana Silva.

Aos colegas que ganhei no Laboratório de Controle de Processos LCP: Vitor, João Inácio Jorge Macfly e Jorge Farid, que foram os primeiros alunos que me receberam no LCP e mostrar o ritmo de trabalho, e à Aline, Renan e Fábio. Também à Samy Flores pela sua dedicação na captura e organização dos dados das entrevistas durante as visitas na VALE e seminários. Ao Ivanildo pelas as discussões durante sua participação nos seminários da VALE realizados no LCP.

Ao funcionário Alcides, pela atenção em fornecer informação sol andamento do mestrado e à bibliolecária Rosivalda pelo seu excelente trabalho.

Aos demais professores do mestrado na área de ciências da computação: Prof. Francisco, Prof. Alexandre Muniz, Prof. Mário, Prof. Aristófanes, Prof. Denivaldo Lopes e especialmente à professora Rosário Girardi que me ajudou muito a entender as diferenças.

À FAPEMA e à CAPES pelo fomento recebido durante esta pesquisa.

Enfim, agradeço a todas as pessoas que contribuíram direta ou indiretamente para a realização desta pesquisa.

RESUMO

Este trabalho apresenta a modelagem e implementação de um aplicação multiagente para a seleção de falhas e tomada de decisão em viradores de vagões da VALE. A proposta investiga abordagens de desenvolvimento e arquiteturas baseadas no conhecimento para criação do SADDEM, um sistema multiagente aplicado ao apoio a decisão em células de descarga de minério. As particularidades das estruturas cognitivas de agentes são baseadas no mecanismo inferência Jess, objetivando encadear decisões e recomendações de planos de manutenção feitas pelo sistema. A ontologia do sistema é elaborada por meio das informações e conhecimento eliciados durante entrevista e atividades operacionais referentes ao conjunto carro posicionador observadas no equipamento virador de vagões VV311-K01. Os artefatos produzidos durante as fases do ciclo de desenvolvimento têm a aplicação das técnicas da metodologia PASSI, Inteligência Artificial e das engenharias de apoio ao desenvolvimento de software com uso do framework JADE. O módulo de tempo real do sistema está vinculado ao software de supervisão e monitoramento das células de produção do setor de descarga de minério da VALE.

Palavras-chave: Sistemas Multiagente. Viradores de Vagões. Inteligência Artificial. JESS. JADE.

ABSTRACT

This work leads to the modeling and implementation of a multi-agent application of faults selection and decision making for VALE's rotary railcar dumpers. The work explore development approaches and knowledge-based achtectures aiming to SADDEM build, a multi-agent system applyed to decision making in ore unload cells. The particularities of cognitives agents structures are based on Jess inference engine in order to link decisions and maintain plans recommendations made by the system. The ontology is developed throught elicitation of information and knowledge during interviews and VV311-K01 car positioner set operating activities. The artifacts produced during the development clycle phases employs PASSI methodology technics, Artificial Intelligence and support engineering to software development using JADE framework. The real time module is linked to the software of supervision and monitoring of the unloading cell productions of VALE.

Keywords: Multi-agent Systems. Rotary Railcar Dumpers. Artificial Intelligence. JESS. JADE.

LISTA DE SIGLAS

ACL	Agents Common Language
BDI	Beliefers Desires Intentions
CCO	Centro de Controle Operacional
CLP	Controlador Lógico Programável (Programable Logic Controller-PLC)
CPU	Central de Processamento de Dados
DCOM	Distribuited Component Object Model
DDE	Dynamic Data Exchange
EC	Engenharia do Conhecimento
EC	Engenharia do Conhecimento
EPS	Enterprise Production Systems
ER	Engenharia de Requisitos
ES	Engenharia de Software
FIPA	Foundation for Intelligent Phisycal Agents
IA	Inteligência Artificial
IEEE	Institute of Electrical and Eletronics Engineers
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standart Edition
JADE	Java Agent Development Environment
JESS	Java Expert System Shell
KQML	Knowledge Query and Manipulation Language
LSI	Laboratório de Sistemas Inteligentes
MES	Manufacturing Execution Systems
OLE	Object Linking Embedding
OPC	OLE Process for Control
PASSI	Process for Agent Society Specification and Implementation
PEAS	Performance, Environment, Actuators, Sensors
PIMS	Plant Information Managment Systems
RCM	Reliability Centred Maintenance
SADDEM	Sistema de Apoio a Decisão em DEscarga de Minério

SCADA	Controle Supervisório e Aquisição de Dados (Supervisory Control Acquisition Data)
SE	Sistema Especialista
SMA	Sistema Multiagentes
TCP/IP	Transfer Control Protocol/Internet Protocol
TMPM	Terminal Marítimo de Ponta da Madeira
UML	Unified Modeling Language
XML	eXtensible Marckup Language

LISTA DE QUADROS

LISTA DE FIGURAS

SUMÁRIO

p.

1 INTRODUÇÃO

As transformações de ordem política, social e econômica do século XVII remontaram os primeiros cenários das grandes revoluções, principalmente as envolvidas com as origens do processo de industrialização e, sobretudo as que foram responsáveis pelo surgimento da microeletrônica e do computador moderno por volta de 1940, desencadeando uma série de mudanças no perfil competitivo das organizações.

1.1 Contexto

No século XX, a preocupação em estudar simultaneamente o comportamento humano e mecanismos para o processamento não numérico de dados, proporcionou o surgimento de uma nova divisão da computação. O progresso dessa subdivisão permitiu ganhar meios e massa crítica para se estabelecer como ciência íntegra, nascendo assim o termo Inteligência Artificial (IA).

A partir da sua primeira definição em 1956, a IA vem desenvolvendo diversas técnicas para concepção de sistemas que sejam capazes de resolver tarefas que exigem conhecimento e raciocínio. Neste meio século, uma gama de trabalhos inseridos no campo da Ciência da Computação vem tentando entender as complexidades da mente humana, através de um amplo estudo de aprendizagem (FERNANDES e cols., 2004).

Visando imitar seu comportamento humano, o computador é apontado como modelo mais adequado para entender e simular o funcionamento do cérebro humano acrescentam Labidi *et al*. (2003). Para isso, utilizam-se modelos eletrônicos para a exploração do comportamento inteligente embutido em entidades de software que atuam em ambientes complexos envolvendo: percepção, raciocínio, aprendizagem, comunicação, ação e planejamento.

Isso possibilitou que a IA direcionasse seus estudos no desenvolvimento de sistemas potencialmente aptos à realização de tarefas que os humanos ainda executam melhor, contudo a redução na distância cognitiva em tarefas como reconhecimento da fala, robótica, representação do conhecimento e visão tem mostrado resultados promissores, como é o caso do BigDog (BOSTON DYNAMICS,

2008), da SeteZoom (INBOT, 2008) e das ferramentas de tradução do Google (GOOGLE, 2007) e Altavista (ALTAVISTA, 2007).

Além disso, os diferentes recursos da IA também norteiam conhecimentos em algoritmos genéticos, redes neurais, sistemas difusos (*fuzzy*), redes bayesianas, sistemas especialistas (SE) e sistemas Multiagentes (SMA) (LABIDI *et al*., 2006). Estes conhecimentos são os que apresentam maior aplicabilidade no campo da IA (RUSSEL e NORVIG, 2004).

Dentre os recursos citados, Luger (2004) explica que os SEs são sistemas que reproduzem o conhecimento de um especialista adquirido durante um longo tempo de experiência, e por essa razão, devem ser construídos com o auxílio de um especialista humano, que dessa forma irá fornecer uma base de informações para a sua concepção.

Os SEs foram aplicações periciais que lançaram a IA no domínio comercial. O SE Mycin, com a utilização do conhecimento médico especializado no tratamento de pacientes com meningite e bacteriomia, e o SE Dendral, voltado para a inferência da estrutura de moléculas orgânicas, destacam-se entre os SEs pioneiros que marcaram a passagem da IA para a fase moderna (GIRARDI, 2005).

A evolução dos SEs está no surgimento dos SMAs, envoltos em uma abordagem da IA Distribuída (FLORES, 2003). Os SMAs realçam soluções para resolverem tarefas de forma colaborativa e cooperativa, focando abstrações sociais, independentes e orientadas a objetivos. Na realidade, enquanto os SEs são estruturas centralizadas em uma única entidade de software, os SMAs apresentam arquitetura descentralizada de duas ou mais entidades de software, convencionalmente chamadas de agentes.

SMAs e SEs são sistemas baseados no conhecimento que proporcionam estruturar o computador em um modelo hábil para a atuação e comportamento mais próximos ao modo com que os humanos usam o seu raciocínio diante de situações de escolha.

Nesse sentido, considerando que em boa parte dos mecanismos da produção industrial têm sido empregadas soluções capazes de automatizar

processos indispensáveis envolvendo controle padronizado, tolerância à falhas e tomada de decisão (Su, *et al.*, 2005), nas próximas seções serão desenvolvidos como alvo de estudo deste trabalho parte do complexo produtivo da VALE, em São Luís-MA e os estágios de construção do Sistema de Apoio a Tomada de Decisão em Descaga de Minério com abordagem multiagente.

1.2 Descrição do problema

O número de organizações que empregam sistemas informatizados objetivando a coleta e o tratamento de informação para a tomada de decisões aumentou consideravelmente. Na sociedade pós-industrial, as tecnologias do conhecimento tornam-se objeto de uso intenso, proporcionando melhorias na comunicação entre as pessoas, na supervisão e otimização dos procedimentos oraganizacionais e produtividade em geral.

Indagando-se sobre como os modernos critérios da Engenharia de Conhecimento (EC) e IA podem apoiar decisões em ambientes não determínísticos como a indústria, nota-se que tais critérios preocupam-se em não aborver somente as informações factuais (e.g. tangíveis). Além disso, armazenam as informações intangíveis que estão ligadas ao relacionamento entre as pessoas e incluem no seu espaço tecnológico artefatos que raciocinam e usam conhecimento especializado em busca de soluções.

Davenport (2000) destaca a força da informação e do conhecimento explicando que “a única vantagem sustentável que uma organização tem é aquilo que ela coletivamente sabe, a eficiência com que ela usa o que sabe, a prontidão com que ela adquire e usa novos conhecimentos”.

De acordo com Prazeres, (2007) o conhecimento quando é compartilhado tende a aumentar, ao passo que o capital diminui quando é dividido. Com essa premissa conclui-se que a “briga” no mercado ocorre pelo capital intelectual, reforçada principalmente quando se ouve falar da bolsa de valores da NASDAQ.

Com o aumento da complexidade para se solucionarem os problemas da era da informação e do conhecimento, aplicações desenvolvidas apenas com as tecnologias convencionais já não acompanham o gerenciamento e o ritmo de mudança das regras de negócios das organizações. O mundo globalizado exige

cada vez mais velocidade nas decisões tomadas em todos os níveis, ou seja, operacional, tático e estratégico.

Os problemas de volume de informação e conhecimento especializado, associados à seleção de falhas e tomada de decisão do virador de vagões da VALE, refletem as dificuldades enfrentadas em situações de falhas de módulos como o conjunto carro posicionador do virador, normalmente chamado carro empurrador. A análise sistêmica de falhas realizada pelas equipes de manutenção da VALE ocorre por meio de consultas às planilhas eletrônicas com modos de falhas catalogados, procedimento este que resulta na atual demora para se chegar às causas da falha.

Levando-se em consideração o intervalo de tempo total para descobrir se a causa de uma comutação deficiente no motor de corrente contínua do conjunto acionamento do carro posicionador pode estar relacionada a uma deficiência do sistema do motor, um travamento do freio ou até mesmo uma sobrecarga - dentre uma série de causas - há um atraso considerável.

Uma falha no carro posicionador do VV311-K01 pode acarretar perdas operacionais com a redução de 1100 ciclos de descarga de minério. Segundo relatórios da VALE, isto é equivalente a uma perda de 208 vagões de minério de uma média de 16120 ciclos por mês (LCP, 2007).

Nesse sentido, desperta-se o interesse de se adicionar melhorias no processo de descarga de minério com à modelagem e implementação de um SMA para automatizar os procedimentos de seleção de falhas e tomada de decisão em viradores de vagões em tempo real, gerenciando o grande volume de dados operacionais e o tempo necessário para a realização de manutenção.

1.3 Objetivos e abordagem

O propósito deste trabalho é desenvolver uma aplicação multiagente de apoio à tomada de decisão para a manutenção de falhas ocorridas em viradores de vagões.

Especificamente, pretende-se:

Analisar o processo operacional produtivo de descarga de minério que ocorre virador de vagões VV311-K01 da VALE;

Estudar os mecanismos de automação e sistemas de supervisão dos equipamentos dos centros de controle operacional de descarga da VALE;

Elaborar a especificação de Análise e Projeto do Sistema Multiagentes junto aos critérios das Engenharias do Conhecimento, de Software e de Agentes usando a metodologia PASSI;

Implementar o Sistema Multiagentes para Apoio a Tomada de Decisão em Descaga de Minério e promover suporte em tempo real à tomada de decisão usando as plataformas JADE de sistemas Multiagentes, Jess como máquina de inferência para codificação de agentes cognitivos para diagnóstico e recomendação.

Reduzir o atraso na identificação de subsistemas defeituosos do carro posicionador do VV311-K01, detectando as causas e recomendando planos de manutenção.

Associar o desenvolvimento do sistema no Laboratório de Sistemas Inteligentes (LSI) da Universidade Federal do Maranhão à sua respectiva validação.

As abordagens deste trabalho propõem melhorias quanto à aceleração das atividades ligadas ao diagnóstico de falhas ocorridas no virador de vagões VV311-K01, visto que a tomada de decisão ainda não possui nenhum suporte computacional inteligente que funcione em linha de produção.

Os resultados capturados por meio do desenvolvimento SMA possibilitam ao especialista obter um plano de ação correspondente à manutenção ou até mesmo acionar uma equipe responsável para a resolução da falha que ocorre no carro posicionador do VV311-K01 em menor tempo.

A ferramenta *Java Agent DEvelopment* (JADE) reúne uma série de bibliotecas prontas para uma boa infraestrutura de agentes. Essas bibliotecas tornam a implementação de sistemas multiagentes mais simples e robusta. Além do JADE, nesta dissertação são utilizados ainda a metodologia *Process for Agent Societies Specifications and Implementation* (PASSI) (COSSENTINO e

BURRAFATO, 2002), e da máquina de inferência *Java Expert System Shell* (JESS) (FRIEDMAN-HILL, 2003, FRIEDMAN-HILL, 2006).

O emprego da metodologia PASSI se faz necessário para o gerenciamento das técnicas orientadas a agentes e suporte ao ciclo de desenvolvimento do SMA, desde à concepção até a construção do mesmo, enfatizando-se a modelagem e implementação de apenas três dos quatro agentes. O processo reúne os passos e atividades a serem percorridas de forma sistematizada de acordo com a Engenharia de Software (ES).

Por outro lado, o motor de inferência Jess, que é usado no desenvolvimento de condições e ações declaradas no raciocínio dos agentes e está vinculado à base regras onde as instâncias do conhecimento do ambiente de descarga estão modeladas. A escolha do Jess é justificada por ter sido escrito em linguagem Java, o que contribui para os aspectos de portabilidade.

Em termos de hardware, algumas interfaces de comunicação, entre o supervisório e drives de importância comercial, apresentam restrições de sistemas operacionais. Nesse sentido, arquiteturas de acesso e protocolos de comunicação baseados tecnologias OLE (*Object Linking Embedding*)/ DCOM (*Distribuited Component Object Model*) não são utilizados neste trabalho, visto que foram desenvolvidas necessariamente para Windows.

Contudo, o sistema proposto tem capacidade de utilizar tais tecnologias, desde que as interfaces de comunicação dos dispositivos de campo (CLPs, balanças, sensores, etc.) estejam abertas e sejam autorizadas pela gerência de tecnologia de automação da VALE para a codificação, ressaltando que só terão efetividade em ambiente Windows.

A validação é obtida com a comparação dos resultados durante a execução do sistema pelos técnicos e engenheiros do Centro de Controle Operacional do Terminal Marítimo de Ponta da Madeira da VALE, em São Luís-MA, em ambiente simulado nos Laboratório de Sistemas Inteligentes e Automação e Controle de Processos da UFMA.

Este trabalho foi objeto de publicação do artigo Real Time Expert System for Faults Identification in Rotary railcar Dumpers na International Conference on Informatics in Control, Automation and Robotics - ICINCO2008 e aceito como capítulo para publicação do livro Robotics, Automation and Control, ISBN 978-953-7619-39-8, da IN-TECH disponível a partir de agosto de 2009.

1.4 Justificativa

A Indústria engloba um ambiente complexo, abrangente e dotado de um enorme fluxo de informações. Considerando-se as transformações que a Indústria mineradora sofreu nas últimas décadas com a compra de novos nichos de mercado, a gestão dos recursos e o planejamento empresarial, desde o chão de fábrica até a cúpula administrativa, vem exigindo constantemente sistemas baseados em informação e conhecimento.

Isso tem gerado o diferencial competitivo e estratégico dos empreendimentos de pequeno à grande porte, pois a economia mudou de uma economia industrial para uma economia que tem valorizado bastante o conhecimento e a informação.

Além disso, a globalização vem tentando consolidar a era da convergência assim como o fez com a economia digital. Observa-se que com a VALE não é diferente. Ao contrário das chamadas empresas da velha economia a VALE vem se mantendo no mesmo patamar das empresas baseadas no conhecimento em termos de valores no mercado mundial.

A comunidade econômica vive uma mudança de época onde os processos do domínio se tornam mais complexos à medida que os negócios se expandem. Os critérios da Engenharia do Conhecimento (EC) e Engenharia de Requisitos (ER) associados a IA, tendem a se apresentar como meios para o alcance de sistemas inteligentes por meio de sociedades computacionais que simulam o raciocínio humano para obtenção de soluções de problemas complexos aplicados à indústria dos dias atuais (SU *et al.*, 2005).

Várias soluções puderam contribuir para a gestão empresarial e industrial, bem como para os negócios eletrônicos. Na medicina, os médicos utilizam sistemas de conhecimento ao combate anual de infecções de dois milhões de pacientes,

explorando e analisando dados contidos no *data mining* em busca de padrões clínicos que poderiam ser de outra forma ignorados (TURBAN *et al*., 2005).

Na indústria, a maneira clássica de retirar tinta antiga do casco do navio está associada a uma quantidade enorme de trabalho, tempo e mão-de-obra. Para esse propósito, a empresa *Ultrastrip Systems* desenvolveu um robô para executar este processo de limpeza por meio de um *joystic* nas mãos de um operador. O resultado foi redução de tempo e qualidade superior a de dez homens que antes realizavam o procedimento (TURBAN *et al*., 2007).

No Brasil, a IA evoluiu no setor bancário, principalmente em relação a análise de crédito e seguradoras. Além dessas áreas, na medicina também estão presentes sistemas de tomada de decisão para auxiliar enfermeiros, médicos e outros profissionais de saúde nos diagnósticos em lugares remotos, onde a presença desses não é tão fácil, principalmente em regiões endêmicas como a Amazônia.

Atualmente, diversos estudos da IA no Brasil têm explorado o setor industrial no desenvolvimento de aplicações inteligentes em problemas da engenharia elétrica, automação industrial e controle de processos industriais, automobilístico e mineração, principalmente em setores de controle de qualidade, tomada de decisão, tratamento de falhas, energia e montagem.

A evolução das atividades industriais foi a principal responsável pela complexidade presente hoje nos processos, contudo o monitoramento do controle automático tem adicionado preocupações com o tamanho das plantas.

Outra preocupação diz respeito às carências na obtenção de uma gestão mais efetiva das falhas, a fim de evitar perdas operacionais e incrementar a produção e disponibilidade do sistema de descarga. Paralelamente, projetar e construir software industrial de alta qualidade que atenda aos requisitos da organização tem sido tarefa difícil, principalmente com o uso de soluções da informática tradicional.

O crescente interesse em agentes de software e SMA tem proporcionado inovações tecnológicas contendo conhecimento e elementos cognitivos para ajudar

a entender as sociedades complexas. A orientação a agente deixa de existir apenas nos laboratórios de pesquisa das universidades e passam a integrar as soluções usadas nas aplicações industriais e comerciais.

Diante do exposto, este trabalho apóia-se no número de aplicações voltadas para a produção industrial utilizando técnicas da IA. O levantamento de falhas em viradores de vagões está associado à construção de um SMA para a detecção de anormalidades nestes equipamentos, visando fornecer mais produtividade a partir de uma combinação de entidades de software inteligentes, informação e conhecimento.

1.5 Organização

Esta dissertação está organizada em seis capítulos.

No capítulo 2, buscou-se evidenciar a importância da informação e o significado do conhecimento para as organizações com o foco voltado para a indústria. O capítulo 3 aborda as principais características do sistema de descarga de minério da VALE, sendo detalhando o funcionamento sistema virador de vagões.

O Capítulo 4 apresenta as particularidades da engenharia de software multiagentes e destaca as contribuições dadas pelas práticas da engenharia de software tradicional. Adicionalmente, são tratados os conceitos de agentes, tipos e arquiteturas de agentes, barreiras encontradas no desenvolvimento orientado a agentes, padronizações, formalismos de representação do conhecimento e metodologias.

No capítulo 5, todos os esforços se concentram no ciclo de desenvolvimento do SADDEM – Sistema de Apoio a Decisão em Descarga de Minério. Na construção do sistema é empregada a metodologia PASSI para a decomposição dos requisitos em modelos UML.

Há ainda a descrição dos agentes e a arquitetura do SMA adotada. Em seguida, a concretização dos artefatos é detalhada em componentes executáveis com a descrição da implementação do sistema. Interfaces gráficas do sistema também são apresentadas com discussão dos resultados dos experimentos.

Por fim, as conclusões e contribuições da dissertação são reveladas no capítulo 6, bem como os trabalhos futuros.

2 SISTEMA VIRADOR DE VAGÕES

Devido à complexidade do equipamento virador de vagões, dedicou-se este capítulo para uma abordagem mais detalhada de suas funcionalidades em termos de subsistemas, conjuntos e itens, destacando-se o VV311-K01. Atualmente a VALE dispõe de três equipamentos viradores de vagões. No estudo dos três, as diferenças percebidas são restritas a potência e motores e mecanismos de deslocamento de vagões tais como cremalheiras e pinhões (MOURA, 2003). Estes últimos estão presentes somente nos viradores VV311-K02 e VV311-K03. Assim a modelagem do VV311-K01 também pode ser aplicada a eles, adaptando-se apenas as regras de conhecimento para a composição das inferências.

Neste capítulo aborda-se o sistema de descarga da VALE. Inicialmente faz-se uma introdução sobre os elementos gerais deste sistema e em seguida destaca-se o objeto de estudo desta dissertação fornecendo o detalhamento do processo do sistema virador de vagões VV311-K01.

2.1 Sistema de Descarga da VALE

Dentre os complexos mineradores do sistema produtivo da VALE o TMPM (Terminal Marítimo de Ponta da Madeira), situado no sistema norte da companhia e local de estoque e embarque de minério de ferro, encontra-se no *rank* dos maiores portos do mundo.

No Brasil, o TMPM está em segundo lugar em circulação de cargas tais como soja, manganês e principalmente exportação de minérios de ferro.

Este terminal é constituído de três viradores de vagões – cada um com capacidade de 8.000 toneladas/hora –, duas recuperadoras – 8.000 toneladas/hora cada –, duas empilhadeiras – uma com capacidade de 16.000 toneladas/hora e a outra 8.000. Há também dois carregadores de navios, sendo um com autonomia para 16.000 toneladas por hora e o outro com 8.000 por hora (LCP, 2006).

Para o mecanismo de empilhamento e recuperação de minério, o TMPM conta com uma máquina que integra estes dois processos e suporta 8.000 toneladas/hora. Além destes equipamentos, o complexo ainda possui uma planta para: descarga, estocagem e embarque de grãos e uma plataforma para o

condicionamento do ferro-gusa que é descarregado. A Figura 1 ilustra as células de produção do TMPM.

Em termos de atracagem deste porto toma-se como exemplo o *Berge Stahl* – o maior navio transportador do mundo, com autonomia de 365.000 toneladas – que só atraca em três portos do mundo. No Brasil, somente o TMPM tem essa capacidade de atracá-lo, pois no porto de Santos (SP), a capacidade de atracagem está em torno de 75.000 toneladas.

No TMPM as ilhas de produção realizam os processos de descarga, empilhamento, recirculação, peneiramento, recuperação e embarque de minérios. Todas as informações pertinentes ao processo de descarga são insumos para esta pesquisa. Conforme dito há pouco, neste processo encontram-se os três viradores de vagões.

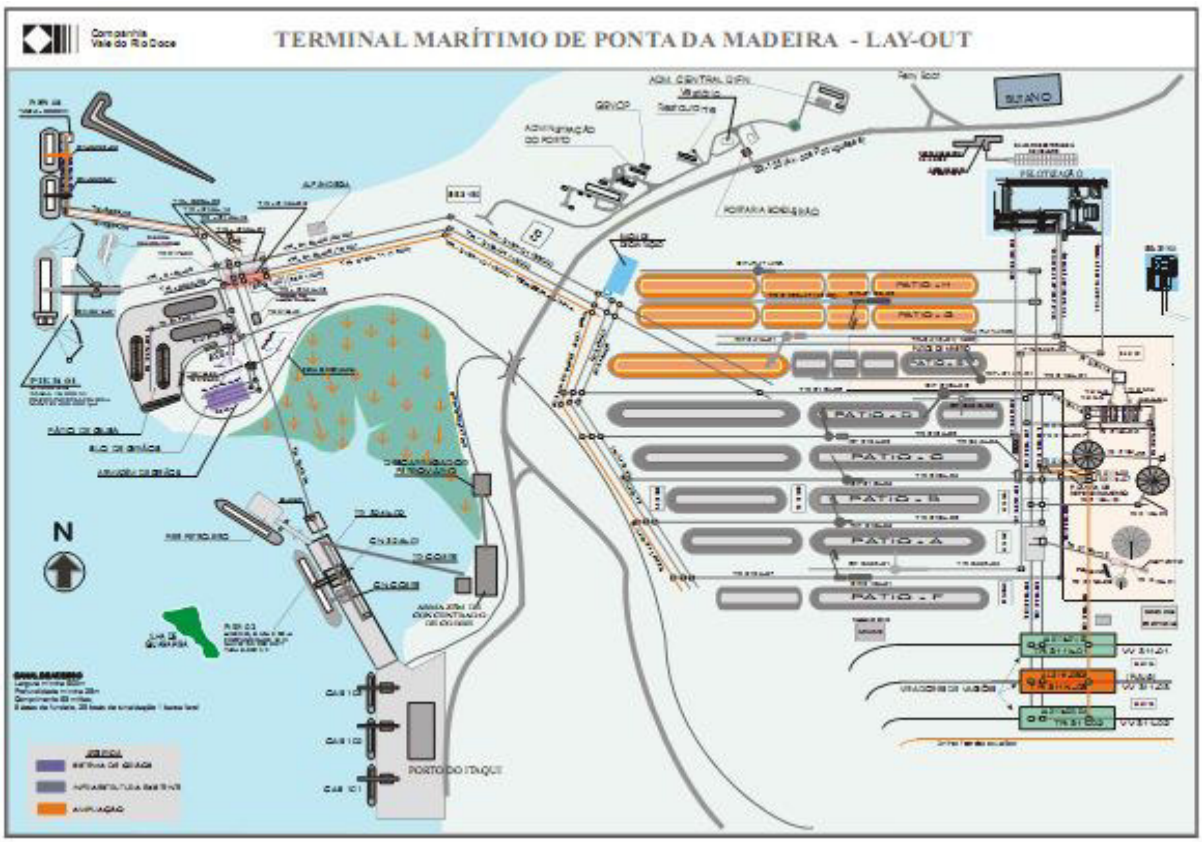

Figura 1 – Planta de Descarga e Embarque da VALE

A descarga de minérios é iniciada nos viradores de vagões (e.g. VV 311-01) com a chegada da locomotiva conduzindo uma média de 102 a 104 vagões que são

posicionados no virador. A partir do ajuste dos vagões, o propósito de cada virador passa a ser descarregar dois deles por giro.

Cada giro representa um ciclo de descarga. O minério é despejado em alimentadores constituídos por esteiras, que estão em nível abaixo do virador de vagões. As esteiras ou tremonhas (e.g. AL 311-01) são responsáveis por conduzir o material descarregado até as correias (e.g. TR 311-01) transportadoras (LCP, 2006).

As correias transportadoras são responsáveis por grande parte do translado do minério de ferro tanto do setor de descarga até o pátio de estocagem quanto deste para o navio, dependendo exclusivamente das empilhadeiras (e.g. EP 313-02) ou empilhadeiras recuperadoras (e.g. ER 313-01 entre os pátios A e F. ver Figura 1), que armazenam o minério, e das recuperadoras (e.g. RP 312-01) para que o percurso do material seja completado até a embarcação. A Figura 2 ilustra os mecanismos de processamento de descarga de minério do TMPM destacando-se o VV311-K01 (GUIMARÃES, 2001, MOURA, 2003).

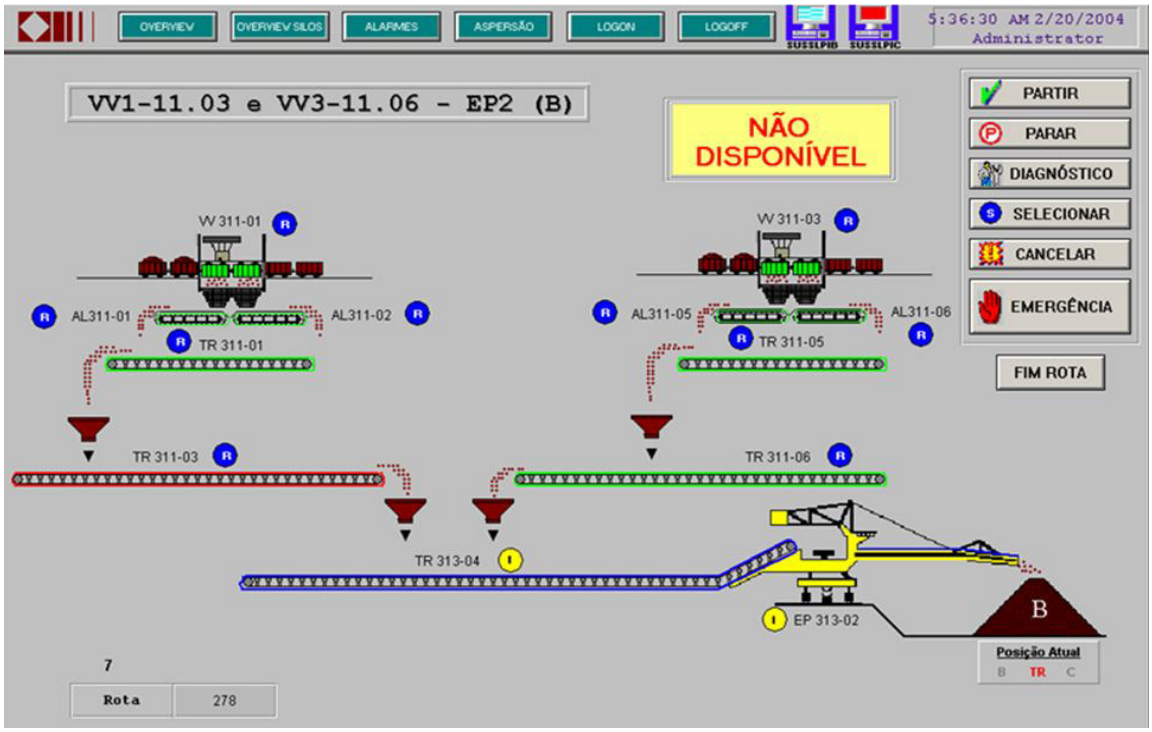

Figura 2 – Processos de estocagem e embarque de minério.

As cores vermelha, verde e azul na Figura 2 estão associadas ao funcionamento ou não das esteiras ou transportadores, além de servirem como

seleção de rotas de transporte do minério e nesse sentido, fogem do escopo desta dissertação.

2.2 Detalhamento do processo

O virador de vagões VV311-K01 é um equipamento utilizado para descarga de minério e possui autonomia para girar dois vagões por vez. É importante enfatizar que, apesar do sistema de descarga da VALE possuir três viradores de vagões, sendo o VV311-K02 e o VV311-K03 um pouco diferente do VV311-K01, no que diz respeito a alguns fatores como potência de motores, seus passos de operação são os mesmos.

Por essa razão, investigam-se os principais módulos do VV311-K01 se ter uma visão geral desse equipamento e suporte para a modelagem conceitual do desenvolvimento do SMA SADEEM. Posteriormente os artefatos são aproveitados como modelo de negócios para a concepção dos primeiros insumos da metodologia de desenvolvimento de agentes utilizada nesta proposta, partindo da descrição mais detalhada do seu processo de operação. Para inicializar a análise do VV311-K01, apresenta-se uma ilustração do referido equipamento na Figura 3.

Figura 3 – Virador de Vagões VV311-K01.

Na VALE, módulos que compõem a infraestrutura do VV311-K01 são organizados tecnicamente em: Sistema; Conjunto e Itens. Por essa razão, as partes que o constituem estão descritas em: Sistema de *estrutura de giro*; Conjuntos tais como *carro empurrador* (i.e. carro posicionador), *comando do braço*, *comando do carro*, *acionamento do carro*, *acionamento do braço* e *braço trava* e finalmente os

Itens, abrangendo os *motores de corrente contínua*, *redutores*, *sensores*, *pinhões*, *rolamentos* e etc (FONSECA NETO e MOURA, 2003).

A estrutura de giro possui um acionamento de corrente contínua e é um dos sistemas que diferencia o VV311-K01 do VV311-K03. Este último tem seu acionamento controlado por um inversor de freqüência. A Figura 4 apresenta a estrutura de giro do VV311-K01 no momento de descarga de minério.

Figura 4 – Estrutura de Giro do VV311-K01.

A Figura 5 ilustra o carro posicionador. Este conjunto possui autonomia para puxar um total de 120 vagões e é constituído de acionamento em corrente contínua.

Figura 5 – Braço Empurrador

Por outro lado, o braço trava é composto de um sistema hidráulico para manter travados os vagões que se encontram afastados da área de giro do VV311-

K01 e evitar a possibilidade de descarrilamento. A Figura 6 apresenta o conjunto braço trava.

Figura 6 – Braço Trava

A parte interna do VV311-K01 é conhecida como estrutura de barril ou giro. Nesta área, oito travas hidráulicas (i.e. grampos sendo quatro em cada lado) são utilizadas para manter presos os vagões que estão sendo descarregados, já os que estão fora são travados, pelos rodeios, por uma trava auxiliar (i.e. tesoura) instalada na saída do barril do virador, de acordo com a Figura 7.

Figura 7 – Travas hidráulicas - grampos

O VV311-K01 possui duas opções de operação, uma com 180º ou 160º. Os grampos possuem três posições definidas, que são: levantado ao máximo para permitir a passagem da locomotiva, levantado para permitir o avanço dos vagões e

baixo para que no momento da virada dos vagões o mesmo não tombe. O braço empurrador e o braço trava possuem as posições levantado (85º) e abaixado (0º), sendo que na operação dos dois últimos vagões o braço empurrador possui a posição levantada a 20º. O carro posicionador tem as seguintes posições: posição inicial, posição inicial para último vagão, posição final e posição final para último vagão. As travas auxiliares possuem as posições abertas ou fechadas.

A locomotiva posiciona o lote de geralmente 104 ou 102 vagões com o acoplamento do primeiro vagão na direção do braço trava. Da posição onde fica localizado o braço de trava e a estrutura do VV311-K01 existe uma distância equivalente a dois vagões, enquanto que entre o conjunto empurrador (carro posicionador e braço empurrador) em sua posição inicial até o braço trava há uma distância equivalente a um vagão (LCP, 2006). Essas distâncias e posicionamentos estão mais resumidos conforme releva a Figura 8.

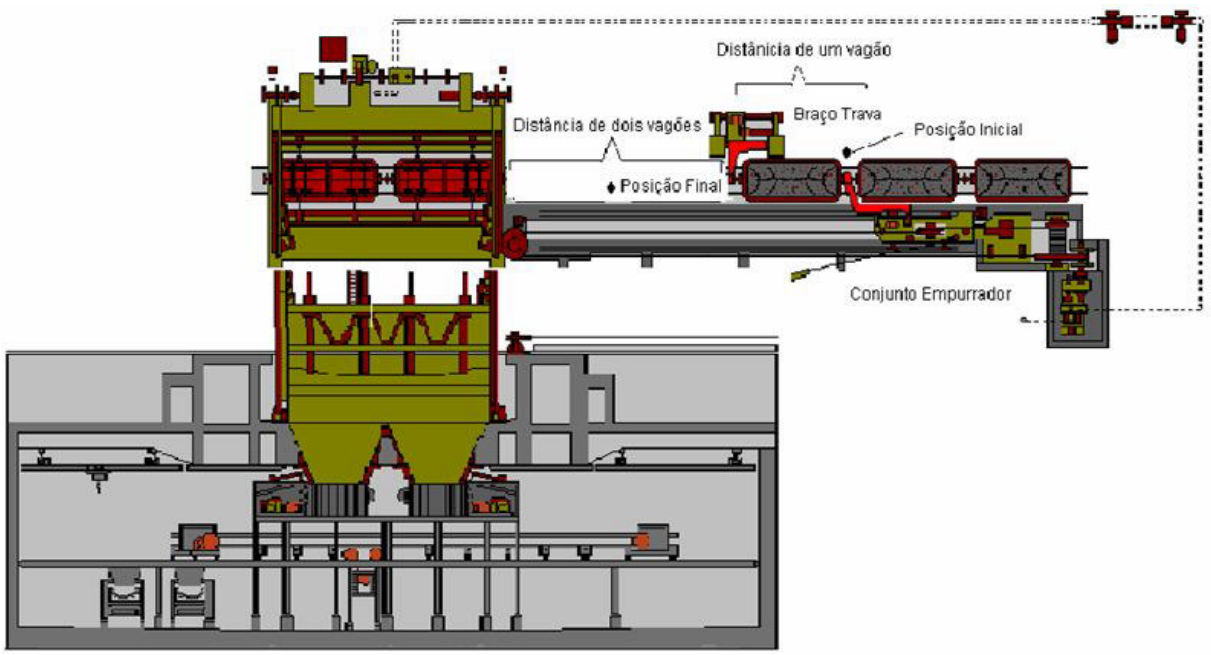

Figura 8 – Posicionamento inicial do carro posicionador e braço trava

Antes da saída da locomotiva, o técnico de operação ferroviária ajusta o lote na posição inicial e executa o dreno de vagões com o auxilio da locomotiva. Essa operação consiste em aliviar o ar do cilindro de freios da composição, já que seu sistema de freios é à base de ar. A realização desta operação evita a sobrecarga no motor do carro posicionador quando este mover a composição de vagões.

A operação é iniciada com os grampos na posição levantada. O carro posicionador procura pela posição de engates dos vagões. Encontrada a posição dos engates, o braço empurrador abaixa até a posição “sobre engate” (abaixado). O carro posicionador avança com a composição até a posição final e então o braço trava abaixa até a posição “braço trava sobre engate”. O braço empurrador levanta até 85º, o carro posicionador retorna até a posição inicial e então o braço empurrador abaixa e o braço trava levanta. No passo seguinte, o carro posicionador avança até a posição final e o abaixa. Neste instante, os dois vagões ficam posicionados dentro do giro e são virados, sendo que os grampos iniciam o movimento de baixar no início do giro. Enquanto o giro está sendo realizado, o braço empurrador levanta. Depois de o giro ter completado 180º e já está retornando para 0^{o}, o carro posicionador retorna para posicionar o próximo par de vagões (LCP, 2006).

No momento em que o giro volta à posição de 0^{o}, o braço empurrador está baixo para que se inicie um novo ciclo de operação que se repetirá até a descarga do penúltimo par de vagões. Neste instante é iniciada uma rotina de operação para descarga dos dois últimos vagões.

Na operação do penúltimo vagão a tesoura é fechada, a trava é levantada, o carro posicionador retorna até a posição inicial para o último vagão. O braço empurrador baixa, a tesoura abre, o carro posicionador avança até a posição final do último vagão, a tesoura fecha, o braço empurrador levanta até 20º, o carro posicionador retorna até a posição inicial do último vagão e o giro é realizado. Depois que o último par de vagões é descarregado, o giro retorna, a tesoura abre, os grampos se levantam ao máximo, o braço empurrador se levanta até 85º, o carro posicionador retorna até a posição inicial e a operação de descarga de um lote é concluída, estando o equipamento disponível para descarregar o próximo lote de vagões.

Para facilitar a descrição da instrumentação do VV311-K01, foi feita uma divisão do mesmo nos seguintes conjuntos: carro posicionador, braço empurrador, braço trava, estrutura de giro e sistema de tesouras. Esses conjutos podem ser visualizados na Figura 9.

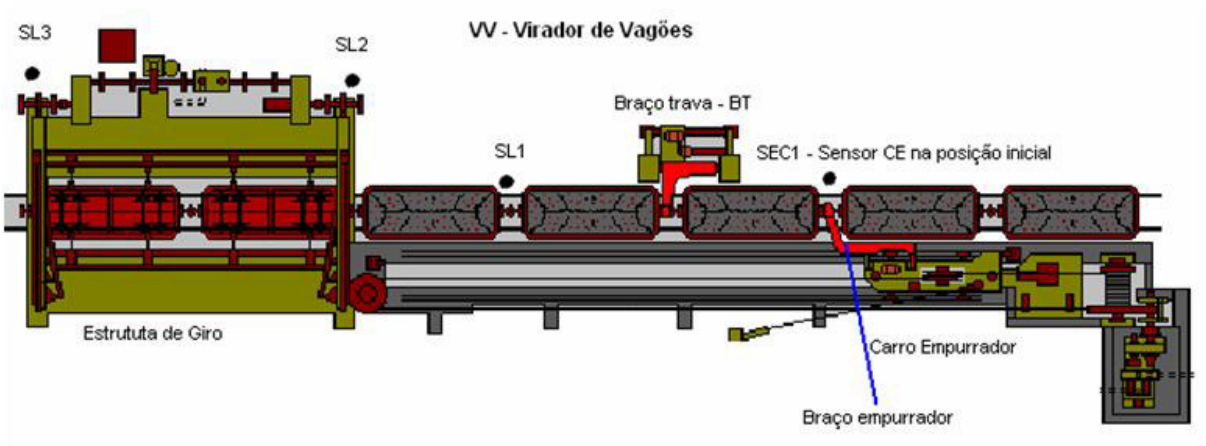

Figura 9 – Módulos do VV311-K01

O VV311-K01 possui instrumentação constituída de sensores indutivos e fotoelétricos, chaves limites eletromecânicas, células de carga, pressostatos, termostatos e chaves rotativas tipo cames. Esta instrumentação faz a função de corte ou parada de movimentos, sinalização de posição, monitoração de temperatura e de pressão, etc.

Nesse sentido, toda a distribuição física e lógica dos equipamentos da área de descarga da VALE representa sua arquitetura de automação onde se procura destacar maior atenção para os dispositivos físicos da instrumentação, tratados nesta pesquisa como elementos que compõem o hardware de tempo real.

2.3 Hardware de Tempo Real

Os elementos do hardware de tempo real revelam um importante meio de comunicação entre o sistema SADDEM e os componentes do VV311-01. É um aparato físico constituído por dispositivos que comandam os viradores através de periféricos tais como sensores fotoelétricos e indutivos, células de carga, pressostatos e termostatos, chaves limites eletromecânicas e chaves cames.

Por meio dos dispositivos e periféricos, os dados provenientes do chão de fábrica são enviados para os equipamentos que executam a coleta de informações relevantes das células de produção. A partir de então, tem-se a composição de toda a gerência de informação de processos. Além disso, os periféricos dos viradores têm um papel significativo para os comportamentos de função de corte ou parada de movimentos, sinalização de posição, monitoração de temperatura e de pressão,

incluindo os aspectos descritos no início desta seção. Nesse sentido, é nesta camada que o sistema deve acessar os dados do VV311-K01.

Para melhor representar a arquitetura do hardware de tempo real, dividiu-se a mesma em níveis, baseando-se no modelo da pirâmide de automação. Este modelo representa a organização do atual momento da tecnologia de comunicação do setor industrial, pois até o inicio da década de 1990, o controle se dava por meio de ilhas de automação onde cada sistema controlava o seu parque sem integração de informação (SOUZA *et al.*, 2005).

Na arquitetura de informação da VALE, a base da pirâmide revela a organização dos sensores e atuadores tais como sensores de nível, pressão, temperatura, de fim de curso, válvulas, inversores de freqüência e etc.

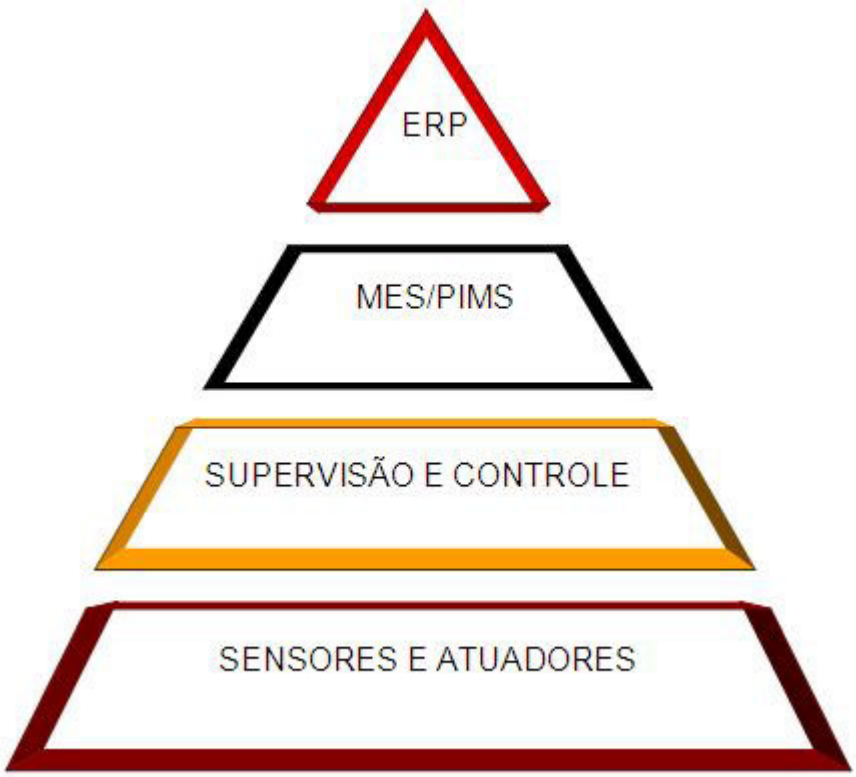

Figura 10 – Pirâmide da automação da VALE.

O nível seguinte apresenta os mecanismos de supervisão e controle, onde estão localizados os Controladores Lógicos Programáveis (CLP ou PLC – *Programable Logic Controler*) e os sistemas de Controle Supervisório e Aquisição de Dados (SCADA). Nesse nível, os sistemas supervisórios se comunicam com os CLPs para a obtenção de dados dos equipamentos de campo da VALE (LCP, 2007).

O próximo nível surgiu graças aos esforços para a disponibilização dos dados de produção, desde o chão de fábrica até a entrega do produto. Assim, pode-se alcançar mais um nível na pirâmide e encontrar os sistemas de gestão de informação de processos normalmente conhecidos como *Enterprise Production Systems* (EPS).

No EPS também se encontram os *Plant Information Managment Systems* (PIMS) e os *Manufacturing Execution Systems* (MES). Alguns estudos apontam existir uma grande confusão no mercado sobre PIMS e MES, mas, mesmo não sendo objetivo deste trabalho entrar no mérito da questão, o que se observa é que PIMS são ferramentas de gerenciamento de informações, já o MES é específico para apenas um tipo de processo.

No topo da pirâmide estão os sistemas corporativos de gerenciamento da planta, *Entrerprise Resource Planning* (ERP), responsáveis pelas informações de negócio. Nesse sentido, a integração dos níveis da pirâmide (i.e. negócio-manufatura) é um fator estratégico para as indústrias que exige intercâmbio constante de informações entre os processos de negócio e manufatura.

No sistema de controle da VALE estão instalados PLCs da Rockwell que recebem informações da planta nos cartões de entrada e enviam comandos, através dos cartões de saída, para os *drives* de acionamentos tais como conversores CA/CC e centro de comando de motores e atuadores. Os PLCs são do modelo SLC500 com CPU 5/05 e recebem via rede *ETHERNET TCP/IP* informações das estações de operação e supervisão, disponibilizando assim os dados para serem lidos.

A programação desenvolvida em *LADDER,* estruturada de maneira que as primeiras malhas são destinadas às falhas, depois aos comandos e finalmente às saídas. O programa é desenvolvido em sub-rotina por movimentos, sendo uma sub-rotina para cada componente (e.g. carro posicionador, giro, travas e etc.) presente no VV311-K01. As malhas de comando foram desenvolvidas de maneira que depende apenas do comando do supervisório para serem fechadas.

Além das sub-rotinas para os movimentos, foram criadas outras para comunicação, automatismo e seqüência da operação normal e de último vagão. O programa foi desenvolvido com uma estrutura definida, onde o primeiro bloco da

sub-rotina é chamado de principal e contém a chamada de todas as outras sub-rotinas.

Já o sistema supervisão e controle do VV311-K01 é composto por quatro clientes, que coletam os dados para o sistema SCADA através do DDE (*Dynamic Data Exchange*) da Microsoft e dois servidores que executam o software *InTouch* da *Wonderware* ambos com o sistema operacional *Windows.* Nas estações do VV-01 há uma placa de rede com padrão *Ethernet* utilizada para a comunicação com o PLC Rockwell e o *software* de supervisão usado é o *InTouch 7.0* (LCP, 2006, LCP, 2008).

Analisam-se também os principais estágios da pirâmide de automação observando o fluxo de dados desde o chão de fábrica até as estratégias de negócios. Dentro destas atividades foi possível atingir um entendimento do processo de identificação das falhas. As falhas são mapeadas em tipos de dados chamados *tagnames* que são declaradas com o seguinte formato: ASC_V10@VV311K01.

O texto antes do @ descreve o elemento de falha e o que fica após este, identifica o equipamento, neste caso, o virador de vagões VV311-K01. Essas *tagnames* compõem as informações *on-line* que descrevem as situações de conjuntos, sistemas e itens do VV311-K01.

2.4 Considerações Finais

Este capítulo apresentou os pricipais aspectos referentes ao sistema de descarga da VALE. A visão geral dos equipamentos dos complexos mineradores foi abordada com destaque para o entendimento do processo e funcionamento das células de produção do TMPM.

Em seguida, descreveu-se o trajeto do minério de ferro desde a sua descaga até estocagem ou embarcação, dedicando-se atenção mais aprofundada ao principal equipamento do setor de descarga de minério, o virador de vagões VV311-K01, importante referencial teórico para o desenvolvimento deste capítulo, obtido por meio de visitas à VALE e pesquisas em fontes documentadas (LCP, 2006, LCP, 2007).

Associado ao rastreamento da arquitetura de automação da VALE, observou-se que VV311-K01 é um equipamento que funciona no chão de fábrica

desta indústria mineradora. Com essa observação, análisou-se a pirâmide de automação da VALE, visando identificar os segmentos de automação que o sistema SADDEM dispõe para interagir com o VV311-K01. Esta análise verificou a viabilidade de aquisição de dados de campo, revelando ainda a forte influência do nível de Supervisão e Controle para as entradas do referido sistema em linha de produção.

O próximo capítulo trata da engenharia de software multiagente com breves considerações sobre o papel da engenharia de software tradicional e as principais terminologias do desenvolvimento oriantado a agentes.

3 ENGENHARIA DE SOFTWARE MULTIAGENTES

Este capítulo aborda a Engenharia de Software tradicional e a Engenharia de Software Multiagentes. Uma introdução sucinta destaca algumas diferenças entre elas e descreve os conceitos e terminologias de programação utilizadas e hierarquia genealógica das metodologias.

3.1 Introdução

O conceito de Engenharia de Software vem descrever uma disciplina que estuda melhorias nos procedimentos de desenvolvimento de sistemas, metodologias, ferramentas e ambiente de desenvolvimento de software (PRESSMAN, 2007). Atualmente, as terminologias de trabalho em que a Engenharia de Software está aplicada têm sido mais discutidas no contexto da programação orientada a objetos, orientada a agentes e orientada a aspectos.

A Engenharia de Software tradicional estuda e desenvolve suas técnicas baseada em sistemas que apresentam comportamentos predeterminados e aplica o vocabulário de programação orientada a objeto. Tudo o que será realizado nesses sistemas é exatamente o que foi planejado, pois utilizam métodos imperativos para a execução de seus programas (ABDELOUAHAB e SILVA, 2006).

Em contrapartida, a Engenharia de Software Multiagentes (i.e. orientada a agentes) revela grandes melhorias em termos de abstrações do mundo real. Além disso, oferece suporte para a representação de uma estrutura organizacional, pois suas técnicas permitem obter maior comunicabilidade no que se refere às necessidades dos usuários, dos produtos gerados nas etapas de projeto e codificação (SILVA e LABIDI, 2007).

Associado a isso, há também uma proximidade dos conceitos reais às abstrações de atores, objetivos, papéis, cooperação e colaboração. No ambiente coorporativo, a Engenharia de Software orientada a agentes se apresenta mais adequada para facilitar os esforços de desenvolvimento de sistemas. Esta afirmação pode ser comprovada quando se pergunta qual dos atuais paradigmas melhor representa as particularidades e percepções mundo real (GIRARDI, 2005).

Massonet *et al.*, (2002), lembram que a Engenharia de Software Multiagentes torna-se interessante por permitir o desenvolvimento de aplicações capazes de atender às necessidades organizacionais, mesclando tanto simplicidade e velocidade quanto comportamentos detalhados e sofisticados por meio de entidades de software altamente modulares, mais conhecidas como agentes.

É importante destacar que antes de se empregar o paradigma baseado em agentes deve-se realizar um estudo prévio para verificar se é apropriado, pois nem sempre pode ser a escolha certa. De acordo com Wooldridge, (2002), o uso de agentes como solução deve ser empregada quando: o ambiente é aberto, ou pelos menos altamente dinâmico, apresentar incertezas e complexidade; os agentes atuam como uma metáfora natural; o controle, dados e *expertise* precisam ser distribuídos e atuarem como componentes autônomos ou semi-autônomos e sistemas legados.

Assim, pode-se considerar que a orientação a objetos resolveu o problema do reuso e manutenção de software enquanto que a orientação a agentes, além de aperfeiçoar os métodos existentes e reduzir a distância cognitiva entre o mundo real e o mundo computacional tem sido vista como uma inovação tecnológica que codifica melhores práticas de como organizar entidades colaborativas e concorrentes (ODELL, 2000).

3.2 Agentes

A definição de agente possui um acervo diversificado e associado a pontos de vistas distintos, apresentando uma variação da visão e concepção que cada autor tem do seu conceito devido a particularidades e funcionalidades do agente em discussão (FARIA, 2004).

Para Sodré (2001), um agente “[...] é uma entidade ativa e autônoma, ou seja, ele pode agir sem a intervenção humana ou outra entidade e tem controle sobre seu próprio comportamento”. Na realidade a autonomia é uma das características que demonstram a principal diferença dos agentes com relação aos objetos, visto que estes só executam aquilo que está predefinido e não são capazes de estimular suas próprias ações.

Para Santos *et al.* (2008), a definição de agente é descrita como uma entidade que realiza uma ação sobre algo. A entidade pode ser humana ou artificial e a ação pode ser a respeito de si ou do ambiente (i.e. percepção) e tudo isso deve produzir um efeito.

Combinando tais características, chega-se a uma síntese de que um agente é aquele que tem independência para atuar e apresenta disposição para perceber suas fronteiras, através de componentes sensitivos, agir sobre esse ambiente e interagir com outras entidades, nele contidas, por meio de atuadores. A Figura 11 resume a interação de um agente com o seu ambiente.

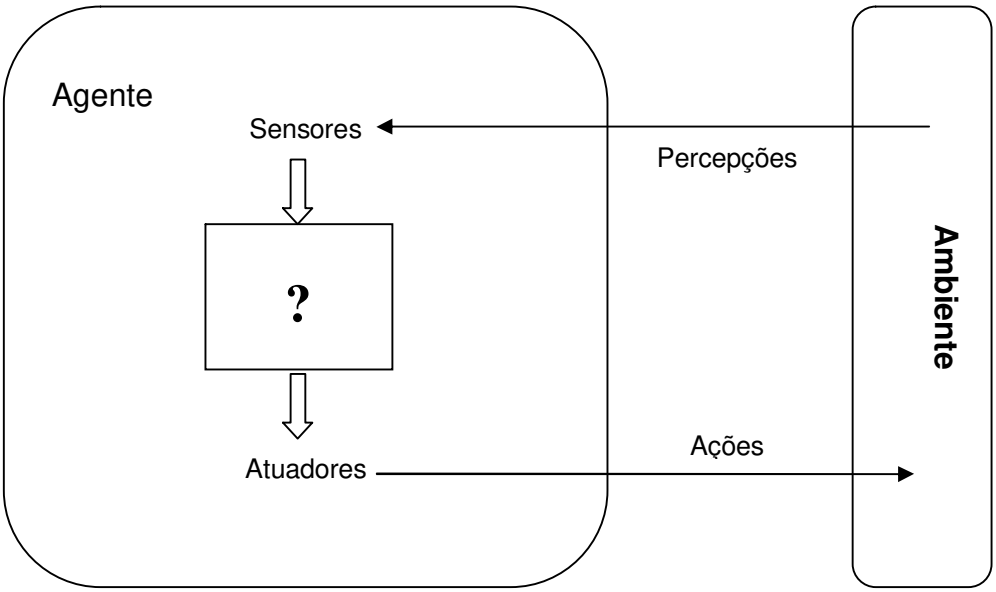

Figura 11 – Interação de um agente com o seu ambiente (Russell e Norvig, 2004).

Visualizando-se os elementos da Figura 11, para uma comparação entre um agente humano e um agente robótico, ao primeiro podem-se associar os sensores aos olhos, ouvidos, olfato e etc. Os atuadores são representados pela boca, mãos e pernas, além de outros. Ao segundo, podem ser associados câmeras, dispositivos de detecção de presença, como sensores, e motores que para a locomoção funcionam como atuadores.

As entradas perceptivas ou sensórias de um agente podem ter origem em uma seqüência de teclas digitadas, pacotes transmitidos na rede, o conteúdo de um arquivo gerado pelo supervisório em determinado instante. Toda a história um

agente pode ser obtida, pois a seqüência de suas ações e tudo o que este já percebeu são referenciadas justamente a essas entradas durante o seu trabalho.

No ambiente da Figura 11 estão embutidos os problemas para os quais os agentes devem apresentar soluções. Esse ambiente é fortemente estudado quando se vai projetar um agente, principalmente porque os demais itens que os caracterizam estão agrupados para formar um ambiente de tarefas, normalmente conhecido como PEAS (*Performance*, *Environment*, *Actuators and Sensors* – Performance, Ambiente, Atuadores e Sensores).

Russel e Norvig (2004) reforçam a importância do ambiente de tarefas descrevendo as propriedades mais relevantes em categorias da seguinte forma:

- Completamente observável vs Parcialmente observável – se os sensores de um agente permitem ingresso completo ao estado do ambiente, a cada fração de tempo, dizemos que este é completamente observável.

- Determinístico vs Estocástico – quando o próximo estado do ambiente e ação do agente é completamente conhecida, diz-se que o ambiente é Determinístico. Não há preocupações com incertezas, caso contrário ele é Estocástico. O ambiente determinístico pode ser estratégico quando este não leva em conta as ações executadas por outros agentes.

- Episódico vs Seqüencial – quando a experiência do agente é dada em divisões chamadas episódios, que por sua vez constituem a percepção e execução de uma única ação daquele, diz-se que o ambiente é Episódico. No ambiente seqüencial, quando há uma decisão, a próxima pode ser influenciada pela anterior.

- Discreto vs Contínuo - o ambiente é discreto quando há uma limitação na diversidade de estados, caso contrário ele é contínuo.

- Único vs Multiagente – o ambiente pode ser ocupado por um único agente ou por vários agentes, mas segundo Russelll e Norvig (2004), a análise dos dois ambientes depende de questões que envolvem competição, cooperação, comunicação e comportamento estocátisco, para que o projeto de agente possa melhor tratar as sucintas diferenças existentes.

Nesse sentido, é instrutivo considerar que a ação de um agende depende do conjunto de percepções que ele captura no ambiente e seu relacionamento com outros agentes em função de um momento específico. Depende também do mundo que o agente vive, pois se este for simples pode-se descrever tudo o que acontece.

Considerando-se a incapacidade de estabelecer, naturalmente, qual será a atitude a ser tomada por um agente, em um ambiente complexo, e sim as condições, pois se isso fosse possível, conhecer-se-ia quase tudo sobre ele (RUSSEL e NORVIG, 2004), para reforçar essa ideia é importante visualizar os agentes como artefatos de software que apresentarem traços com as seguintes propriedades:

- Autonomia – o agente é autônomo quando é capaz de controlar suas próprias ações;
- Reatividade – o agente responde a mudanças ocorridas no ambiente;
- Sociabilidade – os agentes possuem canais de comunicação uns com os outros, por meio de normas e regras de interações.
- Pró-atividade – capacidade de antecipar suas ações, independente do que ocorrer no ambiente, para alcançar suas metas.
- Aprendizagem – capacidade de aprender analisando os sucessos e os fracassos, de acordo com as ações realizadas.

A maioria dos agentes pode possuir ou não uma ou mais propriedades para executar suas tarefas. Em razão disso, alguns estudos apontam para a classificação de agentes em: Fixos, Móveis, Inteligentes e etc. A seção a seguir apresenta os principais tipos de agentes.

3.3 Tipos de agentes

Diversas disciplinas envolvidas com o estudo da tecnologia de agentes têm contribuído com ideias, pontos de vista, métodos e classificações. Nesta seção, concentra-se a contribuição dada à classificação de agentes por apresentar aspectos de maior importância em termos de mobilidade, relacionamento inter agentes e competências de raciocínio, buscando uma melhor compreensão das categorias de agentes e suas particularidades (SANTOS, 2007).

Agentes Móveis – possuem a mobilidade como principal característica, seu ambiente é distribuído e com isso é capaz de se movimentar, por exemplo, em uma *Intranet* ou até mesmo na *Web*. Com a demanda por heterogeneidade e outros aspectos sócio-econômicos o uso desses agentes vem aumentando principalmente para auxiliar as tomadas de decisões em grandes volumes de informação.

Agentes Reativos – possuem reflexos sem qualquer conhecimento prévio do que já foi realizado no passado no ambiente no qual atuam, muito menos da ação que deve ser tomada posteriormente. A autonomia desses agentes é somente a execução.

Agentes Cognitivos – reagem no ambiente de acordo com um histórico de agente, agrupam estratégias de adaptação sofisticadas. Diferentemente dos agentes reativos, podem raciocinar sobre suas ações tomadas no passado e planejar como deverá agir no futuro. Têm capacidade de seguirem planos e metas para alcançar o seu propósito. O "*know how"* de um agente cognitivo está condicionado a uma base de conhecimento contendo fatos e restrições do ambiente de tarefas que o mesmo atua.

Vale ressaltar que esses aspectos não apresentam suporte suficiente para a construção de uma sociedade de agentes. Adicionalmente, é preciso que estes sejam coordenados trabalharem de modo cooperativo e competitivo. Esse equilíbrio entre agentes coordenáveis e seus relacionamentos, associado às demais características descritas, deu origem aos Sistemas Multiagentes.

3.4 Sistemas Multiagentes

Com a velocidade da tecnologia e aumento da dificuldade para resolver problemas complexos, destacou-se há pouco que a Engenharia de Software evoluiu para um novo paradigma: O paradigma orientado a agentes. Este aborda uma metodologia de desenvolvimento envolvendo o trabalho colaborativo e cooperativo com aspectos comportamentais, sociais e cognitivos (SAMPAIO *et al.*, 2007).

Tais elementos estão presentes em estruturas de software chamadas agente. Todo agente possui um conjunto de atributos associados ao seu papel na sociedade que está inserido. Um sistema desenvolvido com estas estruturas é

chamado de Sistema Multiagentes. Em um SMA os agentes atuam de forma autônoma e com habilidade social para a comunicação com os demais agentes.

Particularmente, quando se fala de SMA, está se tratando de um ambiente de tarefas Multiagente. A sociedade que será composta por esses agentes descreve quais estruturas de software devem ser visualizadas como agentes, pois de repente pode ser indagado se um agente piloto de fórmula 1 deve visualizar o próprio carro como um agente ou um objeto que possui comportamento estocástico, tal como as folhas espalhadas pelo vento ou como o comportamento do tráfego.

Para uma questão sutil como essa, a distinção fundamental é identificar qual dos comportamentos possui descrição mais relevante em termos de desempenho. No caso da fórmula 1, um piloto A está sempre tentando maximizar sua medida de desempenho e acaba minimizando a dos demais pilotos, considerando-se um cenário de sucesso, caracterizando assim um ambiente competitivo.

No caso de um agente motorista de táxi uma descrição mais relevante poderia permitir maximizar o desempenho da sociedade para evitar acidentes. Além disso, há também um aspecto competitivo se qualquer um deles estiver disputando uma vaga no estacionamento. Assim, no primeiro caso o ambiente tem característica cooperativa, enquanto que no segundo há uma característica competitiva.

Conforme o que foi visto, o ambiente de tarefas pode sinalizar vários aspectos que devem ser analisados tanto em sistemas com um único agente quanto naqueles com vários agentes. A comunicação, por exemplo, tem mais influência em ambientes multiagentes e normalmente é tratada como comportamento racional. Outro comportamento geralmente associado à racionalidade é o estocástico, pois este é capaz de evitar armadilhas da previsibilidade (RUSSELL e NORVIG, 2004).

3.5 Arquiteturas de Sistemas Multiagentes

A arquitetura de um SMA fornece meios para entender a maneira como um sistema está implementado. Sua finalidade é ter uma visão das propriedades e estruturas dos agentes, descobrindo de que forma os agentes do sistema interagem e asseguram sua funcionalidade em termos de cooperação e coordenação (RUSSEL e NORVIG, 2004).

A cooperação e coordenação conduzem ao planejamento do SMA. A cooperação está associada à ideia de compartilhamento de tarefas e informações. A coordenação assegura os mecanismos de gerenciamento e organização de um grupo de agentes, que certamente é alcançada por meio da comunicação.

A comunicação é uma propriedade fundamental para a cooperação e coordenação e deve seguir uma linguagem comum a todos os agentes da sociedade. Exemplos dessas linguagens são a ACL (*Agents Common Language*), KQML (*Knowledge Query and Manipulation Language*) e a FIPA-ACL (*Foundation for Intelligent Phisycal Agents*) que serão tratadas na seção 3.8.

O tipo de comunicação pode ser direto, considerando-se um ambiente onde os agentes tenham conhecimento uns dos outros e não precisam de intermediador para estabelecer troca de informações entre as partes, ou indireto se os agentes não se conhecem e toda a comunicação é baseada em uma área compartilhada. Além disso, outras características tais como tipologia, mobilidade e regras dos agentes também definem as formas de comunicação.

A especificação do processo de comunicação pode ser influenciada quando a sociedade apresenta ou não a mesma arquitetura de agentes, ou seja, sociedades homogêneas ou heterogêneas. Adicionalmente, os agentes da sociedade podem se apresentar com atuação fixa ou migratória, definindo um tipo de comunicação fechado ou aberto do ponto de vista da mobilidade dos agentes.

Outra característica que trata das questões sociais, tal como ocorre no mundo real, diz respeito às regras de comportamento. No ambiente multiagente pode haver regras explícitas de comportamento definidas para toda a sociedade, estabelecendo uma comunicação baseada em leis, ou até mesmo não existem regras explícitas classificando a comunicação como não-baseada em leis.

O processo de comunicação para a arquitetura multiagente é um assunto que deve ser bastante analisado, pois é nele que as interações, cooperação e colaboração se apóiam. Essa premissa dá embasamento para cooperação classificar as arquiteturas de SMAs em Quadro-Negro (*blackboard*), Troca de Mensagens e Federativa, ao passo que a coordenação estabelece a classificação das arquiteturas de SMA em Mestre-Escravo e Mecanismo de Mercado.

Nesse sentido, há duas possibilidades de se definir a arquitetura de um SMA. A primeira se baseia nos objetivos e planos conjuntos e a segunda no estabelecimento de acordos para coordenar atividades interdependentes, porém estas arquiteturas serão definidas de forma geral, descrevendo a maneira como o sistema implementa suas estruturas e propriedades e como ocorrem as interações entre os seus agentes em cada uma delas.

Na arquitetura Quadro-Negro os agentes se comunicam de forma indireta por meio de uma área compartilhada persistente chamada repositório. Agentes “escrevem” o pedido no quadro e esperam que os outros agentes “leiam” e respondam. A Figura 12 esboça um exemplo de arquitetura Quadro-negro.

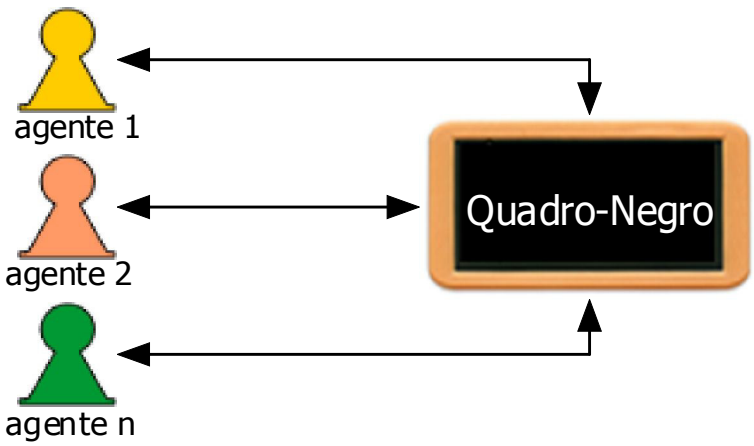

Figura 12 – Arquitetura Quadro-Negro

Uma desvantagem desse tipo de arquitetura pode ser a demora no processo de gravação e recuperação das mensagens, principalmente para ambientes com tempo de resposta crítico.

Na arquitetura Troca de Mensagens os agentes se comunicam através de mensagens, como o próprio nome sugere. A forma de comunicação é direta, pois cada agente sabe o nome e o endereço dos demais agentes da sociedade. Não existe um intermediário como na arquitetura Quadro-Negro, mas pode haver a presença de um agente facilitador de comunicação.

As interações da arquitetura Troca de Mensagens são organizadas em protocolos que ditam e impõem o formalismo adequado das etapas de conversação entre os agentes e assim a comunicação seja estabelecida de forma compreensível. O custo desse tipo de arquitetura pode ser caro quando há um crescimento

exponencial na troca de mensagens, pois devem ser modeladas todas as possibilidades de interação.

A arquitetura Federativa foi elaborada para solucionar os possíveis problemas de mensagens em *broadcast* enviadas em ambientes com um número muito grande de agentes. Nessa arquitetura os agentes são reunidos em federações baseados em um critério de agrupamento escolhido.

Em cada federação há um agente que facilita a entrega da mensagem para o seu destinatário verificando se a mesma foi encaminhada para algum agente de sua federação. A presença do agente facilitador evita que uma mensagem seja enviada para todos os agentes.

Na arquitetura Mestre-Escravo os agentes são classificados como gerentes (mestres) e trabalhadores (escravos). O agente gerente é o coordenador de um grupo de agentes trabalhadores para os quais distribui as tarefas. É comum a presença de um agente facilitador se os agentes formarem grupos.

Na arquitetura Mecanismos de Mercado os agentes ocupam o mesmo nível de atuação e, portanto, conhecem as atribuições que cada um é capaz de realizar. O propósito deste tipo de arquitetura é reduzir a quantidade de mensagens trocadas, pois todos são conhecidos. Os agentes do Mecanismo de Mercado são classificação em produtores e consumidores e são importantes para maximizar lucros em um processo de negociação (GIRARDI e MARINHO, 2007).

Analisar as principais propostas de arquiteturas de software utilizadas na construção de SMAs foi essencial para a observação das possíveis limitações que podem surgir durante o desenvolvimento do sistema proposto neste trabalho. Baseando-se na relação da arquitetura com a solução global e detalhada do projeto como um todo, essa investigação procurou ponderar as vantagens e desvantagens presentes nas diferentes abordagens para instanciar um modelo arquitetural mais apropriado ao problema em questão.

3.6 Desafios do Desenvolvimento Orientado a Agentes

Tiveit (2001) explica que o desenvolvimento orientado a agentes ainda apresenta problemas específicos da tecnologia. Wooldrige (2000) acrescenta que há

obstáculos que ainda devem ser superados para que a Engenharia de Software orientada a agentes se torne popular.

Na visão do autor, um dos obstáculos diz respeito à organização do relacionamento dos agentes com outros paradigmas. Não há muita clareza de como se dá o convívio de agentes com outras terminologias de programação tais como odesenvolvimento orientado aobjetos.

Outro entrave citado por Wooldridge compara as metodologias de desenvolvimento orientado a agentes descrevendo o pouco consenso que há entre elas e praticamente nenhum acordo entre suporte técnico que a metodologia é capaz fornecer, além dos conceitos do mundo real que tem competência para representar.

A escalabilidade também é outro obstáculo. Deve alcançar um entendimento de como fazer a engenharia segura e previsível de sistemas que possuem um grande número de agentes interagindo dinamicamente. A falta de reastreamento de requisitos também é considerada por Woodridge como um obstáculo, pois gerenciar as constantes mudanças nas funcionalidades por parte do cliente em todo processo de desenvolviemnto de SMAs.

Adicionalmente aos problemas específicos da tecnologia de agentes, tais como imprevisibilidade dos padrões e efeitos de suas interações, - que são decorrentes de sua característica autônoma - requisições que podem ser tanto simples quanto prolongadas, comportamento emergente e etc, há também desafios comuns que surgem quando uma nova tecnologia é utilizada. Segundo Tivet (2001) tais desafios são classificados em cinco categorias: *Política*, *Conceitual*, *Análise* e *Projeto*, *Nível de Agente* e *Nível de Sociedade*.

Na categoria *Política* ainda é preciso superar o excesso de entusiasmo ao se propor soluções baseadas em agentes. Normalmente, há um fracasso resultante de uma falta de entendimento da aplicação adequada dos agentes, principalmente por esta tecnologia revelar uma abordagem inovadora, apresentar uma importante maneira de se conceituar e implementar software o que em muitas situações não se dá importância às suas restrições.

Nessa categoria também está incluída a adoção dos agentes como um dogma o que, apesar de existirem diversas aplicações utilizando agentes, nem sempre se pode solucionar problemas com agentes. Há situações em que as terminologias convencionais de desenvolvimento são mais adequadas, como por exemplo, a orientação a objetos.

A categoria *Conceitual* destaca aspectos tais como métodos confiáveis de desenvolvimento e soluções padronizadas. É nesse sentido que as práticas de engenharia de software devem ser analisadas para que não haja falha do projeto como um todo, caso isso seja ignorado, já se sabe que a falha não foi originada por problemas específicos de agente.

Há também uma forte tendência de se esquecer que o sistema que está sendo desenvolvido pode conter várias linhas de execução, as chamadas *threads*, que permitem trabalhar com diferentes fluxos de controle em um único processo. Por essa característica, os SMAs são classificados como ambiente *mutithreaded*. Os problemas de programação concorrente continuam a existir. Portanto, o consumo de processador e memória e desempenho do sistema não devem ser ignorados.

A categoria *Analise e Projeto* descreve as dúvidas de uma solução orientada a agentes, pois não é perceptível como os agentes podem ser usados para trazer melhorias para um produto ou até mesmo como aqueles podem se tornar aptos para criar novas linhas de produtos. Pode haver também um mau entendimento sobre o planejamento de uma arquitetura genérica para uma família de aplicações que na realidade só atende um único problema.

Além disso, é importante frisar que o paradigma de agentes ainda não foi testado em sua essência e é comum equipes de desenvolvimento considerarem como solução única. A pesquisa sobre agentes é um tópico aberto e há bons argumentos que apontam essa tecnologia como solução para software complexo, mas quantitativamente falando, ainda não há comprovação suficiente.

Outro problema que normalmente pode ocorrer é o desenvolvimento de uma arquitetura própria de agentes. Segundo Tivet, (2001) este é um erro considerado gravíssimo, pois devem ser pesquisadas as várias arquiteturas de agentes existentes, bem como os padrões de projeto global e de unidade de agentes que

atendam os requisitos do sistema, cultivando a seleção, adaptação e composição de soluções de software que já foram testadas.

Na categoria *Nível de Agente*, problemas com o excesso de Inteligência Artificial podem sobrecarregar o *framework*, principalmente se a arquitetura for genuinamente deliberativa que geralmente apresentam técnicas de processamento de linguagem natural, planejadores, base de conhecimento e etc.

Por fim, a categoria *Nível de Sociedade* aborda a ausência de uma preocupação com a concorrência comentada nas categorias anteriores. Por exemplo, se o sistema apresenta um único fluxo de controle, portanto deve-se questionar: A solução baseada em agentes é realmente útil para este caso? Em contrapartida, deve-se evitar usar agentes para tudo?

É importante reconsiderar as outras categorias que demonstram possíveis falhas de projetos decorrentes de sobrecarga gerada para gerir os agentes e troca de mensagens entre eles o que, em muitos casos, tem mais impacto nos próprios benefícios que se pretende alcançar com uma solução baseada em agentes. Nesse sentido, é indispensável ter um projeto que explore suficientemente o potencial que os agentes oferecem sem que haja carga de agência e comunicação excessivas.

O estudo destes desafios tem como intuito, reconhecer as armadilhas que podem causar surpresas desagradáveis tanto no paradigma orientado a agentes quando no orientado a objetos. Portanto, não faz parte do escopo desta dissertação indicar que uma abordagem é mais propensa a erro do que a outra.

3.7 Implementação de Agentes de Software

Os Sistemas Multiagentes têm uma natureza diferente do que tradicionalmente se observa na informática clássica. Por essa razão, os mesmos não podem ser desenvolvidos com essas terminologias de trabalho, pois aqueles apresentam restrições quanto à distribuição e interoperabilidade (SILVA *et al.*, 2005).

Foi considerando independência de aplicação e criando meios para facilitar o desenvolvimento desses sistemas, que surgiram os chamados *frameworks*. Normalmente se ouve falar de *framework* e plataforma indiscriminadamente, mas

cada um tem o seu propósito durante a concepção de um Sistema Multiagente, embora estejam sempre juntos. Um *framework* é um esqueleto de software que agrupa uma estrutura genérica de códigos.

Por outro lado, uma plataforma é o local onde o *framework* será instanciado, estendido ou complementado. Resumindo, o *framework* é o farol que indica o caminho a seguir, mas que por si só, não é suficiente para executar um Sistema Multiagente. Para isso, é necessária uma plataforma de desenvolvimento.

Pelo fato de o *framework* oferecer todas as funcionalidades essenciais de um SMA, soluções assim desenvolvidas possuem abordagem horizontal ou *middleware*, pois é possível que um projetista se preocupe apenas com o design dos agentes, ao passo que a abordagem vertical soluções *ad-hoc* específicas são codificadas (SILVA, 2005).

3.8 Especificação da FIPA

Na década de 80 os primeiros empreendimentos para a criação de *frameworks* começavam a aquecer o interesse de pesquisas em SMA, porém somente quando a bolha da internet explodiu foi que a comunidade acadêmica visualizou que a rede global seria o espaço ideal para o surgimento de vários SMAs, devido ao dinamismo e crescimento associados à dificuldade dos usuários em recuperar e filtrar suas necessidades informação e serviços específicos.

Mesmo com o grande aumento da rede, ainda existia o problema de padronização. Um obstáculo para o cenário de comunicação que se consolidava e impedia, desse modo, a interoperabilidade entre os sistemas. Uma fundação sem fins lucrativos, vinculada à IEEE (*Institute of Electrical and Eletronics Engineers*), começou a trabalhar na padronização e interoperabilidade de agentes em ambientes híbridos, hoje conhecida como *Foundation for Intelligent Phisycal Agents*.

Apesar da primeira divulgação das especificações da FIPA terem ocorrido no final da década de 90, somente em 2002 elas foram finalizadas e reconhecidas como um *standard*. As especificações da FIPA trouxeram importantes contribuições para o desenvolvimento de *frameworks* genéricos (e.g. FIPA-OS e JADE), assegurando que os componentes de diferentes SMAs pudessem coexistir e trabalhar em conjunto formalizados por um padrão comum.

A FIPA tem um compromisso em promover o aperfeiçoamento das tecnologias para a interligação entre as aplicações buscando a interoperabilidade entre os sistemas autônomos, tanto na indústria quanto no comércio. É importante destacar que, além de uma linguagem de comunicação, a FIPA descreve serviços e instruções para a qualidade e controle dos principais agentes que participam do gerenciamento do sistema, para a ontologia e para o nível de transporte dos protocolos.

Em termos de implementação, o padrão FIPA fornece um conjunto de especificações apenas como um modelo de referência. Como se trata de uma descrição em alto nível, em comparação a uma linguagem de programação, limita-se ao comportamento externo dos componentes do sistema. Os detalhes relativos à estrutura interna e codificação ficam a critério do desenvolvedor.

Para melhor demonstrar os serviços que devem ser fornecidos pelas plataformas compatíveis com o padrão FIPA, observa-se que os mesmos estão descritos em normativos e opcionais, conforme a Figura 13.

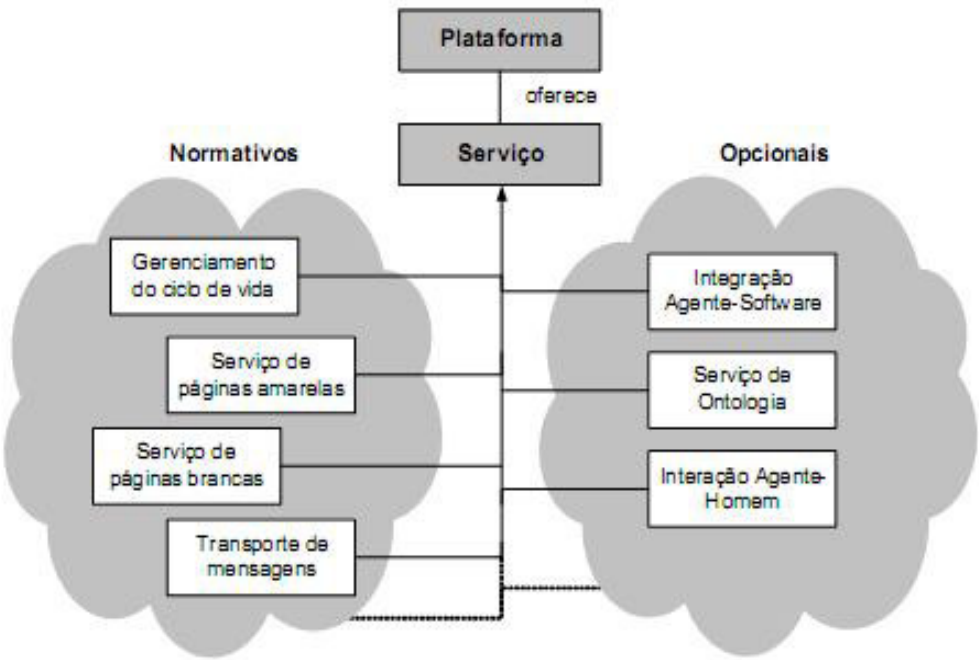

Figura 13 – Serviços fornecidos pelas plataformas compatíveis com o padrão FIPA (SILVA, 2005)

Em uma plataforma, é imprescindível que haja um serviço que gerencie a criação, a retirada e o intercâmbio de agentes entre plataformas. Tais eventos ocasionam mudança de estados, como por exemplo, quando um agente é criado,

devendo-se atribuir um identificador único a ele para permitir que outros agentes possam encontrá-lo sem ambigüidade. Daí a responsabilidade do serviço de gerenciamento do ciclo de vida.

Para permitir que um agente tenha acesso a outros agentes capazes de fornecer determinado serviço, o padrão FIPA especifica o serviço de páginas brancas. Caso a procura seja pela lista de serviços oferecidos, é o serviço de páginas amarelas que tem competência para atender o agente. Os agentes têm liberdade para efetuar seu cadastro, e até mesmo alterá-lo ou removê-lo, em qualquer um destes serviços.

Para a interação dos agentes o padrão FIPA descreve um dos mais importantes serviços da plataforma: o serviço de transporte. São de responsabilidade deste serviço às entregas e troca de mensagens entre os agentes, independente de estarem na mesma plataforma, além de instruções para a implementação de meios de segurança no transporte de mensagens.

Além dos serviços normativos, há também especificações facultativas para as plataformas que seguem o padrão FIPA. Essas opções estendem as capacidades que cada plataforma pode empreender para contribuir ainda mais com a produtividade no desenvolvimento de sistemas.

Os serviços de integração Agente-Software e Interação Agente-Homem têm sido bastante usados em pesquisas que implementam agentes pedagógicos para humanizar as interfaces de ferramentas para o ensino à distância e presencial, melhorando a percepção do aluno em relação conteúdo (RIBEIRO *et al.*, 2007).

Dentre os serviços opcionais, destaca-se com maior ênfase o serviço de ontologia, usada como um repositório contendo as ontologias utilizadas pelos agentes do sistema a fim de evitar ambigüidade no conhecimento compartilhado entre os agentes.

É importante ressaltar que a FIPA também propõe um modelo de plataforma padrão, um tipo de arquitetura que serve de referência para as plataformas, conforme mostrado na Figura 14 (SILVA e FURTADO, 2003).

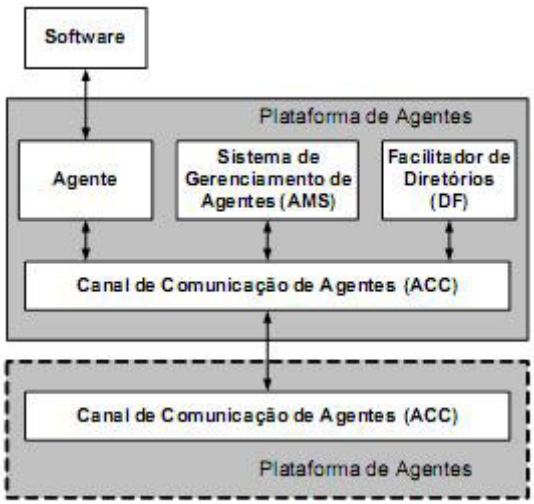

Figura 14 – Modelo de plataforma padrão definido pela FIPA (SILVA, 2005).

Na composição do modelo da Figura 14 observa-se a representação do agente propriamente dito que, conforme o padrão FIPA, deve obter suas tarefas de acordo com a definição do propósito da aplicação. Todo o processo de comunicação do agente é realizado por meio de troca de mensagens, tendo relacionamento dentro e fora da plataforma.

O sistema de gerenciamento de agentes (AMS – *Agent Managment System*) é um agente que tem responsabilidades de supervisão de acesso e utilização da plataforma multiagentes e deve existir somente um. Os serviços de páginas brancas e ciclo de vida da Figura 13 são fornecidos por este agente através da gestão de um repositório para identificação de agentes (AID - *Agent Identification*) e outro para os próprios agentes. O AID válido só é obtido após o registro do agente no AMS.

O facilitador de diretórios (DF - *Directory Facilitator*) é outro agente presente no modelo com a finalidade de prover serviço de páginas amarelas visto na Figura 13. Diferente do AMS, plataforma suporta outros DFs e quando isso ocorre, são organizadas as federações de agentes.

O canal de comunicação dos agentes (ACC – *Agent Communication Channel*), também conhecido na literatura como sistema de transporte de mensagens, é o agente que empreende toda a comunicação interna e externa à plataforma, sendo que este canal de comunicação é utilizado até mesmo pelo AMS e DF (ZHOU, 2006).

A FIPA tem grande preocupação com a criação das especificações para SMAs e principalmente a atualização dessas descrições e por essa razão definiu um ciclo de vida destacando as fases por que passam as definições do Padrão reveladas por meio da Figura 15. Uma especificação percorre as fases do ciclo de vida para atingir a maturidade e quando isso ocorre à mesma atinge status de padrão.

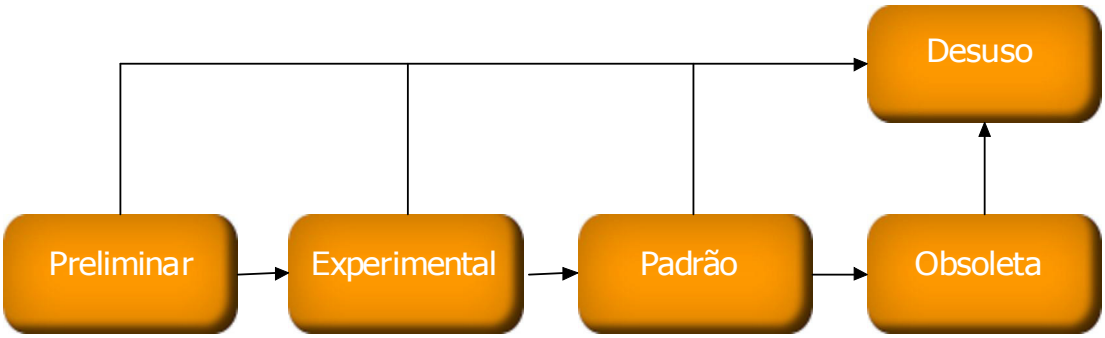

Figura 15 – Ciclo de vida das especificações FIPA.

O primeiro trabalho da FIPA para elaborar uma especificação teve como resultado a linguagem FIPA-ACL (*Agent Communication Language*) que tinha como propósito a padronização de um protocolo de comunicação usado pelos agentes baseado na teoria dos atos de fala ou comunicativos. Nessa especificação, a comunicação de agentes é dividida em três partes:

FIPA *Interaction Protocols* (IPS) – tratam de protocolos para o intercâmbio de pré-estabelecidos para mensagens ACL;

FIPA *Communicatives Act* (CA) – especificações que tratam das diferentes expressões para mensagens ACL;

FIPA *Content Language* (CL) – especificações que tratam das diferentes representações dos conteúdos das mensagens ACL. Maiores detalhes sobre a especificação completa do padrão podem ser encontrados em http://www.fipa.org/specs/aclpecs.tar.gz.

O formato das mensagens trocadas entre os agentes foi definido pela FIPA compreendendo campos específicos para: o agente transmissor da mensagem; receptores da mensagem; ato de comunicação – normalmente chamado de performativa –; conteúdo da mensagem; linguagem utilizada para representar o

conteúdo; ontologia e outros campos para o tratamento de comunicação concorrente. A Figura 16 ilustra o formato de uma mensagem FIPA.

```
(CFP
    :sender Supervisor Agent
    :receiver Interface Agent
    :content (<Plans>
                <planname id="3A"/>
                  <plandetail>
                       Medir vibração
                  </plandetail>
              </plans>)
    :language XML
:ontology plans)
)
```

Figura 16 – Formato de uma mensagem FIPA-ACL

A performativa de comunicação de uma mensagem FIPA pode ser REQUEST se o transmissor deseja que o agente receptor realize alguma ação. Se o agente transmissor quer informar o receptor a respeito de alguma ocorrência, a performativa utilizada é INFORM. QUERY_IF é utilizada quando há indagação de um agente a outro sobre a veracidade de uma proposição. CPF quando há solicitação de propostas. As performativas PROPOSE, ACCEPT_PROPOSAL e REJECT_PROPOSAL quando os agentes estão negociando.

```
(Situacao :equipamento "vv311-k01"
          :sistema "giro"
          :conjunto "acionamento"
          :modo_falha "vibracao"
          :causa "rolamento sujo"
)
```

Figura 17 – Linguagem de conteúdo FIPA-SL

Há uma grande preocupação com a semântica dos atos comunicativos, pois como a FIPA-ACL inicialmente não representava naturalmente o conteúdo da mensagem, foi criado o FIPA-SL (*Semantic Language*) que se baseia em lógica de primeira ordem, permitindo aos agentes expressar relacionamento entre os objetos de um domínio particular e suas características definindo o conteúdo através de uma ontologia. Um exemplo de representação do conteúdo com a linguagem FIPA-SL é mostrado na Figura 17.

Neste trecho, o conteúdo da mensagem revela que a situação do conjunto de acionamento do equipamento VV311-K01 apresenta um modo de falha (e.g.

evento que causa uma falha funcional) ocasionado pelo rolamento sujo. O conteúdo dessa mensagem é trocado entre os agentes que resolvem os problemas relacionados com a situação das unidades, divididas em *sistema*, *conjunto* e *ite*m, de um equipamento em particular (SCAPIN, 2007, SMITH, 2007).

Além da FIPA-ACL há também a linguagem KQML (*Knowledge Query and Manipulation Language*), desenvolvida pelo KSE (*Knowledge Sharing Effort*). É uma linguagem com que contém um protocolo de comunicação de alto nível para a troca de mensagens independente de conteúdo e da ontologia.

As sintaxes da FIPA-ACL e KQML são semelhantes, contudo não é objetivo desta dissertação abordar a KQML, ressaltando-se ainda que, mesmo sendo a FIPA-ACL o padrão estabelecido, a KQML tem se apresentado de forma significativa a uma boa parcela das implementações.

3.9 Plataformas de desenvolvimento

O desenvolvimento de SMA possui dificuldades que são peculiares à própria complexidade da tecnologia e foi por esse motivo que surgiram plataformas e *frameworks* genéricos. A maioria deles segue as especificações FIPA buscando a simplicidade de construção e interoperabilidade.

Atualmente, existem várias plataformas que fornecem assistência ao desenvolvimento de SMA. Os que apresentam maior destaque e importância comercial são: JACK (COBURN, 2001), FIPA-OS (FIPA-OS, 2005) e JADE (BELLIFEMINE *et al*., 2007).

JACK *Intelligente Agents* (JACK, 2008) é inteiramente escrito em Java e possui um ambiente orientado a agentes. Há uma extensão em código Java específica para implementação dos comportamentos e um rico suporte para a construção de agentes com arquitetura BDI (*Beliefers Desires Intentions*) (RAO, 1996). A programação de agentes BDI no JACK conta com cinco métodos construtores que são: agentes, capacidades, relações da base de dados, eventos e planos.

As capacidades são um composto de eventos, planos, base de dados e até mesmo outras capacidades, sendo que cada uma delas revela determinada função

combinada ao agente. A base de dados possui relações que armazenam crenças e dados de um agente. Os eventos identificam as circunstâncias e mensagens que um agente pode responder. Os planos são instruções que um agente segue para perseguir suas metas, ao mesmo tempo em que tratam os eventos aplicados a eles. (SILVA *et al.*, 2005).

FIPA-OS é um *framework* resultante da primeira codificação livre dos padrões FIPA. Sua implementação é toda feita em Java e provê um pacote de ferramentas baseadas em componentes para o desenvolvimento de agentes seguindo as especificações FIPA.

JADE é um *framework* elaborado pelo TILAB (*Telecom Itália Lab*) para o desenvolvimento de aplicações multiagentes distribuídos, usando arquitetura de comunicação P2P (*peer-to-peer*). Poggi *et al.* (2001) comentam que JADE é uma arquitetura de software implementada para facilitar o desenvolvimento de aplicações de agentes com a maior parte da infraestrutura relatada no padrão FIPA-OS.

Em termos de plataforma de desenvolvimento, o foco da abordagem está centrado no *framework* JADE que tem como princípios fundamentais: Interoperabilidade - proporcionando interação com outros agentes; Uniformidade e portabilidade – fornece um conjunto de APIs, independentes de camadas de rede ou de versão do Java (J2EE, J2SE, J2ME) - e Facilidade de uso.

Neste trabalho, a plataforma JADE foi escolhida pelo fato de ser um *framework* implementado em Java e seguir as especificações FIPA que, conforme visto, simplifica o desenvolvimento de sistemas multiagentes. A outra decisão foi pelo fato de o JADE ser um software livre e distribuído em código aberto.

Associado a isso, destaca-se também que a empresa proprietária do JAVA *Sun Microsystem*, liberou recentemente o código fonte da linguagem JAVA (REVISTA AREDE, 2007), aderindo ao padrão de licença de software livre GPL (*General Public License*), o que acaba incorporando a mesma licença do JADE. Detalhes da GPL podem ser encontrados em http://www.gnu.org/licenses/gpl-3.0.html. A seção 4.3 é dedicada ao desenvolvimento do SADDEM com o *framework* JADE.

3.10 Ontologia

O termo ontologia tem sua origem no campo filosófico onde a temática sobre a existência é o ponto de partida para buscar uma especificação do que se pode dizer do mundo. Na realidade, no campo computacional também se adiciona o aspecto existencial se for levado em consideração à capacidade que um agente deve possuir sobre as coisas que existem em sua volta e como ele deve representá-las (RUSSEL e NORVIG, 2004).

Mas o fato de somente existir não basta para o agente. É preciso que ele compreenda, por exemplo, que a existência de um ferro quente pode ser descrita em termos de implicações ativas, reativas ou deliberativas desenvolvidas em função do conhecimento deste agente. Há uma produção de conhecimento e compartilhamento das ideias necessárias para que o mesmo se comunique com os demais agentes da sociedade.

Perez *et al.* (2001) destacam uma definição para o termo ontologia onde os elementos compartilhados, reutilização e conhecimento convergem com o que foi comentado até agora. Na visão do autor, as ontologias têm por finalidade eliciar o conhecimento consensual de um modo genérico, normalmente construído por um grupo de pessoas em diferentes locais.

A definição de ontologia, portanto, segue como uma descrição formal e explícita dos conceitos que representam um domínio em particular. O termo explícito indica que os tipos de conceitos utilizados bem como as limitações decorrentes de seu emprego são claramente definidos. A palavra conceito revela basicamente uma noção do mundo que uma pessoa ou organização pode ter (CAIRE e CABANILLAS, 2005).

De acordo com Borst (1997), quando se afirma que uma ontologia é uma "especificação formal de uma conceitualização compartilhada", o termo 'formal' se refere ao fato de que ela deve ser entendida por uma máquina, no caso mais utilizado, esta máquina é o computador. Já a palavra 'compartilhada' tem sua influência no conhecimento consensual aceito por um grupo.

Enquanto estudos tratam a ontologia como um tipo de taxonomia (HAKEEM e SAH, 2004), ao mesmo tempo há literaturas (DOGAC *et al.*, 2002) que julgam

importante a distinção de ontologias que representam modelos do domínio de uma maneira mais rica em detalhes e mais aprofundada quando se observam as restrições sobre a semântica do domínio.

3.11 Construção de ontologia

Embora uma ontologia possua uma variedade de formas, ela não pode deixar de incluir um vocabulário de termos acompanhados de suas respectivas especificações. Adicionalmente a esses termos deve-se tratar das inter-relações entre os conceitos bem como as suas restrições para evitar interpretações irrelevantes entre os mesmos.

Os elementos que descrevem uma ontologia são genericamente denominados termos. Os termos podem ser de três tipos específicos: conceito, predicado e ações de agente e podem formar grupos, denominados classes, que por sua vez podem ser especializadas ou generalizadas dando origem a uma hierarquia conforme visto na Figura 18. A definição dos termos conceito, predicado e ações de agente será detalhada na seção 3.12.

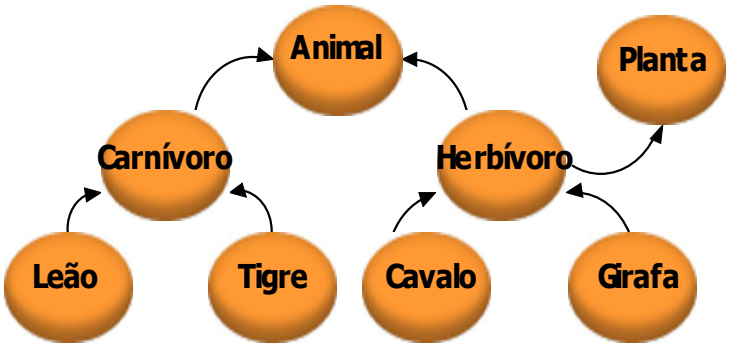

Figura 18 - Exemplo de classes de uma ontologia

A construção de ontologias normalmente envolve: i) determinar o domínio e escopo; ii) Definir a as classes na ontologia; iii) Organizar hierarquicamente as classes em superclasses e subclasses. É importante destacar que não há uma fórmula para a definição de uma ontologia perfeita, as melhores práticas e reutilização daquelas já construídas colaboram para o aprimoramento das técnicas mais utilizadas. A Figura 18 ilustra a construção de uma ontologia representando os conceitos de dois grandes reinos.

Para expressar a construção de uma ontologia utilizam-se linguagens formais como *Ontology Web Language-OWL* (OWL, 2008) e *Resource Description*

Framework-RDF (LASSILA e SWICK, 1999). Enquanto o RDF é aplicado para a definição de metadados, o OWL, que é uma extensão do RDF, define em instância uma ontologia na *Web*. As principais ferramentas para o desenvolvimento de ontologias são: Protégé (2008), OntoEdit (SURE *et al*., 2002), WebODE (ARPÍREZ *et al*. 2001), além de outros.

Uma vez construída a ontologia da Figura 18, os agentes podem trocar conhecimento por meio de uma comunicação utilizando uma linguagem semântica (e.g. FIPA-SL) baseada em lógica de primeira ordem representando o conteúdo da mensagem de acordo com o trecho da Figura 19 a seguir:

```
(Animal:nome "cavalo"
       :reino "Animal"
       :tipo "Herbivoro")
)
```

Figura 19 – Conteúdo de uma mensagem FIPA.

Para que se estabeleça a comunicação entre os agentes é necessário que os mesmos entendam o conteúdo da mensagem. O agente que irá receber a mensagem deve ser capaz de entender os símbolos e termos que o outro agente lhe enviou. Para expressar a ideia de que um cavalo pertence ao reino animal e este por sua vez é Herbívoro, os agentes envolvidos na comunicação compartilham os mesmos conceitos, predicados e ações de agente existentes na ontologia.

3.12 Ontologias em JADE

O mecanismo de comunicação entre os agentes permite a transferência das informações por meio de mensagens contendo vários campos denominados *slots*. O conteúdo e a linguagem para a representação desse conteúdo são *slots* contidos na mensagem para facilitar o que deve ser transmitido de um agente ao outro, podendo ser uma string ou seqüência de bytes, de acordo com o padrão FIPA.

As informações contidas no conteúdo de uma mensagem constituem uma forma complexa de comunicação entre agentes, pois o mesmo pode transmitir informações adicionais a respeito de algum termo de um domínio específico, como por exemplo, informações sobre as faixas de um *compact disc*. As ontologias permitem este tipo de comunicação e são regidas por uma sintaxe bem definida, conforme visto na linguagem de conteúdo FIPA-SL.

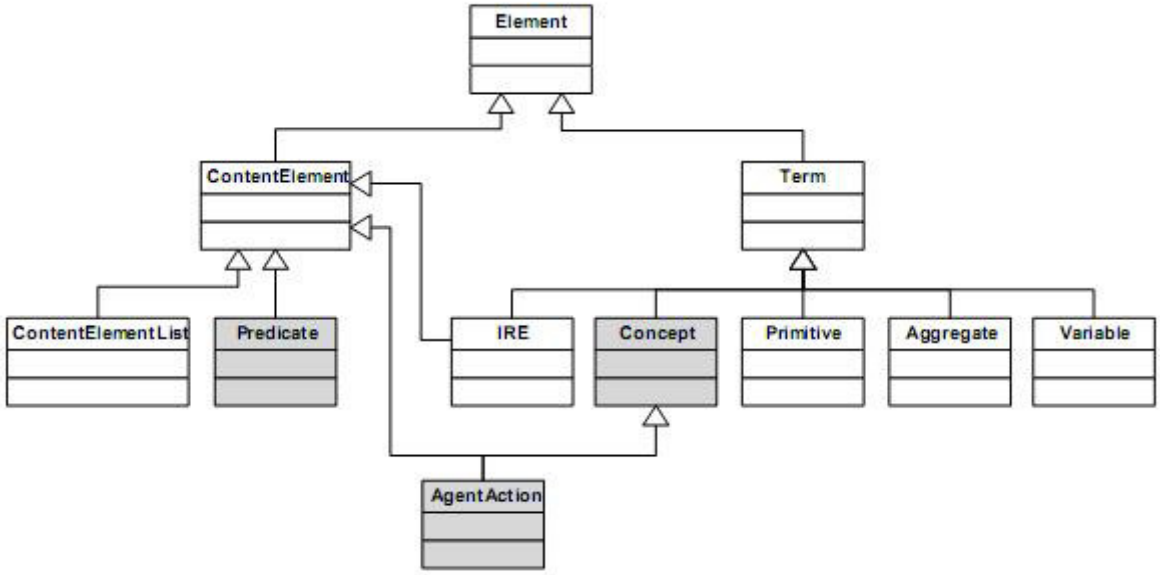

Figura 20 – Modelo de referência de conteúdo em JADE (BELIFFEMINE *et al.*, 2007).

A construção de ontologias no *framework* JADE deriva as especificações da FIPA para a organização das mensagens. Isso leva em consideração tanto a linguagem de conteúdo quanto as performativas. Os elementos da ontologia em JADE formam o Modelo de Referência de Conteúdo (Figura 20) e são definidos como Predicados, Termos, Conceitos, Ações de Agentes, Agregados e Expressões de Identidade Referencial (*Identifying Referential Expressions*- IRE), incluindo ainda uma Lista de Elemento de Conteúdo e Elemento de Conteúdo.

Um Predicado (*Predicate*) representa expressões que dizem algo a respeito do ambiente e pode assumir valores verdadeiros ou falsos. Um Termo (*Term*) aponta as expressões que identificam entidades, tanto abstratas quanto concretas, que existem no ambiente, que os agentes usam para inferir os fatos que lhes são perceptíveis. A Figura 21 exemplifica um predicado na ontologia JADE.

```
(Come
  (Animal :nome "tigre")
  (Animal :nome "girafa")
)
```

Figura 21 – Predicado na Ontologia JADE

Um elemento Termo pode ser classificado em Primitivo (*Primitive*) e Conceito (*Concept*). O primeiro representa entidades atômicas como tipos Inteiros e Strings, ao passo que o segundo define as entidades complexas com vários campos ou *slots*, semelhante às propriedades vistas no exemplo da Figura 19.

As Ações de Agentes (*AgentActions*) são conceitos especiais que indicam as ações que os agentes podem realizar, inclusive para outros agentes. As próprias performativas são tratadas como Ações de Agentes, pois estas são associadas ao conteúdo de certos tipos de mensagens ACL como, por exemplo, a performativa REQUEST discutida na seção 3.8. A Figura 22 ilustra uma ação de agente.

```
(Alimentar
   (Animal :nome "cavalo")
   (Tipo: "herbivoro")
)
```

Figura 22 – Ações de Agentes na Ontologia JADE

Agregados (*Agregates*) são entidades resultantes do agrupamento de outras entidades. Um exemplo seria uma seqüência de nomes de pessoas no conteúdo de uma mensagem passada para um agente, de acordo com a Figura 23.

```
(Sequence
   (Herbivoro :nome "cavalo")
   (Herbivoro :nome "girafa")
)
```

Figura 23 – Agregado na ontologia JADE.

Por outro lado, elementos IREs são bastante úteis quando se deseja identificar uma ou mais entidades para as quais um determinado predicado é tido como verdadeiro. Um exemplo seria o envio de uma mensagem de um agente que perguntasse se determinados animais comem planta. A Figura 24 ilustra o uso de um IRE na ontologia JADE.

```
(all ?x (comem-planta
   (Herbivoro :nome "cavalo")
   (Herbivoro :nome "girafa")
)
```

Figura 24 – IRE na ontologia JADE.

Observa-se que para a declaração de um elemento IRE é necessário o emprego de variáveis representado por '?x' na Figura 24. As variáveis representam as expressões utilizadas em consultas e normalmente são conteúdo de uma mensagem com a performativa QUERY_IF, para indicar as entidades que não se tem conhecimento prévio.

Os dois últimos elementos da ontologia JADE são a Lista de Elemento de Conteúdo (*ContentElementList*) e o Elemento de Conteúdo (*ContentElement*).

Enquanto o primeiro é uma lista de tipos primários (e.g. predicados, ações de agentes e IREs) o segundo é a forma genérica que o primeiro especializou para conter a própria lista e os tipos primários.

Isso significa que somente predicados, ações de agentes, IREs e lista de elementos desses três tipos (a superclasse *ContentElement* da Figura 20) são conteúdo significativo de pelo menos uma mensagem ACL, lembrando que todos são herança da classe Elemento de Conteúdo.

Uma ontologia para o domínio estudado nesta dissertação será representada como um conjunto de esquemas definindo a estrutura dos predicados, conceitos e ações de agente pertencentes aos recursos operacionais do carro posicionador do virador de vagões VV311-K01.

Essas informações são a base para se entender quais abstrações são necessárias para se desenvolver uma ontologia em JADE para o SADDEM, uma vez que o foco desta pesquisa não é, exclusivamente, o desenvolvimento de ontologias, contudo todos os detalhes de uma ontologia em JADE podem ser encontrados em (CAIRE e CABANILLAS, 2004) e (BELLIFEMINE *et al*. 2007).

3.13 Metodologias orientadas a agentes

A construção e o projeto de software industrial de qualidade é um desafio que exige ferramentas e técnicas adequadas. Os paradigmas de implementação de software surgiram como propostas para melhorar gerenciamento da crescente demanda por fatores computacionais de alta qualidade (SILVA *et al*., 2005).

A visão de qualidade normalmente envolve duas dimensões: i) Usuário - entende que software de qualidade é aquele que vai além dos requisitos a serem satisfeitos, em outras palavras deve obedecer a custo e prazo; ii) Projetista do software – entende que o bom software é aquele que atende aos requisitos de flexibilidade, adaptabilidade, manutenibilidade, reusabilidade, adequação a código legado, interoperabilidade, escalabilidade e robustez (SILVA *et al*., 2003).

Quando se desenvolve um SMA, englobam-se todas as situações por que passam os sistemas distribuídos e concorrentes, da mesma forma que os problemas adicionais resultantes de interações sofisticadas e flexibilidade. Adicionalmente,

todas as metodologias que propõem técnicas para a Engenharia de Software Multiagente precisam ser capazes de disponibilizar mecanismos de abstração apropriados, bem como as ferramentas de modelagem de tarefas individuais e coletivas dos agentes.

Para uma solução baseada em agentes adequada, Wooldridge (2002) lembra que devem ser considerados os seguintes aspectos: i) o ambiente é aberto, ou pelo menos altamente dinâmico, incerto e complexo; ii) os agentes representam uma metáfora natural; iii) o controle, dados e expertise devem ser distribuídos e atuarem como componentes semi-autônomos e iv) sistemas legados.

Considerando-se esses e outros elementos, diversas metodologias vêm sendo propostas para o desenvolvimento de SMAs. Dentre as metodologias atualmente disponíveis estão a MaSE (DELOACH *et al.*, 2001), a Gaia (ZAMBONELLI *et al.*, 2003), a MESSAGE, Tropos (MYLOPOULUS e CASTRO, 2000, BRESCIANI *et al.*, 2004, GIORGINI *et al.*, 2005) e a PASSI (COSSENTINO e POTTS, 2006), sendo que esta última foi escolhida para se empreender o desenvolvimento do SADDEM por apresentar uma abordagem centrada em técnicas e suporte notacional bastante conhecidos tais como orientação a objetos e UML durante o ciclo de desenvolvimento. A seção a seguir descreve detalhadamente a metodologia PASSI. Maiores informações à cerca das demais metodologias podem ser encontradas em (LINDOSO, 2006).

3.14 Metodologia PASSI

A PASSI (*Process Agent Society Specification and Implementation*) é uma metodologia resultante de um longo período de estudo e experimentos, sobretudo na área de robótica. As fases cobertas pela PASSI empreendem tanto a análise quanto o projeto de aplicações baseadas em agentes, integrando métodos de orientação a objetos e a notação UML, bem como suporte às abstrações de software tais como agentes, papéis e tarefas (COSSENTINO e POTTS, 2006).

Os papéis são desempenhados por um agente durante as interações com outros agentes para alcançar seus objetivos (i.e. representação dos interesses dos agentes sobre o estado do mundo que gostariam de alcançar). Além disso, um papel

tem uma coleção de tarefas a desempenhar para atingir metas específicas. Nesse sentido, uma tarefa é definida como uma atividade que o papel desempenha.

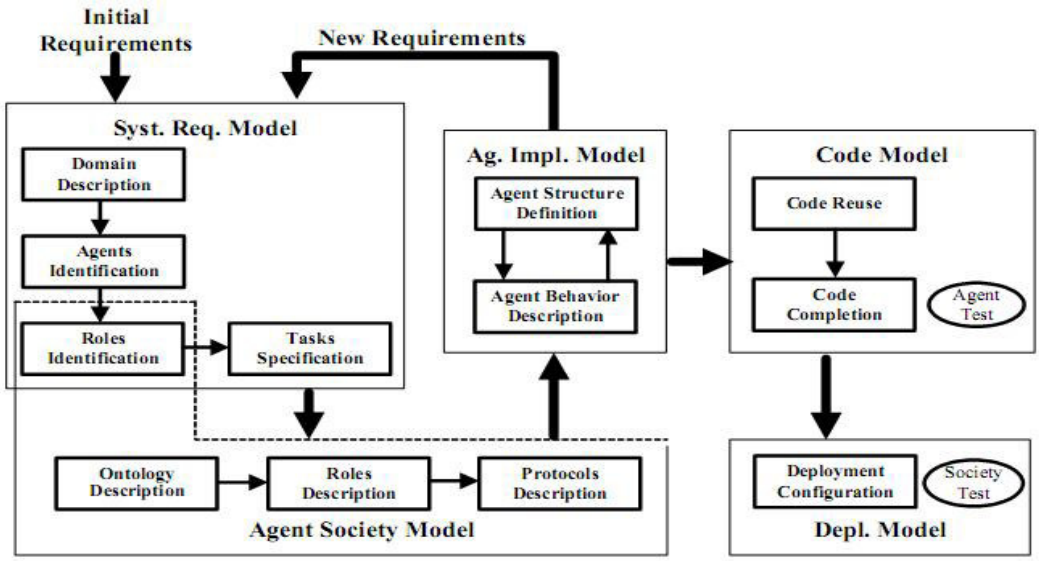

Figura 25 - Representação do Processo da Metodologia PASSI (COSSENTINO e POTTS, 2006)

O desenvolvimento de um SMA que emprega a metodologia PASSI deve percorrer cinco fases que são: *Modelos de Requisitos do Sistema*, *Modelo da Sociedade dos Agentes*, *Modelo de Implementação dos Agentes*, *Modelo de Código* e *Modelo de Distribuição*. A Figura 25 ilustra as etapas e modelos das fases da metodologia PASSI.

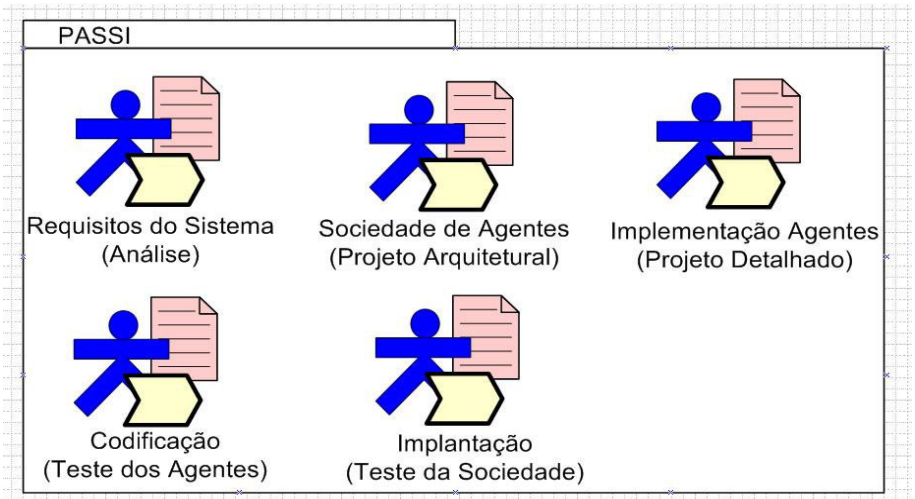

Figura 26 – Fases do Processo da metodologia PASSI.

Fragmentando-se os modelos e fases da metodologia PASSI, conforme visto na Figura 26, o processo da metodologia PASSI, representado com a notação SPEM (SPEM, 2007), tem como produto de cada fase um modelo.

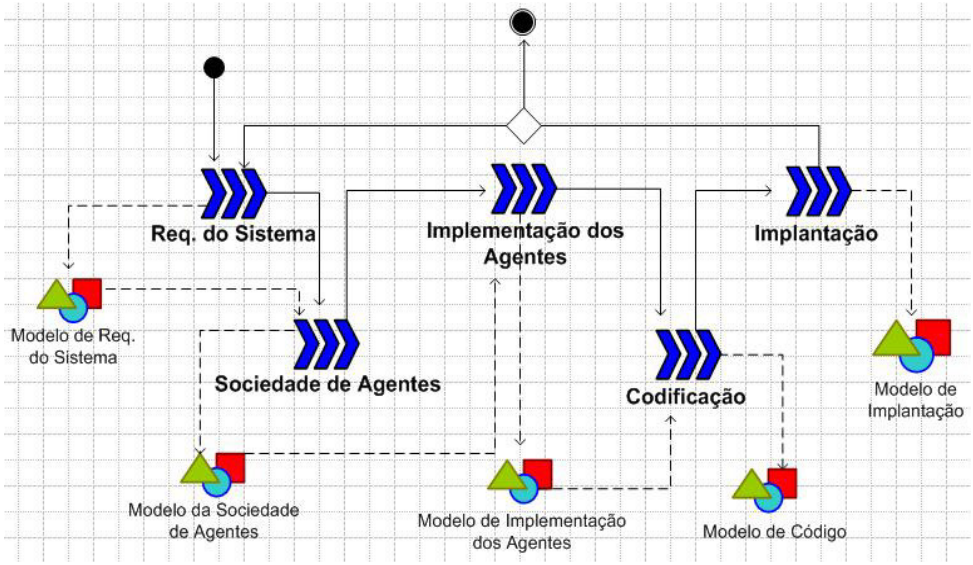

Figura 27 - Representação do Processo PASSI em notação SPEM.

Cada um oferece diferentes visões. No total, podem ser realizadas quinze etapas durante seu processo de desenvolvimento (COSSENTINO e POTTS, 2006).

Para um melhor entendimento do seu ciclo de vida em termos de fases, etapas e passos para a produção dos modelos e do próprio processo de desenvolvimento da metodologia PASSI, representaram-se na Figura 27 as fases do desenvolvimento de SMA com o propósito de se detalhar, posteriormente, a aplicação de suas técnicas.

De acordo com os artefatos da Figura 26 e Figura 27, a primeira fase tem como propósito o desenvolvimento do *Modelo de Requisitos do Sistema* em termos de ações e objetivos, consistindo em quatro etapas para o seu desenvolvimento que são:

- Descrição do Domínio – tem por finalidade a produção de uma série hierárquica de caso de uso convencional na notação UML para descrever as funcionalidades do sistema que serão decompostas nos passos restantes por meio do detalhamento dos cenários usando diagramas. Há também a possibilidade de se desenvolver o Diagrama de Contexto que fornece uma perspectiva inicial do sistema em alto nível de abstração, do qual mais diagramas detalhados podem ser desenhados.

- Identificação dos agentes: neste passo separam-se as funcionalidades internas dos agentes, representado estes como estereótipos em pacotes da UML.

- Identificação dos papéis: Utiliza os diagramas de seqüência para explorar cada responsabilidade dos agentes através de cenários de papéis específicos.

- Especificação de Tarefas: Especificação através dos diagramas de atividade e descrições auxiliares das habilidades de cada agente.

A segunda fase é a do desenvolvimento do *Modelo de Sociedade de Agentes* que é um modelo de interações sociais e dependências entre os agentes envolvidos na solução. No desenvolvimento deste modelo são envolvidas três etapas que adicionam uma parte do modelo anterior:

- Descrição da Ontologia de Domínio: Utiliza o diagrama de classes e OCL (*Object Constraint Language*) para descrever o conhecimento relacionado aos agentes individualmente e as suas interações práticas. Os elementos da ontologia utilizam seguintes relações da UML: i) Generalização - permite generalizar relações entre duas entidades. É um dos principais operadores para a construção da ontologia; ii) Associação - relacionamento lógico entre duas entidades e permite a especificação do papel de entidades envolvidas para esclarecer a estrutura; iii) Agregação - pode ser utilizada para construir grupos onde restrições de valores podem ser especificadas explicitamente.

- Descrição dos Papéis: Utilizam os diagramas de classes para mostrar os diferentes papéis executados pelos agentes, as tarefas que abrangem os papéis relacionados, habilidades de comunicação e as dependências interagentes.

- Descrição do Protocolo: Utilizam os diagramas de seqüência para especificar a gramática de cada protocolo de comunicação em termos de performativas.

A terceira fase é o desenvolvimento do *Modelo de Implementação dos Agentes* que é o modelo da arquitetura da solução em termos de classes e métodos, no qual seu desenvolvimento envolve as seguintes etapas:

- Definição da Estrutura do Agente: Utiliza diagramas de classes convencionais para descrever a estrutura da solução das classes dos agentes.

- Descrição dos Comportamentos dos Agentes: Utiliza-se do diagrama de atividade ou diagrama de estados para descrever os comportamentos individuais dos agentes.

A quarta etapa é a do *Modelo de Código* que é o modelo da solução em nível de código, no qual necessita realizar os seguintes passos para o seu desenvolvimento:

- Reuso de Código: Pode ser reutilizados uma biblioteca de classes e alguns diagramas com associação do código reutilizável.

- Complementação de Código: Código fonte do sistema.

A quinta e última etapa é a do *Modelo de Implantação* que é o modelo de distribuição das partes do sistema através das unidades de processamento do hardware e suas migrações entre as unidades de processamento. Neste modelo há a seguinte etapa:

- Configuração de Implantação: Utiliza os diagramas de *deployment* para descrever a alocação dos agentes nas unidades de processo ativas e algumas restrições na mobilidade e migração.

Logo abaixo, observam-se as ferramentas que oferecem suporte à metodologia PASSI:

- PTK *Toolkit*: É um *add-in* para a ferramenta *Rational Rose* (RATIONAL ROSE, 2007) que dá suporte a etapa de geração de código Java.

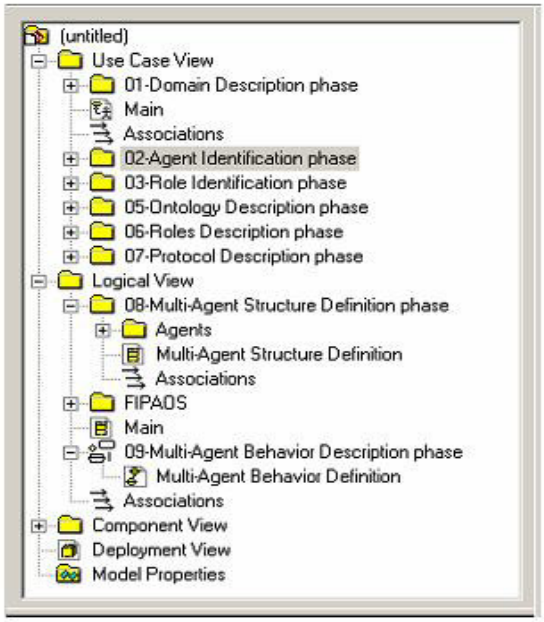

Figura 28 - Navegação do *add-in* do PTK Toolkit

- *AgentFactory* : É uma aplicação que permiti a geração de protótipos de partes de sistemas que utilizam plataformas que estejam de acordo com a FIPA.

Conforme percebido, a metodologia PASSI utiliza os aspectos da orientação a objetos de forma proveitosa através do reuso, bem como adiciona um suporte apropriado as particularidade dos agentes por meio das ferramentas PTK *Toolkit* e *AgentFactory* que fornecem melhorias em projetos robustos e reduzem os esforços desta fase de desenvolvimento, assegurando geração automática de código e rápida prototipagem de grande parte de sistemas compatíveis com o padrão FIPA (COSSENTINO e POTTS, 2006).

3.15 Considerações Finais

Neste capítulo foram descritos os principais elementos da Engenharia de Software Orientada a Agentes e as motivações que contribuíram para se alcançar uma nova perspectiva de desenvolvimento de software. Dentre os elementos destacaram-se os estudos da Engenharia de Software tradicional, como por exemplo, o ganho de produtividade e reuso de código proporcionado pela orientação a objetos.

Nesse sentido, o propósito deste capítulo não foi engrandecer a orientação a agentes, mas a relação à orientação a objetos. Grande parte do desenvolvimento de

SMAs tem como base a orientação a objetos pelas contribuições que esta trouxe em termos de fraco acoplamento e alta coesão. Resumiram-se como principais diferenças entre os paradigmas os aspectos de autonomia e capacidade cognitiva.

Foram apresentadas as definições dos agentes como entidades autônomas, bem como o trabalho de cooperação e colaboração entre eles em uma estrutura denominada multiagente, enfatizando o ambiente que os cercam e as particularidades que devem ser analisadas para um projeto de um sistema multiagente.

Ressaltou-se a importância do avanço de plataformas operativas, descrevendo-se também os problemas que deram a oportunidade de se criarem mecanismos de interoperabilidade e especificações técnicas de comunicação, ambos sintetizados em um padrão para os sistemas multiagentes.

Dentre as técnicas de comunicação, descreveram-se de forma abstrata as principais definições de ontologias que hoje são bastante empregadas para a reutilização e compartilhamento de conhecimento de uma sociedade de agentes. Os exemplos citados servem como insumo para o entendimento dos métodos utilizados para a elaboração da ontologia do sistema SADDEM baseada no *framework* JADE.

Este capítulo também se preocupou em apresentar a metodologia PASSI para o desenvolvimento do SADDEM com a aplicação de grande parte de seus modelos. Selecionou-se a PASSI porque o problema abordado nesta dissertação se trata de uma construção complexa envolvendo programação concorrente e distribuída, tornando-se indispensável para o emprego de técnicas e melhores práticas de desenvolvimento e gerenciamento do processo em cada fase.

Outro ponto em se escolher a PASSI é o emprego da UML para a construção dos modelos - que também é bem aceita tanto no âmbito acadêmico quanto no industrial - reduzindo os possíveis impactos na produtividade resultante da adoção de uma linguagem que não enuncie a facilidade ou a simplicidade de se abstrair e representar seus métodos, independente do nível de especialização do usuário.

Assim, apesar dos SMAs não serem uma solução universal e de ainda existirem importantes desafios a serem superados, estes sistemas são vistos como uma nova direção importante na engenharia de software porque promovem uma forma de se imaginar o fluxo de controle em um domínio extremamente distribuído e a possibilidade de se implementarem melhores práticas de organização de entes colaborativos e concorrentes.

4 SISTEMA DE APOIO A DECISÃO EM DESCARGA DE MINÉRIO- SADDEM

Neste capítulo apresenta-se o processo de desenvolvimento do SADDEM, um sistema multiagente para seleção de falhas e apoio à decisão. O ciclo de vida de construção do sistema é gerenciado pela metodologia PASSI. Os artefatos de análise e projeto são mapeados para componentes implementados no *framework* JADE. As atividades realizadas resumem-se na captura dos requisitos iniciais e caracterização do problema estudado. O acesso ao ambiente de descarga de minério e aos especialistas foi realizado através de visitas programadas para a coleta de informações. Os esforços da pesquisa focaram as atividades que vão desde os requisitos - passando pela a identificação dos agentes, ontologia da sociedade de agentes, responsabilidades e suas tarefas - até a implementação.

4.1 Requisitos do Sistema SADDEM

Entende-se por requisito uma funcionalidade, condição ou habilidade necessária para a que um sistema alcance o seu objetivo, assim as primeiras exigências que devem ser atendidas em um sistema traduzem-se no conjunto de requisitos devem ser satisfeitos.

Os requisitos do SADDEM têm como meta o apoio à tomada de decisão no processo de descarga de minério, selecionando as falhas que são medidas pelos sensores do equipamento virador de vagões VV311K01. Na primeira camada da pirâmide de automação, as falhas são capturadas por PLCs e armazenadas em banco de dados histórico com os endereços de memória dos dispositivos de campo.

A partir do banco de dados histórico, há uma exportação e conversão dos endereços de memória do PLC para um banco de dados relacional em Oracle para a consulta da camada superior, onde estão os sistemas MES e PIMS. Nesta camada, o SADDEM realiza a leitura de um repositório de dados em XML, programados para serem gerados a partir de procedimentos em SQL (*Structured Query Language*), fornecendo o estado atual das tags do VV311-K01medidas pelos sensores.

Como essas tags mapeiam o comportamento dos dispositivos de campo, incluindo suas não conformidades, obtem-se por meio desta proposta, a situação dos componentes, sistemas e itens do VV311-K01em tempo de produção, sendo um

referencial significativo para o processo decisório tratado pelo SADDEM, que deve fornecer também recomendações de planos de manutenção. Para melhor especificação e entendimento das funcionalidades do SADDEM, listam-se os seguintes requisitos iniciais:

- Monitorar as *tagnames* no banco de dados relacional Oracle por meio de programação de rotinas SQL;
- Gerar repositório de dados em formato XML;
- Verificar tipo de falha;
- Recomendar planos de manutenção.

4.1.1 Base de Dados

O SADDEM armazena planos de manutenção para o VV311-K01elaborado pelo corpo técnico da VALE baseado na metodologia de Manutenção Centrada na Confiabilidade (*Reliability Centred Maintenance*- RCM), modelada em um banco de dados separado, construído em *Microsoft Access* 2003. Esses planos serão modelados nos mecanismos de inferência dos agentes para o monitoramento das *tagnames* e recomendação dos respectivos planos para a execução dos reparos.

4.1.2 Descrição dos Agentes

A composição do sistema SADDEM é organizada em quatro agentes: Karem, Posicionador, Interfaceador e Monitor. Os dois primeiros têm suas estruturas definidas por meio do motor de inferência em JESS, para formar a arquitetura de agentes deliberativos (WOOLDRIDGE, 2002), e atuam como agentes do núcleo da aplicação. Os outros agentes são modelados com arquitetura interna reativa.

4.1.3 Agente KAREM

O *Kernel Agent for REcoMmendation* é um agente núcleo, responsável pela recomendação de planos para manutenção dos dispositivos do carro posicionador do VV311K01. Além disso, Karem é um agente com arquiterura cognitiva, pois ele carrega uma base de conhecimento que é alimentada pelo resultado da leitura de um repositório de dados em XML, realiza inferências sobre as causas de falha do VV311-K01e escolhe os referidos planos para o conjunto carro posicionador.

4.1.4 Agente Positioner

O agente Positioner foi caracterizado com esse nome por estar envolvido com as inferências para descoberta das causas de falhas do carro posicionador. As entradas para este agente concentram-se no repositório de dados, de onde são extraídos os fatos que encadeam o comportamento definido na sua arquitetura cognitiva. Vale ressaltar que o agente Positioner coopera com o agente Karem lhe enviando um diagnóstico das falhas lidas no repositório de dados.

4.1.5 Agente Monitor

Agente que responde pelo monitoramento das tagname no PIMS. Nesse monitoramento estão incluídas as requisições do SADDEM por meio de programação de *stored procedures* no banco de dados a cada ciclo de tempo necessário para o processo decisório completar uma iteração.

4.1.6 Agente Interface

Agente que apresenta o resultado das colaborações dos agentes Monitor, Karem e Positioner em interface gráfica. É através deste agente que o usuário do SADDEM interage para verificar as recomendações e o raciocínio do agente KAREM.

4.2 Arquitetura SMA adotada no SADDEM

Como descrito na seção 3.5, os agentes representam papéis e responsabilidades importantes no ambiente em que atuam, cooperam entre si, comunicam-se e se organizam para solucionar os eventuais conflitos que venham ocorrer na sociedade.

Como todas as propriedades abordadas convergem para uma forma de arquitetura de sistema multiagente, a arquitetura do SADDEM, aqui proposta está baseada de nas tarefas realizadas pelos três agentes no contexto da metodologia PASSI.

Para o mecanismo de cooperação, a partilha de tarefas e os resultados são realizados por meio de troca de mensagens diretas, pois no sistema SADDEM os agentes se conhecem e trocam informações diretamente entre si. No caso da

coordenação, a organização dos agentes está apoiada nas atividades dos agentes Monitor e Positioner que trabalham para o agente Karem.

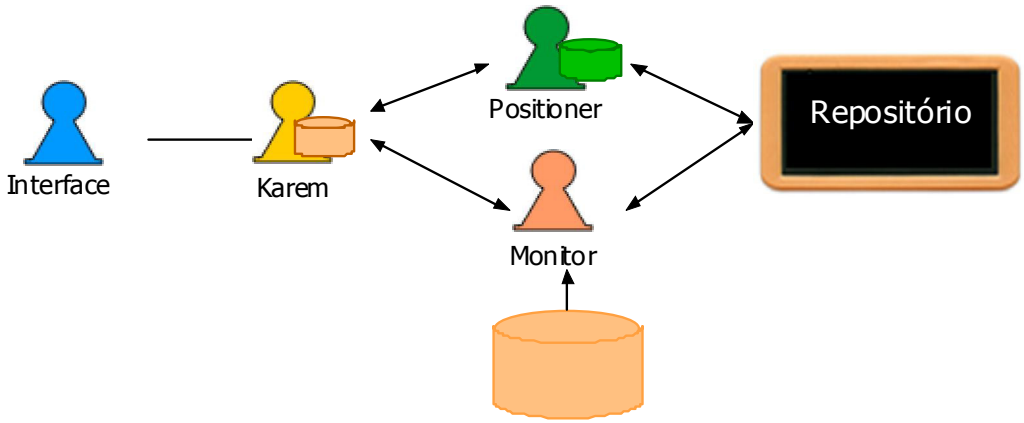

Figura 29 – Arquitetura SMA adotada no SADDEM.

Na Figura 29 é possível destacar a dependência do agente Karem no processo de inferência, já que ele aguarda o término das tarefas para a seleção de falhas e decisões para então enviar as recomendações de planos de manutenção para o agente Interface.

Com essa arquitetura, Karem e Positioner são apresentados com suas bases de conhecimento. O agente Monitor extrai os dados do PIMS e disponibiliza em formato XML para o agente Positioner no repositório.

4.3 Desenvolvimento do SADDEM

Após explanação dos requisitos, inicia-se a descrição do processo de desenvolvimento do SADDEM. As fases, etapas e passos serão guiados pela metodologia PASSI conforme explicado anteriormente. Para o esboço dos modelos, a ferramenta de apoio utilizada foi a PTK (PASSI *Tool Kit*), configurada no *Rational Rose*, e que tem integração com os *frameworks* JADE e FIPA-OS. O raciocínio do agente Karem é modelado com a máquina de inferência Jess.

4.3.1 Identificação dos Agentes

Conforme a metodologia PASSI, a definição dos agentes neste primeiro estágio tem como finalidade uma análise informal dos requisitos por meio de diagramas de caso de uso, reunindo um conjunto inicial de conceitos em alto nível de abstração.

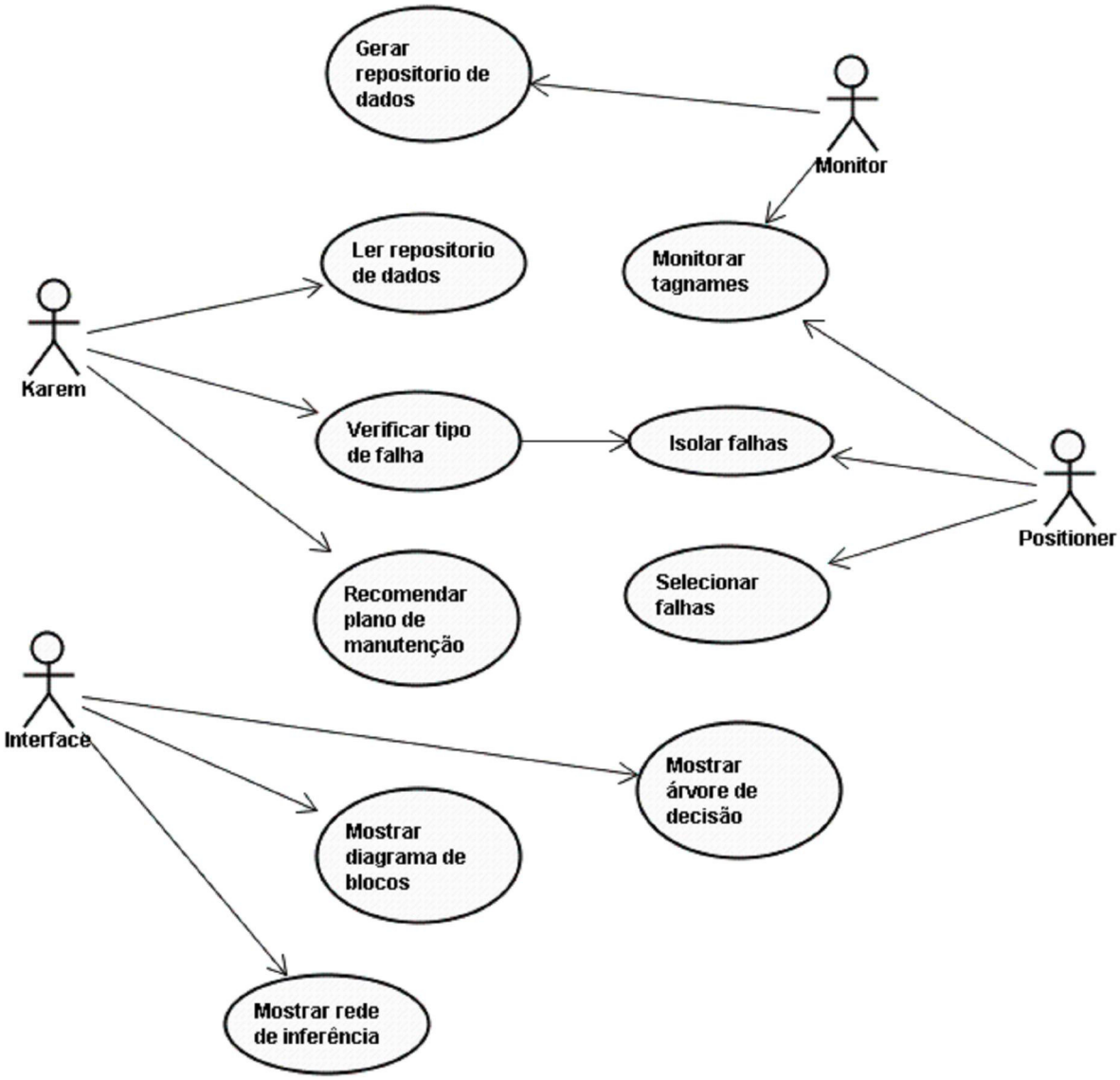

Figura 30 – Descrição do Domínio do SADDEM

Esses passos convergem para a descrição do domínio que é parte do modelo de requisitos do sistema e contém o detalhamento funcional do sistema SADDEM, conforme visto na Figura 30.

A identificação propriamente dita dos agentes ocorre com a organização dos casos de uso em pacotes. Os insumos produzidos durante a identificação dos agentes destacam os casos de uso e pacotes representados no mesmo diagrama. Como resultado desta etapa, os agentes do sistema SADDEM estão alocados de acordo com os pacotes da Figura 31.

Os pacotes do SADDEM são caracterizados por abstrações de agentes. Esse modelo permite a observação dos subsistemas e seus controles de fluxos para

a verificação das possíveis dependências entre eles durante determinada configuração de comunicação.

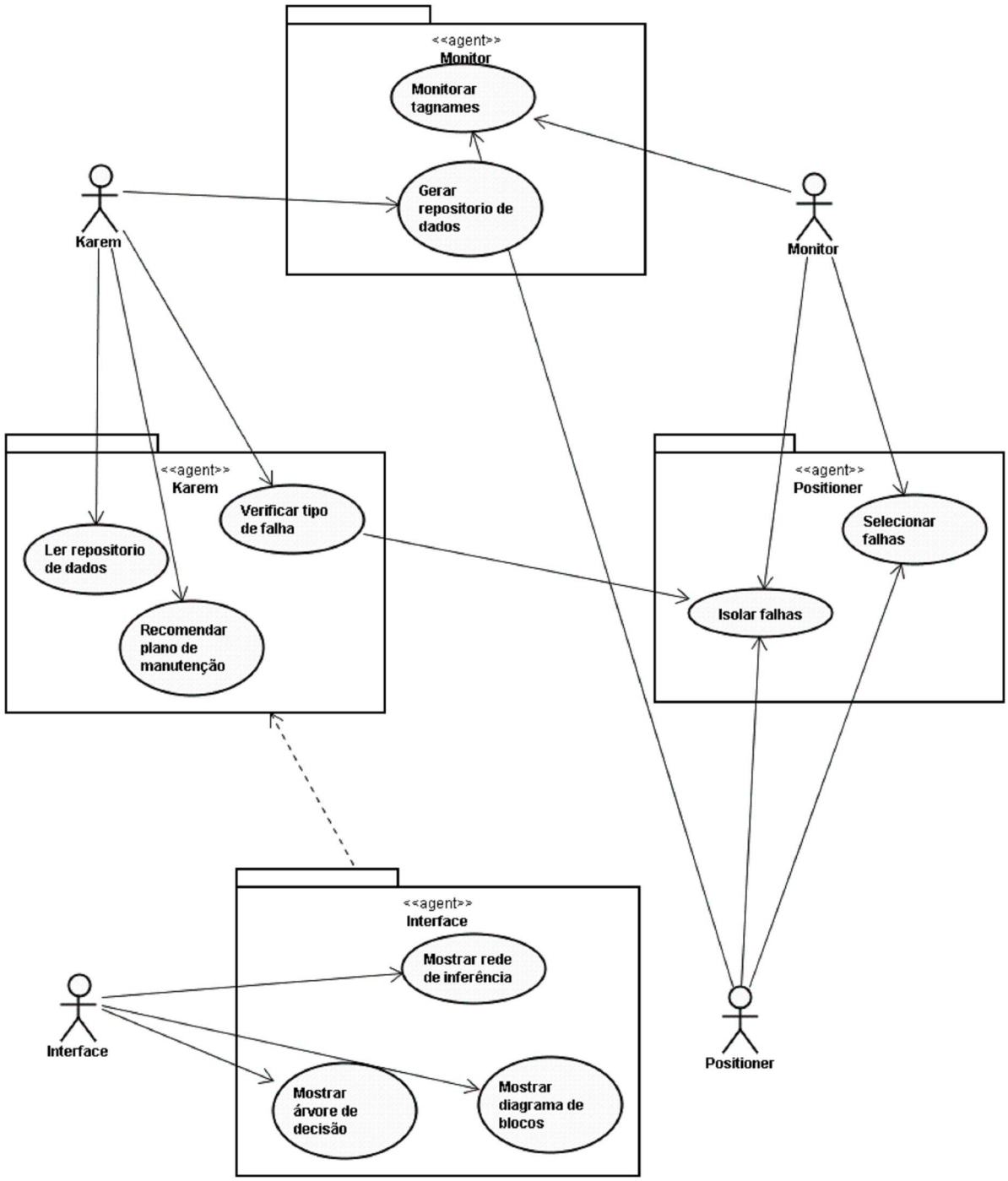

Figura 31 – Identificação dos Agentes do SADDEM.

Além disso, cada pacote apresenta as funcionalidades que serão atribuídas aos agentes do sistema SADDEM, ou seja, nessa organização os casos de uso expõem as atribuições dos agentes da sociedade. As relações entre os casos de uso vistas na Figura 30 serão convertidas em atos de comunicação.

4.3.2 Identificação de papéis

Essa atividade tem por objetivo a identificação de cenários de interação do SADDEM por meio da descrição dos papéis dos seus agentes. As funcionalidades dos agentes, que foram apresentadas na seção anterior, são usadas para explorar as responsabilidades por meio de estruturas da metodologia PASSI que capturam o comportamento dinâmico do sistema proposto, representando essas interações com diagramas de seqüência da UML. O primeiro cenário é mostrado na Figura 32.

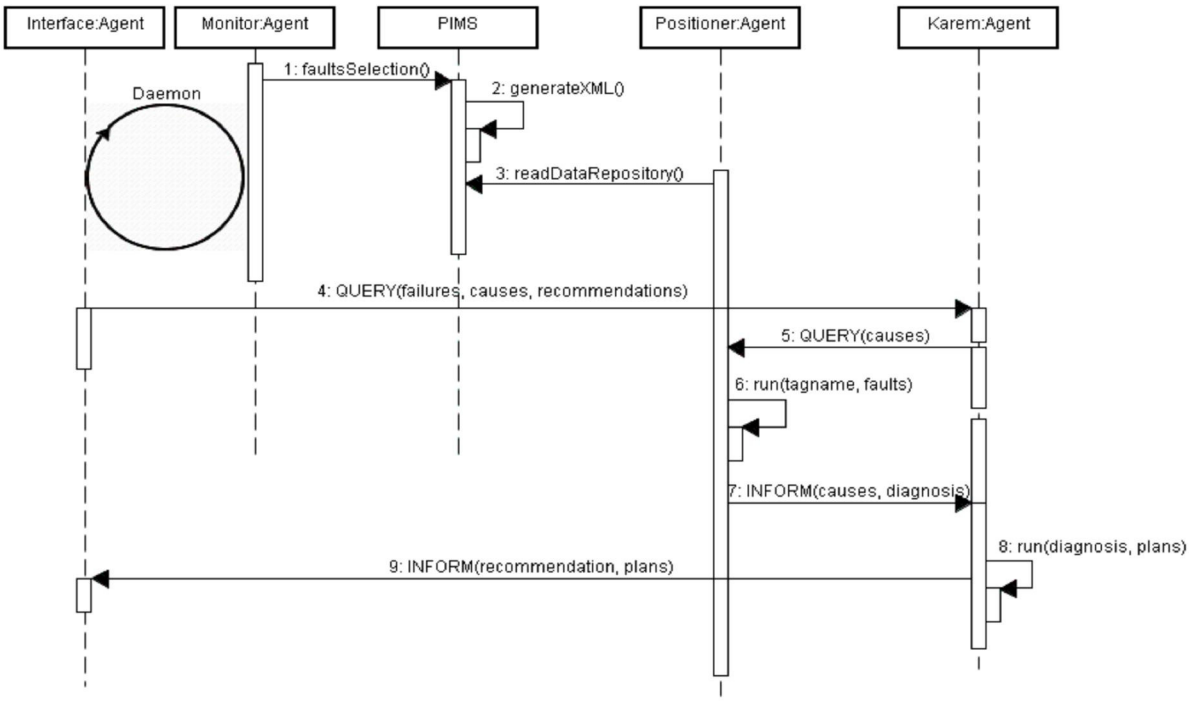

Figura 32 – Interações do Sistema SADDEM: Recomendação de Planos.

O comportamento dinâmico do SADDEM, exposto na Figura 32, revela a presença de quatro agentes e uma entidade externa que representa o sistema de informação PIMS. O cenário inicia com um *loop* para obtenção dos dados do repositório através das mensagens 1 e 2. Seqüencialmente, participam desta interação os agentes Interface e Monitor. O PIMS é principal provedor dos dados de campo a cada requisição.

Após isso, obedecendo a uma escala temporal, o agente Interface novamente envia uma mensagem QUERY desejando descobrir qual recomendação pode ser dada pelo agente Karem para uma determinada causa encontrada no repositório de dados. Antes dar um retorno ao agente Interface, o agente Karem

consulta o agente Positioner que contém as regras que utiliza para tomar decisões quando tem conhecimento de alguma falha que ocorreu.

O agente Positioner realiza uma combinação dos dados existentes com regras que ele possui em sua base de conhecimento. Dependendo do retorno do agente Positioner, agora é a vez do agente Karem executar um processo semelhante para recomendar os planos de manutenção a serem enviados para o agente Interface.

As interações da Figura 32 revelam os aspectos dinâmicos do sistema SADDEM, mas para melhor entendimento dos principais elementos envolvidos nesta etapa separaram-se os agentes mais relevantes para a observação e análise isolada, sendo que cada fluxo de informação trocada entre os agentes e entidades externas ocorrem por meio de cenários para melhor contextualizar os elementos do sistema.

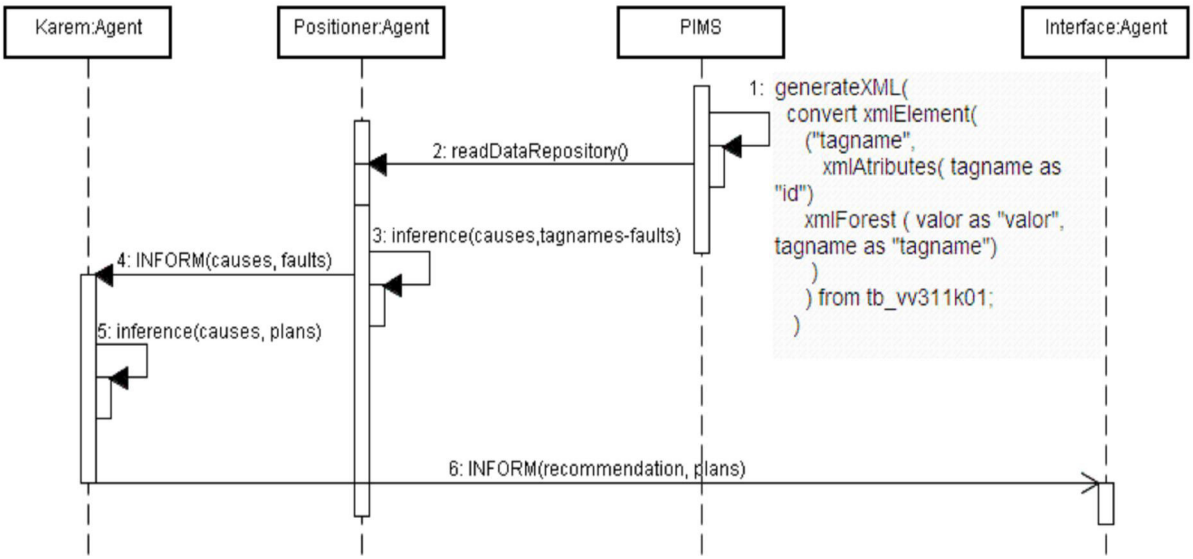

Figura 33 – Interações do agente Karem.

A Figura 33 caracteriza o cenário seleção de planos que tem como foco as interações do agente Karem. Este agente apresenta um fluxo de informações em que participam os agentes Interface e Positioner. Após a geração da consulta e conversão desta para o formato XML na entidade externa PIMS, as informações do repositório (arquivo XML) são capturadas pelo agente Positioner que efetua um raciocínio de acordo com os dados de falhas encontrados no repositório.

Após a inferência das falhas, o agente Positioner envia o resultado contento as causas das falhas que foram identificadas no repositório para o agente Karem. Este por sua vez, realiza um processo semelhante ao do agente Positioner (ver Figura 36) combinando as causas encontradas com os planos de manutenção do VV311-K01e os envia ao agente Interface que apresenta para o usuário.

O próximo cenário está associado ao monitoramento das *tagnames* onde o papel principal é realizado pelo agente Monitor. Participa também a entidade externa PIMS que concentra os dados manipulados pelos os agentes da sociedade durante as interações. Na Figura 34, observam-se as propriedades dinâmicas por meio de mensagens trocadas entre o agente Monitor e o PIMS enfatizando como ocorre captura dos dados operacionais.

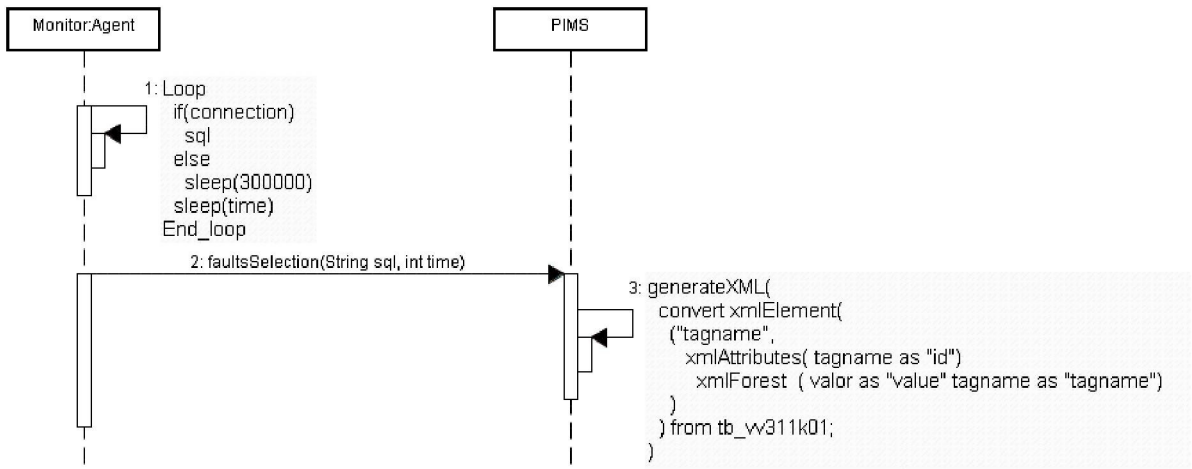

Figura 34 – Interações do agente Monitor

As interações do agente Monitor foram modeladas levando em consideração a vantagem de se realizar a aquisição das *tagnames* somente com o valor igual a um, descartando aquelas com valor igual a zero, ou de se selecionarem ambas. Como essa decisão ocorre em tempo de projeto detalhado, e nesse sentido analisam-se as caixas pretas, associaram-se as soluções alternativas ao requisito não funcional freqüência das falhas como um atributo que o SADDEM deve atender.

Esse requisito foi esclarecedor para se optar por capturar do PIMS tanto as tagnames com valor igual a zero quanto aquelas com valor igual a um, devido às melhorias acrescentadas com o acompanhamento da freqüência com que um item do VV311-K01pode apresentar não conformidade, o que teoricamente não seria possível saber se somente as *tagnames* com valor igual a um fossem selecionadas.

Nesse cenário há um ciclo de requisições (*loop*), disparadas pelo agente Monitor para selecionar as *tagnames* no PIMS a cada cinco minutos. Essas instruções são declaradas em uma *thread* - um fluxo de controle no mesmo processo executado pelo agente monitor do agente – que aguarda um tempo de 300000 milisegundos até que a próxima instrução SQL seja executada.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <tags>
 - <tagname id="ASC_B11@VV311K01">
      <value>0</value>
   </tagname>
 - <tagname id="AIN_ALI_001@VV311K01">
      <value>1</value>
   </tagname>
 - <tagname id="AFL_BEP_001@VV311K01">
      <value>0</value>
   </tagname>
 - <tagname id="ACV_BRE_001@VV311K01">
      <value>0</value>
   </tagname>
 - <tagname id="ATA_BRT_M01@VV311K01">
      <value>0</value>
   </tagname>
   ...
  </tags>
```

Figura 35 – Documento XML com as *tagnames* do PIMS

A partir deste loop, o PIMS executa a instrução SLQ na tabela de *tagnames* do VV311-K01formatando as linhas da consulta de acordo com a sintaxe XML onde os elementos filhos são constituídos pelos campos tagname e valor da tabela tb_vv311k01. Cada *tagname* possui um atributo identificador que tem como chave a própria *tagname*. A

Figura 35 resume o formato do documento XML.

O próximo cenário de interações do sistema SADDEM destaca a seleção de falhas. Neste fluxo de dados, o Agente Positioner é o centro do processo que se inicia com a obtenção do arquivo XML, gerado no cenário que monitora as *tagnames*. O documento é lido pelo agente Positioner e depois combinado com uma base de conhecimento que encadeia e declara as condições para se diagnosticar as causas de uma falha no sistema do carro posicionador do VV311K01.

Antes de se representar as interações do agente Positioner, ressalta-se que tais condições estão declaradas em um modelo de raciocínio que captura as instâncias de conhecimento módulo decisório do agente Positioner visando um melhor entendimento do processo de inferência.

Além disso, toda aquisição de conhecimento dos agentes Positioner e Karem foi obtida por meio de entrevista estruturada, ou seja, fazendo uso de questionário e consultas em fontes documentadas, diretamente com o especialista em diagnóstico de falhas do setor de descarga da VALE para composição dos modelos de raciocínio dos agentes cognitivos do sistema SADDEM.

Baseando-se no nível de inferência do modelo de conhecimento da metodologia *CommonKADS* (LABIDI, 1997), o modelo de racionínio da Figura 36 destaca a inferência realizada pelo agente Positioner para o conjunto carro posicionador, lembrando que o mesmo modelo foi adotado para o agente Karem.

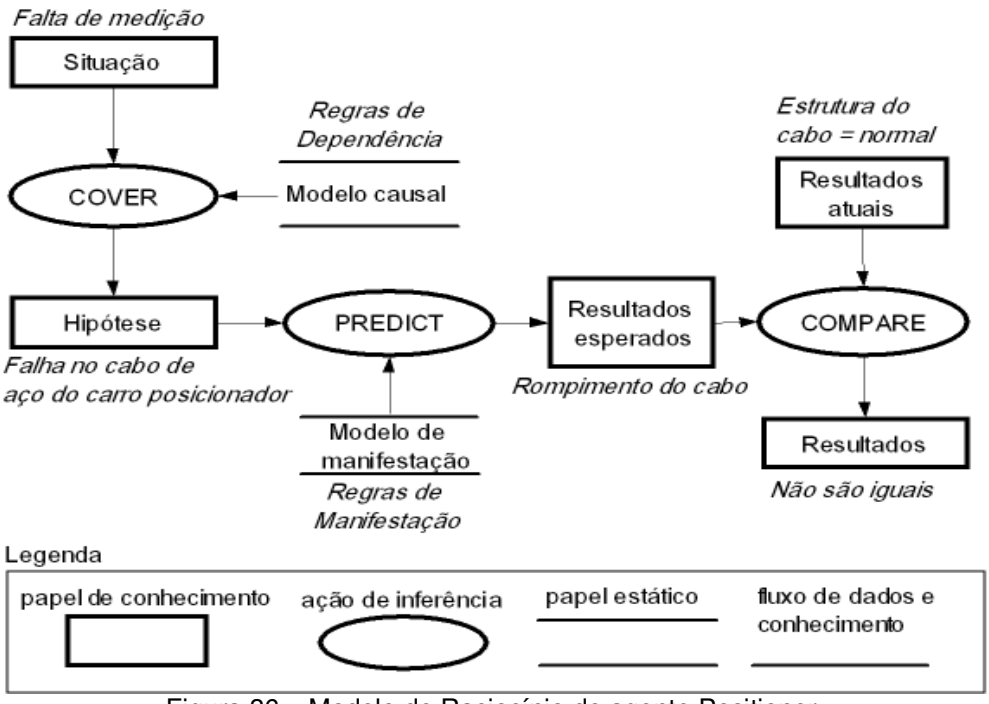

Figura 36 – Modelo de Raciocínio do agente Positioner.

No modelo da Figura 36, os papéis de conhecimento são nomes funcionais que capturam os elementos que participam do processo de raciocínio para diagnosticar as situações do carro posicionador e podem apresentar variabilidade tais como outras situações (i.e. travamento, aquecimento etc). As ações de

inferência tomam como entrada papéis estáticos, representados pelo modelo causal e de manifestação.

No modelo causal estão presentes as regras que relacionam os modos de falha do carro posicionador levando-se em consideração os valores dos seus atributos, enquanto que no modelo de manifestação reúnem-se as regras que manifestam suas responsabilidades a partir dos valores destes atributos que satisfazem determinadas condições.

Os conceitos de inferência *COVER*, *PREDICT* e *COMPARE* representam axiomas de raciocínio que serão mapeados pela máquina de inferência pelo agente Positioner. Basicamente uma transição de conceitos abstratos como artefatos de entrada para conceitos concretos sintetizados em um conjunto de asserções.

Após as considerações internas, para melhor visualização das interações do agente Positioner da Figura 37, principalmente com relação à mensagem 3, o que o agente Positioner faz é processar os dados de entrada, fatos presentes no arquivo XML, para encontrar as causas, e por meio de um mecanismo de inferência, compara-os com a sua base de conhecimento que é composta por um conjunto de regras.

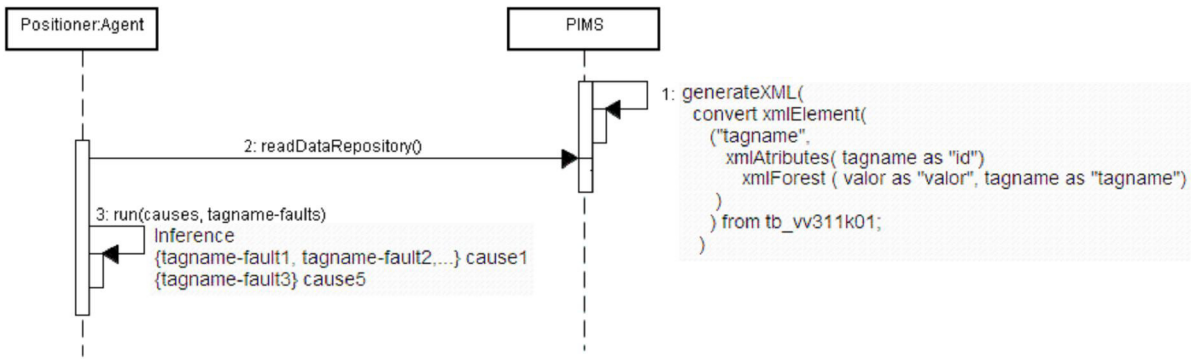

Figura 37 – Interações do agente Positioner.

Com os resultados dos modelos produzidos em cada cenário, as responsabilidades dos agentes puderam ser identificadas, desenhando os papéis específicos, bem como a visualização da comunicação interagentes. Os próximos recursos utilizados no desenvolvimento do sistema SADDEM terão como foco as capacidades dos agentes modeladas em termos de tarefas que cada um deve desempenhar no ambiente.

4.3.3 Identificação das tarefas

A atuação dos agentes no seu ambiente de trabalho reúne competências que envolvem destreza, independência, planejamento e todas as propriedades citadas na seção 3.2. Adicionalmente a esses atributos, as capacidades dos agentes de sistema SADDEM devem ser identificadas neste passo da metodologia PASSI. As tarefas identificadas estão desenhadas em três diagramas de atividade da UML e definem os comportamentos dos agentes Karem, Monito e Positioner.

Na Figura 38, uma das primeiras tarefas do agente Karem é receber as causas das falhas contidas no repositório, que na realidade é o arquivo XML com as *tagnames*, e de acordo com as referidas causas selecionar os planos de manutenção no banco de dados do sistema SADDEM que serão enviados posteriormente ao agente Interface como recomendação.

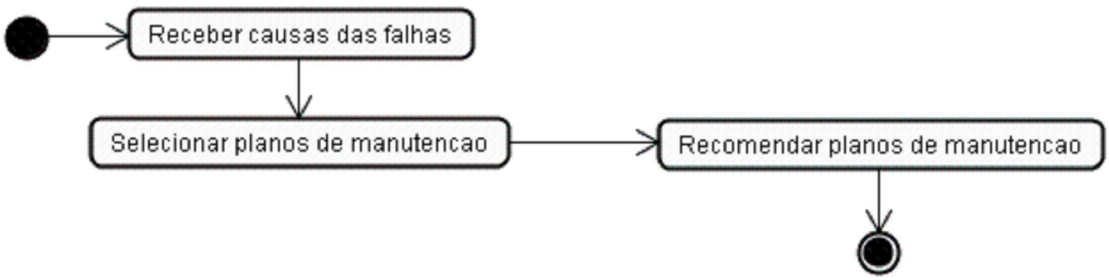

Figura 38 – Tarefas do agente Karem.

O agente Monitor possui tarefas que dependem do estabelecimento de conexão com o banco de dados do PIMS. Uma vez conectado, o agente Monitor inicia então as demais atividades, começando pela seleção dos dados que estarão no repositório. O uso do XML nas tarefas do agente foi uma alternativa para se armazenar temporariamente o resultado das consultas para evitar problemas em tempo de resposta entre a aplicação e o banco de dados do PIMS na leitura das tuplas da tabela do vv311k01.

Como se trata de uma fonte de dados de um sistema legado, esses dados precisam ser convertidos para um formato extensível e rápido para manter aspectos não funcionais como desempenho e interoperabilidade entre as aplicações, o que é conseguido com o desempenho das tarefas ‘converter para XML’ e ‘gerar arquivo XML’.

Após essas tarefas, a conexão é encerrada e o agente Monitor passa a executar uma tarefa de espera para realizar um novo processo após o tempo necessário para a utilização do repositório pelo agente Positioner, que consome os dados que ali estão para a identificação das causas das falhas que ocorrem no chão de fábrica. O tempo de espera de 5 minutos foi estipulado pela própria equipe de manutenção visando um melhor gerenciamento das recomendações do SADDEM antes da próxima atualização das *tagnames*. A Figura 39 ilustra o cenário 'seleção de falhas' das tarefas do Monitor.

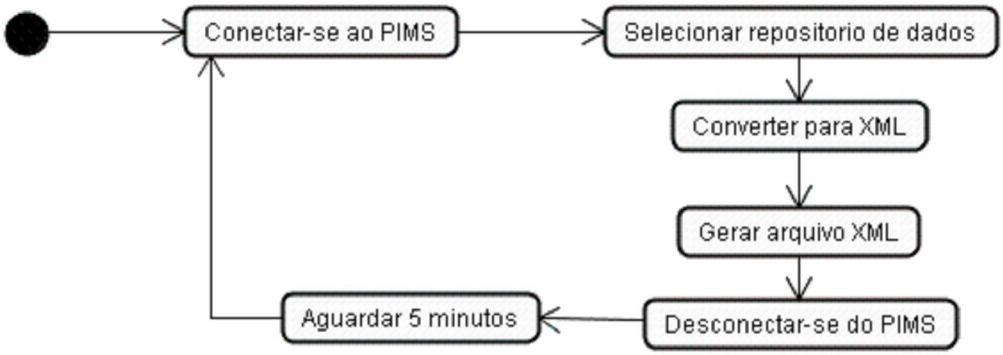

Figura 39 – Tarefas do agente Monitor

O agente Positioner é um agente que possui tarefas cognitivas concentradas em uma máquina de inferência para prover decisões das causas de falhas ocorridas no sistema carro posicionador. A primeira tarefa é a obtenção do repositório de dados para extrair as *tagnames* do arquivo XML através da leitura do conteúdo suas marcações – que neste caso são as tags do XML. Por exemplo, a tag <valor> 1 </valor> tem conteúdo 1 e indica no repositório que algum item falhou.

A próxima tarefa do agente Positioner é encadear os fatos por meio do mecanismo de inferência combinando as ocorrências encontradas no repositório. Há também uma sincronia nesse processo onde a inferência opera sobre um conjunto de regras e raciocínio simbólico complexo que determina como o agente Positioner deve reagir aos fatos e qual ação ele deve executar.

Vale lembrar que as tarefas de inferência, combinação e agrupamento das combinações são representadas apenas como uma interação realizada em alto nível, ou seja, o mecanismo de inferência de um agente é uma caixa preta e toda a complexidade computacional é algo que o implementador de sistemas multiagentes não precisa se preocupar em detalhar.

Neste mecanismo de inferência há um módulo de software feito com recursos heurísticos, otimizados e de bom desempenho para tratar problemas complexos, por essa razão as tarefas 'Inferir sobre os elementos do repositório', 'Combinar causas e *tagnames* de falhas' e 'Agrupar combinação causa-falha' foram desenhadas genericamente na Figura 40, pois essas tarefas sintetizam em parte a capacidade da própria máquina de inferência a ser utilizada.

Figura 40 – Tarefas do agente Positioner.

De posse da combinação causa-falha, o agente Positioner envia o resultado para o agente Karem, que por sua vez observa quais são os planos de manutenção adequados para as causas das falhas encontradas no sistema carro posicionador do vv311k01. As interações projetadas concluem as atividades deste estágio de desenvolvimento. A partir dos comportamentos, tarefas e comunicação envolvida nos cenários eliciam-se os elementos necessários para a construção da ontologia.

As próximas tarefas serão trabalhadas na fase de *Sociedade de Agentes* da metodologia PASSI. Nessa fase, os agentes serão desenhados conforme suas particularidades mais internas. Todos os cenários serão examinados levando em consideração a organização social dos agentes refinando as funcionalidades e avaliando os papéis envolvidos, capacidades de comunicação, bem como as dependências pelos agentes do sistema SADDEM.

4.3.4 Descrição da ontologia

A descrição da ontologia corresponde à etapa que apresenta o *Modelo da Sociedade de Agentes* da metodologia PASSI. Todos os passos realizados para a construção da ontologia `onto-positioner` do sistema SADDEM permitirão a definição dos mecanismos de comunicação dos agentes. É importante ressaltar que para cada comunicação são especificados três itens significativos que são: ontologia, linguagem e protocolo de interação. Normalmente os dois últimos são

padronizados pela FIPA, enquanto que a ontologia segue uma definição que é específica do ambiente do sistema SADDEM.

Para a composição da ontologia do sistema SADDEM inicialmente definiram-se seus elementos em conceitos, predicados e ações de agentes. Os conceitos foram organizados de acordo com as entidades consideradas significativas, ou seja, são entidades chave para a ontologia que normalmente são estruturadas com muitos detalhes e por isso muitas vezes dotada de certa complexidade.

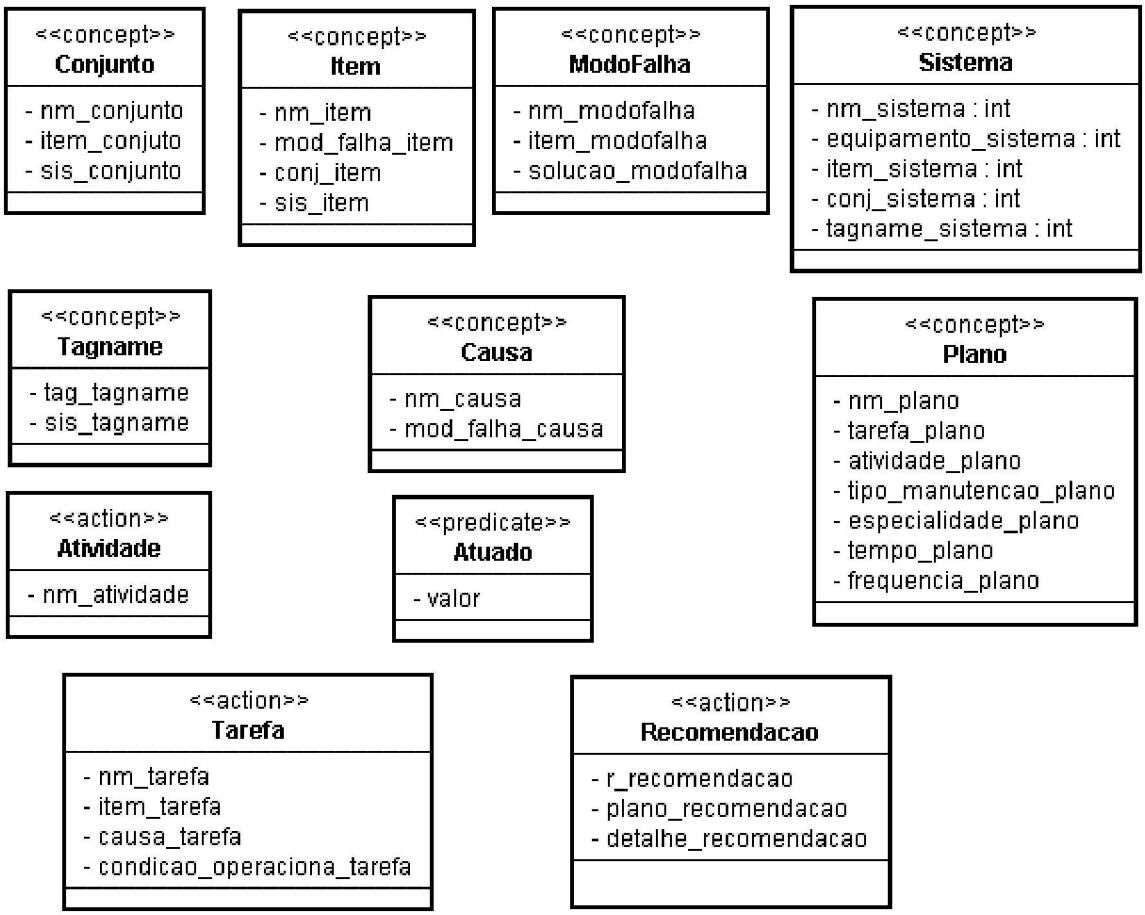

Figura 41 – Conceitos da ontologia do SADDEM.

Os conceitos presentes da ontologia, assim como os demais elementos (e.g. predicados e ações de agentes), foram selecionados com critérios baseados na classificação *Failure Mode and Effect Analysis* – FMEA (SCAPIN, 2007), estabelecida para equipamentos industriais e na metodologia *Reliability Centred Maintenance* - RCM (SMITH, 2007). A Figura 41 ilustra as primeiras entidades da ontologia do SADDEM em notação UML.

Para chegar aos elementos da Figura 41, poderiam ser organizadas outras formas de composição da ontologia do SADDEM. A partir da identificação dos conceitos mais importantes, a modelagem da ontologia de domínio pode ser esboçada com adição dos relacionamentos dos predicados, ações de agente e suas respectivas multiplicidades, conforme mostrado no diagrama da Figura 42.

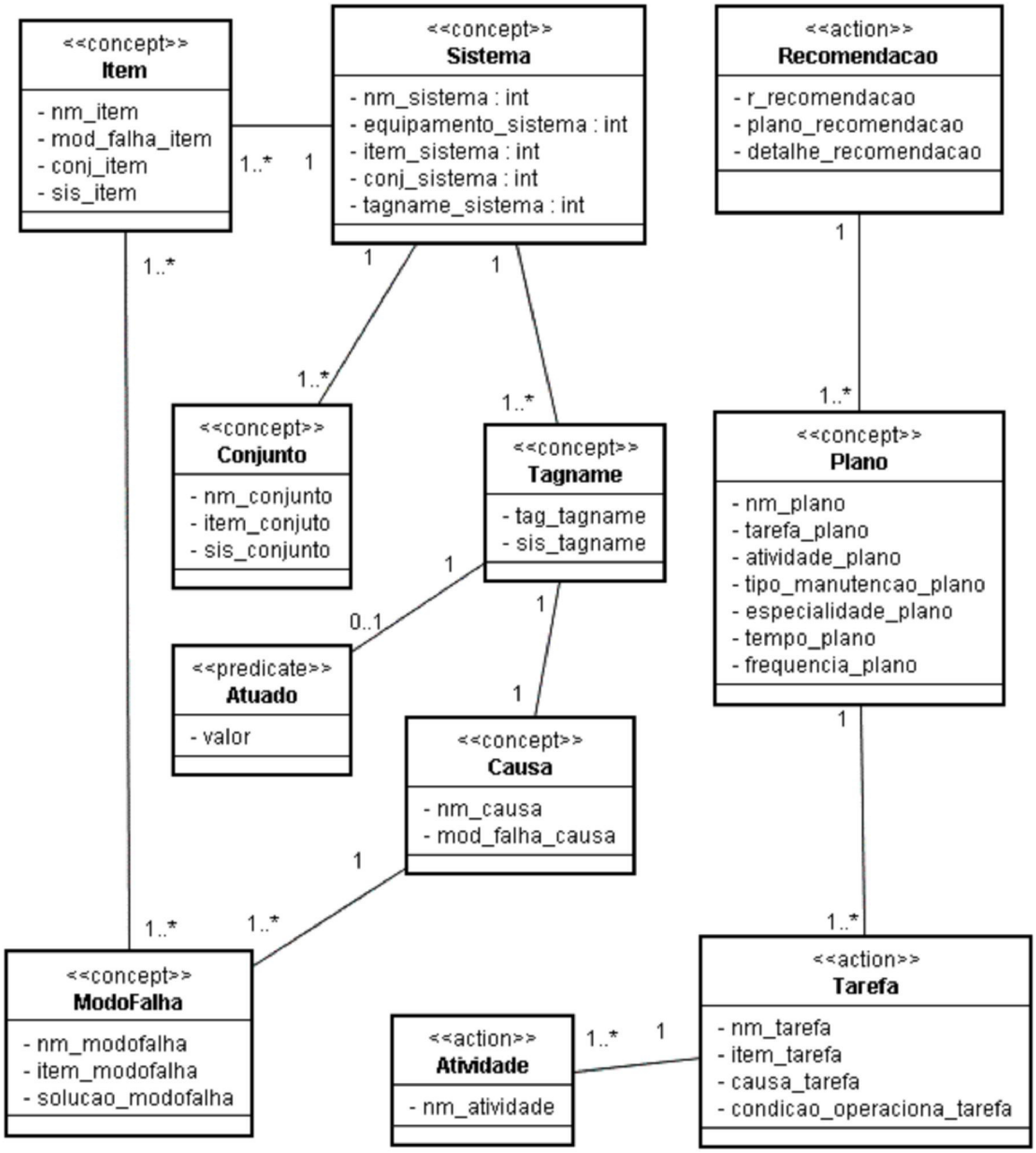

Figura 42 – Ontologia de domínio do sistema SADDEM – onto-positioner

4.3.5 Descrição de papéis

O esforço da metodologia PASSI, durante essa etapa, é projetar o ciclo de vida dos agentes focando seus papéis, colaborações e conversações envolvidos na

sociedade. O suporte notacional é dado em termos de diagramas de classe, destacando os papéis e pacotes para representar os agentes. Portanto, cada agente será visualizado como um pacote contendo dois ou mais diagramas classe.

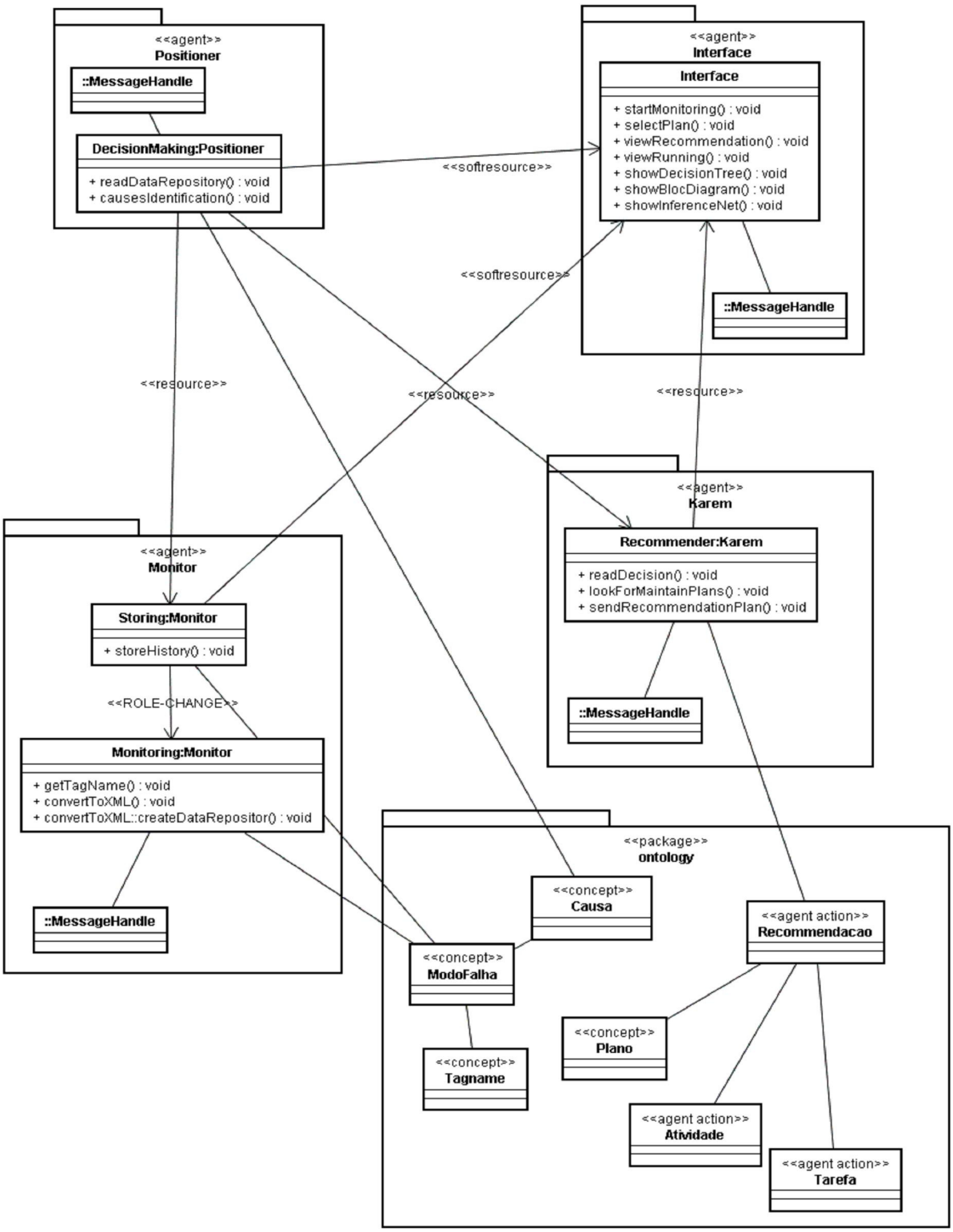

Figura 43 - Descrição dos papéis dos principais agentes do SADDEM.

Na Figura 43, além das classes da ontologia, estão aquelas com operações que revelam a síntese das tarefas de cada agente, destacadas em métodos resultantes dessa composição. Por exemplo, o agente Karem para alcançar seu objetivo de recomendação utiliza os meios de leitura da decisão do agente Positioner e de procura pelos planos de manutenção com seu respectivo envio. Essa prática pode melhorar a identificação de padrões de reuso, pois através dos métodos é que se podem selecionar quais são as capacidades dos agentes e em quais situações eles podem ser selecionados, adaptados e compostos.

Na descrição dos papéis da Figura 43, o agente Monitor representa dois papéis. Essa troca de papéis, caracterizada pela PASSI com o esteriótipo [ROLE_CHANGE], indica as capacidades deste agente para o monitoramento das *tagnames* e organização das falhas mais freqüentes, fazendo com que o mesmo controle as buscas no PIMS (ver Figura 34) e manipule as operações de armazenamento para consultas posteriores.

Os relacionamentos resources e softresources (CONSSENTINO e BURRAFATO, 2002) são dependências entre os agentes que indicam quais deles precisam disponibilizar entidades para consumo de outros. Os agentes Karem, Monitor e Positioner, estabelecem entre si um compromisso envolvendo as entidades modeladas na ontologia (Figuras Figura 41 e Figura 42) para contemplar a seleção de falhas, recomendação de planos e tomada de decisão.

Do lado do agente Interface, as dependências softresource são caracterizadas pelos recursos não essenciais, ou seja, basicamente requisitos não funcionais em que se adicionam propriedades qualitativas ao sistema, mas que não afetam o seu funcionamento caso os agentes Karem e Monitor não entregarem recursos para os métodos `showDecisionTree()` e `ShowInference()`.

4.3.6 Definição da Estrutura dos agentes

A etapa de estrutura dos agentes influencia e é influenciada pelos comportamentos dos agentes. Suas interações permitem visualizá-la sob duas dimensões: uma simples e a outra composta. Na primeira, leva-se em consideração a perspectiva multiagente e a segunda para um único agente (CONSSENTINO e BURRAFATO, 2002).

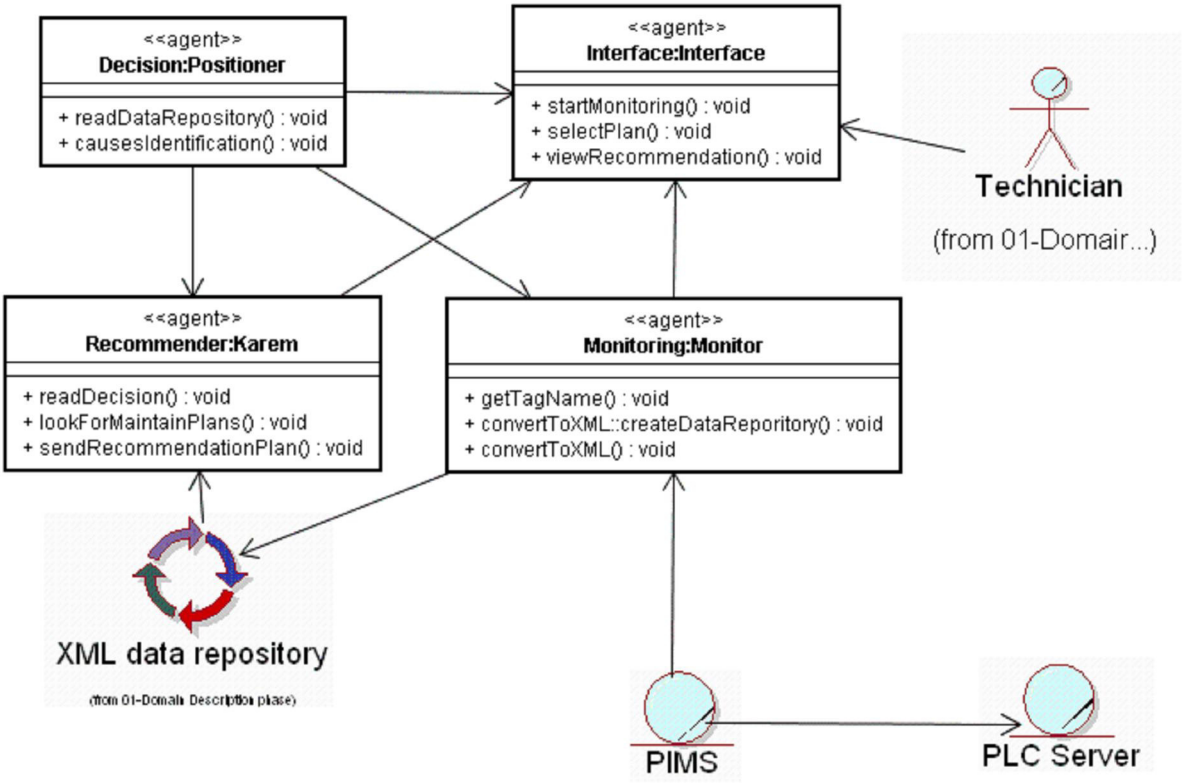

Figura 44 – Estrutura multiagentes do sistema SADDEM – Dimensão composta.

Inicialmente, toda a atenção dos agentes da Figura 44 se volta para o nível de grupo. Os agentes descobertos na etapa de identificação de agentes são visualizados agora como classes. O compartimento dos atributos representa o conhecimento dos agentes, já detalhado no diagrama da ontologia da Figura 42. O compartimento de operações é utilizado para representar as tarefas de um agente.

Modelaram-se também esteriótipos adicionais tais como PIMS, PLC Server, XML data repository e Technician. O propósito desta modelagem está na representação dos *stakeholders* de negócios, associados ao processo realizado pelo vv311k01, e na importância deles no ambiente multiagente, melhorando a compreensão de quais entidades o sistema SADDEM necessita consumir para produzir conhecimento durante o seu fluxo de trabalho.

Continuando a definição da estrutura, um diagrama de classe é usado por cada agente para ilustrar a composição interna dos agentes representados na dimensão composta. A classe Karem foi selecionada do modelo da Figura 44, juntamente com suas classes internas para identificar as tarefas em unidade. Na

Figura 45, apresenta-se o detalhamento do agente Karem, visando ajustar os atributos e métodos tanto das classes de agentes quanto das classes de tarefas.

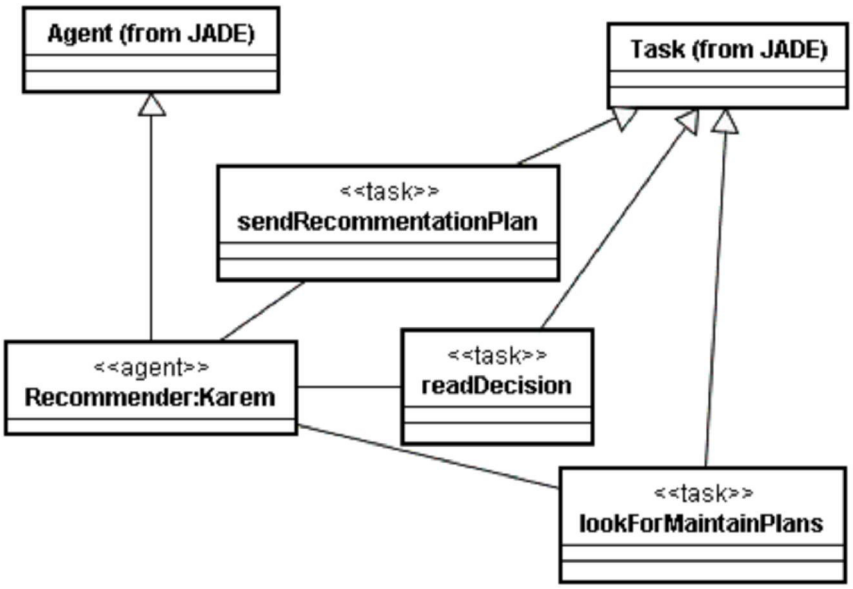

Figura 45 – Estrutura de unidade do sistema SADDEM para o agente Karem – Dimensão simples.

A modularização das interações da Figura 45 é uma prática importante para descrever a análise das funcionalidades dos agentes. Os módulos tratados como “caixas pretas” na Figura 44, focados na visualização externa do comportamento de um grupo de agentes, tornam-se “caixas-brancas” para concentrar os esforços de projeto na perspectiva de cada agente (CONSSENTINO e BURRAFATO, 2002).

Nesse sentido, o resultado desse estágio é obter uma estrutura detalhada do sistema, pronto para ser implementado quase que automaticamente. A verificação das alternativas encontradas para o desempenho coletivo e individual dos agentes é fundamental, pois um sistema multiagente exige muita cautela nas soluções empregadas para evitar problemas com os fluxos de controle resultante das operações *multithreading*.

4.3.7 Descrição do comportamento dos agentes

Para a descrição do comportamento dos agentes, elaborou-se um diagrama de atividades com *swiming lines* para destacar o fluxo de eventos que ocorre nos métodos das classes relacionadas. De acordo com a PASSI, o propósito deste modelo é detalhar os métodos. Assim, em cada *swiming line* coloca-se um agente e

uma única classe com seus métodos associados. A Figura 46 resume a organização comportamental dos agentes do SADDEM.

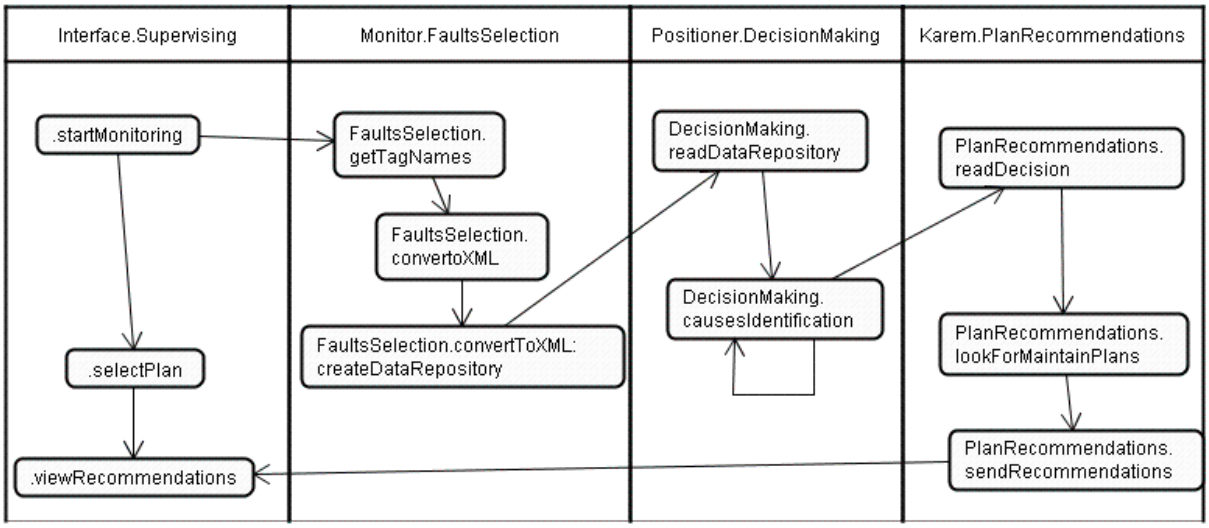

Figura 46 – Comportamento dos agentes do sistema SADDEM

Na Figura 46, cada agente responde por sua classe de competência, onde as operações agora podem ser visualizadas em detalhes. Nessa interação, o agente Interface inicia a visualização do estado do carro posicionador invocando o método `startMonitoring()` da classe `MainScreen`. Ainda nesta mesma guia, os métodos `selectPlan()` e `viewRecommendation()` oferecem a supervisão pelo usuário do sistema SADDEM para acompanhamento das falhas e os respectivos planos de manutenção.

A classe mais significativa do agente Monitor trata da seleção de falhas e suas metas são atingidas com os métodos `getTagnames()`, `convertToXML()` e `createDataRepository()`, já discutidos nos modelos anteriores. Em síntese, invocação deles envolve o processo de triagem do estado de funcionamento vv311k01, extraindo as *tagnames* relacionadas com o posicionador, convertendo-as para XML e disponibilizando um repositório para a leitura.

O agente Positioner apresenta a sua classe de tomada de decisão vinculada com os métodos para leitura do repositório criado pelo agente monitor e o mecanismo de inferência para identificar as causas de falhas ocorridas no posicionador do vv311k01. Respectivamente, essas operações são alcançadas pela invocação dos métodos `readDataRepository()` e `causesIdentification()`. Após isso, a decisão é enviada ao agente Karem pelo método `sendDecision()`.

A partir da decisão enviada pelo agente Posicionador, o agente Karem trabalha para atingir o seu objetivo no sistema que é recomendar os planos de manutenção para a equipe técnica. Os métodos relacionados com a classe `PlanRecommendation` revelam as operações vinculadas com a procura pelos referidos planos para as respectivas causas de não conformidades encontradas durante o procedimento de descarga.

Com o entendimento dos principais comportamentos dos agentes do sistema SADDEM, parte-se agora para o próximo estágio que é a concretização de toda a abstração realizada com o uso da metodologia PASSI e da ferramenta notacional UML. Nesse sentido, a solidificação da arquitetura do sistema, esboçada por meio da etapa de definição da estrutura de agentes, será o insumo de entrada para as etapas de implementação do SADDEM.

4.3.8 Implementação

A etapa de implementação da metodologia PASSI tem como objeto a realização do sistema, que neste contexto, inclui a arquitetura do SADDEM moldada durante as etapas da fase *Modelo da Sociedade de Agentes*. De forma específica, o sistema é implementado em termos de componentes de softwares que envolvem a codificação das classes de agentes e ontologia, subsistemas da descrição dos papéis de agentes, scripts, executáveis e etc.

Para a implementação do sistema SADDEM e seus agentes na plataforma JADE é necessário a linguagem de programação Java. As interações com as entidades externas utilizam as linguagens SQL e XML para o tratamento das informações contidas no PIMS. O agente Positioner foi implementado com a linguagem Java e Jess (FRIEDMAN-HILL, 2003) de forma embutida, ou seja, o Java para a definição da estrutura do agente e o Jess para o módulo cognitivo do mesmo.

O primeiro artefato codificado foi ontologia `onto-positioner`, feita com a linguagem Java, conforme a Figura 42. Cada elemento da ontologia possui uma classe com métodos *get* e *set* para os atributos que ela define. É importante lembrar que a linguagem semântica FIPA-SL foi utilizada nesta etapa de implementação para comunicação interagentes através da codificação dos elementos da ontologia

embutidos no *slot content* das mensagens ACL instanciando a classe SLCodec do pacote `jade.content.lang.sl`.

Partindo-se para a definição dos agentes, que na realidade são agentes JADE, na Figura 47 observam-se as declarações na implementação do agente Monitor. Essa é a parte mais trivial de toda a construção do sistema SADDEM, pois a estrutura é geral para praticamente todos os agentes.

```
01  package agents;
02  [...]
03  public class Monitor extends Agent{
04        private Codec codec = new SLCodec();
05        private Ontology ontology = Onto.getInstance();
06        private ContentManager contentManager = new ContentManager();
07
08        private Agent myAgent;
09
10        protected void setup(){
11
12              getContentManager().registerLanguage(codec);
13              getContentManager().registerOntology(ontology);
14          /*This behaviour contain the getTagname, convertToXML
15              and createRepositoryData methods */
16              this.addBehaviour(new FaultsSelection());
17          /*This behaviours handle the agent messages */
18              this.addBehaviour(new MessageHandle ());
19    }
```

Figura 47 – Implementação do agente Monitor.

Os pontos mais relevantes na classe do agente Monitor são as definições do *Codec* para a codificação do conteúdo das mensagens. Na seqüência, a definição da ontologia do sistema para a comunicação inter e intra-agentes do gerenciador do conteúdo sendo este responsável pela interpretação do conteúdo de uma performativa ACL.

No método setup do agente Monitor estão inseridos os comportamentos `FaultsSelection` e `MessageHandle` desenvolvidos como extensão da classe `CyclicBehaviour` do pacote `jade.core.Behaviours`. Enquanto o primeiro comportamento seleciona as *tagnames* no PIMS, o segundo trata das mensagens que o agente Monitor recebe dos demais agentes.

A classe `FaultsSelection` é composta por dois métodos: `getTagName()` e `convertToXML()`. Esses métodos foram reunidos em uma classe chamada `HandleXMLData`, separada especificamente para manipulação de documentos XML

e consulta de dados, importando respectivamente as API's `jdom.jar`, `xerces.jar`, `xercesImpl.jar`, `xml-api.jar` e `ora18n.jar`. A Figura 48 ilustra a parte do código da classe `FaultsSelection`.

```
01  package agents;
02  import utilities.xml.HandleXMLData;
03  [...]
04  //behaviour class in order to get tagnames from PIMS database
05  class FaultsSelection extends CyclicBehaviour{
06
07    private Hashtable <String, String> tb =
08                                      new Hashtable<String, String>();
09    private HandleXMLData          xmlData =
10                                      new HandleXMLData(null);
11    public void action() {
12      try{
13         //this method perform the queries in the PIMS database
14         xmlData.getTagName();
15         /*converts the sql query to XML file creating the data
16         repository*/
17         xmlData.convertToXML();
18        }
19        catch(Exception ex) {
20          ex.printStackTrace();
21        }
22     }
23  }
```

Figura 48 – Comportamento FaultsSelection do agente Monitor.

No método `getTagName()` foram definidos fluxos de controle cíclicos para a seleção de falhas no banco de dados do PIMS através da execução do arquivo `generate_xml.sql`, ficando sempre ativo e operando no sistema como um ouvinte dos dados operacionais do chão de fábrica relativos ao posicionador do vv311k01. Na Figura 49, resume-se parte da classe `FaultsSelection` onde está implementado o método `getTagName()`.

```
01  /** This method perform a connection with PIMS database */
02  public void getTagName(){
03
04      try{
05          Process p = Runtime.getRuntime().exec("generate_xml.slq");
06          InputStream ips = p.getInputStream();
07          int c = 0;
08          //this loop shows the execution result
09          while ((c = ips.read()) != -1) {
10             System.out.print((char)c);
11          }
12       }catch (Exception e) {
13           System.out.println("Problems during getting tagnames!");
14      }
15  }
```

Figura 49 – Método getTagName da classe FaultsSelection.

Em termos de fluxo de controle, o método `convertToXML()` é semelhante ao `getTagName()`, pois a conversão também realiza iterações executando um arquivo em lote chamado `generate_xml.bat`. É neste arquivo que está a consulta ao PIMS com a implementação necessária tanto para a conversão dos dados quanto para a geração do repositório de dados. A Figura 50 demonstra parte do código do método `convertToXML()`.

```
01  public void convertToXML(){
02      try {
03          Process p = Runtime.getRuntime().exec("generate_xml.bat");
04          InputStream ips = p.getInputStream();
05          int c = 0;
06          //this loop shows the execution result
07          while ((c = ips.read()) != -1) {
08              System.out.print((char)c);
09          }
10      }catch (Exception e) {
11         System.out.println("Problems during file convertion!");
12      }
13  }
```

Figura 50 – Método convertToXML da classe FaultsSelection.

Outro comportamento necessário para este agente diz respeito à comunicação com os demais agentes. Foi definida uma classe para recepção e envio das mensagens ACL e manipulação dos objetos da ontologia do sistema compartilhados durante a troca dessas mensagens. Para ilustrar a implementação da classe `MessageHandle` do agente Monitor, apresenta-se um fragmento do seu código na Figura 51.

```
01  class MessageHandle extends Behaviour{
02    public void action(){
03    //send messages with faults to Positioner agent
04    ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
05    msg.addReceiver(positionerAID);
06    [...]
07    if (msg != null) {
08      try {
09           ArrayList list = new ArrayList();
10           ContentElement ce = null;
11           ContentManager cm =  getContentManager();
12           ModoFalha modFalha = (ModoFalha) ce;
13           //handle the faults sent from Monitor
14           modFalha.setNmModoFalha(ce);
15           list.add(modFalha);
16           cm.fillContent(msg,modFalha);
17           send(msg);
18      }
19  [...]
```

Figura 51 – Comportamento MessageHandle do agente Monitor.

Na classe `MessageHandle` do agente Monitor uma instância da classe `ACLMessage` é criada para configurar o agente receptor (i.e. Positioner), a ontologia e o *codec*. Esse conteúdo é adicionado em uma lista, que contém os valores dos atributos da ontologia do sistema SADDEM, acessados pelo método `setNmModoFalha()` da classe ModoFalha. Em síntese, este comportamento permite ao agente Monitor enviar os modos de falhas para o agente Positioner poder realizar o seu processo de inferência.

Conforme mencionado, o agente Positioner é responsável por tarefas que exigem deliberação, ou seja, este agente passa por uma fase de raciocínio que decide quais causas podem estar relacionadas com determinados modos de falha. A parte cognitiva, conforme visto na Figura 52, implementa um comportamento para a tomada de decisão.

```
01  /** This agent has Jess engine embedded in Java classes*/
02  public class Positioner extends Agent{
03
04        private Codec codec = new SLCodec();
05        private Ontology ontology = Onto.getInstance();
06        private ContentManager contentManager = new ContentManager();
07        private Agent myAgent;
08        public void setup(){
09           getContentManager().registerLanguage(codec);
10           getContentManager().registerOntology(ontology);
11          /* Decision behaviours */
12          this.addBehaviour(new DecisionMaking("positioner.clp"));
13          /* behaviours in order to handle the agent messages */
14          this.addBehaviour(new MessageHandle());
15        }
16  [...]
```

Figura 52 - Implementação do agente Positioner.

Na classe de comportamento `DecisionMaking`, embutiu-se a linguagem Jess para utilizar os recursos da máquina de inferência. Esta classe implementa os objetos Jess na linguagem Java tendo como parâmetro a base de conhecimento para referenciar o arquivo com as regras do carro posicionador do VV311K01.

As regras do agente Positioner estão integradas aos métodos de leitura e acesso do repositório de dados, assim a inferência utiliza os dados do arquivo para combiná-los dedutivamente. Na classe `DecisionMaking`, utilizou-se o recurso de herança para compartilhar os objetos da classe `Reasoning`, incluindo a linguagem Jess.

É importante ressaltar que na classe `Reasoning` também existe uma herança de comportamento do tipo `CyclicBehaviour`, que por sua vez, é passada para a classe que a estende. Por essa razão, no método construtor da classe `DecisionMaking`, foi definido o parâmetro `jessFile` para referenciar o arquivo da base de conhecimento do agente Positioner.

O método `readDataRepository()` tem como parâmetro o arquivo XML com as *tagnames* e o método `causesIdentification`() da Classe `Reasoning` lista as causas dos modos de falha. Na Figura 53, estão presentes trechos da implementação da classe `DecisionMaking`.

```
01  //Inner behaviour class for decision making over positioner faults
02  class DecisionMaking extends Reasoning{
03
04      private XMLDataHandle xmlData = new XMLDataHandle();
05
06          public DecisionMaking(String jessFile) {
07              super(agent, jessFile);
08          }
09          public void action() {
10              try {
11                  //performs data repository the read
12                  xmlData.readDataRepository("dataRepository.xml");
13                  //causes identification
14                  super.causesIdentification();
15              }
16              catch(Exception ex) {
17                  ex.printStackTrace();
18              }
19          [...]
```

Figura 53 - Comportamento DecisionMaking do agente Positiner.

No segundo método do agente Positioner, foram declaradas as instruções para o tratamento das mensagens enviadas pelo agente Monitor que agrupa as falhas na classe `ModoFalha` da ontologia `onto-positioner` (Figura 42). A partir de então, o agente Positioner recebe a mensagen ACL com a informação das falhas lidas no repositório. Por essa razão, a performativa utilizada foi INFORM.

O conteúdo da mensagem é extraído através do método `getContentExtract()` da classe `ContentManager`. O agente Positioner verifica se no gerenciador de conteúdo há uma instância da classe `ModoFalha` da ontologia do SADDEM para chamar o método `getNmCause()` da classe `Causa` definida em `onto-positioner`.

No método `getNmCause()` são realizadas chamadas para a classe `Reasoning`, com o objetivo de iniciar o comportamento cognitivo do agente Positioner, encadeando-se os fatos presentes no objeto `modoFalha` utilizado como parâmento do referido método. A Figura 54 descreve o comportamento `MessageHandle` do agente Positioner.

```
01  class MessageHandle extends Behaviour{
02
03  public void action(){
04    //Receive the message
05    MessageTemplate mt = MessageTemplate.and(
06    MessageTemplate.MatchLanguage(codec.getName()),
07    MessageTemplate.MatchOntology(ontology.getName()) );
08    ACLMessage msg = blockingReceive(mt);
09    //receive agent local name
10    String rAgent = msg.getSender().getLocalName();
11      if (msg != null) {
12        //Message received. Process it
13        ACLMessage reply = msg.createReply();
14        try {
15             ArrayList list = new ArrayList();
16             ContentElement ce = null;
17             Causa cause = (Causa) ce;
18             ContentManager cm =  getContentManager();
19             ce = (ContentElement) cm.extractContent(msg);
20               if (msg.getPerformative() == ACLMessage.INFORM){
21                 if (ce instanceof ModoFalha){
22                   ModoFalha modoFalha = (ModoFalha) ce;
23                   //handle the faults sent from monitor
24                   cause.getNmCausa(modoFalha);
25                 }
26               }
27               reply.addReceiver(new AID(rAgent,AID.ISLOCALNAME));
28               reply.setPerformative(ACLMessage.INFORM);
29               list.add(cause);
30               cm.fillContent(reply,cause);
31               //send Decision to Karem agent
32               send(reply);
33               }
34               catch (OntologyException oe) {
35                 oe.printStackTrace();
36                 reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
37               }
38               catch (CodecException cde) {
39                 cde.printStackTrace();
40                 reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
41               }
42             }
43  }
```

Figura 54 - Comportamento MessageHandle do agente Positioner.

A próxima implementação a ser desenvolvida é a do agente Karem para recomendar planos de manutenção. Praticamente a definição da classe deste agente segue o padrão dos outros agentes. Dentre seus comportamentos, o mais

significativo está representado pela classe `PlanRecommentation`, que tem como parâmetro um arquivo contendo os planos de recomendação em formato lido pela máquina de inferência Jess, conforme visto na Figura 55.

```
01  public class Karem extends Agent{
02  [...]
03     protected void setup(){
04       getContentManager().registerLanguage(codec);
05       getContentManager().registerOntology(ontology);
06       /* contain the Jess rules in order to plans recommendation */
07       this.addBehaviour(new
08                      PlanRecommendation("RecommendationPlans.clp"));
10       this.addBehaviour(new MessagesHandle());
11  }
```

Figura 55 - Implementação do agente Karem.

Basicamente, outra diferença na implementação dos agentes Positioner e Karem está no comportamento implementado pelos métodos que tratam das causas de falhas codificados no método `action()` da classe `PlansRecommendation`. O método `readDecision()`, por exemplo, satisfaz a leitura das causas encontradas pelo agente Monitor.

Por outro lado, o método `lookForMaintainPlans()` executa a seleção de planos de recomendação, tomando como parâmetro a mensagem recebida, contento uma lista de causas inferidas pelo agente Positioner. A Figura 56 ilustra os métodos da classe de comportamento `PlanRecommendation` do agente Karem, ressaltando-se que `readDecision()` é chamado pela classe `XMLDataHandle`.

```
01  class PlanRecommnedation extends Reasoning{
02    [...]
03    public PlanRecommnedation(String jessFile) {
04      super(agent, jessFile);
05    }
06    public void action() {
07      try{
08         //read the positioner decision list from superclass
10         super.readDecision();
11         super.lookForMaintainPlans(jessFile);//search maintains plans
12   [...]
```

Figura 56 - Comportamento PlanRecommendation do agente Karem.

A partir das causas enviadas na mensagem, o agente Karem inicia o processo de procura pelos planos adequados às falhas do carro posicionador. Essa tarefa está implementada no método `lookForMaintainPlans()` que, por sua vez, foi declarado na superclasse `Reasoning`. É importante lembrar que o motor ou

máquina de inferência está codificado nesta superclasse e tem comportamento cíclico.

Outros métodos significativos da classe `Reasonig` são: i) `action()`, ii) `ACLJessTemplate()` e iii) `JessFactToACL()`. No primeiro método, foram implementados fluxos de controle condicional, repetição e exceção para o recebimento de mensagens ACL e inserção de fatos na memória de trabalho do agente Karem.

Outro ponto importante do método `action()` está relacionado com as mensagens que o agente Karem recebe do agente Positioner. Ao receber as decisões como conteúdo de uma determinada mensagem ACL, deve haver um mecanismo que permita o término de um ciclo de inferência do agente Karem. Para isso, utiliza-se uma chamada bloqueante por meio do método `blockReceiving()` do pacote `jade.core.Agent`.

```
01  public class Reasoning extends CyclicBehaviour{
02    public void action() {
03      ACLMessage msg;
04      if (totalPassos < numMaxPassos) {
05        System.out.println(myAgent.getName()+"Karem agent is blocked");
06        msg = myAgent.blockingReceive();
07        assertFato(mgsAClJess(msg));
08      } else {
10        System.out.println(myAgent.getName()+" Verifying messages");
11        msg = myAgent.receive();
12        if (msg != null)
13           assertFato(mgsAClJess(msg));
14        }
15       try{
16         if (numMaxPassos > 0) {
17           totalPassos = rete.run(numMaxPassos);
18           System.out.println("Completed "+totalPassos+" passe(s)");
19         }
20         else
21           rete.run();
22       }catch (JessException re) {
24         [...]
```

Figura 57 - Comportamento Reasoning do agente Karem.

Assim, enquanto o agente Karem estiver processando uma mensagem ACL em sua pilha, o mesmo entra em estado de bloqueio. Vale ressaltar que o bloqueio ocorre na *thread* do agente, ou seja, nos fluxos de controle relacionados com a recepção de mensagens. Na Figura 57, apresenta-se um fragmento do código do método `action()` da classe `Reasoning`.

Na realidade, bloquear os agentes - em situações como a do agente Karem - resulta em economia de recursos e constitui uma boa prática de implementação. Essa medida poupa consumo de CPU e evita a parada do sistema por falta de memória para a execução.

A implementação do segundo método da classe `Reasoning` cria o template `aclTemplate`, em semântica Jess, buscando descrever os fatos que serão inseridos na memória de trabalho do agente Karem. No template, foram implementadas variáveis que representam os slots de uma mensagem ACL. A Figura 58 ilustra um trecho de código do método `aclTemplate()`.

```
01   public String aclTemplate() {
02      String cmd = "(deftemplate aclTemplate " +
03                   "(slot performativa) " +
04                   "(slot remetente (type FLOAT)) " +
05                   "(multislot destinatario) " +
06                   "(slot conteudo))";
07      return cmd:
08   }
```

Figura 58 - Metodo aclTemplate da classe Reasoning.

Uma vez definido o template `aclTemplate`, é necessária a definição das instâncias da classe `Plano` para tratar os fatos referentes aos planos de manutenção. Para isso, na Figura 59, implementou-se o método `plansTemplate()` que gera um template para se obter acesso aos métodos `set` e `get`. Observa-se a importação do pacote `ontology`, onde foram definidas as classes da ontologia do sistema SADDEM para criar o template `plans`.

```
01   public String plansTemplate() {
02      String cmd = "(import ontology.*)" +
03                   "(defclass plans Plano)" +
04                   "(ppdeftemplate plans)";
05      return cmd;
06   }
```

Figura 59 - Método plansTemplate da classe Reasoning.

A implementação dos templates `aclTemplate` e `plansTemplate` armazena os comandos da linguagem Jess na variável do tipo `String`, que é retornada, respectivamente, pelos métodos `aclTemplate`() e `plansTemplate`. Além disso, ambos têm como meta o tratamento das mensagens intra e interagentes. Após a definição dos referidos métodos, foi possível implementar o

`jessFactToACL()`, método da classe `Reasonig`, conforme mostrado na Figura 60.

```
01  public ACLMessage jessFactToACL(Context context,jess.ValueVector vv){
02    int perf = ACLMessage.getInteger(vv.get(0).stringValue(context));
03    ACLMessage msg = new ACLMessage(perf);
04    System.out.println("**Remetente** " + vv.get(1).toString());
05    if (vv.get(1).stringValue(context)!="nil")
06      msg.setSender(tratAID.obtemAIDAgentes(vv.get(1).
07                                          stringValue(context)));
08      if (vv.get(2).toString() != "nil") {
09        List l = tratAID.obtemListaAgentes(context,
10                                        vv.get(2).listValue(context));
11        for (int i=0; i<l.size(); i++)
12          msg.addReceiver((AID)l.get(i));
13        }
14      if (vv.get(3).stringValue(context) != "nil") {
15        msg.setContent(tratMsg.semCote(vv.get(3).stringValue(context));
16      }
17      return msg;
18  }
```

Figura 60 – Método jessFactToACL da classe Reasoning.

O método `JessFactToACL()` traduz os fatos codificados em linguagem Jess para ACL. Nota-se que nos três métodos da classe `Reasoning` são mostradas as codificações tando em Java quanto em Jess. As abordagens Jess, Jade e Java oferecem subsídios para concretização dos modelos de um agente congnitivo em artefatos executáveis assim como foi escolhida para os agentes Karem e Positioner.

```
01  public lookForMaintainPlans(String jessFile){
02     rete = new Rete();
03     try {
04         rete.eval(plansTemplate());
05         rete.eval(aclTemplate());
06         [...]
07         FileReader freader = new FileReader (jessFile);
08         Jesp jesp = new Jesp(freader, rete);
09         jesp.parse(false);
10       [...]
```

Figura 61 - Método lookForMaintainPlans da classe Reasoning.

Nesse sentido, após uma visão interna da classe `Reasoning`, o método `lookForMaintainPlans()` instancia os objetos de inferência, por meio da classe `Rete` do pacote `jess.Rete`, que são responsáveis pelo mecanismo *matching-pattern* ou casamento de padrões, tais como as causas encontradas pelo agente Positioner e os respectivos planos de manutenção. A Figura 61 resume a codificação do método `lookForMaintainPlans()` do agente Karem.

No método `lookForMaintainPlans()`, as declarações de código mais importantes destacam as intâncias das classes `Rete`, `FileReader` e `Jesp`. Ambas pertencentes ao pacote `jess.*`. A instância `rete` realiza uma chamada à função `eval()` do Jess para executar os comandos definidos nos templates `aclTemplate` e `plansTemplate`.

A instância `freader` garante acesso à base de conhecimento do agente Karem ao carregar o conteúdo do arquivo Jess (`.clp`). A base de conhecimento é formada por regras que incluem as condições para a recomendação dos planos de manutenção. Além disso, `freader` também é um dos parâmetros da função `parse` da instância `jesp`, utilizada para análise sintática de códigos em escritos em Jess.

```
01   class MessageHandle extends Behaviour{
02       [...]
03       public void action(){
04       [...]
05           ACLMessage reply = msg.createReply();
06
07                try {
08                   ArrayList list = new ArrayList();
09                   ContentElement ce = null;
10                   Recomendacao rec = (Recomendacao) ce;
11                   ContentManager cm =  getContentManager();
12                   ce = (ContentElement) cm.extractContent(msg);
13                   if (msg.getPerformative() == ACLMessage.INFORM){
14                        if (ce instanceof Cause){
15                            Cause cause = (Cause) ce;
16                            rec.getPlano();
17                            rec.getDetalhe_Recomendacao();
18                        }
19                   }
20                   reply.addReceiver(new AID(rAgent,AID.ISLOCALNAME));
21                   reply.setPerformative(ACLMessage.INFORM);
22                   list.add(cause);
23                   cm.fillContent(reply,cause);
24                   send(reply);
25
26       [...]
27   }
```

Figura 62 - Comportamento MessageHandle do agente Karem.

O outro comportamento do agente Karem foi definido no método `MessageHandle()`, que é semelhante àquele declarado para o agente Positioner. Os elementos da ontologia `onto-positioner` são instanciados para codificar o fluxo de mensagens deste agente, acessando os métodos declarados na classe `Recomendação`. A Figura 62 resume o comportamento do agente Karem.

O tratamendo das mensagens que chegam ao agente Karem tem como entrada os objetos da classe de ontologia `Causa`. O conteúdo é extraído da mensagem para que haja a inserção dos fatos no mecanismo de raciocínio do agente. Na classe de ontologia `Recomendacao`, são executas ações baseadas em inferências de planos de manutenção por meio de chamadas aos métodos `getPlano()` e `getDetalhe_Recomendacao()`.

Após especificar toda a implementação necessária para os agentes cognitivos Positioner e Karem, suas respectivas bases de conhecimento puderam ser construída. Para o agente Positioner, foram declaradas 250 regras com base na situação do carro posicionador do VV311K01, em relação aos modos de falhas apresentados. O Quadro 1 exemplifica o formato das regras em linguagem natural e seu equivalente em Jess.

Linguagem Natural	Linguagem Jess
SE a tagname AFF_CEP_F01@VV311K01 atuar ENTAO modo falha indica situação de fim da vida útil	(defrule regra20 (test (eq TRUE (tagAtuada "AFF_CEP_F01@VV311K01"))) => (store RESULT21 "Fim da vida util") (assert (VidaUtil falhas)))
SE a tagname AFL_BEP_001@VV311K01 atuar ENTAO Modo falha indica situação de graxa contaminada	(defrule regra20 (test (eq TRUE (tagAtuada "AFL_BEP_001@VV311K01"))) => (store RESULT20 "Graxa contaminada") (assert (GraxCont falhas)))
SE a tagname ATA_BRT_M01@VV311K01 atuar ENTAO Modo falha indica situação de travamento do freio	(defrule regra21 (test (eq TRUE (tagAtuada "ATA_BRT_M01@VV311K01"))) => (store RESULT21 "Travamento do Freio") (assert (TravFrei falhas)))

Quadro 1 – Regras da base de conhecimento do agente Positioner.

Na regra 20 do Quadro 1, a função `tagAtuada` recupera o valor de cada tagname do repositório e envia o resultado para a função `test`. Se o resultado for verdadeiro, então o mesmo é armazenado em `RESULT20` para ser usado na classe `ModoFalha` da ontologia. Na sequência, é inserido o fato `VidaUtil` na memória de

trabalho para ativar as regras que serão processadas. O Quadro 2 exemplifica o formato da base de conhecimento do agente Karem constituída de 100 regras.

Linguagem Natural	Linguagem Jess
SE a causa for Falta de lubrificação ENTAO EXECUTAR MP NA UNIDADE HIDRÁULICA DA TRAVA AUXILIAR DO VV-311K-01-P	(defrule regra58 (FaltLubr falhas) => (store RESULT58 "EXECUTAR MP NA UNIDADE HIDRÁULICA DA TRAVA AUXILIAR DO VV-311K-01- Plano 5A"))
Se a causa for sobrecarga Então MP ELÉTRICA VV-311K-01 - MOTOR CC	(defrule regra70 (Sobr falhas) => (store RESULT70 "MP ELÉTRICA VV-311K-01 - MOTOR CC - Plano 8F;8G"))
SE causa for afrouxamento dos parafusos da base OU Sobrecarga no motor ENTAO MEDIR VIBRAÇÃO NO VV-311K-01	(defrule regra51 (AfroParaBaseE/ouSobrMoto falhas) => (store RESULT51 "MEDIR VIBRAÇÃO NO VV-311K-01 - Plano 1A"))

Quadro 2 – Regras da base de conhecimento do agente Karem.

Com esforços de implementação do sistema SADDEM, finaliza-se o desenvolvimento de seus componentes executáveis mais importantes. As próximas atividades apenas sintetizam as especificações de código no *framework* JADE e o modo de apresentação da interface do usuário dos cenários descritos no processo.

4.3.9 Comunicação entre os agentes no JADE

Para visualização da atuação dos agentes durante a comunicação, os mesmos foram inicializados nas ferramentas RMA e *Sniffer*, conforme a Figura 63. Ambas auxiliam no gerenciamento e depuração do *framework* JADE.

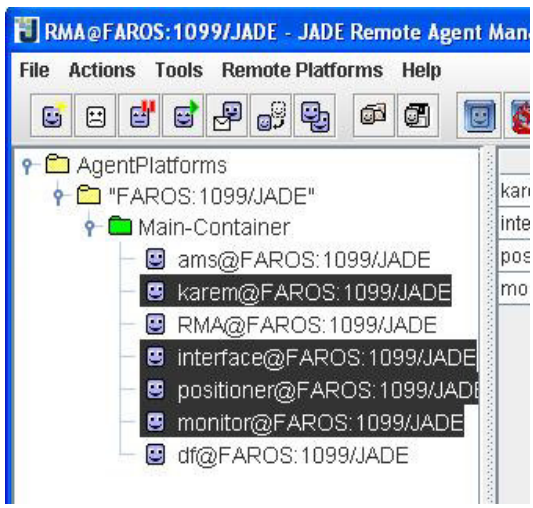

Figura 63 – Agentes do SADDEM inicializados no RMA JADE.

Para o monitoramento dos agentes do sistema SADDEM no ambiente JADE, os mesmos foram carregados na ferramenta de Sniffer para observação das mensagens trocadas entre os mesmos em gráfico semelhante ao diagrama de sequência da UML, conforme mostrado na Figura 64.

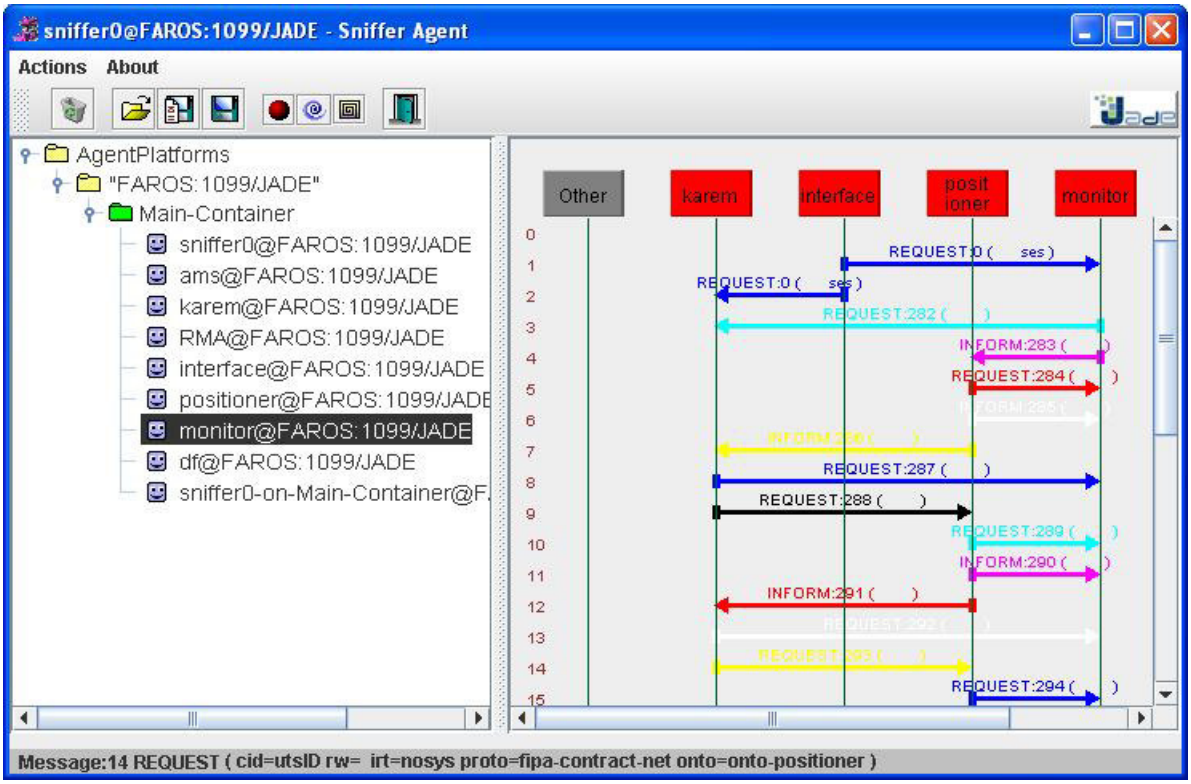

Figura 64 – Mensagens do SADDEM capturadas pelo Sniffer JADE.

Todas as mensagens dos agentes do SADDEM, carregados à esquerda da Figura 64, são rastreadas enquanto está executando. É possível também obter informações das mensagens que são trocadas com apenas um clique sobre as setas.

4.4 Experimentos e Resultados

Para observar as condições de funcionamento do sistema SADDEM e detectar os fatores que influenciam ou não nos resultados, realizaram-se experimentos no Laboratório de Sistemas Inteligente da Universidade Federal do Maranhão para evitar interrupções na linha de produção do virador de vagões, ressaltanto, contudo, que os ensaios experimentais são efetuados seu carro posicionador.

Nesse sentido, as ferramentas empregadas nos experimentos são as mesmas utilizadas nas instalações operacionais de descarga da VALE, dentre as quais estão o Sistema Gerenciador de Banco de Dados Relacional Oracle e instâncias das tabelas contendo as *tagnames* do VV311-K01para geração do repositório de dados.

Como se trata de um ambiente baseado em conhecimento, a avaliação do sistema SADDEM foi feita por um engenheiro de manutenção da VALE para julgamento dos resultados processados pelos agentes cognitivos no que tangem os modos de falha, causas e recomendações de planos. A interação com o usuário ocorre por meio de uma interface gráfica que mostra o *status* do sistema em tempo real, conforme visto na Figura 65.

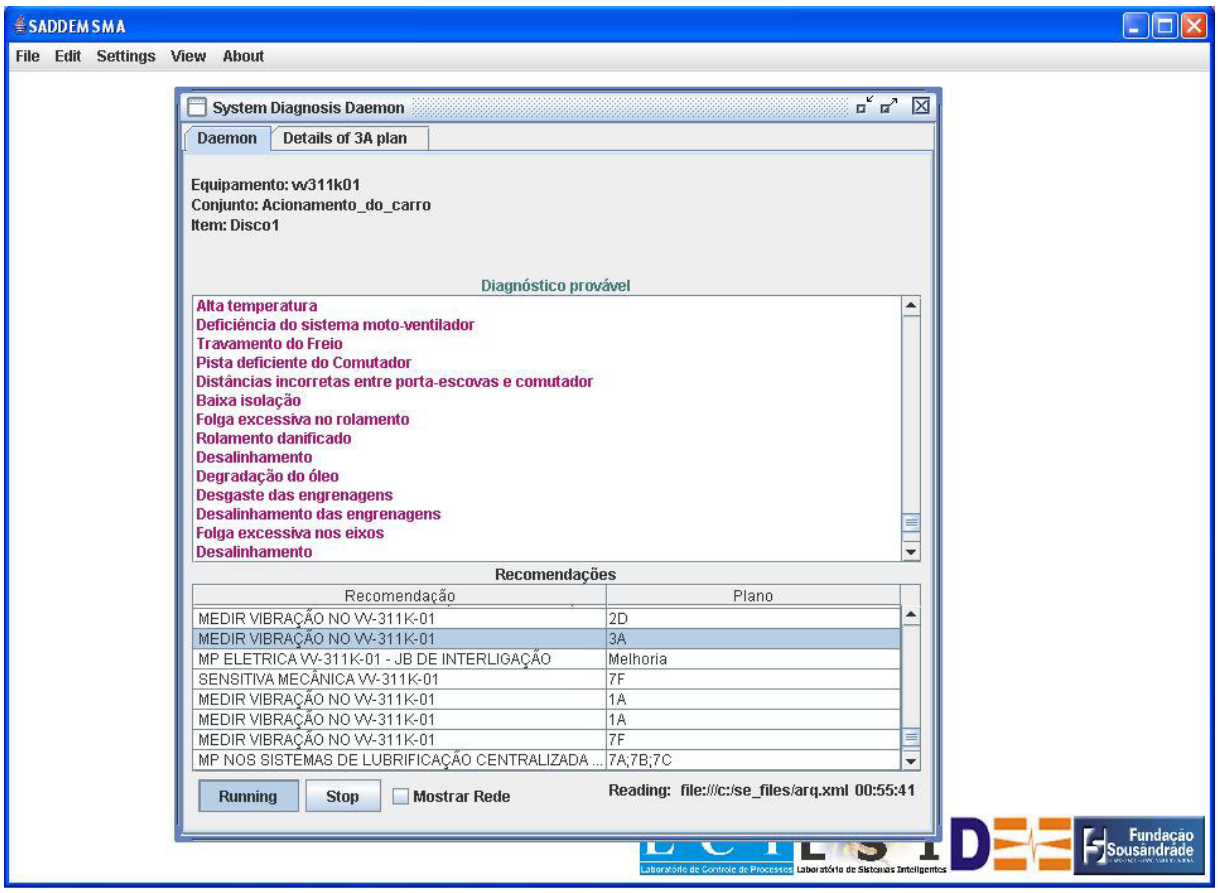

Figura 65 – Interface do Sistema SADDEM.

Na área ‘Diagnóstico provável’ estão as causas relacionadas às falhas do item ‘Disco1’ do conjunto ‘Acionamento do carro’ do VV311K01, inferidas pelo agente Positioner de acordo com as ocorrências encontradas no repositório. Na área

‘Recomendações’ o agente Karem se responsabiliza em mostrar as recomendações e os respectivos planos de manutação.

Na realidade, para cada plano recomendado, o sistema SADDEM detalha as atividades necessárias para a sua execução. Além disso, informa a quantidade de pessoas para a realização do trabalho. A Figura 66 ilustra o detalhamento do plano 3A, que adiciona as informações necessárias para a recomendação ‘MEDIR VIBRAÇÃO NO VV311K01’, selecionado pelo usuário do sistema.

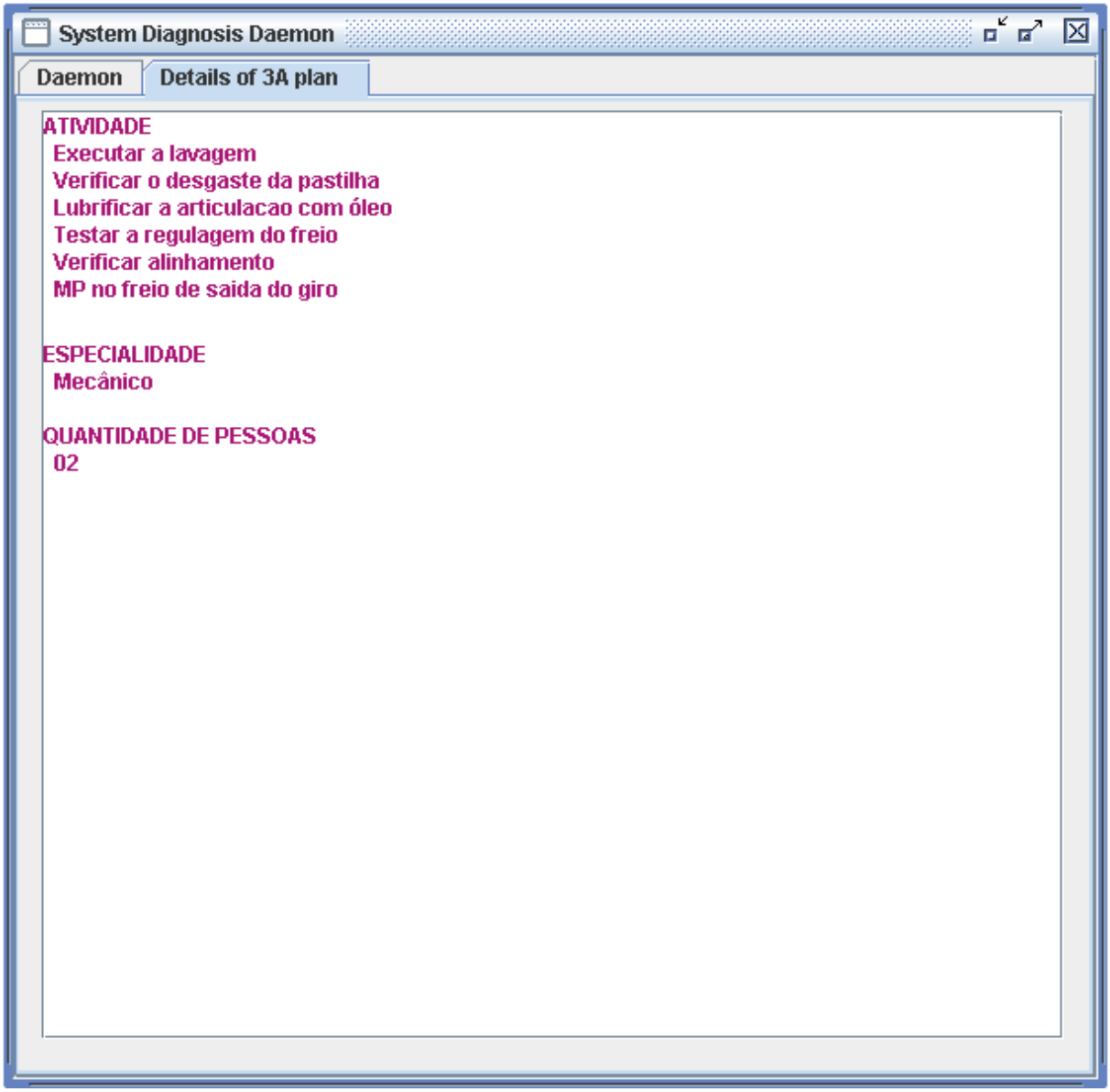

Figura 66 – Detalhamento das atividades dos planos de manutenção.

Durante o funcionamento do sistema, ajustou-se gradativamente o tempo de espera necessário para a realização de consulta, criação do repositório de dados e recomendação de planos de manutenção para intervalos $0 < t < 5$ segundos,

visando observar o desempenho dos agentes cognitivos em ciclos menores em computadores com processador Intel de 3Ghz e memória RAM de 512MB – configuração normalmente usada nas instalações do setor de descarga da VALE – e principalmente por estes possuírem um motor de inferência que demanda grande recurso de processamento e por estarem em ambiente *multithreading*. Para t = 2 segundos, o sistema aprensentou funcionamento adequado.

Neste mesmo computador, para 0 < t < 2 segundos, o SGBD Oracle não conseguiu suportar o número de processos executados com a quantidade de memória RAM na sua configuração padrão e retornou o erro ORA-04031, que está relacionado com o uso de memória além do disponível, fazendo com que o SADDEM acabasse gerando uma exceção.

Para resolver esse problema, ajustaram-se aguns parâmentros do Oracle relacionado ao gerenciamento de memória e alteração do DB_CACHE_SIZE (ORACLE, 2006). Após este ajuste, conseguiu-se apenas retardar o erro e não resolvê-lo. A partir desse primeiro experimento, executou-se o SADDEM em outra máquina com 1GB de memória RAM e mesmo processador do computador anterior, preservando a configuração padrão dos parâmentros do Oracle e foi possível obter o funcionamento apropriado do sistema.

Mesmo com os ensaios realizados, o sistema SADDEM se mostra adequado para o ambiente operacional de descarga por oferecer tempo de resposta menor do que o considerado necessário pelos técnicos e engenheiros da VALE. Ressaltando-se que se for necessário diminuir o tempo de resposta, isto é, para 0 < t < 2 segundos, haverá maior consumo de memória RAM para a execução do sistema.

Também foram realizados testes para validar a eficiência do sistema no diagnóstico de causas e nos planos recomendados. Para que isso fosse possível, o sistema foi avaliado por técnicos do setor de descarga da VALE. Basicamente o teste consiste na análise dos resultados observados e esperados para cada área de manutenção mencionada.

Os resultados do Quadro 3 foram obtidos conforme as decisões tomadas pelo técnico de operações da VALE e aquelas tomadas pelos agentes do SADDEM. As falhas que não foram detectadas corretamente por parte do técnico, marcadas

com um ‘X’, comparadas com as mesmas não encontradas pelo SADDEM, revelam tanto sua utilidade quanto suas restrições para detectar as falhas tais como ‘polias fora do lugar’ e ‘desalinhamento’. Na realidade, estas eram falhas que se auto-recuperavam em alguns ciclos e o sistema não conseguia detectar corretamente. Para situações como estas houve necessidade de revisão das regras.

Conjunto	Falha	Detectada corretamente pelo técnico	Detectada corretamente pelo SADDEM
Comando do braço	bloco contato danificado	✓	✓
Comando do carro	Falta de medição	✗	✓
Comando do acionamento	polia fora do lugar	✓	✗
Comado do acionamento	comutação deficiente	✓	✓
Acinamento do carro	Vibração	✗	✓
Acionamento do carro	aquecimento dos rolamentos	✓	✓
Comando do braço	desalinhamento	✗	✗
Comando do acionamento	Queima	✗	✓

Quadro 3 – Validação do sistema SADDEM.

Visando uma avaliação mais refinada, outro meio adotado para validação do sistema foi baseado na fórmula do quiquadrado, proposto em algumas literaturas (PINTO, 2001, ALVES *et al.*, 2006). Nesta dissertação, utiliza-se essa equação para observação das colaborações dos agentes durante as decisões tomadas com a participação de um engenheiro e dois técnicos para a realização dos testes. Na metodologia aplicada para validação foram selecionadas as falhas mecânicas, elétricas e hidráulicas. A Figura 67 apresenta a fórmula do quiquadrado.

$$X^2 = \sum_{i}^{n} \frac{(fo_i - fe_i)^2}{fe_i}$$

Figura 67 – Fórmula do quiquadrado

Onde, fo_i = freqüência observada dos acertos dos agentes do SADDEM em porcentagem. fe_i = freqüência esperada dos acertos dos agentes do SADDEM. Para

se chegar ao resultado da Figura 67, em cada tipo de falha, utilizaram-se os seguintes critérios: i) Percentual de 90% para freqüência esperada, ii) Para a prova de X^2 > 0,05, não há diferença significativa entre as decisões do SADDEM e dos especialistas e iii) Para a prova de X^2 < 0,05, há diferença nas decisões do SADDEM e dos especialistas.

Pretende-se, primeiramente, comparar o valor encontrado pela estatística do teste com um valor estipulado para validação de softwares, como é o caso do SADDEM. A segunda finalidade é atestar que o comportamento do SADDEM não é diferente do especialista com resultado maior que 0,05. Os esforços do processo de validação são revelados no gráfico da Figura 68.

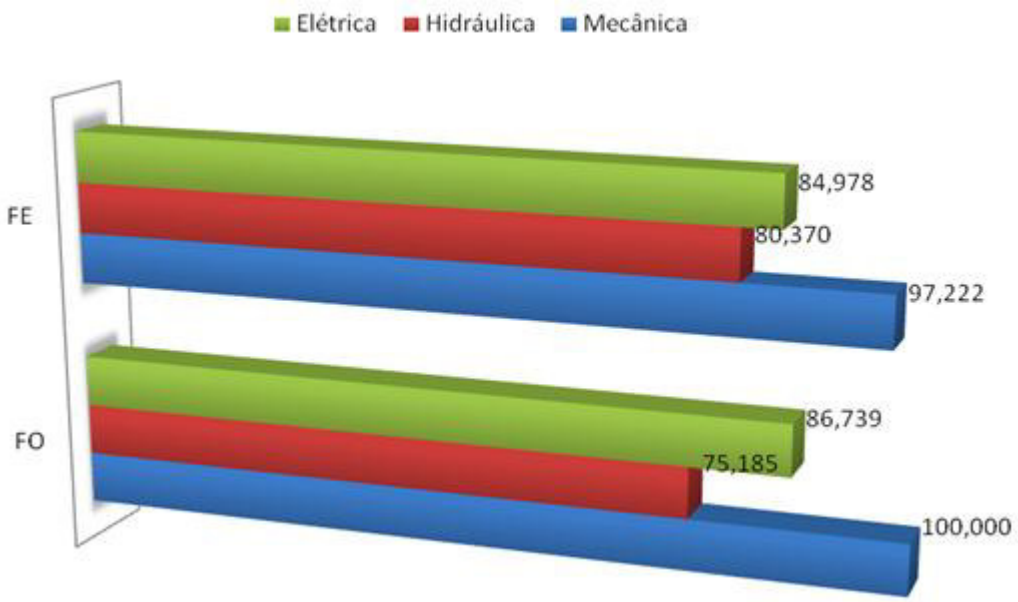

Figura 68 - Quiquadrado - Validação do SADDEM.

Com o gráfico da Figura 68, confirmam-se as limitações do SADDEM com relação a algumas falhas mecânicas observando-se os valores da freqüência observada e esperada, sendo que esta última ficou abaixo do percentual pelo fato de não haver sensores para todos os dispositivos mecânicos. Por outro lado, a parte hidráulica teve freqüência esperada de acordo com a porcentagem estipulada.

Embora a parte elétrica tenha apresentado maior freqüência observada, a freqüência esperada do sistema SADDEM também ficou com pequena alteração. A prova de X^2 ficou igual a 0,388, indicando que não há diferenças significativas entre os participantes da validação. É importante lembrar que os resultados da validação

do sistema também se mostram como excelentes mecanismos para descobrir quando e onde a base de conhecimento necessita de alguma mudança na lógica das regras.

De maneira geral, o sistema SADDEM foi considerado adequado para ser utilizado como apoio a tomada de decisão, haja vista que o sistema também pode ser encarado como uma ferramenta de treinamento para vários níveis de *expertise*. Além disso, com o resultado da avaliação foi possível subtrair o tempo gasto para a identificação das falhas na parte mais crítica do VV311-K01e validar o sistema proposto nesta pesquisa sem interferência nas células produtivas da VALE.

4.5 Considerações Finais

Neste capítulo, abordou-se o desenvolvimento do SADDEM feito a partir da integração de várias ferramentas que dentre as mais relevantes estão: JADE, Jess e IDE de programação Eclipse. Na construção do sistema, foi proposta a metodologia PASSI para o gerenciamento e controle dos níveis de abstração exigidos durante a sua elaboração modelados com notação gráfica baseada em UML utilizando a ferramenta Rational Rose.

Durante o processo de desenvolvimento, os diagramas modelados foram refinados à medida que os artefatos se aproximavam da solução e de acordo com as tecnologias empregadas. Para melhor entendimento da aquisição de conhecimento dos agentes cognitivos e de como o especialista humano toma decisões relacionadas às falhas do carro posicionador, representaram-se as instâncias de conhecimento no modelo de raciocínio do agente Positioner, baseando-se na metodologia CommonKADS.

Os códigos fontes estão resumidos aos trechos mais relevantes. Ressalta-se que outras classes também foram aplicadas no SADDEM, mas como tais objetos são básicos, classes de conexão à banco de dados, consultas e tratamento de strings, não foram julgadas como necessárias.

A ontologia definida para o sistema facilitou a compreenção do conteúdo das mensagens trocadas pelos agentes. O compartilhamento do conhecimento global do SADDEM foi organizado de acordo com conceitos, predicados e ações de agentes, produtores de artefatos concretos das definições do ambiente de descarga da VALE,

pois através desses elementos os agentes puderam compreender qual o significado de um modo de falha, o que fazer com as causas encontradas, e como recomendar os planos de manutenção, além de outros recursos da ontologia.

A sociedade de agentes definida para o sistema SADDEM envolveu o projeto global de quatro agentes e detalhando três deles. Dois agentes possuem arquitetura cognitiva e são os principais responsáveis pelas decisões e recomendações de planos de manutenção. Cada agente foi implementado destacando seus métodos mais significativos.

Completando-se as atividades de desenvolvimento, descreve-se a validação do SADDEM que foi feita por especialistas da área de descarga de minério. Basicamente foram observados seus componentes funcionais e a sua forma de raciocinar nas situações de falhas, comparando os resultados obtidos pelos técnicos e engenheiros com os alcançados pelo sistema.

No próximo capítulo, apresentam-se os potenciais benefícios gerados pela proposta de modelagem e implementação do sistema SADDEM, bem como suas contribuições na visão geral das conclusões e trabalhos futuros.

5 CONCLUSÕES E TRABALHOS FUTUROS

A concretização dos mercados globalizados e as inovações tecnológicas têm exigido crescentemente posicionamento estratégico dos profissionais tomadores de decisão. O fator tempo implica em deliberações aceleradas para a obtenção da informação correta e no momento certo. Nesse contexto o sistema SADDEM surge para o aperfeiçomento dos processos de negócio para seleção de falhas e tomada de decisão com recomendação de planos de manutenção.

Nessa perspectiva, o uso de agentes e sistemas multiagentes apresentou-se como um alicerce tecnológico satisfatório e adequado para os procedimentos organizacionais do setor de descarga de minérios da VALE referentes aos equipamentos viradores de vagões.

As melhorias adicionadas às células produtivas estão relacionadas com o tempo para a identificação das causas de falhas do equipamento e sua correta identificação e aplicação do sistema on-line, pois o mesmo tem comunicação com o supervisório. Adicionando a isso, outros benefícios alcançados destacam a abordagem da Inteligencia Artificial e mecanismos de inferência.

O sistema SADDEM apresentou propostas para a resolução de situações que, mesmo bastante conhecidas pelos tomadores de decisão, na maioria das vezes acabavam permitindo aos mesmos o confronto com processos decisórios que se arrastam indefinidamente por não disporem de ferramentas de conhecimento para o tratamento do volume dos dados operacionais.

Uma das propostas para as situações citadas é baseada na troca de ideias, uma vez que há dois agentes que se comunicam trocando os conhecimentos dos especialistas modelados em uma base de regras. Outra proposta sintetiza o processo de deliberação, ou seja, com o intuito de evitar tomada de decisões rápidas com respostas apressadas demais, o SADDEM foi também considerado uma espécie de sistema facilitador de escolhas em tempo real. Nesse sentido, velocidade é um dos fatores que dão suporte para a minimização de tempo e erros cometidos em determinadas intercorrências.

As principais contribuições desta dissertação são:

1. Construção de um SMA com arquitetura extensível, flexível e portável centrada em agentes inteligentes.

A estrutura do sistema SADDEM foi construída com terminologias reusáveis para a inserção de outros módulos referentes a outros equipamentos do chão de fábrica da VALE, dentre eles, correias transportadoras, empilhadeiras, recuperadoras e carregadores de navios. A flexibilidade está presente na existência de uma ontologia e bases de conhecimento.

Novos conceitos, predicados e ações de agentes podem ser adicionados na aplicação, bem como novas regras. O espaço tecnológico é todo baseado em Java para ser aplicado em qualquer sistema operacional sem recompilação de código fonte.

2. Modelagem do sistema baseado na metodolgia PASSI e notação gráfica UML.

A metodologia PASSI reúne um conjunto de técnicas para o desenvolvimento rápido de sistemas multiagentes de fácil compreenção e com ferramentas para automatização de algumas etapas de modelagem. A UML representa outra colaboração, pois se trata de um padrão e, por isso, permitiu facilidade de entendimentos dos diagramas.

3. Sistema SADDEM on-line.

A integração do sistema na linha de produção é uma das principais contribuições.

4. Interoperabilidade.

A comunicação do SADDEM com o sistema de informações PIMS foi feita com XML para manter a certa independência da solução proposta e possíveis restrições de banco de dados, além de acelerar o processamento necessário para leitura das *tagnames* por meio dos agentes.

5. Validação do SMA SADDEM.

Poucos trabalhos abordam a validação de um sistema baseado em connhecimento como os sistemas multiagente. Esta foi uma etapa onde as características não funcionais do sistema SADDEM foram colocadas à prova para constatar a sua utilidade e refinar as lógicas de decisão não atendidas pelo

conhecimento dos agentes por meio de um modelo estatístico. Em resumo, tais aspectos foram identificados com maior aproveitamento durante a utilização do sistema e comparação dos dados observados com os esperados.

Como trabalho futuro, pretende-se construir um módulo de acesso a dados utilizando o protocolo *OLE Process Control* - OPC e comparar o tempo de resposta com a solução aqui apresentada. Além disso, há também enorme interesse de se estudar outros mecanismos de raciocínio e nas arquiteturas de agentes cognitivos para o suporte ao transporte de minério inferindo as decisões baseadas nas escolhas de rotas realizadas por técnicos de otimização.

REFERÊNCIAS

ABDELOUAHAB, Z.; SILVA, Emanoel Costa Claudino. **Management and Integration of Information in Intrusion Detection System: Data Integration System for IDS Based Multi-Agent Systems**. In: Web Intelligence and International Agent Technology Workshops, 2006. WI-IAT 2006 Workshops. 2006 IEEE/WIC/ACM International Conference on, 2006, Hong Kong. Proceedings of the Web Intelligence and International Agent Technology Workshops, 2006. WI-IAT 2006 Workshops. 2006 IEEE/WIC/ACM International Conference on, 2006. p. 49-52.

ALVES, Marcelo de Carvalho; POZZA, Edson Ampélio; MACHADO, José da Cruz. **Sistema Especialista para Identificar Fungos no Teste de Sanidade em Sementes**. Anais do IV Congresso Brasileiro da Sociedade Brasileira de Informática Aplicada à Agropecuária e à Agroindústria. 2006.

ARPÍREZ, JC.; CORCHO, O.; FERNÁNDEZ-LÓPEZ, M.; GÓMEZ-PÉREZ, A. WebODE: **A Scalable Ontological Engineering Workbench**. In: Gil Y, Musen M, Shavlik J (eds) First International Conference on Knowledge Capture (KCAP'01). Victoria, Canada. ACM Press (1-58113-380-4), New York, pp 6-13.

BELLIFEMINE, F.; CAIRE, G.; GREENWOOD, D. JADE - **Developing Multi-Agent System with JADE**. 2007: Wiley

BORST, WN. **Construction of Engineering Ontologies**. University of Tweenty. Enschede, The Netherlands - Centre for Telematica and Information Technology. 1997.

BRESCIANI, P.; GIORGINI, P.; GIUNCHIGLIA, F.; MYLOPOULOS, J.; PERINI, A.: **Tropos: An Agent -Oriented Software Development Methodology**. In Autonomous Agents and Multi –Agent Systems v. 8 (3): 203-236, May 2004.

CAIRE, G.; CABANILLAS, D. **Jade Tutorial. Application-Defined Content Languages and Ontologies**. JADE 3.3. http://jade.tilab.com/doc/CLOntoSupport.pdf, (2004).

CHUNG, L. K.; NIXON, B. A.; YU, E. and MYLOPOULOS, J. **Non-Functional Requirements in Software Engineering**, Kluwer Publishing, 2000.

COBURN, M.: **Jack intelligent agents**: User guide version 2.0. At http://www.agent-software.com, 2001.

COSSENTINO, Massimo; BURRAFATO, Piermarco; **Designing a multi-agent solution for a bookstore with the PASSI methodology** In: Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002), Toronto (Ontario, Canada), May 27-28, 2002.

COSSENTINO, M. e POTTS C. **PASSI: A Process for Specifying and Implementing Multi-Agent Systems Using UML**. Disponível em: <http://www-

static.cc.gatech.edu/classes/AY2002/cs6300_fall/ICSE.pdf>. Acesso em: 18 mai 2006.
DAM, Khanh Hoa; WINIKOFF, Michael. **Comparing Agent-Oriented Methodologies**, In: Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS@AAMAS'03). July 2003.

DAVENPORT, Thomas H. **Ecologia da informação: porque só a tecnologia não basta para o sucesso na era da informação**. 2. ed. Tradução de Bernadete Siqueira Abrão. São Paulo : Futura, 2000.

DELOACH, Scott A. *et al*.. **Multiagent Systems Engineering**. The International Journal of Software Engineering and Knowledge Engineering, Volume 11 no. 3, June 2001.

DILEO, J.; JACOBS, T.; DELOACH, S. **Integrating Ontologies into Multi-Agent Systems Engineering** In: Proceedings of 4th International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS 2002), pp. 15-16. Bologna, Italy. July, 2002.

DOGAC, Asuman. LALECI, Gokce. KABAK, Yildiray. CINGIL, Ibrahim. **Exploiting WebService Semantics: Taxonomies vs. Ontologies**. Software Research and Development Center. Middle East Technical University (METU). Ankara Turkiye, 2002.

FARIA, Carla Gomes de. **Uma Técnica para a Aquisição e Construção de Modelos de Domínio e Modelos de Usuários Baseados em Ontologias para a Engenharia de Domínio Multiagentes**. Dissertação (Mestrado em Engenharia de Eletricidade – Área de Ciência da Computação), Universidade Federal do Maranhão (CPGEE/UFMA). São Luís-MA. 2004.

FERNANDES, Anita Maria da Rocha, e Col. **Inteligência Artificial: Aplicada à saúde**. Florianópolis, 2004 Visual Books. 196p.

FLORES, C. D. **Fundamentos dos Sistemas Especialistas**. In: BARONE, D.A. C. (Ed.). Sociedades Artificiais: a nova fronteira da inteligência nas máquinas. Porto Alegre: Bookman, 2003. p.332

FONSECA NETO, J. V. da; Moura, J.P.de. **Um Sistema Inteligente para Identificação de Falhas e Tomada de Decisão em Correias Transportadoras de Minério**, VI SBAI, 2003.

FRIEDMAN-HILL, E. J. **Jess in Action: Rule-based systems in java**. USA: Manning Press, 2003.

FRIEDMAN-HILL, E. J., **JESS The Rule Engine for the Java Plataform - Language Reference – version 7.0b7** (11 may 2006) DRAFT, Sandia National Laboratories. Livermore, CA, USA.

GIORGINI, P.; KOLP, M.; MYLOPOULOS, J.; CASTRO, J.: **Tropos: A Requirements-Driven Methodology for Agent-Oriented Software**. In B.

Henderson-Sellers and P. Giorgini (Eds) Agent-Oriented Methodologies, Idea Group, (2005).

GIRARDI, Rosario; LINDOSO, Alisson Neres. **An Ontology-based Methodology for Multi-agent Domain Engineering**, In: 3rd Workshop on Multi-Agent Systems: Theory and Applications (MASTA 2005) at 12th Portuguese Conference on Artificial Intelligence (EPIA 2005), Ed. IEEE. Covilhã, Portugal. December 05-08, 2005.

GIRARDI, R.; MARINHO, Leandro Balby. **A domain model of Web recommender systems based on usage mining and collaborative filtering**. Requirements Engineering (London), v. 12, p. 23-40, 2007.

GUIMARÃES, P. H. R. **Automação e Supervisão de Plantas de Minério de Ferro**. Grad. em Engenharia Elétrica, Universidade Federal do Maranhão. 2001.

HAKEEM, Asaad; SHAH, Mubarak. **Ontology and Taxonomy Collaborated Framework for Meeting Classi cation** in 17th conference of the International Conference on Pattern Recognition, ICPR 2004.

JENNINGS, N. R. **On Agent-based Software Engineering**. Artificial Intelligence: 117, p. 277-296. 2000.

KOLP, M.; CASTRO, J.; MYLOPOULOS, J. **A social organization perspective on software architectures**. In Proc. of the 1st Int. Workshop From Software Requirements to Architectures (5–12). STRAW'01, Toronto, Canada, (2001).

LABIDI, Sofiane; BASTOS FILHO, Othon; AXT, Margarete. **SISTEMA INTELIGENTE DE DESAFIOS ABERTOS IOCS: Uma proposta de adaptação dos padrões do Método Clínico Piagetiano em Plataforma Multiagentes**. RENOTE. Revista Novas Tecnologias na Educação, v. 4, p. 10, 2006.

LABIDI, Sofiane; SOUZA, C. M.; NASCIMENTO, E. **NetClass: Cooperative Learner Modeling in a Web-Based Environment** In: 6th Int. Conf. On Computer Based Learning in Science. Proceedings of the 6th Int. Conf. On Computer Based Learning in Science (CBLIS) Nicosia, Cyprus: University of Cyprus, 2003.

LABIDI, S. **CommonKADS Extension for Supporting Multi-Expertise**. In: The 17th International Conference of the British Computer Society on Expert Systems, (ES'97)., 1997, Cambrige. Proceedings of the 17th International Conference of the British Computer Society on Expert Systems, (ES'97)., 1997.

LCP. Laboratório de Controle de Processos. **Sistema Inteligente para Tomada de Decisão em Operação de Descarga de Vagões do Tipo GDT do Terminal Portuário da Ponta da Madeira** - Projeto 1, Etapa 1-3/18: relatório técnico: São Luís; 2006

LCP. Laboratório de Controle de Processos. **Sistema Inteligente para Tomada de Decisão em Operação de Descarga de Vagões do Tipo GDT do Terminal**

Portuário da Ponta da Madeira - Projeto 1, Etapa 5-8/18: relatório técnico. São Luís; 2007.

LASSILA, O.; SWICK, R. **Resouce Description Framework (RDF) Model and Syntax Specification**. In: W3C recommendation, World Wide Web Consortium. 1999.

LINDOSO, Alisson Neres. **Uma Metodologia Baseada em Ontologias Para a Engenharia de Aplicações Multiagentes**. Dissertação de mestrado, Universidade Federal do Maranhão, 2006.

LUGER, George F. **Inteligência Artificial: Estruturas e estratégias para a solução de problemas complexos**. 4.ed. – Porto Alegre: Bookmann, 2004.

MOURA, José Pinheiro de. **Sistema Especialista para Identificação de Falhas e Tomada de Decisão**. Dissertação de mestrado, Universidade Federal do Maranhão. 2003.

MYLOPOULOS, J.; CASTRO, J. **Tropos: A Framework for Requirements-Driven Software Development**. In: Information Systems Engineering: State of the Art and Research Themes, Brinkkemper, J. and Solvberg, A. (eds.), Springer-Verlag, pp. 261-273. Berlin, Alemanha. 2000.

MASSONET, Philippe; DEVILLE, Yves; NÈVE, Cédric. **From AOSE Methodology to Agent Implementation**. Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1, pp. 27-34, 2002.

ODELL, J.; Parunak, H.; Bauer, B.: **Extending UML for Agents**. In: Wagner, G. Lesperance, Y., and Yu, E. (Eds.), Proceedings of the Agent-Oriented Information Systems, Workshop at the 17th National Conference on Artificial Intelligence, (2000).

PERINI, A. and SUSI, A. **Developing Tools for Agent-Oriented Visual Modeling**. In G. Lindemann, J. Denzinger, I. J. Timm, and R. Unland, editors, Multiagent System Technologies, Proc. of the Second German Conference, MATES 2004, volume 3187 of LNAI, pages169–182, Erfurt, Germany, 2004. Springer.

PINTO, A. C. S. **Sistema especialista para diagnose e manejo de problemas fitossanitários e redes neuronais para descrever epidemias da ferrugem do café**. 2001. 91p. Tese (Doutorado em Agronomia / Fitopatologia). Universidade Federal de Lavras, Lavras-MG.

POGGI, Agostino; RIMASSA, Giovanni; TOMAIULOLO, Michele. **Multi-user and security support for multi-agent systems**. Artigo – Dipartimento di Ingegneria dell'Informazione, Università di Parma, Parma, Italy, 2001. Disponível em: <http:sharon.cselt.it/projects/jade/papers/WOA2001.pdf>.

PRAZERES, Hélvio Tadeu Cury. **Como Administrar Pequenas Empresas**. Viçosa-MG, CPT, 2007 354p.

RAO, A. S. **AgentSpeak(L): BDI agents speak out in a logical computable language**, in 'MAAMAW '96: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world: agents breaking away', Springer-Verlag New York, Inc.1996, Secaucus, NJ, USA, pp. 42–55.

RIBEIRO, João P. A.; REATEGUI, Eliseo; BOFF, Elisa. **Integrando um Agente Pedagógico para Recomendações de Tutores a um Sistema de Gerência de Cursos**. Departamento de Informática, Universidade de Caxias do Sul, 2007. Disponível em: www.cinted.ufrgs.br/ciclo9/artigos/7dJoaoPedro.pdf.

RUSSELL, Stuart; NORVIG, Peter. **Inteligência Artificial**. 2. ed. São Paulo: Campus, 2004. 1040 p.

SAMPAIO, Cláudio H.; LABIDI, Sofiane; FARIAS, Osevaldo. **Unsing Agents for Detection of Fraud in Municipal Taxes** in 7th International Conference on Intelligent Systems Design and Applications - ISDA'07 – Universidade Estadual do Rio de Janeiro - UERJ.

SANTOS, André Luís Silva dos. **Um Modelo de Sistema de Filtragem Híbrida para um Ambiente Colaborativo de Ensino Aprendizagem**. Dissertação (Mestrado em Engenharia de Eletricidade – Área de Ciência da Computação), Universidade Federal do Maranhão (CPGEE/UFMA). São Luís-MA. 2007.

SANTOS, André; LABIDI, Sofiane; ABDELOUAHAB, Z. **Information Filtering for a Collaborative Learning Environment: A Multiagent Approach**. In: The Third International Conference on Internet and Web Applications and Services, 2008, Athens. Proceedings of the The Third International Conference on Internet and Web Applications and Services, 2008. p. 43-48.

SCAPIN, Carlos A. **Análise Sistêmica de Falhas** – Nova Lima: INDG Tecnologia e Serviços Ltda, 2007. 166p.: il.

SILVA, Leonardo Ayres M. & FURTADO, João J.V. **Estudo e Desenvolvimento de Sistemas Multiagentes usando JADE: *Java Agent Development framework***. Monografia de Graduação do Curso de Bacharelado em Informática. Fortaleza: Universidade de Fortaleza, 2003.

SILVA, C.; CASTRO, J.; TEDESCO, P. and SILVA, I.: **Describing Agent-Oriented Design Patterns in Tropos**. In Proceedings of the 19th Brazilian Symposium on Software Engineering, Minas Gerais, Brazil (2005).

SILVA, Ismênia G. L da. **Projeto e Implementação de Sistemas Multia-gentes: O Caso Tropos**. Dissertação de mestrado, Universidade Federal de Pernambuco. 2005.

SILVA, Roosewelt L.; LABIDI, Sofiane. **Representation and Content Aggregation in Learning Object Semantic Web-Based Repository**. In: ICBL - International Conference on Interactive Computer aided Blended Learning, 2007, Florianópolis. ICBL - International Conference on Interactive Computer aided Blended Learning Proceedings. RENOTE. Revista Novas Tecnologias na Educação, v. 1, p. 1-2, 2007.

SODRÉ, Alidia Clicia Silva. **Metodologia Baseada em Agentes Para o Desenvolvimento de Software**. Dissertação de mestrado, Universidade Federal do Maranhão, 2001.

SOUZA, Alessandro J. de; FEIJÓ, Rafael H; LEITAO, Gustavo B. P; MEDEIROS, Adelardo A. D; BEZERRA, Clauber G; ANDRADE, Wany L. S. de; GUEDES, Luiz Affonso; MAITELLE, André L. **Gerência de Informação de Processos Industriais: Um Estudo de Caso na Produção de Petróleo e Gás**. VII SBAI, 2005.

STURM, Arnon; SHEHORY, Onn. **A Framework for Evaluating Agent-Oriented Methodologies**, In: Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS@AAMAS'03). July 2003.

SU, K. W.; HWANG, S.L; CHOU, Y.F. **Applying knowledge to the usable fault diagnosis assistance system: A case study of motorcycle maintenance** in Taiwan. Elsevier Science Ltd, 2005.

SURE, Y.; ERDMANN M.; ANGELE, J.; STAAB, S.; STUDER, R.; WENKE, D. **OntoEdit: Collaborative Ontology Engineering for the Semantic Web**. In: Horrocks I, Hendler J (eds) First International Semantic Web Conference (ISWC'02). Sardinia, Italy. Springer Verlag Lecture Notes in Computer Science (LNCS) 2342. Berlin, Germany, pp 221–235.

SUSI, Angelo; PERINNI, Anna; MYLOPOULUS, John; GIORGINI, Paolo. **The Tropos Metamodel and Its Use**. May 2005.

TURBAN, E.; RAINER, R. K.; POTTER, R. E. **Administração de Tecnologia da Informação: Teoria e Prática**. 3ª Edição. Editora Campus. 2005.

TURBAN, Efraim; RAINER, R. Kelly; POTTER, Richard E. **Introdução a sistemas de informação: uma abordagem gerencial**. Rio de Janeiro: Elsevier, 2007.

TVEIT, A.: **A survey of Agent-Oriented Software Engineering**, First NTNU CSGSC, May 2001.

WOOLDRIDGE, M.; JENNINGS, N. R.; KINNY, D. **The Gaia Methodology for Agent-Oriented Analysis and Design**, Journal of Autonomous Agents and Multi-AgenSystems 3 (3):285-312 (2000).

ZHOU, Bing-hai; LI, Chun-chang; ZHAO, Xin. **FIPA agent-based control system design for FMS**. Springer-Verlag London Limited 2006.

WOOLDRIDGE, M.: **Introduction to Multi-Agent Systems**. Jonh Wiley and Sons, New York (2002).

ZAMBONELLI, F.; JENNINGS, N. R. and WOOLDRIDGE, M.: **Developing Multiagent Systems: the Gaia Methodology**. ACM Trans on Software Engineering and Methodology, 12(3): 317-370, 2003.

ALTAVISTA. Disponível em: <http://www.altavista.com>. Acesso em 16 jul. 2007.

BIGDOG. Disponível em: <www.bostondynamics.com/content/BigDog_World_Record_Oct-28-2008.pdf>. Acesso em set. 2008.
FIPA-OS: A component-based toolkit enabling rapid development of FIPA compliant agents, 2005. Disponível em: <http://fipa-os.sourceforge.net/>. Acesso em out 2007.

GOOGLE. Disponível em: <http://www.google.com.br>. Acesso em 16 jul. 2007.

JACK Intelligent Agents. Disponível em: <http://www.agent-software.com/>. Acesso em: 20 dez. 2008.

ORACLE. Disponível em: <http://download-west.oracle.com/docs/cd/B19306_01/server.102/b14237/initparams043.htm>. Acesso em 20 ago. 2006.

OWL. Ontology Web Language. Disponível em: <http://www.w3.org/TR/owl-ref/>. Acesso em: 20 jan. 2008.

PRESSMAN, Roger S. R.S. Pressman & Associates, Inc. Disponível em: <http://www.rspa.com>. Acesso em 07 abr. 2007.

RATIONAL ROSE. Disponível em: <http://www-01.ibm.com/software/awdtools/developer/rose/>. Acesso em jan. 2007.

RDF. Resource Description Framework. Disponível em: <http://www.w3.org/RDF/>. acesso em: 20 jun. 2007.

REVISTA AREDE. Disponível em: <http://www.arede.inf.br/index.php?option=com_content&task=view&id=980&Itemid=99>. Acesso em maio. 2007.

SMITH, Marc T. Potential failure mode and effects analysis. Apresentação, Cayman Systems.1998. 113p. Disponível em: <http://www.fmeainfocentre.com/handbooks/FMEA-N.pdf>. Acesso em: 18 ago. 2007.

SETEZOOM. Disponível em: <www.inbot.com.br/sete/>. Acesso em: set. 2008.

SPEM - Software Process Engineering Metamodel Specification. Disponível em: <http://www.omg.org/technology/documents/formal/spem.htm>. Acesso em: 5 abr. 2007.

THE PROTÉGÉ PROJECT. Disponível em: <http://protege.stanford.edu/>. Acesso em: 16 jul. 2008.