UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

O-bCNMS: UM SISTEMA DE GERÊNCIA DE CONFIGURAÇÃO DE REDES BASEADO EM ONTOLOGIA

Bruno Campelo Lopes dos Santos

Dissertação de Mestrado em Informática

Vitória, Junho de 2007

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

O-bCNMS: UM SISTEMA DE GERÊNCIA DE CONFIGURAÇÃO DE REDES BASEADO EM ONTOLOGIA

POR:

BRUNO CAMPELO LOPES DOS SANTOS

ORIENTADOR:

PROF. DR. ANILTON SALLES GARCIA

DISSERTAÇÃO DE MESTRADO APRESENTADA À UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO COMO PARTE DOS PRÉ-REQUISITOS NECESSÁRIOS À OBTENÇÃO DO TÍTULO DE MESTRE EM INFORMÁTICA.

ÁREA DE CONCENTRAÇÃO: GERÊNCIA DE REDES

VITÓRIA, ES – BRASIL

2007

O-bCNMS: UM SISTEMA DE GERÊNCIA DE CONFIGURAÇÃO DE REDES BASEADO EM ONTOLOGIA

Bruno Campelo Lopes dos Santos

Dissertação Submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como Requisito Parcial para a Obtenção do Grau de Mestre em Informática.

APROVADA EM 06/06/2007 POR:

_____

PROF. DR. ANILTON SALLES GARCIA, UFES

_____

PROF. DR. GIANCARLO GUIZZARDI, UFES

_____

PROF. DR. JORGE MOREIRA DE SOUZA, FITEC

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

VITÓRIA, JUNHO DE 2007

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Central da Universidade Federal do Espírito Santo, ES, Brasil)

Santos, Bruno Campelo Lopes dos, 1979-
S237o O-bCNMS : um sistema de gerência de configuração de redes baseado em ontologia / Bruno Campelo Lopes dos Santos. – 2007.
200 f. : il.

Orientador: Anilton Salles Garcia.
Dissertação (mestrado) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Redes de computação - Gerência. 2. Ontologia. 3. Gerenciamento de configurações de redes. 4. Agentes inteligentes (Software). I. Garcia, Anilton Salles. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 004

À minha esposa Fernanda, pelo amor incondicional e exemplo de pessoa.
Aos meus grandes companheiros inseparáveis e maiores mestres da minha vida: meus pais, Guilherme e Maricy.
Aos meus irmãos, Juliana e Guilherme, pela demonstração contínua de amizade.

Agradeço a Deus por tornar possível tudo isso.

Aos meus amigos por sempre torcerem por mim.

Ao professor Anilton Garcia, pelo acompanhamento e orientação ao longo do trabalho.

Às equipes das empresas em que trabalhei durante o período do Mestrado, por permitir uma flexibilidade que me possibilitasse realizar o curso.

Enfim, agradeço a todos que, de alguma forma, deram apoio ao projeto.

“No final tudo dá certo. Se não deu certo ainda, é porque não se chegou ao final”.
Fernando Pessoa (1888 - 1935)

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

ACC – *Agent Communication Channel*

AchieveRE – *Achieve Rational Effect*

ACL – *Agent Communication Language*

AID – *Agent Identification*

AMS – *Agent Management System*

API – *Application Programming Interface*

AUML – *Agent Unified Modelling Language*

BID – *Belief, Intention, Desire*

BBL – *Behaviour-Based Layer*

CASE – *Computer-Aided Software Engineering*

CCI – *Common Communication Interface*

CFP – *Call for Proposal*

CIM – *Common Information Model*

CMIP – *Common Management Information Protocol*

CMIS – *Common Management Information Service*

CORBA – *Common Object Request Broker Architecture*

CPL – *Cooperative Planning Layer*

DAML+OIL – *DARPA Agent Markup Language + Ontology Inference Layer*

DF – *Directory Facilitator*

DMI – *Desktop Management Interface*

ENF – *Event Notification Faciliy*

FIPA – *Foundation for Intelligent Physical Agents*

FIPA-ACL - *Foundation for Intelligent Physical Agents - Agent Communication Language*

FIPA SC – *Foundation for Intelligent Physical Agents Standards Committee*

FIPA SL – *Foundation for Intelligent Physical Agents Standard Language*

GDMO – *Guidelines for the Definition of Managed Objects*

GUI – *Graphical User Interface*

HTTP – *HyperText Transfer Protocol*

IA – Inteligência Artificial

ITIL – *Information Technology Infrastructure Library*

IDEF5 – *Integrated Definition Ontology Description Capture Method*

IEEE – *Institute of Electrical and Electronics Engineers*

IP – *Internet Protocol*

ISO – *International Organization for Standardization*

IRE – *Identifying Referential Expressions*

JADE – *Java Agent Development Framework*

JRE – *Java Runtime Environment*

JVM – *Java Virtual Machine*

KBSI – *Knowledge Based Systems Incorporation*

KIF – *Knowledge Interchange Format*

KQML – *Knowledge Query and Manipulation Language*

KSE – *Knowledge Sharing Effort*

LGPL – *Lesser GNU Public License*

LPL – *Local Planning Layer*

LPRM – Laboratório de Pesquisa em Redes e Comunicações Multimídia

MAC – *Media Access Control*

MIB – *Management Information Base*

MIF – *Managed Information Format*

MPLS – *Multiprotocol Label Switching*

MTS – *Message Transport System*

MTU – *MaximumTransmission Unit*

NCCF – *Network Communication Control Facility*

NLDM – *Network Logical Data Manager*

NMA – *Network Management Application*

NME – *Network Management Entity*

NPDA – *Network Problem Determination Application*

O-bCNMS – *Ontology-based Configuration Network Management System*

OID – *Object Identifier*

OSI – *Open System Interconnection*

OWL – *Web Ontology Language*

PDU – *Protocol Data Unit*

RDF – *Resource Description Framework*

RFC – *Request for Comments*

RMI – *Remote Method Invocation*

RMON – *Remote Network Monitor*

SLA – *Service Level Agreement*

SMA – Sistema MultiAgente

SMI – *Structured Information Management*

SNMP – *Simple Network Management Protocol*

TI – Tecnologia da Informação

TCP – *Transmission Control Protocol*

TME – *Tivoli Management Environment*

TOVE – *Toronto Virtual Enterprise*

UML – *Unified Modelling Language*

UFES – Universidade Federal do Espírito Santo

URL – *Uniform Resource Locator*

VTAM – *Virtual Telecommunication Access Method*

WAN – *Wide Area Network*

WBEM – *Web Based Enterprise Management*

WWW – *World Wide Web*

RESUMO

Com o crescente aumento do nível de exigência dos serviços prestados por uma empresa a seus clientes, ter um controle sobre sua própria estrutura é fundamental. Dentro desse processo de gerência, especificamente a área de gerência de configuração, surge como um setor de grande relevância para realização desta missão de forma eficiente. Neste trabalho, é proposto o desenvolvimento e implementação de um protótipo de um sistema de gerência de configuração de redes que trabalhe com agentes, através da plataforma JADE, e definir uma proposta de ontologia para integração dos diferentes modelos utilizados para gerência de redes. O objetivo do sistema é possibilitar uma gerência de configuração centralizada e integrada de ambientes heterogêneos de Tecnologia de Informação, através de um processo de *Discovery* de elementos de rede cujo padrão de informações armazenadas terá como base um modelo ontológico definido.

ABSTRACT

With a growing increasing in the level of requirement of services provided by an enterprise to its clients, having a control about your own stucture is fundamental. In this context, configuration management appears as a great relevant area to complete this mission in a efficient way. This project proposes the development and implementation of a prototype for a configuration management system that works with agents, through JADE platform, and defines an ontology proposal to integrate different network management models. This system desires to provide a vision of a centralized configuration management of heterogeneous information tecnology environments. This will be possible with a discovery process of network elements which stored information pattern will be based on a defined ontological model.

INTRODUÇÃO

O aumento do número e da utilização das redes de computadores aliado a uma busca intensa e constante por uma maior qualidade nos serviços prestados, torna o processo de gerência de redes algo cada vez mais necessário e importante no cotidiano das empresas. Dentro desse gerenciamento, existe também uma necessidade de integração dos sistemas de informação e modelos de gerência existentes nas corporações.

Entretanto, a presença de inúmeras tecnologias distintas e novas aplicações, faz com que este processo seja agregado de uma maior complexidade. O aumento dessa complexidade dos ambientes de Tecnologia da Informação traz muitos problemas para os administradores e usuários de redes. Dessa forma, monitorar e controlar os eventos que ocorrem nesses ambientes torna-se uma atividade primordial, independente da dimensão de cada uma dessas estruturas.

O funcionamento adequado dos sistemas de uma empresa é, atualmente, fundamental para seu negócio como um todo. Este funcionamento, no entanto, é dependente da estrutura de rede que esta empresa possui. Se esta estrutura não é apropriada e sofre constantes paradas, a empresa sofre perda de receitas, podendo ocasionar fortes prejuízos para a organização. Sendo assim, um maior controle se tornou necessário e com ele surgiu o conceito de Acordo de Nível de Serviço (SLA – *Service Level Agreement*), estabelecido entre o provedor do serviço e seus clientes. Este acordo trata de uma garantia de qualidade mínima e constante, levando-se em

consideração vários parâmetros de qualidade, que deve obrigatoriamente ser cumprida pelo prestador dos serviços de rede.

Para se atingir um nível de qualidade de serviço estabelecido precisa-se, na maioria das vezes, ter um controle do seu ambiente para se evitar problemas inesperados, como rotas bloqueadas devido a um equipamento parado. Este controle é proporcionado pelo conhecimento da estrutura do seu ambiente de rede. Tal conhecimento está relacionado com a gerência de configuração e o processo de *Discovery*, no qual se identificam os elementos que compõem a rede gerenciada.

O processo de *Discovery*, devido a sua abrangência e diversidade de tecnologias, pode envolver modelos distintos e com uma linguagem bem particular, o que dificulta uma integração das informações extraídas no ambiente analisado.

Por esses motivos apresentados, este projeto propõe desenvolver e implementar um protótipo de um sistema que defina uma proposta de ontologia para possibilitar uma gerência integrada e centralizada das informações relacionadas à gerência de configuração da rede. Esse sistema tem como pré-requisito o uso de agentes desenvolvidos com base na tecnologia JADE, fazendo uso de padrões abertos, utilizando ferramentas isentas de licenças (*softwares* livres). O sistema será denominado **O-bCNMS** (*Ontology-based Configuration Network Management System*).

O projeto está voltado para a gerência de configuração e a proposta de uma ontologia para a integração de modelos, pois a interoperabilidade semântica tem sido um objetivo de pesquisas constantes na área.

1.1 Motivação

A principal motivação do presente trabalho é atuar no contexto de gerência de configuração de redes, a fim de proporcionar um modelo que permita constituir um vocabulário único e compartilhado das informações extraídas do ambiente gerenciado.

1.2 Descrição do Problema

Extrair informações de um ambiente de rede de modo a possibilitar um maior controle do mesmo é um objetivo cada vez mais presente para se administrar e garantir níveis de serviços pré-estabelecidos. Tal controle é diretamente dependente de um modelo eficiente de gerência de configuração de forma a permitir o conhecimento do contexto gerenciado.

Diante deste cenário, obter os dados de cada ativo da rede e construir um modelo capaz de fornecer uma base de informações compartilhada, com um vocabulário único, surge como um problema importante a ser solucionado para que se tenha uma gestão mais eficiente da estrutura monitorada.

1.3 Justificativa

A principal justificativa para se trabalhar no problema supracitado é a tentativa de proporcionar um modelo de gerência de configuração que possa unificar o conteúdo extraído do ambiente gerenciado. Desta forma, a opção por trabalhar com agentes, para se obter os dados dos dispositivos da rede, e com ontologia, para se construir uma especificação formal que possibilite a formação de uma base de informações que seja independente da tecnologia e protocolo utilizados para o processo de gerência.

1.4 Objetivos Gerais e Específicos

O objetivo geral do trabalho é dar continuidade aos trabalhos realizados pelo grupo de gerência de redes do LPRM (Laboratório de Pesquisas em Redes e Multimídia) da Universidade Federal do Espírito Santo, bem como permitir trabalhos futuros voltados para o uso de Simulação de Eventos Discretos para a tomada de decisão na Plataforma Inteligente para gerência de redes.

O objetivo específico deste trabalho é desenvolver uma proposta de uma Ontologia que contemple o armazenamento de informações extraídas de diferentes

tecnologias em um processo de *Discovery*. Baseado nessa proposta, utilizar agentes que façam a busca e a publicação das informações e as converta para o padrão estabelecido pela Ontologia. Por meio dessa conversão, ter-se-á o modelo integrado de informações sobre a configuração da rede.

1.5 Resultados Esperados

O presente trabalho apresenta os seguintes resultados:

- Uma proposta de um modelo de ontologia, construído em uma ferramenta de distribuição livre, para atender ao processo de *discovery* e voltado para gerência de configuração.
- Uma primeira versão de um sistema de gerência de configuração centralizado, baseado em agentes, que faça uso da plataforma JADE. Esse protótipo foi desenvolvido com base em padrões abertos, plataformas e *softwares* livres.
- Um projeto de graduação de um aluno do curso de Engenharia de Computação ou Ciência da Computação, voltado para a gerência de configuração de redes.

1.6 Principais Contribuições

Dentre os objetivos traçados, espera-se do presente trabalho as seguintes contribuições:

- Desenvolvimento de uma linha de pesquisa voltada para o uso de ontologias no contexto da gerência de configuração de redes;
- Incentivo ao uso de agentes, baseados na plataforma aberta JADE, para se obter uma forma de comunicação e mobilidade nos ambientes gerenciados;

- Estímulo ao estudo e utilização de protocolos de gerência, como o SNMP, para extração de informações relacionadas à estrutura a ser monitorada.

1.7 Organização do Trabalho

A estrutura desta dissertação caracteriza-se por capítulos referentes à teoria abordada no trabalho e por outros relacionados ao modelo de Ontologia proposto e ao projeto do *software* desenvolvido como estudo de caso.

O capítulo 2 apresenta uma revisão bibliográfica, com os principais trabalhos relacionados aos assuntos discutidos.

No capítulo 3, sobre gerência de redes, são apresentados os conceitos que se referem à área com o foco em gerência de configuração, bem como modelos de arquitetura gerente-agente e uma breve apresentação de algumas ferramentas existentes que atuam na referida área.

No capítulo 4, referente a agentes, é apresentado o conceito de agente, sua tipologia e as características necessárias para se construir sistemas multiagentes, com destaque para formas e linguagens de comunicação existentes entre eles.

O capítulo 5 preocupa-se em discutir o *framework* JADE para a construção de agentes. Este segue o padrão FIPA (*Foundation for Intelligent Physical Agents*) focado em interoperabilidade. Tal padrão foi utilizado no presente trabalho para desenvolvimento de agentes.

Os capítulos 6 e 7, são referentes à modelagem da ontologia e do protótipo implementado para o sistema **O-bCNMS**. O capítulo 6 apresenta conceitos envolvidos no contexto de ontologias e o modelo ontológico proposto, detalhando os conceitos envolvidos e os relacionamentos entre eles. O capítulo 7 apresenta o processo de engenharia de software voltado para o desenvolvimento dos agentes modelados para o protótipo: a modelagem arquitetural dos agentes, o projeto detalhado, os padrões sociais utilizados, a arquitetura do sistema, a implementação

em JADE e o modelo de interface utilizado para o protótipo definido para esta dissertação. Por fim, têm-se as considerações finais sobre o presente trabalho.

Para o leitor interessado na teoria abrangida por este trabalho, recomenda-se os capítulo 3, 4 e 5. Para o leitor com conhecimento em ontologias, recomenda-se o capítulo 6, no qual ele pode encontrar a descrição do modelo ontológico proposto voltado para gerência de configuração de redes. Para aquele leitor com conhecimento e interesse em agentes, indica-se o capítulo 7. Neste capítulo, ele poderá conhecer a modelagem feita para os agentes utilizados no sistema **O-bCNMS**, bem como verificar o protótipo desenvolvido.

Este trabalho relaciona-se com as disciplinas ministradas na área de redes de computadores como introdução a redes de computadores e gerência de Redes, ambas opcionais para o curso de pós-graduação Mestrado em Informática. Além da referida área, o trabalho possui fortes ligações com as disciplinas banco de dados e engenharia de software e engenharia de ontologias. Tais disciplinas também são opcionais para o curso de pós-graduação Mestrado em Informática.

Com relação ao Mestrado em Informática, tal projeto proporciona ao mestrando desenvolver conceitos importantes na área de gerência de redes. Estes conceitos são voltados para a definição de um modelo de informações que possa ser compartilhado dentro de um contexto definido. Desta maneira, possibilitou um crescimento profissional de poder aplicar diretamente a teoria, alvo do trabalho e discutida ao longo das disciplinas afins, além de uma experiência de trabalhos acadêmicos, fundamental para quem deseja dar continuidade a outros projetos na área, como um doutorado por exemplo.

TRABALHOS RELACIONADOS

Este projeto tem como objetivo dar continuidade aos trabalhos realizados pelo grupo de gerência de redes do LPRM (Laboratório de Pesquisas em Redes e Multimídia) da Universidade Federal do Espírito Santo, bem como permitir trabalhos futuros voltados para o uso de ontologias que permita a tomada de decisão na Plataforma Inteligente para Gerência de Redes (*NM Intelligent Plataform*). Dentre esses trabalhos destacam-se [45]; [74]; [12]; [59].

A Figura 1 apresenta um esboço da arquitetura correspondente à plataforma na qual o agente atuará de forma a realizar um gerenciamento centralizado através da ontologia proposta. Esta arquitetura baseia-se em redes distintas compostas por diversos ativos, como servidores, impressoras, roteadores entre outros, que podem ser gerenciados por protocolos de gerência diferentes, como o SNMP. Neste cenário, a definição de um modelo ontológico tem um papel essencial na extração das informações acerca das redes e na construção de uma base de dados capaz de estabelecer um padrão para a gerência dos ambientes em questão.

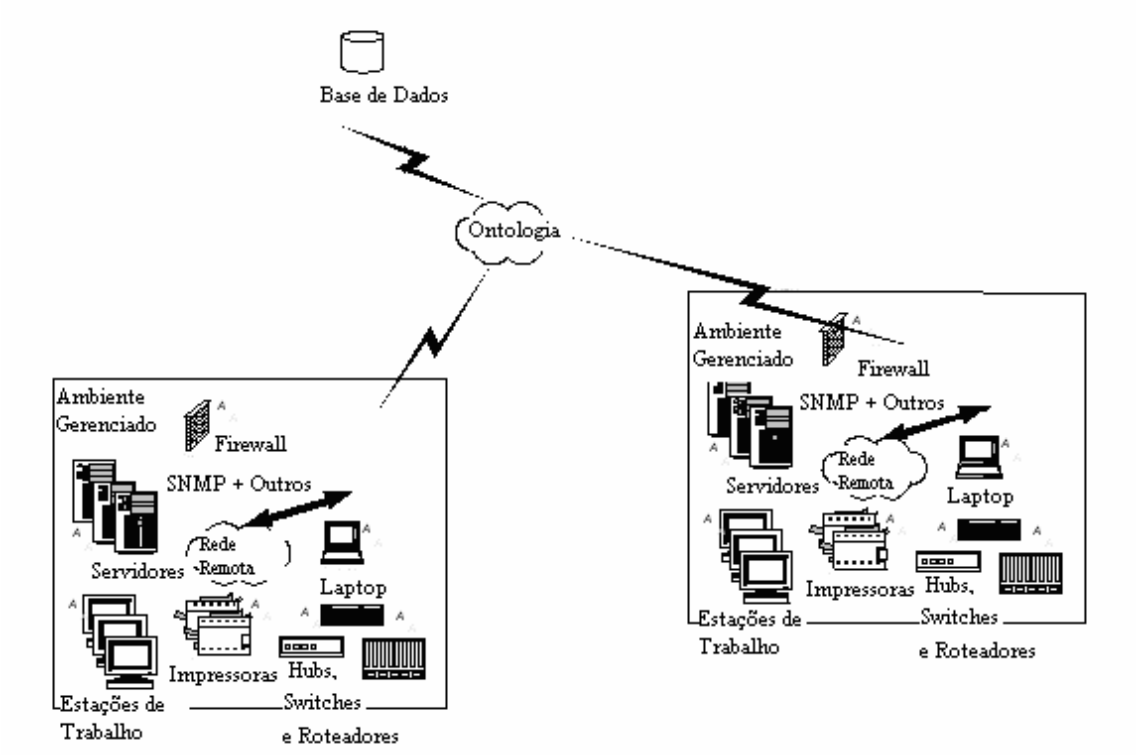


Figura 1 - Arquitetura de Gerência de Configuração baseada em Ontologia

Esta dissertação aborda assuntos de três áreas distintas, porém com uma forte integração entre cada uma delas. Trata-se da gerência de redes, agentes e ontologias.

A gerência de redes tem proporcionado muitos trabalhos por parte de pesquisadores e profissionais da área. Alguns desses trabalhos constituem uma base para este projeto. Em [8], [35] e [42] são discutidos os conceitos, padrões e arquiteturas que envolvem a gerência de redes, foco do capítulo 3 deste trabalho. Em [43], são discutidas algumas das melhores práticas para gerência de redes, alvo deste trabalho no que tange à gerência de configuração, apresentada no capítulo 3. Em [62] e [17], é apresentado o padrão estabelecido pelo protocolo SNMP (Simple Network Management Protocol) nas suas três versões, utilizado no desenvolvimento do protótipo do sistema **O-bCNMS**. Em [12] é apresentado um estudo das ferramentas de gerência de redes utilizadas no mercado e em [59] é proposto um protótipo de ferramenta capaz de realizar um gerenciamento centralizado da configuração de uma rede baseado no protocolo SNMP. Tal estudo foi utilizado como base para o desenvolvimento de operações com este protocolo no protótipo desenvolvido no estudo de caso.

O padrão de desenvolvimento voltado para a tecnologia de agentes também tem se destacado como uma área de pesquisa bastante explorada dentro do contexto de gerência. Os principais conceitos envolvidos com o uso de agentes e a classificação dos mesmos são apresentados, por exemplo, em [5], [9], [29], [49]. Em [21], o foco é o desenvolvimento de agentes autônomos modelados a partir da metodologia *Tropos* [77], assunto este reforçado por [22], com o uso de técnicas de Inteligência Artificial. Em [57], apresenta-se categorias de agentes voltados para gerência de redes. Tais trabalhos foram base para o capítulo 4. Outros assuntos abordados neste capítulo abrangem agentes em ambientes heterogêneos [34], sistemas multiagentes [1] [10] [61] [76] e o processo de comunicação entre agentes [46], principalmente no que diz respeito a linguagens de comunicação.

Uma tecnologia utilizada para o desenvolvimento de agentes de acordo com o padrão FIPA (*Foundation for Intelligent Agents*) [20], JADE (*Java Agent Development Framework*), tem sido um caminho adotado para construção de agentes. Em [61] e [47], são trabalhos com aplicações práticas no desenvolvimento de agentes através da plataforma JADE, que é a preocupação do capítulo 5 desta dissertação. Outros trabalhos que constituíram a base para este capítulo são [3] e [4], nos quais o *framework* JADE é apresentado e discutido do ponto de vista do desenvolvedor.

No processo de modelagem e definição dos agentes a serem construídos para o protótipo do sistema **O-bCNMS**, o trabalho [51] discute algumas questões importantes no processo de abstração de modelos de agente para gerência de redes. Questões relacionadas à *design patterns* de agentes [41] e formas de representação dos mesmos através do padrão UML (*Unified Modelling Language*) [2], foram utilizados como base teórica para confecção do capítulo 7 deste trabalho.

No que diz respeito à ontologia, uma vez que um dos principais objetivos desta dissertação é a proposição de um modelo ontológico voltado para gerência de configuração, têm-se muitos trabalhos de destaque. Em [25], [30], [31] e [32], são discutidas questões conceituais referentes a ontologias e em [48] e [75], o foco está relacionado ao processo de construção de uma ontologia, abordadas no capítulo 6. Nos trabalhos [70], [71], [72] e [73] são apresentadas questões referentes a modelos de ontologia voltados para a gerência de redes e web semântica. Tais pontos também são assunto do capítulo 6 que abrange uma discussão sobre análise do

domínio de conhecimento e construção de ontologias, temas principais dos trabalhos [14], [15], [16], [23], [44], [52], [60] e, em especial, de [13] e [33]. Em [37], o assunto é a proposta de um *framework* baseado em cenários para a implementação de uma ontologia. Assuntos relativos à metodologias para desenvolvimento de ontologias são apresentadas por [39] e de forma mais específica por: [26], [27], [28] e [69], no qual o assunto é a metodologia *TOVE* (*Toronto Virtual Enterprise*); [66], [67] e [68], referente à *Enterprise Model Approach*; [18] e [24] relacionadas à *Methontology* e [40] correspondente à *KBSI IDEF5*.

Dessa forma, esta dissertação tem como propósito principal apresentar um caminho a ser seguido no desenvolvimento de uma estratégia que permita extrair informações de gerência de configuração baseado em uma ontologia, utilizando para isso, agentes desenvolvidos por meio do *framework* JADE.

GERÊNCIA DE REDES

As redes de computadores foram projetadas, inicialmente, como um mecanismo para permitir o compartilhamento de recursos como impressoras, modems de alta velocidade, existindo apenas em ambientes acadêmicos, governamentais, organizações militares, e em empresas de grande porte. Entretanto, a evolução das tecnologias de redes, aliada à uma significativa redução de custos dos recursos computacionais, motivou a proliferação das redes de computadores por todos os segmentos da sociedade. Houve também uma profunda mudança nos serviços oferecidos, pois além do compartilhamento de recursos, novos serviços, tais como correio eletrônico, transferência de arquivos, WWW, aplicações multimídia, foram acrescentados, aumentando a complexidade das redes. Não bastassem esses fatos, o mundo da interconexão de sistemas computacionais convive com a grande diversidade dos padrões de redes, sistemas operacionais, equipamentos, protocolos, etc.

Nesse contexto, em que cada vez é mais rápido o crescimento da complexidade e heterogeneidade das redes de computadores, surge a necessidade de buscar uma maneira consistente de realizar o gerenciamento de redes para, com isso, manter toda a estrutura da rede funcionando e atendendo às necessidades de seus usuários e às expectativas de seus administradores.

Segundo [8], o gerenciamento de redes pode ser entendido como o processo de controlar uma rede de computadores de forma que ela esteja sempre funcionando de maneira eficiente, isto é, “*Keeping the network up and running*”.

Para [62], a gerência estaria associada ao controle de atividades e monitoração do uso de recursos da rede. As tarefas básicas da gerência de redes seriam extrair informações da rede, tratá-las possibilitando um diagnóstico e encaminhar as soluções dos problemas. Para tanto, funções de gerência devem ser embutidas nos diversos elementos de uma rede, possibilitando descobrir, prever e reagir a problemas.

3.1 Áreas funcionais da Gerência de redes

De acordo com a ISO, citado por [62], um ambiente OSI possui cinco áreas funcionais: gerência de falhas, contabilidade, configuração, desempenho e segurança.

A gerência de falhas busca identificar onde uma possível falha ocorreu, isolar o restante da rede para mantê-la em funcionamento, reconfigurá-la ou modificá-la visando minimizar o efeito da falha e, por fim, reparar ou substituir o componente com problemas de modo a restaurar o estado inicial da rede.

A gerência de contabilidade atua na definição de limites de utilização dos objetos gerenciados e na identificação dos custos associados ao uso desses objetos.

A gerência de configuração dedica-se às funções de controlar, identificar e prover dados sobre objetos gerenciados para operações de interconexão de serviços na rede. Preocupa-se também com a manutenção e atualização dos relacionamentos entre os elementos da rede, ou seja, relações de interconexão e operação entre esses elementos.

A gerência de desempenho avalia o comportamento dos objetos gerenciados de uma rede e o resultado de suas atividades. Abrange duas categorias de funções: monitoração e controle. A monitoração permite coletar dados de um conjunto de recursos da rede. O controle acontece através da interpretação desses

dados, de tal forma que proporcione à gerência opções para melhorar o desempenho da rede.

A gerência de segurança atua na proteção e no controle de acesso a uma parte ou mesmo à toda a rede. Visa garantir, também, a integridade, autenticidade e disponibilidade sobre recursos da rede, bem como sobre informações que trafegam na mesma.

3.2 A Gerência de Configuração

Segundo [35], o termo configuração pode ter diferentes significados dentro da gerência de redes. O que vai determinar sua função será o contexto que estará sendo analisado.

Uma funcionalidade, possivelmente útil, é a de fornecer uma descrição de um sistema distribuído baseado na organização física e geográfica dos recursos da rede. Essa descrição inclui como esses recursos estão, no momento, interconectados, além de informações sobre suas relações lógicas [35].

Outra atividade dessa área da gerência é a de atuar na manipulação da estrutura da rede, definindo e alterando parâmetros que controlem operações normais no sistema. Como exemplo, o estado desejado de um dispositivo ou um relacionamento entre recursos de um ambiente de rede [35].

De acordo com [35], atuar na parte de gerência de configuração significa preocupar-se com atividades como criação de bases de dados contendo a documentação do sistema gerenciado, além de prover uma forma de se dispor de um mapa geral da rede. Outras preocupações estão no fato de se desenvolver ferramentas para modificar e invocar parâmetros de configuração e controlar a autorização sobre tais informações. Para tantas atividades, necessita-se de elementos que possam extrair estes dados e que possuam um vocabulário único de comunicação.

3.2.1 Algumas ferramentas existentes

No mercado, existem muitas ferramentas que desempenham atividades relacionadas à gerência de configuração.

Uma delas é o ***HPOpenView***[1] , da empresa ***Hewllet-Packard***. Trata-se de um *software* que atua também nas áreas de gerência de desempenho e de falhas. Provê um sistema de correlação de eventos, bem como um serviço de nomes, relacionamentos e registros de objetos. Oferece a facilidade de integrar outras API's. Suporta aplicações distribuídas, além de especificações de protocolos definidos pelo **ITU**, **OSI**, **X/OPEN**, entre outros. Trata-se de uma ferramenta suportada por estações **SUN**, **HP**, ***Microsoft WINDOWS***. Dispõe de suporte para os protocolos de gerência **SNMP** e **CMIP**. Possui sistema de multi-conexões, controlado pelo ***PostMaster*** e também um mecanismo de segurança, o ***MC Service Guard***. Os principais módulos existentes na solução **HPOpenView** são:

Network Configuration Manager **(NCM)** – responsável por alterar e realizar o controle da configuração da rede, através de características que permitem garantir segurança, disponibilidade e eficiência operacional na rede. Suporta as melhores práticas do ITIL (*Information Technology Infrastructure Library*).

Server Configuration Management **(SCM)** – tem como preocupação gerenciar o ciclo de vida do servidor e manter sua configuração em conformidade com o suporte a ambientes de TI heterogêneos, buscando melhoria de qualidade nos serviços com menores custos.

Asset Management **(AM)** – tem como foco otimizar recursos e rastrear e controlar o uso dos mesmos através de inventário automatizado.

Business Service Management **(BSM)** – possui como meta prover uma solução que conecte TI ao negócio da organização. O BSM oferece meios para se medir como ações de TI podem melhorar processos de negócio, gerando retorno direto para a corporação.

---

1 http://www.openview.hp.com/

***Identity Management* (IM)** – seu foco é o atendimento de chamadas e aprovisionamento de serviços, com o objetivo de reduzir carga de trabalho do setor de *help desk*.

***Network Services Management* (NSM)** – responsável por monitorar, reportar e automatizar requisitos que sejam necessários para gerenciar infra-estruturas heterogêneas e complexas, com serviços como MPLS e Telefonia IP.

Outro *software* de destaque é **Tivoli IBM**[2]. O **Tivoli** trabalha com o conceito de regiões de policiamento ou **TMR**. Uma região de policiamento abrange um conjunto de recursos gerenciados, como contas de usuários, estações de trabalho, roteadores. Em uma rede podem ser definidas mais de uma região de policiamento. Cada região possui suas próprias políticas de controle de acesso e gerenciamento.

Em uma região de policiamento existem alguns elementos que ajudam no gerenciamento, como principalmente:

TMR Server - responsável por controlar toda região de policiamento e define as políticas utilizadas nessa região;

***Managed Node* -** trabalha como um intermediário entre as funções do *Endpoint* e do *Endpoint Gateway*. Executa algumas tarefas a pedido do *TMR Server*;

***Endpoint Gateway* -** controla as comunicações e operações entre os *Endpoints.* Responsável por enviar aos *Endpoints* métodos que os permitam realizar as funções de gerenciamento;

***Endpoint* -** agente que executa as operações de gerenciamento nos recursos finais.

O ambiente de gerenciamento do **Tivoli** (**TME**) é dividido em vários módulos, cujos principais são:

***Framework* -** Módulo principal do Tivoli. Tem funções básicas de administração e serviços de gerenciamento. Facilita no restante das funções de gerenciamento, como no gerenciamento de sistemas e em notificações, e na integração com os outros módulos;

---

[2] http://www-306.ibm.com/software/tivoli/

Outros módulos importantes são: ***User Administration***, que permite o gerenciamento de grupos e contas de usuários, o ***Distributed Monitoring***, responsável por monitorar o status de recursos da rede; o ***Inventory***, que se preocupa em controlar a configuração da rede sobre *hardware* e *software;* o ***Software Distribution***, que permite a distribuição e instalação de *software* em máquinas de uma rede heterogênea; o ***Enterprise Console***, que gerencia eventos baseada em regras.

Além desses módulos de gerenciamento o **Tivoli** utiliza-se do ***Netview***, ferramenta também desenvolvida pela ***IBM***, para o diagnóstico e controle de redes de comunicações. Seu objetivo era substituir uma gama variada de produtos similares já existentes. O mais conhecido desses produtos é o ***Network Communication Control Facility*** (**NCCF**) que funciona como uma aplicação ***Virtual Telecommunication Access Method*** (**VTAM**), fornecendo uma visão do estado da rede através de um monitor de controle e uma base de dados. Outros produtos de controle são o ***Network Problem Determination Application*** (**NPDA**), que opera na detecção de problemas e o ***Network Logical Data Manager*** (**NLDM**), que gerencia dados lógicos.

Há também o ***Net View***[3], desenvolvido pela **Cisco**. Esse *software* permite a gerência da rede através de nós. Possibilita a criação e manipulação de novos nós no sistema por parte do gerente, bem como de subredes. Ao ser iniciado exibe a topologia da rede em questão. Suporta o protocolo SNMP. Fornece também o mapa do ambiente gerenciado por subrede. Preocupa-se mais com informações relativas ao nó, sem gerar estatísticas de desempenho.

Outro *software* importante é o ***What's Up***[4], da empresa ***IPSwitch***. Algumas de suas características são:

SmartScan que usa o SNMP para detectar dispositivos e criar mapas que refletem exatamente a hierarquia da sua rede, com mapas separados para vários níveis de subredes.

---

3 http://www.cisco.com/univercd/cc/td/doc/product/dsl_prod/6700/ems14/netview.htm

4 http://www.sbg.pt/ipswitch_wug.htm

O ***SmartScan*** procura nas tabelas do roteador, descobre automaticamente dispositivos e adiciona-os ao mapa, de acordo com a sua hierarquia de rede, com mapas separados para cada subrede.

Há também o ***Scan IP*** que permite analisar uma vasta quantidade de endereços IP e realizar um mapa. Esta funcionalidade é especificamente útil para monitorar apenas uma porção de uma rede ou de uma subrede.

Fornece gráficos de desempenho com relatórios que proporcionam representações visuais de estatísticas acumuladas para mapas selecionados e de dispositivos, sem ter que exportar dados para um outro programa.

O indicador gráfico de mapa *Web Interface* permite visualizar a rede através da *web* para ambientes remotos de administração. Possui identificação automática de dispositivos nas rotinas ***scan*****,** que podem identificar dispositivos de gerência de SNMP e configurar aparelhos.

O menu possibilita o acesso a um conjunto de ferramentas de rede selecionadas como ***TraceRoute***, ***Ping***, ***Lookup***, ***Finger***, ***Whois***, ***LDAP*** e ***WinNet***, que podem ser utilizadas para diagnosticar possíveis problemas.

Por fim, um outro *software* que merece destaque é o ***Unicenter TNG***[5], desenvolvido pela ***Computer Associates***.

O ***Unicenter TNG***[6] é definido de forma a ser o centro do gerenciamento corporativo. Nele são concentradas todas as disciplinas, permitindo uma visão centralizada das diversas plataformas a serem gerenciadas e uma integração entre as diversas disciplinas de gerência. É disponibilizado com a infra-estrutura necessária para o gerenciamento da maioria das plataformas existentes no mercado. Entre elas, a plataforma *Windows NT*, *UNIX*, *Netware*, *AS/400*, *Tamdem NSK* e ainda os ambientes de *Mainframe*. O ***Unicenter TNG*** inclui funcionalidades básicas dentre as quais se destacam:

Enterprise Discovery - para a realização da descoberta ou *"discovery"* de todos os objetos de rede e recursos existentes na rede corporativa, incluindo sistemas, estações, aplicações e bancos de dados;

---

5 http://h21007.www2.hp.com/dspp/mop/mop_partner_product_detail_IDX/1,1331,1409,00.html

6 http://www.pr.gov.br/celepar/celepar/batebyte/edicoes/2000/bb102/estudo.htm

Manager/Agent Event Management - para monitoração completa de status e logs de eventos, habilitando a resposta automatizada para a mudança do status de objetos;

Calendar Management - para a definição de data e sincronização baseadas em tempo, para fornecer uma coordenação das mudanças que afetam os processos de negócio;

Common Object Repository - o repositório de objetos suporta todas as funções de gerenciamento, sendo capaz de armazenar informações sobre os objetos gerenciados, suas propriedades e relacionamentos. As aplicações, "*hardware*", "*software*", bancos de dados e entidades como contas a pagar, contas a receber e recursos humanos, entre outros, são exemplos destes objetos gerenciados;

Event Notification Facility (**ENF**) e a ***Common Communication Interface*** (**CCI**) - reforçam a confiabilidade e escalabilidade do *framework*, fornecendo uma infra-estrutura de mensagens;

WEBEM Support - para habilitar os usuários corporativos a relacionarem interfaces de gerência *web* a outros dados de gerência de diversas fontes, criando uma visão centralizada dos recursos dos ambientes de tecnologia da empresa;

WakePC Technology - este sistema permite que técnicos executem tarefas de manutenção remotamente;

DMI Support - permite que administradores possam identificar, visualizar e gerenciar sistemas, incluindo componentes de *hardware* e *software* e dados estáticos e dinâmicos;

Real Word Interface - este serviço permite visualizar a topologia da rede e todos os recursos de computação na rede corporativa, através de uma interface real 2-D, 3-D e visualização via *web*.

Para apoiar a escalabilidade e flexibilidade requerida pelo gerenciamento em todos os níveis, o ***Unicenter TNG*** é baseado em uma arquitetura para gerenciamento em múltiplos níveis gerente/agente. Os agentes do ***Unicenter TNG*** são ativos, capazes de implementar políticas e coordenar, juntamente com outros

agentes ativos, as funções de monitoração de eventos e estados, gerenciamento da configuração de ambientes distribuídos, entre outras.

A Tabela 1 apresenta um resumo das principais características das ferramentas descritas nesta seção:

Características	**HPOpenView**	**Cisco NetView**	**IBM Tivoli**	**What´s Up (IPSwitch)**	**Unicenter TNG**
Áreas da Gerência de Redes	Desempenho e Falhas	Configuração	Configuração e Desempenho	Configuração	Configuração e Desempenho
Suporte ao protocolo SNMP	Possui	Possui	Possui	Possui	Possui
Sistema de Correlação de Eventos	Possui	Não Possui	Possui	Não Possui	Possui
Gerência de Problemas	Possui	Não Possui	Possui	Não Possui	Possui
Integração com o Negócio	Possui	Não Possui	Possui	Não Possui	Possui
Gerência Multi-Nível Agente/Gerente	Não Possui	Não Possui	Não Possui	Não Possui	Possui
Gerenciamento baseado no conceito de	Recursos	Nós	Região de Policiamento	Dispositivos	Plataformas

Tabela 1 – Características das principais ferramentas de gerência

3.3 Sistemas de gerenciamento de redes

Um sistema de gerenciamento de redes é uma coleção de ferramentas de monitoração e controle. Tal coleção é integrada, no sentido de possuir uma única interface de operação que seja amigável, para realizar a maioria das tarefas de gerenciamento de rede. Também deve oferecer uma quantidade mínima de equipamentos separados. Isto é, a maioria dos elementos de *hardware* e *software* requeridos para o gerenciamento de rede está incorporado ao equipamento do usuário [62].

De forma simplificada, pode-se dizer que um sistema de gerenciamento de redes contém dois elementos: um gerente e vários agentes.

De acordo com [43], o papel do gerente torna-se fundamental no controle e solução dos problemas na rede e, muitas vezes, é executado por uma equipe capaz de atuar em quatro tarefas distintas: *help desk,* como captador e identificador

de problemas; suporte técnico, responsável por agir operacionalmente na solução dos problemas; operação, tendo a função de acompanhar eventuais alarmes gerados pela estação de gerência; gerente propriamente dito, encarregado de avaliar a equipe, providenciar treinamentos necessários, solicitar aquisição de equipamentos, redefinir as prioridades de atendimento entre outras atividades.

3.3.1 Arquitetura de gerenciamento

Uma das mais utilizadas arquiteturas de gerenciamento baseia-se na teoria de orientação a objetos. O sistema representa os recursos gerenciados através de entidades lógicas, as quais recebem a denominação de objetos gerenciados.

Nessa arquitetura, cada nó da rede contém *softwares* dedicados a tarefas de gerenciamento. Eles caracterizam uma entidade gerenciável, ou de acordo com a Figura 2, um NME (*Network Management Entity*). Cada NME constitui um agente, que se caracteriza por ser um *software* com funções de coletar e armazenar dados desses objetos gerenciados e responder aos comandos de um controle central da gerência de rede. Esses comandos são, sobretudo, para solicitar informações e dados sobre os objetos ou para mudar algum parâmetro dos mesmos. Cabe também ao agente, enviar mensagens ao controle central quando algo diferente em relação ao funcionamento normal ocorre com esse objeto [62]. Este acontecimento pode ser uma dificuldade de processamento do objeto em questão, causada, por exemplo, por excesso de solicitações em um mesmo momento.

No mínimo, uma estação na rede é designada para ser a central de controle. Além de possuir um *software* agente, essa estação central inclui um programa com uma interface de operação que permite ao usuário autorizado gerenciar a rede através das informações extraídas dos agentes. Tal programa, denotado na Figura 2 como NMA (*Network Management Application*), caracteriza o gerente. A comunicação agente-gerente é feita por meio de um protocolo de camada de aplicação [62].

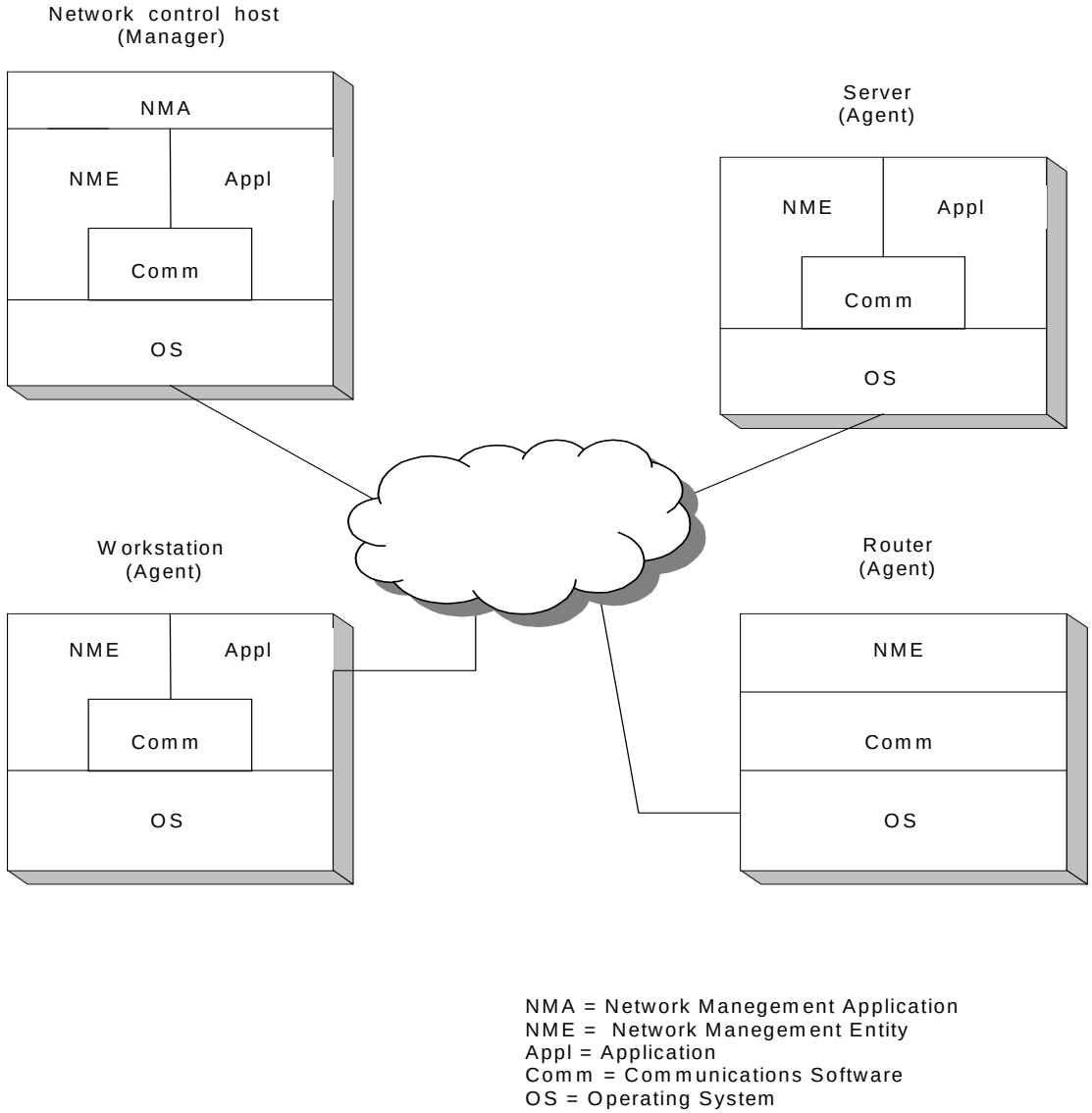


Figura 2 - Elementos de um sistema gerenciável [62]

3.3.2 Configurações de gerenciamento

Segundo [62], há basicamente dois modos de se fazer um gerenciamento de uma rede. O primeiro deles seria um gerenciamento centralizado (Figura 2), onde toda a gerência da rede concentra-se em um único ponto desta. Nesse caso, tem-se toda a informação em um único ponto, o que facilita o acesso a todo e a qualquer dado. No entanto, uma possível falha da estação central, deixaria a atividade de gerenciamento comprometida. É uma configuração usada em sistemas mainframe ou nos quais existe apenas um gerente responsável por toda a estrutura.

Outra maneira seria o gerenciamento distribuído (Figura 3). Nessa configuração, a gerência é feita em vários pontos da rede, o que minimiza o *overhead*[7] de tráfego nesta e aumenta a escalabilidade do modelo. A razão desse aumento explica-se pelo fato de que para adicionar recursos gerenciáveis à rede, um dos principais problemas passa a ser a escolha do local desejado para instalá-los. Isto porque, com uma gestão distribuída, a adição de novos recursos será administrada por entidades distintas, de acordo com o local em que o mesmo for inserido. Outra vantagem seria a de eliminação de uma total paralisação da atividade de gerenciamento, no caso de uma falha de uma estação gerente. Uma desvantagem seria o limitado acesso a informações da rede, uma vez que este seria limitado por cada gerente. É indicada para empresas que possuem várias subredes distribuídas divididas por departamentos.

[7] Carga de controle que trafega na rede ocupando o barramento da mesma.

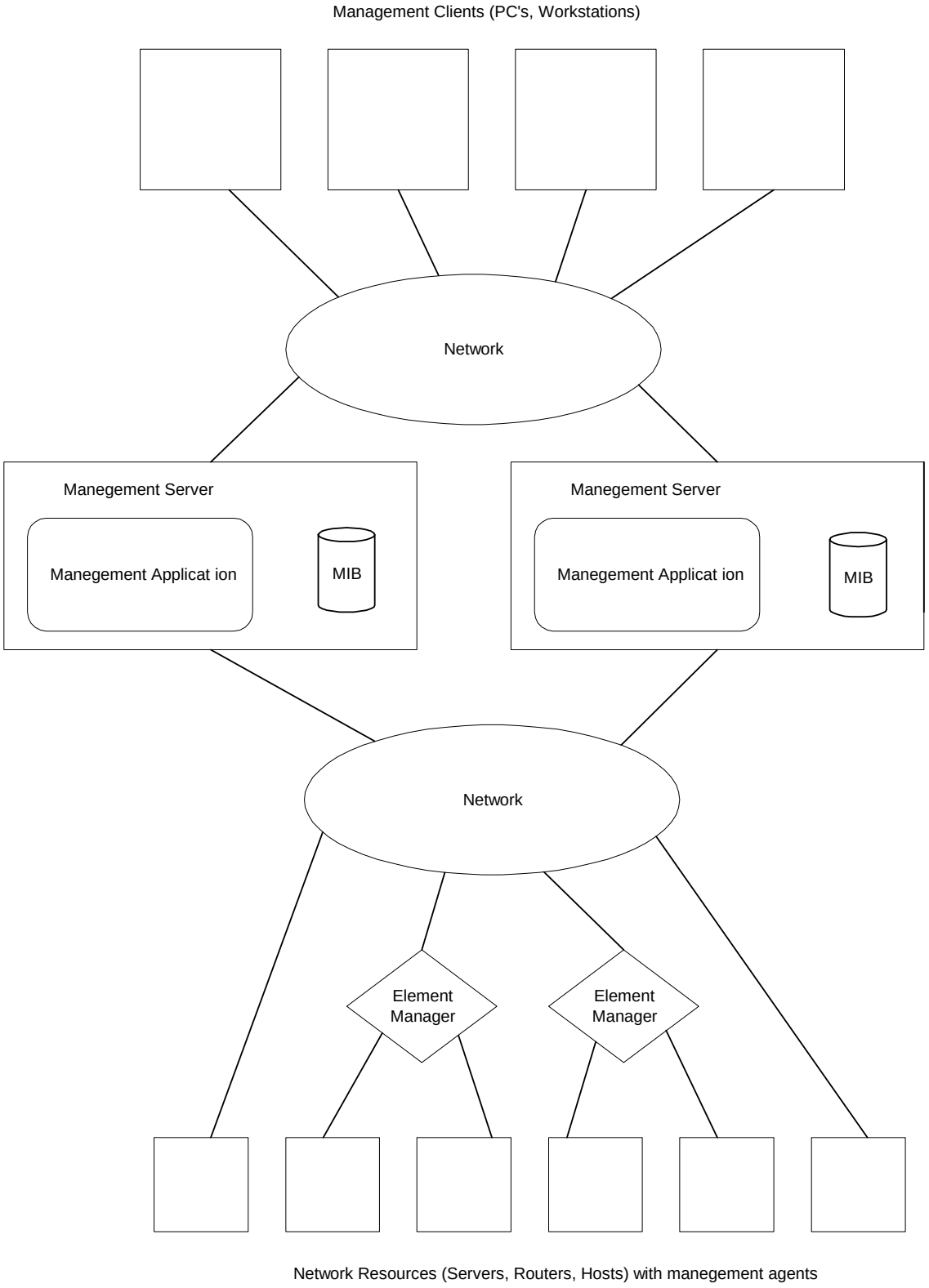


Figura 3 - Arquitetura de sistema de gerenciamento distribuído [62]

3.3.3 Proxies

Em muitos casos, uma rede possui elementos que não suportam um mesmo padrão de gerenciamento. Um exemplo seria um sistema com componentes antigos, construídos com arquiteturas e tecnologias da época em que foram concebidos, e outros mais atuais, desenvolvidos nos padrões tecnológicos mais explorados no momento. Neste cenário, o processo de comunicação entre tais componentes de natureza distinta torna-se um problema. Uma solução para tal fato é fazer um dos agentes atuar como *proxy* de um ou mais nós da rede [62].

Dessa forma, a aplicação gerente, ao solicitar uma informação de um determinado nó, se comunicaria com o agente *proxy*. Este, então, traduz a solicitação do gerente para o protocolo de gerência adequado para o referido elemento da rede. Este elemento, por meio do seu *software* agente, fornece a resposta ao *proxy*, que traduz de volta para o protocolo usado pelo gerente e encaminha tal informação a este.

A Figura 4 ilustra o mecanismo de funcionamento de um agente *proxy*, onde a aplicação gerente recorre ao *proxy* para comunicar-se com o agente do elemento gerenciável.

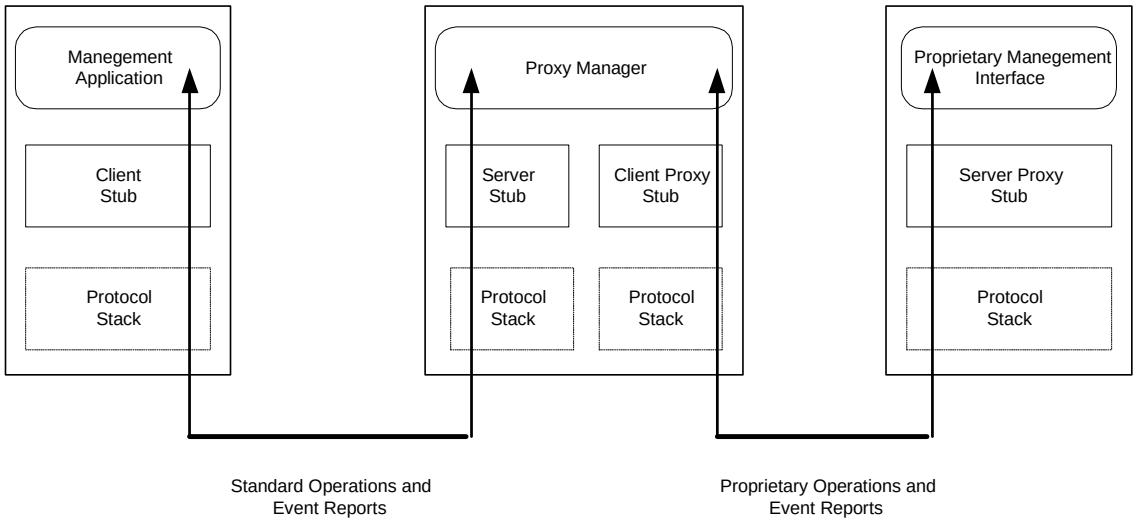


Figura 4 - Arquitetura usando um agente *proxy* [62]

3.4 Protocolos de gerenciamento

Os principais protocolos desenvolvidos para gerenciamento de redes são os padronizados na Internet e pela ISO, respectivamente o SNMP e o CMIP. Com o crescimento de aplicações sendo executadas em ambientes distribuídos, tais protocolos têm-se firmado como solução para gerenciamento de grandes redes [17].

A ISO especifica o ***Common Management Information Protocol*** (**CMIP**) e o ***Commom Management Information Services*** (**CMIS**) como protocolo e serviço de gerenciamento de rede do nível de aplicação do modelo OSI.

A necessidade de mecanismos de gerenciamento nas redes baseadas em TCP/IP é atendida pelo ***Simple Network Management Protocol*** (**SNMP**). Uma das vantagens do SNMP é a simplicidade e facilidade de implementação, o que faz com que se resolva boa parte dos problemas relacionados à gerência de redes.

3.5 Considerações Finais

Neste capítulo foram trabalhados os conceitos relacionados à gerência de redes, em especial voltado para gerência de configuração de redes.

O objetivo principal desta dissertação é atuar nesta área, de modo a apresentar através do sistema **O-bCNMS** proposto, formas de se aplicar a teoria envolvida em um processo de gerência de configuração buscando um controle do ambiente gerenciado. Deste modo, algumas características apresentadas pelas principais ferramentas de mercado são exploradas no presente trabalho, como principalmente a extração de dados dos ativos em funcionamento para se construir um modelo de informações gerenciais que permita ao administrador realizar uma gestão do domínio de conhecimento estudado.

AGENTES

O conceito de agentes tem origem em diversas áreas distintas. De acordo com [76], agente pode ser definido como “um sistema encapsulado situado em um determinado ambiente, com a capacidade de ser flexível, ou seja, mutável de acordo com o cenário que se apresenta, e autônomo, isto é, independente em suas ações”. Ou seja, um sistema computacional situado dentro de um ambiente específico, com a capacidade de atuar de forma reativa e pró-ativa, visando atingir os objetivos projetados para o mesmo. Segundo [10], um agente possui como características principais a reatividade, a autonomia, a pró-atividade e a cooperação. A primeira delas está relacionada à capacidade do agente reagir, sempre da mesma forma, a alguma situação para a qual foi programado. A autonomia constitui-se em uma das principais características de um agente. Tal comportamento permite a este, a partir de um modelo do ambiente em que está inserido, tomar decisões capazes de realizar ou rejeitar uma solicitação feita a ele. A pró-atividade, também relacionada a este comportamento, traduz a capacidade do agente antecipar uma eventual necessidade de mudança e executá-la, antes mesmo que o problema ocorra. A última característica, a cooperação, refere-se à comunicação entre agentes de modo a contribuir para alcançarem o mesmo objetivo.

Os agentes podem ou não serem desenvolvidos utilizando-se técnicas de Inteligência Artificial. De acordo com [21], os agentes podem ser classificados, segundo seu comportamento, conforme a Figura 5. Segundo [57], esta classificação

é adequada para agentes envolvidos em gerência de redes de computadores. O comportamento passivo é atribuído àqueles agentes que não possuem autonomia e não utilizam técnicas de IA. São agentes que se assemelham a simples programas e são construídos através de instruções, o que proporciona a eles um comportamento pré-definido. Um exemplo importante é o modelo agente-gerente tradicional, no qual o agente é um *software* com funções de coletar informações e enviá-las ao gerente para que este possa tomar as decisões adequadas. Os agentes desenvolvidos com técnicas de IA são classificados como ativos ou autônomos [21] [57]. Os agentes ativos são desenvolvidos com o auxílio de heurísticas. Na área de gerência de redes, estas heurísticas, que definem regras de produção ou redes neurais diretas para apoiar na solução de problemas, são fornecidas pelo administrador da rede. Tais agentes são categorizados como sistemas de raciocínio lógico, além de serem similares a sistemas especialistas e não possuírem autonomia.

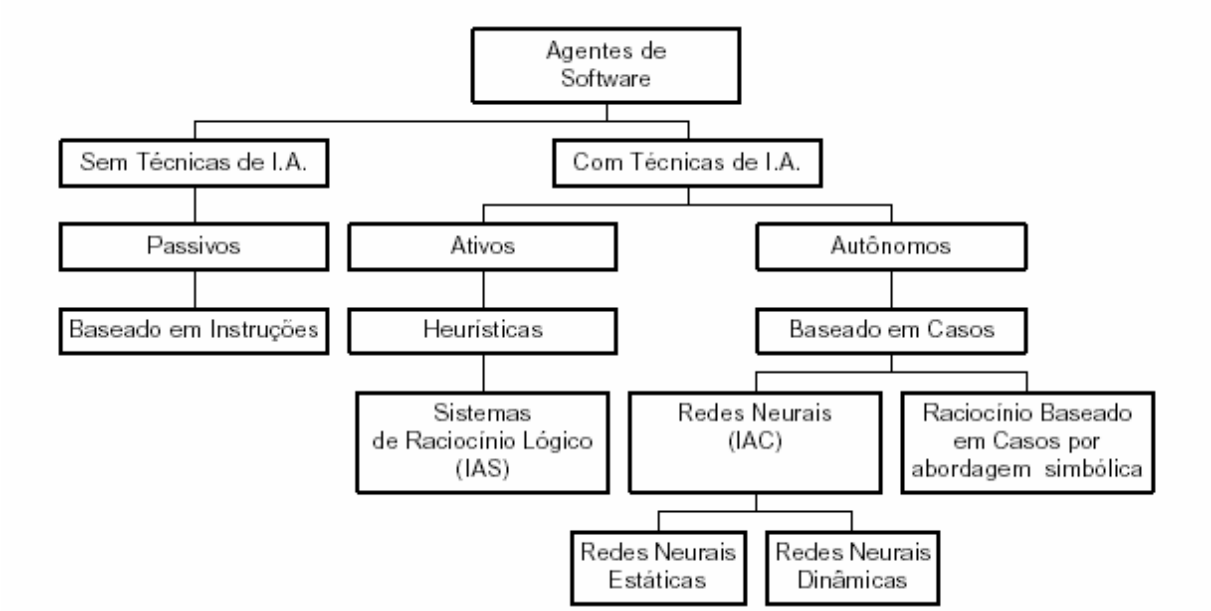


Figura 5 - Classificação para o paradigma de agentes [57]

Agentes autônomos, segundo [57], são sistemas computacionais que operam em ambientes dinâmicos e imprevisíveis, sendo capazes de interpretar dados obtidos por eventos ocorridos no ambiente e executar ações que produzam efeitos nesse mesmo contexto em que está inserido. São independentes, capazes de aprender e modelar sua forma de agir, através da observação e coleta de exemplos de ações que acontecem no ambiente em que está inserido.

Com base na Teoria de Autômatos [57], agentes autônomos são classificados em estáticos, que refletem uma aplicação direta de modelos de ambientes pré-definidos, e dinâmicos, capazes de expandirem seus modelos internos de ambiente em função de acontecimentos a que são submetidos. O processo para se desenvolver tais tipos de agentes, envolve uma coleta de exemplos de como deve ser o funcionamento de cada um deles. Se o agente for implementado a partir de uma rede neural, estes exemplos servirão para realizar o treinamento da rede. Este pode fazer uso de redes neurais diretas, também conhecidas como estáticas, ou de redes neurais dinâmicas.

Outra categoria, conforme [57], seria construir os agentes através de um sistema de raciocínio baseado em casos, que faz uso do raciocínio por analogia. Nesta situação, tem-se um conjunto de casos resolvidos de um mesmo problema e, dado um novo caso, necessita-se saber como resolvê-lo. Seguindo a heurística de desprezar inteligentemente casos, seleciona-se uma distância entre casos e desprezam-se todos os que estão mais distantes do exemplo a resolver, retendo-se ao mais próximo. A solução do novo caso será a solução do caso mais próprio.

4.1 Tipologia de agentes

Segundo [49], os agentes podem ser classificados como na Figura 6.

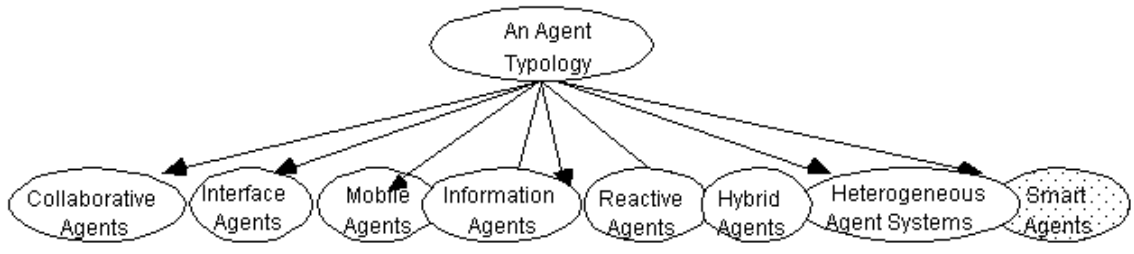


Figura 6 - Tipologia de agentes [49]

Esta classificação não é ortogonal, uma vez que um determinado agente pode ser categorizado em mais de um tipo simultaneamente. Ou seja, um agente

pode ser móvel e inteligente ao mesmo tempo. Porém, cada categoria pode ser caracterizada da seguinte forma:

Agentes colaborativos (*Collaborative Agents*) dão ênfase à autonomia e cooperação com outros agentes, em detrimento de realizar tarefas solicitadas pelo seu gerente. Eles são capazes de aprender, porém esta não é uma de suas características principais. Normalmente, devem ser capazes de agir racionalmente e de forma independente em ambientes multiagentes que tenham restrições de tempo. Tendem a ser estáticos, isto é, atuam estritamente no ambiente em que estão inseridos sem mover-se por outras redes.

Agentes de interface (*Interface Agents*) caracterizam-se pela autonomia e pela capacidade de aprender. Suportam e provêem assistência ao usuário para que este aprenda a utilizar uma aplicação específica. Esta assistência fornecida ao usuário ocorre basicamente de quatro maneiras: através da observação e reprodução do comportamento do usuário; através das respostas fornecidas pelo usuário; por meio de instruções recebidas ou através da solicitação de informações a outros agentes.

Agentes móveis (*Mobile Agents*) são *softwares* capazes de mover-se, através de *roaming*, por redes geograficamente distribuídas (WANs). São capazes também de interagir com estações externas, obtendo informações das mesmas. São considerados agentes porque são autônomos e cooperativos, porém de uma forma diferente dos agentes colaborativos, visto que estes priorizam a interação com outros agentes em detrimento de atender a uma solicitação realizada pelo gerente. Já os agentes móveis não dão prioridade a este tipo de cooperação, dando ênfase aos seus próprios objetivos.

Agentes de informação (*Information Agents*) têm como função a gerência, manipulação e comparação de informações extraídas de diferentes fontes distribuídas. São definidos pelo que são capazes de fazer, enquanto os agentes colaborativos e de interface são definidos pelo que são.

Agentes Reativos (*Reactive Agents*) representam uma categoria especial de agentes que não possuem modelos internos de ambientes. Eles agem sempre em resposta a um estímulo ao estado atual do ambiente em que estão inseridos.

São vistos como uma coleção de módulos que operam de maneira autônoma e são responsáveis por tarefas específicas.

Agentes híbridos (*Hybrid Agents*) são os agentes formados pela composição dos anteriores, sobretudo agentes colaborativos e de interface com agentes reativos. Uma arquitetura típica de um agente híbrido consiste em uma base de conhecimento associada com uma unidade de controle, localizada no topo de um componente responsável pela percepção do meio, que também gerencia comunicações de baixo nível. Há três camadas nesta unidade: a camada baseada em comportamento - *behaviour-based layer* (BBL), a camada local de planejamento - *local planning layer* (LPL) e a camada de planejamento cooperativo - *cooperative planning layer* (CPL).

Sistemas de Agentes Heterogêneos (*Heterogeneous Agents System*) são referentes a um conjunto de dois ou mais agentes integrados que pertençam a duas ou mais classes de agentes distintas. Pode conter um ou mais agentes híbridos.

Agentes Inteligentes ou Cognitivos (*Smart Agents*) são agentes cuja inteligência deriva do comportamento emergente da interação de vários módulos. Pode ser desenvolvido a partir de agentes simples. Possuem a capacidade de aprendizado e pró-atividade como pontos fortes.

4.2 Sistemas Multiagentes

Segundo o contexto da Inteligência Artificial Distribuída, de acordo com [65], um sistema multiagentes (SMA) caracteriza-se por possuir um conjunto de agentes que interajam de forma autônoma, independente e que trabalhem de forma cooperativa para resolver um determinado problema ou alcançar um objetivo específico. Pode-se dizer que sistemas multiagentes são redes fracamente acopladas e formadas por múltiplos agentes que interagem entre si e trabalham em conjunto, visando realizar tarefas ou objetivos. Esses objetivos podem ou não ser comuns a todos os agentes. Tais agentes podem ser heterogêneos ou homogêneos, colaborativos ou competitivos, entre outros. A definição dos tipos de agentes depende da finalidade da aplicação que o sistema multiagente está inserido.

Os sistemas multiagentes com agentes reativos são constituídos por um grande número de agentes. Estes interagem utilizando um comportamento de ação/reação. A inteligência surge como decorrência dessas grandes trocas de interações entre os agentes e o ambiente. Ou seja, os agentes não são inteligentes individualmente, mas o comportamento global deles é. Já os sistemas multiagentes constituídos por agentes cognitivos são geralmente compostos por uma quantidade bem menor de agentes se comparado aos sistemas multiagentes reativos. Estes são inteligentes e contêm uma representação parcial de seu ambiente e dos outros agentes. Tais agentes comunicam-se entre si e, desta forma, conseguem negociar uma informação ou um serviço e prever uma ação futura.

Esse planejamento é possível, pois, em geral, tais agentes são dotados de conhecimentos, competências, intenções e crenças, o que lhes permite coordenar suas ações visando à resolução de um problema ou a execução de um objetivo. Entende-se por conhecimento as informações que o agente possui sobre o mundo que ele ocupa. Competência é a capacidade de atuação do mesmo dentro do contexto a que pertence. Intenções são informações que guiam as ações do agente. Crenças referem-se ao estado do mundo e mental dos agentes que compõem o meio de modo a ser base para a tomada de decisão de um agente.

Atualmente a pesquisa em sistemas multiagentes tem mantido seu foco na coordenação das ações e comportamentos dos agentes, bem como na unificação dos *frameworks* de representação e avaliação das primitivas de modelagem. Segundo [38], o crescimento do interesse em pesquisas nesta área tem se concentrado em buscar capacidade de fornecer robustez e eficiência, de permitir interoperabilidade entre sistemas legados e de resolver problemas cujo controle é distribuído. Nesta linha, um sistema multiagentes tem como desafios a serem superados a comunicação entre agentes, a linguagem a ser utilizada para realizar a interação entre os agentes e a necessidade de garantir a coordenação entre os agentes para que a solução do problema seja coerente.

4.3 Comunicação entre agentes

A comunicação é fundamental para permitir que haja colaboração entre os agentes independentes. Para que seja possível uma comunicação entre dois agentes, ambos devem: possuir capacidade de se comunicar, compartilhar um conteúdo de conhecimento básico e uma maneira de representar esse conhecimento. Em sistemas multiagentes, é necessário que a comunicação siga um padrão. Para tanto, deve-se utilizar uma linguagem que possa ser entendida pelos demais agentes que compõem o ambiente. Essa comunicação deve permitir que agentes coordenem suas próprias atividades, além de trocarem informações entre si.

Existem formas distintas para a troca de informações entre os agentes em um sistema multiagentes. Segundo [1], agentes podem trocar mensagens de forma direta, podem comunicar-se através de um agente “facilitador”, também conhecida como comunicação assistida, e podem também utilizar uma comunicação por difusão de mensagens. Existe ainda o modelo de comunicação através de *blackboard* ou quadro-negro.

De acordo com [46], a comunicação direta prevê que cada agente se comunique diretamente com qualquer outro sem a necessidade de uma entidade intermediária. Eles estabelecem uma ligação direta ponto-a-ponto entres os agentes, através de um conjunto de protocolos que garanta a chegada de mensagens com segurança. Nesse caso, é preciso que cada agente envolvido na comunicação conheça a existência dos demais, bem como o modo de endereçar mensagens para eles. A principal vantagem deste tipo de comunicação entre agentes é a não necessidade de um agente coordenador da comunicação. Em contrapartida, o custo da comunicação é elevado, principalmente quando o número de agentes no sistema é grande, o que torna a implementação mais complexa.

Na comunicação assistida, ou sistema federado, segundo [46] citando [1], os agentes utilizam outro *software* ou um agente especial para coordenar suas atividades. Define-se uma hierarquia entre os agentes participantes e a troca de mensagens ocorre por meio desses agentes facilitadores ou mediadores. Trata-se

de uma forma de comunicação menos custosa e menos complexa. Recomenda-se para sistemas em que o número de agentes é muito elevado.

A comunicação por difusão de mensagens ou *broadcast*, na maioria das vezes, de acordo com [1], é utilizada em casos em que se necessita enviar a mensagem para todos os agentes do meio ou quando o agente remetente desconhece o agente destinatário ou seu endereço.

Comunicação por quadro-negro ou *blackboard*, segundo [1], é utilizada na Inteligência Artificial como modelo de memória compartilhada. Trata-se de um repositório no qual os agentes postam mensagens a outros agentes e conseguem obter informações sobre o ambiente em que estão inseridos.

4.4 Linguagens de Comunicação entre agentes

As linguagens de comunicação entre agentes definem a capacidade que cada agente possui de se comunicar. Deve ser compartilhada por todos os agentes do sistema e possuir um número limitado de primitivas de comunicação. Um dos problemas nessas linguagens, conforme [46], refere-se à interpretação do conteúdo destas mensagens. Deste modo, é necessário que a mensagem na comunicação entre agentes seja clara e explícita.

4.4.1 KQML

Knowledge Query and Manipulation Language (KQML) é uma linguagem de comunicação de alto nível para troca de mensagens. Trabalha de forma independente de conteúdo, plataforma e da ontologia utilizada. É utilizada principalmente para consulta, manutenção, ações e serviços de bases de conhecimento. Trata os envolvidos como bases de conhecimento virtuais. Foi desenvolvida pelo KSE – "*Knowledge Sharing Effort*" – com o intuito de ser um formato padrão de mensagem e um protocolo de gerenciamento de mensagens. Possui foco na especificação da informação necessária à compreensão do conteúdo da mensagem.

Segundo [29], o significado de padrões no KQML normalmente está associado à intuição, ou seja, parte-se do princípio que os termos são de domínio público, através do conhecimento intuitivo que cada um já possui. Tal fato causa ambigüidade e existência de termos vagos, além de performativas reservadas com nomes inadequados. Possui sintaxe semelhante ao LISP e é composta da ação (performativas) e de parâmetros não posicionais. É baseada em comunicação ponto a ponto e permite outras formas de comunicação, além da participação de agentes especializados, que são agentes com habilidades e conhecimento específicos para desempenharem determinadas funções. A Figura 7 ilustra a sintaxe definida para uma mensagem KQML, na qual o termo **:sender** identifica o agente emissor da mensagem, o termo **:receiver** identifica o agente destino da mensagem enviada. O termo **:reply-with** apresenta o rótulo esperado para a mensagem de resposta. O termo **:language** traz a linguagem de representação definida para o conteúdo da mensagem, enquanto o termo **:ontology** define o nome da ontologia utilizada por este conteúdo, que refere-se ao termo **:content**. Este representa a informação sobre a qual a performativa expressa uma atitude.

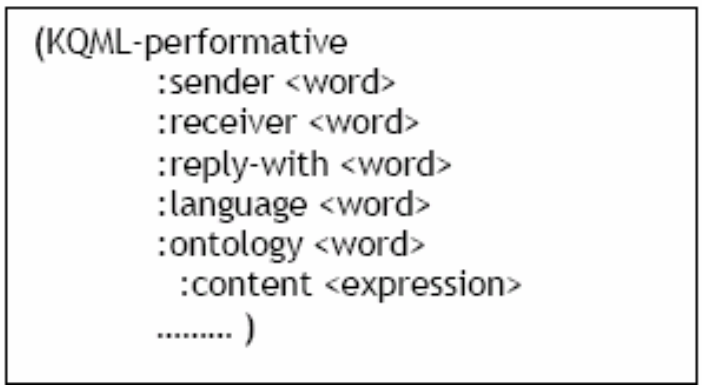


Figura 7 - Formato de mensagem KQML [46]

4.4.2 FIPA-ACL

A FIPA-ACL (*Foundation for Intelligent Physical Agents - Agent Communication Language*) é uma linguagem baseada em ações próprias de um diálogo. Possui uma sintaxe semelhante ao KQML, porém o conjunto de performativas é mais bem definido [76]. É formada por um conjunto de tipos de

mensagens e descrições das conseqüências da mensagem sobre os agentes emissores e receptores.

Trata-se de uma linguagem baseada em atos de fala (*speech acts*) que permite aos agentes utilizarem distintos mecanismos de transporte. De acordo com [76], possui um serviço de mensagem confiável, que informa ao agente, quando houver, os erros ocorridos. Permite ao agente determinar se o processamento de mensagens será assíncrono ou síncrono. Cada agente possui um nome que permite identificar sua localização. Suas primitivas são chamadas de ações ou atos comunicativos (*communicative acts*).

Um ato de fala, ou *speech act*, é baseado no modelo BID – *Belief, Intention, Desire* – ou seja, crença, intenção e desejo. Ele pode ser caracterizado por quatro pontos: pré e pós-condições de estados mentais, questões normativas de ação e reação, aspectos sociais de comunicação e pragmáticas de atos de fala.

O primeiro ponto relacionado às pré-condições e pós-condições dos estados mentais de um agente pode ser entendido da seguinte forma: se um agente envia ao outro uma consulta sobre a temperatura da cidade de Vitória, ele expressa um desejo (de conhecer a temperatura) e uma crença (de que o agente destino a conhece). Estas seriam pré-condições ao processo de comunicação. Da mesma forma, o agente receptor percebe a intenção do agente emissor que ele conheça de fato a temperatura, assim como ele sabe que o agente emissor tem o desejo de possuir esta informação que ele, no momento não possui.

Com relação às questões normativas de ação e reação, um ato de fala expressa com clareza quem deseja saber que tipo de informação, isto é, o desejo do agente ao realizar esta comunicação e quem será responsável por concedê-la, qual agente tentará responder ao desejo do agente emissor.

Os aspectos sociais de comunicação indicam o tipo de performativa associado à comunicação, se está referenciando uma crença ou de uma requisição que deve ser executada pelo agente destino. Por exemplo, se uma mensagem possui o seguinte conteúdo ***authority(a,b, request)***, significa dizer que a requisição enviada por **a** deve ser cumprida por **b**.

As pragmáticas de atos de fala referem-se ao modo como os agentes se interagem e interpretam as respostas que recebem. Sendo assim, considerando o

exemplo anterior no qual o agente emissor deseja conhecer a temperatura da cidade de Vitória, assume-se que este seja o seu real objetivo. Deste modo, ele espera receber algum tipo de resposta do agente para o qual enviou a solicitação. Caso o agente destino deseje manter uma relação cooperativa, ou seja, cordial com o emissor, ele deve necessariamente enviar uma resposta, ainda que não seja a resposta exata da requisição feita.

Por resolver a maioria dos problemas identificados no KQML, surgiu como uma linguagem a ser estabelecida para a comunicação de agentes. A sintaxe de uma mensagem FIPA-ACL é mostrada na Figura 8. Possui elementos em comum com a sintaxe do KQML (**:sender**, **:receiver**, **:content**, **:language**, **:ontology**, entre outros), além de termos adicionais que trazem mais informação à mensagem, como **:conversation-id**, que trata-se de um identificador único para um processo de comunicação específico.

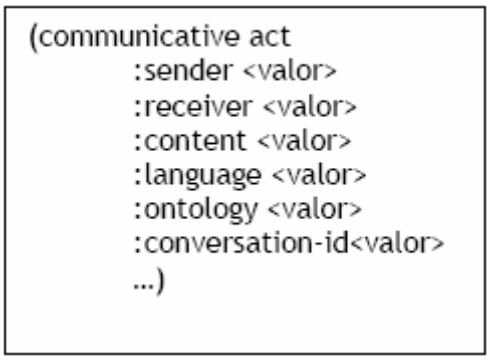


Figura 8 - Formato de mensagem FIPA-ACL[76]

4.5 Coordenação entre agentes

De acordo com [76], para sistemas multiagentes no qual há uma colaboração entre eles, a coordenação é um fator primordial.

Trata-se de compartilhar o conhecimento entre os agentes envolvidos, de modo que as ações específicas de cada agente sejam coordenadas para se atingir o objetivo final do sistema.

A importância da coordenação se justifica pelo fato de muitos dos objetivos de um sistema multiagentes, não poderem ser alcançados por um agente

de forma individual. Sendo assim, a necessidade dos agentes compartilharem informações para chegar ao objetivo do sistema é essencial.

Segundo [76], uma cooperação demanda que cada agente mantenha um modelo dos demais agentes com os quais ele se relaciona e também um modelo de interações possíveis. O processo se divide em cooperação e negociação.

Para [76], negociação pode ser definida como a coordenação entre agentes antagônicos, ou seja, entre agentes que trabalham isoladamente com o foco nos seus próprios objetivos (*self-interested*). Na maioria das vezes, usam-se protocolos de negociação para determinar as regras e definem-se os conjuntos de atributos sobre os quais se pretende chegar a um acordo sobre o que está sendo negociado.

Já cooperação, de acordo com [76], é uma coordenação com agentes que não sejam competitivos e que se ajudem mutuamente, ainda que para realizar esta cooperação provoquem custos para cada um dos agentes participantes. O foco, nesta situação, é o trabalho em equipe visando o objetivo final do sistema. Porém, a cooperação somente ocorre quando há, entre os agentes envolvidos, uma dependência mútua, de forma que o objetivo que provocou a dependência entre eles possa ser composto por objetivos individuais, isto é, cada agente pode ter a sua própria meta, mas não a atinge totalmente sem a cooperação com outro(s) agente(s).

4.6 Ambientes Multiagentes

Conforme [76], ambientes de sistemas multiagentes devem prover uma infra-estrutura que defina os protocolos de comunicação e interação entre os agentes. São geralmente abertos, não-centralizados e são compostos por agentes autônomos e distribuídos, que podem ser competitivos ou cooperativos.

As características principais estão descritas na Tabela 2, na qual o ponto mais relevante está relacionado à infra-estrutura de comunicação, que pode prover um mecanismo sincronizado ou não para troca de mensagens entre os agentes, orientado ou não à conexão, entre outros. Outros pontos importantes são o fornecimento de:

- serviços de mediação com suporte a ontologias;
- serviços de segurança, como autenticação;
- definição do protocolo utilizado pelas mensagens, como FIPA-ACL, KQML;
- funcionalidades de diretório de serviços relacionados à busca e localização dos agentes no ambiente para comunicação com os demais.

Propriedade	Valores
Autonomia de Projeto	*Plataforma/Protocolo de Interação/Linguagem/Arquitetura Interna*
Infra-estrutura de Comunicação	*Memória Compartilhada (blackboard) ou Baseada em Mensagens, Orientada à Conexão ou Não Orientada à Conexão(e-mail), Ponto-a-Ponto,Multicast ou Broadcast, Síncrono ou Assíncrono*
Diretório de Serviço (Directory Service)	*White pages, Yellow pages*
Protocolo de Mensagens	*KQML, FIPA-ACL, HTTP OU HTML,OLE , CORBA*
Serviço de Mediação	*Baseado em Ontologia ou Transações*
Serviço de Segurança	*Timestamps/Autenticação*
Operações de Suporte	*Armazenamento/Redundância/Restauração/ Accounting*

Tabela 2 - Características de ambientes multiagentes [76]

4.7 Padrão FIPA

Foundation for Intelligent Physical Agents (FIPA) é uma fundação, sem fins lucrativos, fundada em 1996 em Genebra. A partir de junho de 2005, tornou-se integrante da comunidade IEEE. Seu objetivo é gerar padrões para sistemas baseados em agentes, com o foco na interoperabilidade de agentes heterogêneos e interativos. Preocupa-se com a conexão de agentes e sistemas multiagentes entre múltiplos fornecedores de ambientes para tais entidades. O padrão FIPA é focado no comportamento externo dos componentes do sistema, deixando em aberto detalhes internos de implementação e a arquitetura. Trouxe padronização e maior

agilidade no desenvolvimento da tecnologia de agentes, visto que anteriormente havia uma competição entre sistemas proprietários, sem um padrão definido. O modelo proposto pela FIPA, ilustrado na Figura 9, caracteriza-se por apresentar três níveis e cinco elementos.

O primeiro nível, *Applications*, define requisitos para construção de modelos de domínio de conhecimento e serviços que possam ser utilizados pelos agentes que compõem a plataforma aderente ao padrão.

O segundo nível, *Abstract Architecture*, estabelece entidades relevantes para construção da plataforma de agentes.

O terceiro nível é formado por três elementos: *Agent Communication*, *Agent Management* e *Agent Message Transport*. O primeiro deles tem como preocupação definir o formato e padrão de mensagem a ser trocada entre os agentes, no caso o padrão ACL (*Agent Communication Language*). O segundo estabelece requisitos para controle e gestão dos agentes na plataforma. O último deles preocupa-se em garantir que as mensagens trocadas entre os agentes possam ser transportadas pela rede por quaisquer protocolos de transporte capazes de desempenhar esta função.

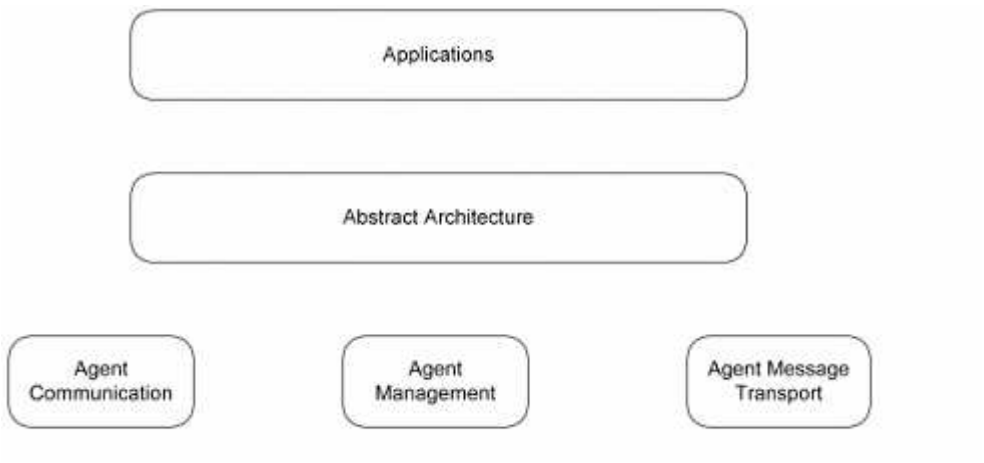


Figura 9 - Camadas do modelo FIPA [20]

FIPA é composta por um comitê principal, o *IEEE FIPA Standards Committee*. Este é formado por dois tipos de grupos. São eles:

- Grupos de trabalho (*Working Groups* - WG) – Responsáveis por projetos de padronização que estejam no escopo de FIPA SC (*Foundation for Intelligent*

Physical Agents Standard Committee). Os grupos de trabalho já aprovados até o momento são:

- Interoperabilidade de agentes e *Web Services* (AWSI WG);
- Comunicações Homem-Agente (HAC WG);
- Agentes Móveis (MA WG);
- Agentes Nômades *Peer to Peer* (P2PNA WG).

- Grupos de estudos (*Study Groups* - SG) - A FIPA SC pode formar um grupo de estudos por 12 meses para criar padrões de uso ou de desenvolvimento de projetos. Deve possuir um plano de trabalho que inclua entregas específicas. O grupo de estudos aprovado até o momento é:
  - Revisão da Especificação FIPA (ROFS SG).

4.8 Considerações Finais

Este capítulo teve como foco apresentar conceitos envolvidos no trabalho com agentes. A definição de metodologias de desenvolvimento de *softwares* orientados a agentes, conforme [61], cresce de um modo promissor na engenharia de software. Algumas metodologias de desenvolvimento orientadas a agentes ganham destaque como AUML (*Agent Unified Modelling Language*) [2], que realizam aplicações de conceitos do paradigma orientado a objeto ao mundo de agentes.

Outra metodologia de desenvolvimento de sistemas orientada a agentes importante é *Tropos* [77]. Inspirada na análise de requisitos iniciais e fundamentada em conceitos sociais e intencionais. *Tropos* adota o *framework* de modelagem organizacional *i** durante todas as fases de desenvolvimento de agentes. O *framework i** possui uma estrutura conceitual que distingue motivações, intenções e raciocínios sobre as características de um processo [77].

A principal característica da metodologia orientada a agentes utilizada neste projeto, mais especificamente no estudo de caso, foi a modelagem com o foco em comportamentos. Dessa forma, identificam-se os agentes a partir de tarefas que os mesmos devam cumprir. Esta abordagem estimula o reuso, uma vez que os

comportamentos podem ser utilizados por agentes distintos, e auxilia uma melhor definição do papel de cada agente em um sistema multiagente, no qual a interação entre estes é o ponto forte a ser explorado.

A escolha de agentes para o presente trabalho está pautada principalmente nas características comportamentais apresentadas pelos mesmos, como a interação constante com o meio e o modo dinâmico como atuam e capturam informações do contexto em que estão inseridos. A cooperação e a capacidade de atuarem de forma reativa ou, até mesmo, de forma pró-ativa, são pontos a serem bastante utilizados quando se busca a realização de tarefas em sistemas voltados para gerência de redes. Tal fato se explica porque em um cenário de elementos diversos a serem gerenciados, a capacidade de agir em função de um evento e de trocar informações e ações visando a completude de uma tarefa específica, torna-se essencial para se ter um processo de controle eficiente, capaz de prover ao administrador o conhecimento de seu ambiente e, consequentemente, uma gerência efetiva da configuração da estrutura a ser monitorada.

JADE

JADE (*Java Agent Development Framework*) [36] [4] é um *middleware* desenvolvido e distribuído pelo TILAB [64], de acordo com a licença LGPL (*Lesser GNU Public License*) [63], para o desenvolvimento de aplicações distribuídas multiagentes baseada na arquitetura de comunicação *peer-to-peer*. Completamente implementado em Java™, seu objetivo é facilitar o desenvolvimento de sistemas orientados a agentes. JADE tem como principais características:

- Fornecer um ambiente de agentes compatível com as especificações da FIPA, conforme seção 4.7;
- Possuir uma interface visual que suporte o gerenciamento de agentes distintos e sistemas multiagentes;
- Prover ferramentas que auxiliam o desenvolvimento e a depuração de aplicações multiagentes implementadas em JADE;
- Dar apoio a execuções de ações de agentes de modo paralelo, concorrente ou sequencial, através dos modelos de comportamentos disponíveis na sua API;
- Oferecer uma biblioteca de protocolos padrão FIPA, para interação entre agentes, pronta para ser usada;
- Permitir que o registro dos agentes seja transparente para o desenvolvedor;

- Realizar o transporte de mensagens no formato FIPA-ACL dentro da mesma plataforma de agentes;Possuir um serviço de nomes em conformidade aos padrões FIPA, permitindo que os agentes recebam um identificador único na sua iniciação, válido para todo ambiente;
- Oferecer mecanismos que tornam possível a sua integração com aplicações externas que executem agentes de JADE.

Em JADE, comportamentos são *threads* de execução lógica que podem ser compostos de várias maneiras para atingir padrões de execução, podendo ser iniciados, suspensos ou gerados em qualquer momento, segundo [3]. O paradigma adotado na comunicação entre os agentes é o de troca de mensagens de modo assíncrono. Cada agente possui um tipo de fila de mensagens na qual o ambiente de execução JADE disponibiliza mensagens enviadas por outros agentes. As mensagens trocadas por agentes de JADE seguem a linguagem ACL, definida pelo padrão internacional da FIPA.

5.1 *Middleware*

Middleware, segundo [47], pode ser definido como *software* que permite a interconexão entre aplicações distintas e completamente segregadas, como ilustrado na Figura 10. No contexto do JADE, segundo [47], este comportamento seria representado pelas bibliotecas de alto nível, que permite, de modo efetivo e mais simples, o desenvolvimento de aplicações, fornecendo serviços genéricos como: acesso a dados, controle de recursos, entre outros. Os serviços são fornecidos pelos sistemas operacionais. De acordo com [47], a idéia por detrás do *middleware* é fornecer API’s independentes da plataforma que adicione facilidades nativas dentro de simples blocos reutilizáveis, o que fornece uma redução no custo e tempo de desenvolvimento.

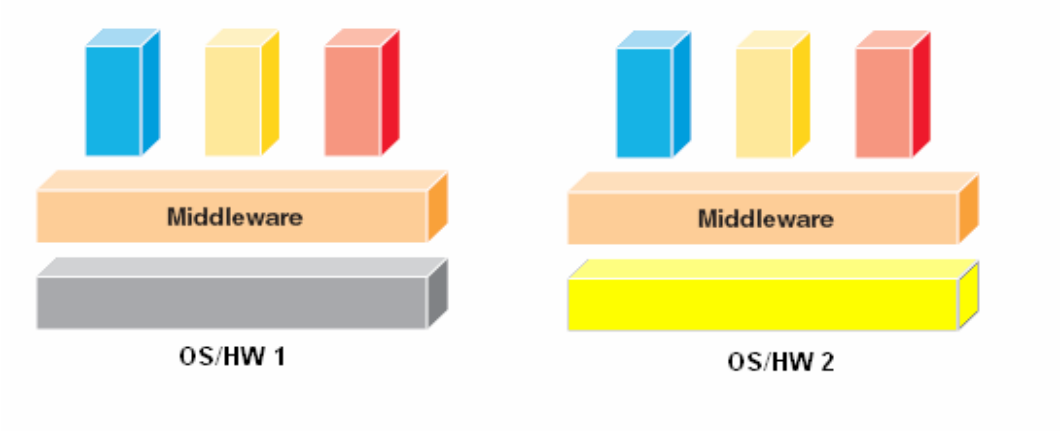


Figura 10 - Funções de um *Middleware* [47]

5.2 Arquitetura

O modelo de arquitetura do JADE, mostrado na Figura 11, tem as seguintes características:

- Composta por agentes que possuem um único nome;
- Cada agente é capaz de se comunicar com outros agentes de forma bidirecional;
- Cada agente tem o seu próprio repositório (*container*) associado a uma máquina virtual Java,
- Integra-se a sistemas mais complexos como plataformas J2EE e .NET.

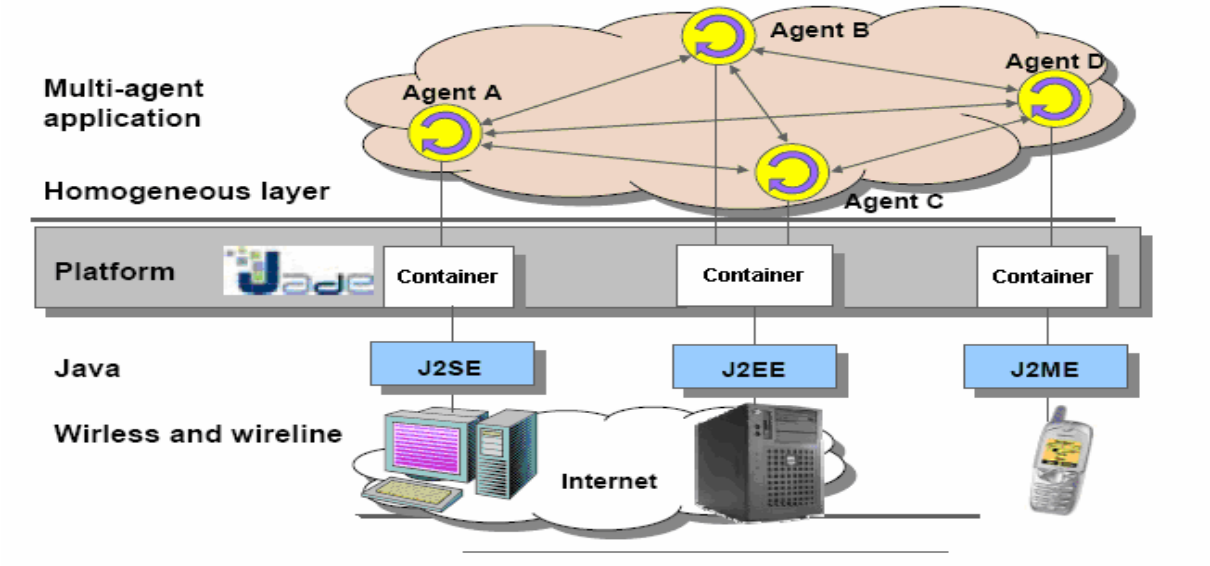


Figura 11 - Arquitetura JADE [3]

5.2.1 Plataforma

São três, os agentes necessários na plataforma JADE, conforme a Figura 12: *Agent Management System* (AMS), *Message Transport System* (MTS), *Directory Facilitator* (DF).

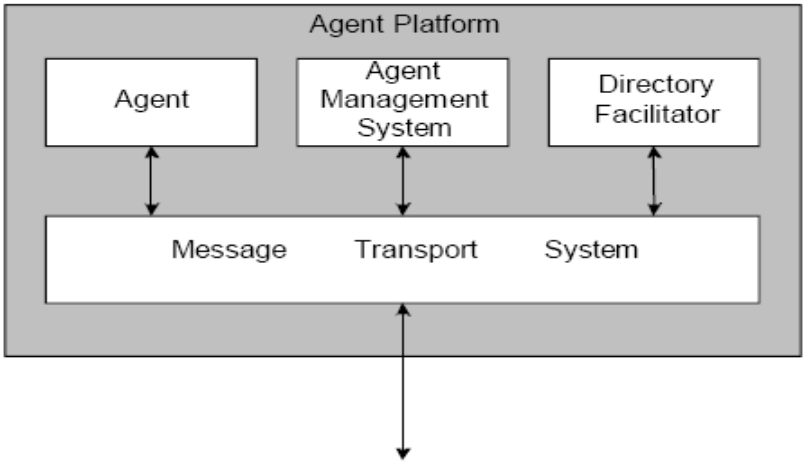


Figura 12 - Arquitetura da plataforma de agentes FIPA [3]

Segundo [4], o *Agent Management System* (AMS) é o agente responsável por controlar o acesso e o uso da plataforma. So há um AMS em cada plataforma. O AMS fornece serviço de páginas brancas, ou seja, permite catalogar a localização dos agentes dentro da plataforma, e de ciclo de vida, além de manter um diretório de identificadores de agentes (AID) e o estado do agente. Todo agente deve se registrar no AMS para ganhar um ID valido.

O *Directory Facilitator* (DF) é o agente que fornece um serviço padrão de páginas amarelas para a plataforma, que segue a especificação FIPA [4]. Este serviço de páginas amarelas tem o papel de facilitador no processo de localização dos agentes na plataforma, uma vez que cabe a ele informar em que ponto da plataforma está o container em que o agente ao qual se deseja comunicar está.

De acordo com [4], o *Message Transport System* também conhecido como *Agent Communication Channel* (ACC) é um componente de *software* que controla as trocas de mensagens, incluindo mensagens enviadas e recebidas de plataformas remotas.

O JADE, ao ser executado, o *Agent Management System* (AMS) e o *Directory Facilitator* (DF) são criados e o módulo ACC é configurado de modo a possibilitar a comunicação, [4]. De acordo com [4], o suporte para linguagens de conteúdo e ontologias fornecido por JADE é projetado para realizar a conversão de uma representação interna da informação do agente dentro da representação da expressão de conteúdo ACL. Este processo ocorre da seguinte forma: define-se um modelo de domínio para o qual os agentes deverão se comunicar. Todas as informações trocadas entre os mesmos deverão estar mapeadas nos conceitos identificados neste modelo, isto é, nesta ontologia. Porém, ao enviar um conjunto de informações a outro agente, este está em uma seqüência de bytes. Cabe ao JADE, checar semanticamente o conteúdo desta mensagem e convertê-lo em objetos de classes Java que se referem às classes definidas no modelo ontológico adotado pelos agentes participantes desta comunicação.

Segundo [61], outra característica importante de JADE é que sua plataforma de agentes pode ser distribuída por várias estações, cada uma executando apenas uma JVM (*Java Virtual Machine*). Os agentes são implementados como *threads* Java e inseridos dentro de repositórios de agentes. Estes repositórios são chamados de *containers*, cuja função é prover o suporte para a execução do agente e representar o ambiente de execução das aplicações de agentes [61]. O conjunto de todos os *containers* caracteriza uma plataforma. Esta, segundo [4], fornece uma camada homogênea, que encapsula, para os agentes e para os desenvolvedores de aplicações, a complexidade e a diversidade de características importantes para os agentes, sendo as principais: *hardware*, sistemas operacionais, tipos de redes ou JVMs.

Segundo [61], cada *container* equivale a um processo e podem-se ter diferentes *containers* na mesma plataforma, sendo que haverá uma JVM por *container*. Além disso, de acordo com [4], diferentes agentes podem conviver no mesmo *container*, uma vez que cada agente tem seu próprio *thread* de execução e a JVM escalona um ambiente *multi-threaded* preemptivo. O que define se um agente deve conviver com outros no mesmo *container*, é o fato da necessidade ou não do mesmo estar distribuído na rede em relação ao *container* onde está um determinado conjunto de agentes. Caso o novo agente necessite estar em um ponto específico da rede, ele então deverá ser associado a um novo *container*, visto que estará

associado a uma máquina virtual Java diferente. A comunicação entre JVMs é feita através da invocação remota de métodos Java, através da tecnologia RMI (*Remote Method Invocation*). De acordo com [4], a Figura 13 apresenta a estrutura da plataforma de agentes JADE distribuída. Ao se iniciar um agente em JADE, este será associado a um *container* principal, localizado na estação 1, que é o *container* onde se encontra o AMS (*Agent Management System*), o DF (*Directory Facilitator*) e registro RMI (*Remote Method Invocation Registry*). Esse registro RMI é o meio que JADE utiliza para manter as referências aos outros *containers* de agentes que se conectam à plataforma. Os outros *containers* de agentes, que correspondem a repositórios de agentes distribuídos pela rede, são conectados ao *container* principal, o que possibilita ao desenvolvedor abstrair a separação física das estações ou as diferenças existentes entre estas. Cada *container* possui uma única máquina virtual Java, indicado pelo elemento JRE 1.2 (*Java Runtime Environment*) da Figura 13.

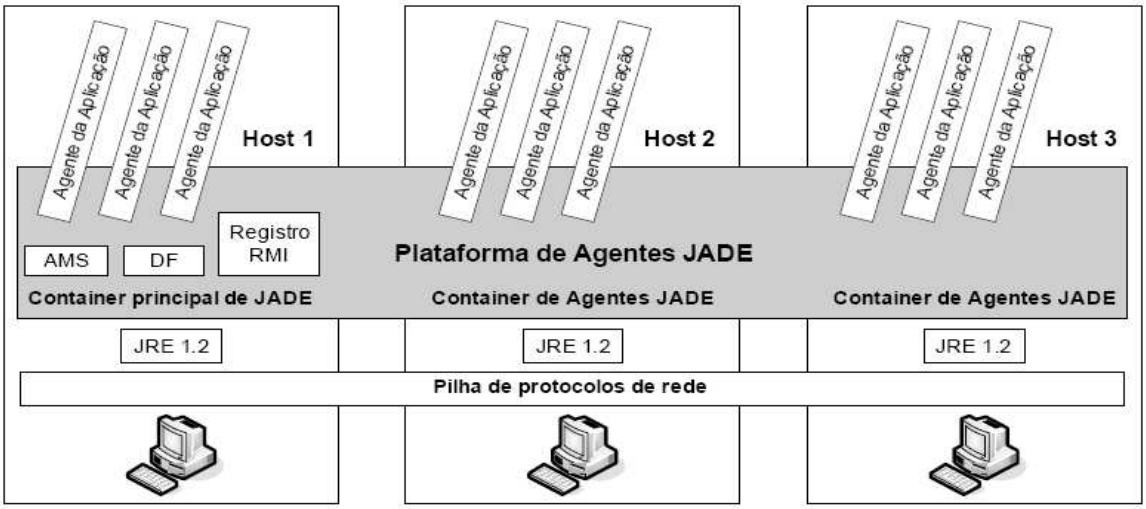


Figura 13 - Plataforma de agentes JADE distribuída por vários *containers* [4]

5.3 Agentes em JADE

Agentes em JADE são processos que representam entidades autônomas independentes, que possuem uma identidade e a capacidade de se comunicar com outros agentes para atingir os seus objetivos de execução [36], podendo realizar múltiplas tarefas e trocas de mensagens de forma simultânea.

De acordo com [4], um agente em JADE é caracterizado como uma instância da classe *Agent*, contida no pacote *jade.core*, ou de alguma subclasse de *Agent*. Esta classe facilita o desenvolvimento de agentes, visto que fornece todas as características necessárias para realizar as interações básicas com a plataforma, tais como métodos para registro, configuração ou gerenciamento remoto do agente. A estrutura da classe *Agent* também fornece o conjunto básico de métodos capazes de implementar o comportamento específico do agente, como por exemplo o envio de mensagens e a utilização de protocolos de interação [61].

Cada serviço ou funcionalidade de um agente deve ser implementado como sendo um comportamento. Os comportamentos de JADE modelam arquiteturas internas reativas. Porém, a abstração de comportamento do modelo do agente de JADE permite a integração com *softwares* externos. Isso é possível porque se pode integrar JADE com alguma estrutura para projetar, por exemplo, a base de conhecimentos do agente [61].

Segundo [61], JADE oferece também bibliotecas para implementação de diversas características de agentes, como, por exemplo, mobilidade e clonagem. Tais características possibilitam a criação de sistemas sofisticados. Isto torna possível que em uma mesma aplicação existam agentes de aprendizado, agentes baseados em utilidade, agentes móveis, e assim por diante.

Segundo [61], a definição de ontologias para utilização por agentes móveis em JADE, está disponível no pacote *jade.domain.mobility*. Este inclui uma lista de símbolos utilizados para mobilidade e de classes que implementam estas ontologias. Todo agente possui uma identificação única, denominada de AID (*Agent Identifier*). Esse identificador é constituído por um nome globalmente único que deve obedecer à estrutura: <*localname*>@<*hostname*>:<*port*>/*JADE*, como por exemplo: **ams@G6C15:1099/JADE**; um conjunto de endereços de agentes (uma vez que cada agente herda os endereços de transporte de sua plataforma de origem); e um conjunto de elementos que provêem páginas brancas, nas quais o agente é registrado para ser localizado posteriormente por outro agente que necessite se comunicar com o mesmo.

Os principais métodos da classe *Agent* são, de acordo com [4]:

- ***protected void setup()***: utilizado para inserção da codificação de iniciação do agente. Este método pode ser sobrescrito para fornecer os comportamentos necessários ao agente, além de tarefas comuns de iniciação, como, por exemplo, o registro no DF;
- ***public void addBehaviour(Behaviour comportamento)***: adiciona um novo comportamento ao agente, que será executado concorrentemente com outros comportamentos do agente. Geralmente é chamado dentro do método ***setup ()*** para disparar algum comportamento inicial, mas pode ser usado também para gerar comportamentos de modo dinâmico. Inversamente, o método **removeBehaviour ()** realiza o processo de remoção de determinado comportamento do agente;
- ***public final void send(ACLMessage mensagem)***: envia uma mensagem ACL para outro agente. Esse agente destino é especificado no campo *receiver* da mensagem na qual um ou mais agentes podem ser definidos como receptores;
- ***public final ACLMessage receive()***: recebe uma mensagem ACL da fila de mensagens do agente. Este método não é bloqueável e retorna a primeira mensagem da fila, caso ela possua mensagens, ou o valor nulo, caso a fila esteja vazia.

A criação de uma nova instância de agente em JADE ocorre do seguinte modo, segundo [61]: executa-se o método construtor do agente e este recebe um AID. Em seguida, o agente é registrado no serviço de páginas brancas (AMS) e colocado no estado ***AP_ACTIVE***. Então, o seu método ***setup()*** é executado. O método ***setup()*** deve ser implementado para iniciar um agente, devendo adicionar pelo menos um comportamento para o agente através do método ***addBehaviour()***. Ao final da execução do método ***setup()***, JADE executa automaticamente os comportamentos, a partir do que foi primeiramente declarado, usando um escalonamento circular não-preemptivo.

As estruturas internas dos agentes não são especificadas pela FIPA. Desta forma, a interoperabilidade entre os agentes é garantida através de estruturas especificadas para os comportamentos externos dos agentes. Ou seja, através de protocolos e linguagens de conteúdo, definidos em conformidade com a FIPA [20].

A Figura 14 apresenta a descrição de uma arquitetura interna para um agente genérico em JADE. Segundo [61], cada agente tem uma fila privativa de mensagens ACL, podendo decidir quando e quais mensagens recebidas ele irá tratar. O subsistema de comunicação de JADE foi projetado de forma a alcançar um menor custo possível na passagem de mensagens. Se uma mensagem é enviada e o subsistema não consegue encontrar o destinatário, ele a retorna para ser gerenciada pelo AMS. Associado ao agente tem-se um escalonador de comportamentos e um gerenciador do ciclo de vida do agente. O fato de cada agente em JADE possuir o seu próprio gerenciador permite que ele seja mais autônomo, uma vez que controla completamente sua *thread* de execução. Os recursos associados aos agentes e que dependem de suas aplicações são as crenças que ele possui para tomar as decisões e a capacidade de atuação do mesmo no contexto ao qual está inserido.

De acordo com [61], um agente genérico em JADE possui estruturas dependentes de cada aplicação para armazenar recursos. Cada agente possui também um conjunto de comportamentos ativos associados que representam as tarefas, funcionalidades ou serviços que eles podem realizar. Esses comportamentos podem ser executados concorrentemente, visto que o modelo computacional de um agente em JADE é multitarefa.

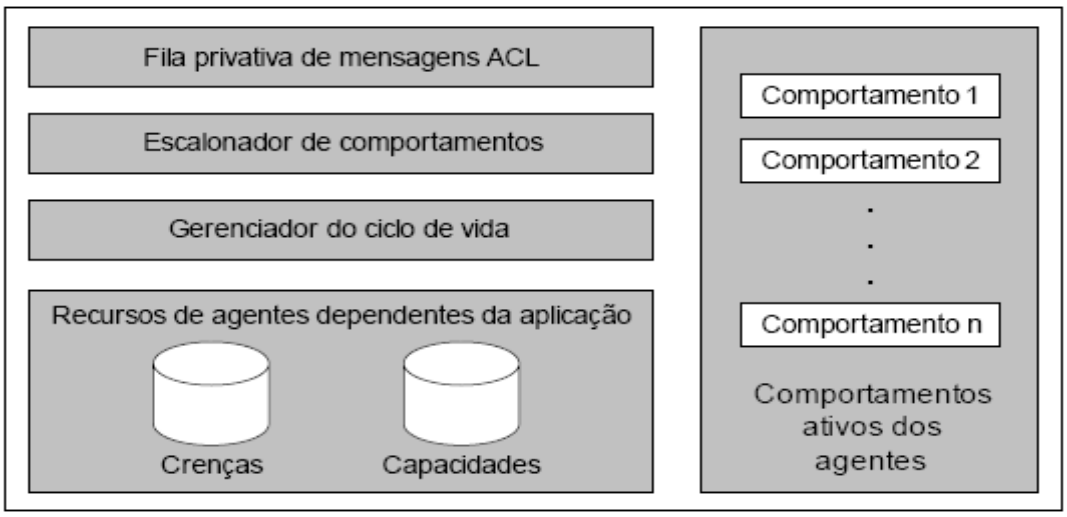


Figura 14 - Arquitetura interna de um agente genérico em JADE [61]

5.4 O Ciclo de Vida de um agente

De acordo com a especificação da FIPA para ciclo de vida de uma plataforma de agentes [20], um agente pode estar em um dos seis estados: *desconhecido*, *ativo*, *suspenso*, *iniciado*, *em trânsito* ou *em espera*. Existem dois estados, *copiado* e *migrado*, que são utilizados internamente pelo JADE. Sua representação em JADE é feita através de constantes estáticas da classe *Agent*. As constantes são as seguintes, segundo [61]:

- ***AP_INITIATED* (INICIADO)**: o objeto da classe *Agent* foi instanciado, mas o mesmo ainda não solicitou registro no AMS, não possuindo ainda nome ou endereço. Nesse estado, não consegue se comunicar com outros agentes;
- ***AP_ACTIVE* (ATIVO)**: o objeto da classe *Agent* foi registrado no AMS. Possui um nome formal e um endereço, além de ter acesso a todas as funcionalidades do JADE;
- ***AP_SUSPENDED* (SUSPENSO)**: o objeto da classe *Agent* está suspenso. Sua *thread* interna está suspensa e nenhum comportamento está sendo executado;
- ***AP_WAITING* (EM ESPERA)**: o objeto da classe *Agent* está bloqueado. Sua *thread* interna está em estado de espera e somente será reativada quando alguma condição vier a ser atendida;
- ***AP_DELETED* (ENCERRADO)**: o agente foi encerrado, ou seja, removido da plataforma. Sua *thread* interna terminou a execução e o agente não possui mais registro no AMS;
- ***AP_TRANSIT* (EM TRÂNSITO)**: próprio de agentes móveis. Ele entra nesse estado quando estiver migrando para outra plataforma. O sistema continua a armazenar as mensagens para o agente em uma fila, mas estas mensagens serão redirecionadas para sua nova localização;
- ***AP_COPY* (COPIADO)**: JADE utiliza internamente este estado para indicar os agentes que foram clonados. Pode representar, na prática, outro estado também, como *em espera*, por exemplo;

- ***AP_GONE* (MIGRADO)**: usado internamente por JADE para indicar que um agente móvel migrou para uma localização distinta e já se encontra em uma situação estável. Assim como o estado anterior, um agente *migrado* pode também estar em outro estado, como o estado *em espera*.

A Figura 15 apresenta o ciclo de vida de um agente, segundo o padrão FIPA. Uma vez iniciado, o agente migra para o estado de ativo. As demais transições de estado são sempre entre o estado de ativo e o estado a ser migrado. O estado em espera é um estado ativo, porém sem estar sendo executado no momento.

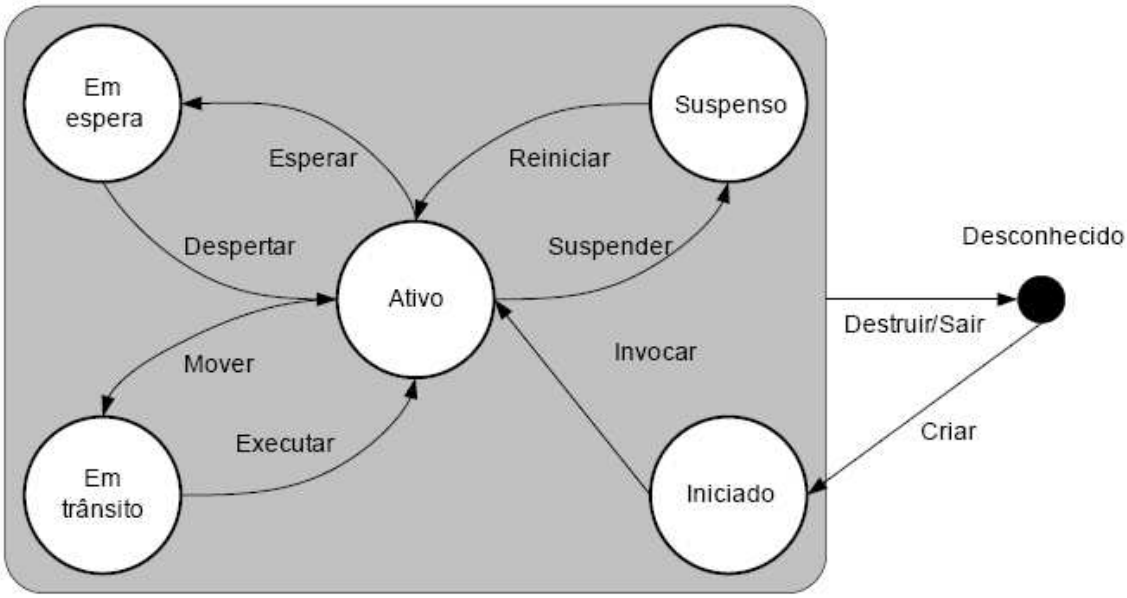


Figura 15 - Ciclo de vida de um agente definido pela FIPA [4]

A troca de estados dos agentes, feita pelo JADE, ocorre através de chamadas aos seguintes métodos fornecidos pela classe *Agent*, segundo [4]:

- ***public void doActivate ()***: realiza a transição do estado *suspenso* para os estados *ativo* ou *em espera*. Este último refere-se ao estado no qual o agente estava quando o método ***doSuspend()*** foi executado. Esse método, chamado pelo JADE, reinicia a execução do agente. Somente é indicado para agentes que estejam no estado de *suspenso*;
- ***public void doSuspend ()***: realiza a transição do estado *ativo* ou *em espera* para o estado de *suspenso*. O estado original do agente é armazenado e será

restaurado pela chamada ao método ***doActivate()***. Pode ser chamado pelo JADE ou pelo próprio agente, que ao fazer isso interrompe todas suas atividades. Mensagens que chegam para um agente suspenso são armazenadas pelo JADE e entregues tão logo este reinicie suas atividades;

- ***public void doWait ()***: realiza a transição do estado *ativo* para o estado *em espera*. Pode ser chamado pelo JADE ou pelo próprio agente, que, ao fazer tal ação, bloqueia-se, interrompendo todas as suas atividades até que alguns eventos ocorram. Um agente em estado de *espera* sai desse estado quando método ***doWake ()*** é invocado ou quando alguma mensagem é recebida;
- ***public void doWake ()***: executa a transição do estado *em espera* para o estado *ativo*. Pode ser chamado pelo JADE e causa o reinício da execução do agente;
- ***public void doDelete ()***: executa a transição dos estados *ativo*, *suspenso* ou *em espera* para o estado *migrado*. Tal ação provoca a destruição do agente e pode ser chamado pelo JADE ou pelo próprio agente;
- ***public void doMove* (*Location* destino)**: realiza a transição do estado *ativo* para o estado *em trânsito*. Suporta a mobilidade de agentes e é chamado pelo JADE ou pelo próprio agente para iniciar o processo de migração;
- ***public void doClone* (*Location* destino, *java.lang.String* nomeClone)**: realiza a transição do estado *ativo* para o estado *copiado*. Dá suporte à mobilidade de agentes e pode ser chamado pelo JADE ou pelo próprio agente para iniciar o processo de clonagem.

Os agentes em JADE somente conseguem executar seus comportamentos quando estiverem no estado ativo, [61].

5.5 Comportamentos de agentes

Os comportamentos que um agente desempenha dentro de um SMA são fundamentais para se alcançar o objetivo final do sistema, de acordo com [61]. JADE contém comportamentos específicos para a maioria das tarefas comuns na programação de agentes e também para construção de tarefas mais complexas. No

desenvolvimento orientado a agentes com JADE, o agente deve obrigatoriamente ser uma instância direta ou indireta da classe *Agent* e implementar tarefas específicas, através da implementação de um ou mais comportamentos através das subclasses de *Behaviour*. Estas deverão ser instanciadas e adicionadas ao agente.

A estrutura dos comportamentos de JADE é suportada por um escalonador, interno à superclasse *Agent* e transparente para o desenvolvedor. Este gerencia o escalonamento desses comportamentos usando a política de escalonamento *Round-Robin* não-preemptivo [4]. Ou seja, cada comportamento é executado por vez e por um determinado tempo. Encerrado esse tempo, o comportamento pode voltar para o fim da fila, caso não tenha acabado de executar a tarefa a que se propõe. Não há prioridade entre os comportamentos, isto é, cada comportamento adicionado deve ser encaminhado ao fim da fila e todos terão a mesma fatia de tempo para execução.

Com relação à concorrência, segundo [4], um agente é um objeto ativo com uma *thread* de controle interna. JADE usa o modelo de uma *thread* por agente e não uma *thread* por tarefa. Tal modelo tem como objetivo manter um número reduzido de *threads* necessárias para executar a plataforma de agentes.

5.5.1 Comportamento em JADE - Classe *Behaviour*

Behaviour é uma classe abstrata de JADE existente no pacote *jade.core.behaviours* e que pode ser usada para modelar uma tarefa genérica que o agente deve executar, de acordo com [4]. Tem como finalidade modelar e fornecer a estrutura de comportamentos que podem ser implementados para execução de agentes em JADE.

Os dois métodos principais da classe *Behaviour* são *action* e *done*. Ambos são abstratos. O método ***action()*** é responsável por realizar ações que serão desempenhadas pelo agente através deste comportamento. O método ***done()*** tem a função de informar ao escalonador do agente se o comportamento executado terminou ou não. Caso o comportamento em questão não tenha sido encerrado, o método ***action()*** é novamente executado.

Os principais métodos da classe Behaviour, segundo [61], são:

- ***onEnd()***: invocado apenas após o comportamento ter sido encerrado;
- ***onStart()***: executado somente antes do início da execução do comportamento do agente;
- ***void block()***: utilizado para bloquear um comportamento. Pode possuir parâmetro referente ao tempo de bloqueio em milissegundos. Permite o desbloqueio do agente caso uma mensagem seja recebida;
- ***void restart()***: reinicia um comportamento que esteja bloqueado;
- ***void reset()***: restaura o estado inicial de um comportamento;
- ***boolean isRunnable()*:** verifica se o comportamento está bloqueado.

A hierarquia das classes derivadas de *Behaviour* é apresentada na Figura 16.

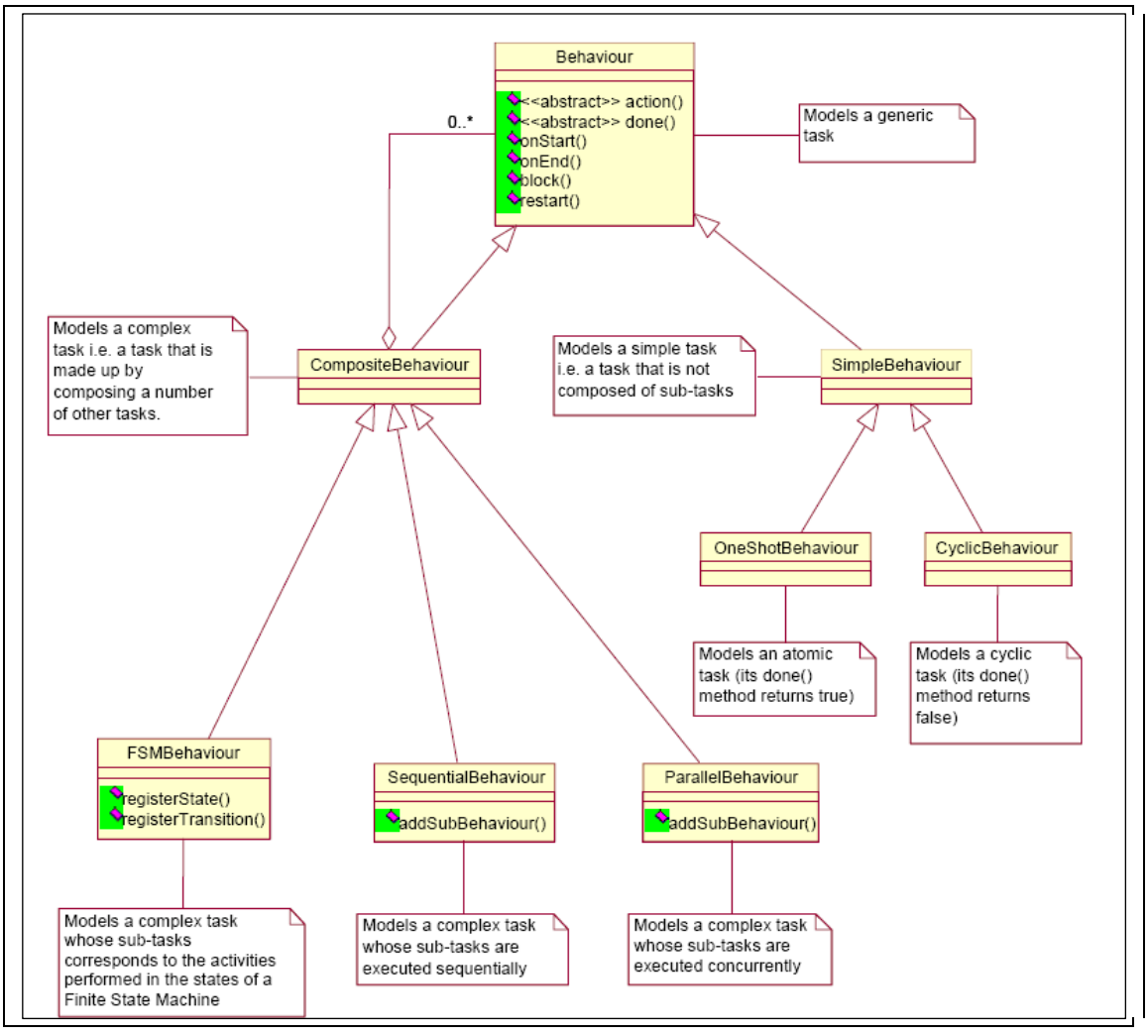


Figura 16 - Hierarquia de comportamentos em JADE [4]

A classe *SimpleBehaviour*, de acordo com [4], é uma classe abstrata que deve ser utilizada para modelar comportamentos únicos, ou seja, que não podem ser interrompidos. Possui quatro classes herdadas disponibilizadas pelo JADE que são:

- ***jade.core.behaviours.CyclicBehaviour***: define um comportamento simples que deve ser executado sempre. Classe abstrata que pode ser herdada para criação de comportamentos cuja execução deve ser contínua;
- ***jade.core.behaviours.OneShotBehaviour***: modela uma tarefa que deve ser executada apenas uma vez e não pode ser bloqueada. Também pode ser herdada para a criação de comportamentos. Tem como subclasse a classe ***SenderBehaviour***;
    - ***jade.core.behaviours.SenderBehaviour***: comportamento utilizado para envio de mensagens ACL. Encapsula o método ***send()*** como uma operação atômica. Após enviar uma mensagem ACL é finalizado;
- ***jade.core.behaviours.TickerBehaviour***: embora não representada na Figura 16, por não ser um dos tipos de comportamento definido nas primeiras versões do JADE, também é subtipo de *SimpleBehaviour*. Modela comportamentos que necessitam ser executados periodicamente para realizar ações específicas. Tem como objetivo executar, periodicamente, através do método ***onTick()***, um trecho de código definido pelo desenvolvedor em uma determinada freqüência de repetições;
- ***jade.core.behaviours.WakerBehaviour***: embora também não representada na Figura 16, por não ser um dos tipos de comportamento definido nas primeiras versões do JADE, também é subtipo de *SimpleBehaviour*. Modela um comportamento simples a ser executado depois de expirado um tempo definido. A ação deve ser inserida no método ***handleElapsedTimeout()*** o qual é chamado sempre que o intervalo de tempo for transcorrido.

Segundo [4], a classe *CompositeBehaviour* é uma classe abstrata utilizada para modelar comportamentos que caracterizam ações complexas, ou seja, tarefas que possuam sub-tarefas. Controla internamente seus sub-comportamentos e seus intervalos de execução. Possui três subclasses disponibilizadas pelo JADE:

- ***jade.core.behaviours.FSMBehaviour***: comportamento composto que escalona seus sub-comportamentos de acordo com uma máquina de estados finitos definida pelo desenvolvedor, na qual cada sub-comportamento representa um estado. Esta classe possui métodos para registrar estados e transições que definem como será realizado o escalonamento desses sub-comportamentos. Um comportamento *FSMBehaviour* deve ser definido da seguinte forma:
    - Determina-se um único comportamento como estado inicial, através do método ***registerFirstState()***, que tem como parâmetros o comportamento e o nome do estado;
    - Define-se um ou mais comportamentos como estados finais, através do método ***registerLastState()***;
    - Define-se um ou mais comportamentos como estados intermediários através do método ***registerState()***;
    - Para cada estado, são definidas as transições para os outros estados através do método ***registerTransition()***.
- ***jade.core.behaviours.ParallelBehaviour***: comportamento composto que escalona seus sub-comportamentos de forma concorrente. É finalizado quando uma condição relacionada à execução de seus sub-comportamentos for atendida. Essas condições são determinadas através de um parâmetro no construtor da classe. Este pode ser: um valor inteiro que indicará o número de sub-comportamentos que necessitam ser finalizados para que a condição seja atendida, ou uma constante que pode ser ***WHEN_ALL*** ou ***WHEN_ANY***. Estas são empregadas para definir que todos os comportamentos ou algum dos comportamentos, respectivamente, devem ser encerrados para que a execução da instância da classe ***ParallelBehaviour*** seja finalizada;
- ***jade.core.behaviours.SequentialBehaviour***:comportamento composto que escalona seus sub-comportamentos de modo seqüencial. Dessa forma, um comportamento que tenha sido decomposto apenas pode ser encerrado quando todos os seus sub-comportamentos tiverem finalizado suas respectivas execuções. Para adicionar sub-comportamentos, utiliza-ze o método ***addSubBehaviour()***.

5.6 Mensagens em JADE

De acordo com [61], sistemas baseados em agentes possuem intrinsecamente uma arquitetura *peer-to-peer*, onde cada agente é um *software* sendo executado em um ponto da rede e que necessita se comunicar com os demais a quem provê ou consome serviços. O modelo de comunicação em JADE é fundamentado em três características principais, segundo [3]:

- Agentes são entidades ativas e fracamente acopladas. O grau de dependência entre os agentes é minimizado, uma vez que a plataforma permite que o agente escolha os destinatários das mensagens que deseja ou necessita enviar. Cada agente é responsável por decidir que mensagens ele irá interpretar ou descartar, bem como controlar seu ciclo de execução para aguardar respostas a mensagens que enviou.
- Agentes realizam ações e a comunicação é considerada um tipo de ação. Sendo assim, os agentes tanto podem realizar ações físicas quanto ações comunicativas. Relacionado a essas ações, pode-se fazer o planejamento da comunicação. Este planejamento necessita que se definam efeitos e pré-condições associados a cada comunicação possível.
- A comunicação possui um significado semântico. Quando um agente recebe uma mensagem, ele deve ser capaz de compreender o significado desta ação e qual a intenção do agente que enviou a mensagem. Desta forma, torna-se necessário a existência de uma semântica e de um padrão de comunicação.

JADE é compatível com o padrão da FIPA definido para resolver a necessidade de comunicação e sociabilidade dos agentes, a Linguagem de Comunicação de Agentes (ACL – *Agent Communication Language*) [19]. A Figura 17 apresenta os componentes do modelo de comunicação FIPA, no qual há um nível (Envelope) que encapsula a mensagem ACL permitindo que esta seja transmitida pela rede por quaisquer protocolos de transporte. Esta mesma mensagem é associada a uma linguagem de conteúdo que pode ser especializada em símbolos que representarão o modelo de domínio a ser explorado pelos agentes, isto é, a ontologia que definirá o padrão do conteúdo das mensagens trocadas entre os agentes da plataforma.

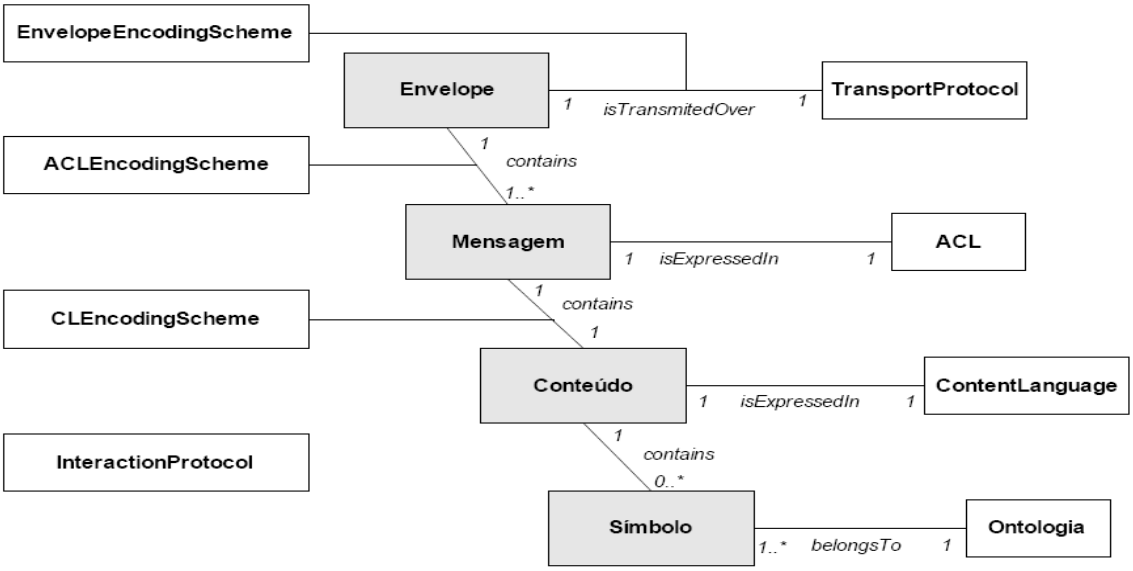


Figura 17 - Componentes do modelo de comunicação da FIPA [5]

Diversos padrões de comunicação comuns foram definidos pela FIPA. Esses padrões, denominados protocolos de interação, são utilizados para se conseguir realizar tarefas comuns, como delegar uma ação.

Na comunicação entre agentes na plataforma JADE, os agentes se comunicam enviando mensagens uns para os outros através do envio assíncrono de mensagens ACL. Ao receberem estas mensagens, os agentes as interpretam como atos de fala (*speech acts*). Como o padrão FIPA define que somente mensagens ACL são transportadas na plataforma e não define nenhum padrão para o conteúdo dessas mensagens, JADE as codifica de forma transparente para o desenvolvedor.

A classe *ACLMessage* fornecida pelo JADE representa as mensagens seguindo as especificações da FIPA [19]. A Figura 18 ilustra o um exemplo de uma mensagem FIPA-ACL trocada entre agentes JADE.

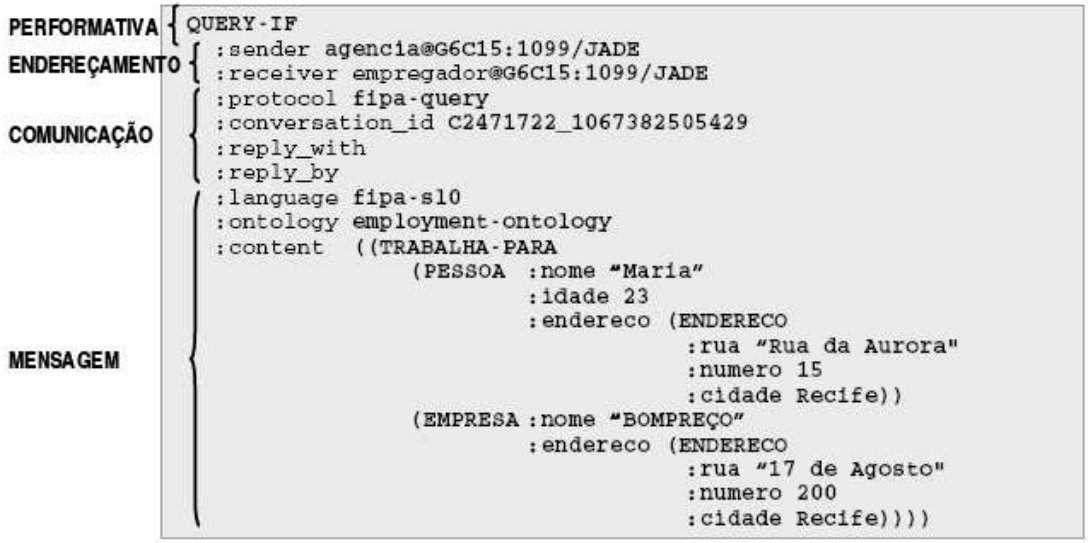


Figura 18 - Exemplo de mensagem para o padrão ACL [61]

As performativas de FIPA para os atos comunicativos (*communicative acts*) são identificadas na mensagem ACL por constantes, como ***ACCEPT_PROPOSAL***, ***AGREE***, ***CANCEL***, ***CFP***, ***CONFIRM***, entre outras [61].

De acordo com [4] e [61], para que um agente consiga enviar mensagens a outros agentes, o desenvolvedor deve criar uma instância da classe ***ACLMessage***, preencher a mensagem com as informações necessárias em cada campo correspondente e invocar o método ***send()*** da classe *Agent*. Já quando o agente desejar receber mensagens, deve chamar o método ***receive()*** ou ***blockingReceive()*** da classe *Agent*. Outra forma de enviar ou receber mensagens em JADE é através do uso das classes ***SenderBehaviour*** e ***ReceiveBehaviour***. O uso dessas classes faz com que as trocas de mensagens possam ser escalonadas como atividades independentes. Os principais métodos da classe *ACLMessage*, segundo [4], são:

- ***public void addReceiver*(AID idAgente)**: adiciona o AID de um agente como receptor ou destinatário da mensagem ACL;
- ***public void removeReceiver*(AID idAgente)**: remove um determinado AID da lista de agentes receptores;
- ***public ACLMessage createReply()***: cria uma nova mensagem ACL para ser resposta à uma outra;

- ***public static java.lang.String[] getAllPerformativeNames()***: tem como saída uma coleção de *strings* com todas as performativas que podem ser utilizadas em JADE;
- ***public Iterator getAllReceiver()***: possui como saída um objeto *iterator* contendo todos os AIDs dos agentes receptores da mensagem;
- ***public java.lang.String getContent()***: obtém uma *string* com o conteúdo da mensagem;
- ***public void setContent*(*java.lang.String* conteúdo)**: define o conteúdo da mensagem a ser enviada.
- ***public void setOntology*(*java.lang.String* ontologia)**: define a ontologia que a mensagem irá seguir. Caso haja um repositório de ontologias, a aplicação deverá conhecer cada um dos modelos ontológicos disponíveis, de modo que ao registrar uma determinada ontologia para a mensagem, a aplicação manipule de forma adequada à mesma, as classes Conceitos envolvidas;
- ***public void setPerformative*(int performativa)**: define a constante recebida como parâmetro para ser a performativa da mensagem ACL em questão.

5.7 Protocolos de interação

De acordo com [4], o padrão FIPA-ACL [19] especifica que cada mensagem representa um ato comunicativo (*communicative act*). Para cada ato comunicativo, a FIPA estabelece as condições que precisam ser verdadeiras antes do agente executar uma ação e o efeito esperado dessa ação. Também é especificado pelo padrão que o agente receptor pode decidir ignorar a mensagem recebida.

Desta forma, como as condições definidas na comunicação devem ser verdadeiras antes que a solicitação contida na mensagem seja atendida, é importante que os agentes façam uso de protocolos de interação. A utilização de protocolos de interação, iniciados por um determinado agente, permite que este agente possa verificar se a consequência esperada para o ato comunicativo, executado por ele, foi atingida ou não.

A FIPA específica um conjunto padronizado de protocolos de interação. Segundo [4], para cada conversação entre os agentes, JADE define dois papéis, o *Initiator* e o *Responder*. *Initiator* representa o agente que iniciou a conversação, enquanto que *Responder* representa o agente que está em conversação após ter sido acionado para tal por algum outro agente. JADE provê classes, existentes no pacote *jade.proto*, para o comportamento de ambos os perfis, de acordo com a maioria dos protocolos de interação da FIPA.

De acordo com [4], todos os comportamentos específicos para *Initiators*, ao serem encerrados, são removidos da fila de tarefas de agentes, assim que estas alcancem qualquer estado final do protocolo de interação. Todos os *Initiators* incluem um número de métodos ***reset()*** com os argumentos apropriados e podem manipular vários *Responders* ao mesmo tempo, com exceção da classe ***FipaRequestInitiatorBehaviour***. Já os *Responders* são cíclicos e são reagendados assim que alcançam qualquer estado final do protocolo de interação, o que permite ao desenvolvedor limitar o número máximo de *Responders* que o agente pode executar em paralelo, através da adição de sub-comportamentos dentro do método ***setup()*** do agente.

5.7.1 *AchieveRE (Achieve Rational Effect)*

Os protocolos de interação especificados pela FIPA, como *FIPA-Request*, *FIPA-Query*, *FIPA-Propose*, *FIPA-Request-When*, *FIPA-Recruiting*, *FIPA-Brokering*, permitem ao agente responsável por ser o *Initiator* verificar o efeito esperado do ato de comunicação realizado por este agente. Para tanto, JADE provê as classes ***AchieveREInitiator*** e ***AchieveREResponder***.

A Figura 19 apresenta a estrutura abstrata do protocolo de interação utilizada pelo *framework* JADE através dos agentes ***AchieveREInitiator*** e ***AchieveREResponder***. O agente *Initiator* envia uma mensagem. O *Responder* pode responder com a indicação de que não entendeu a mensagem que recebeu (*NOT-UNDERSTOOD*), com uma mensagem de recusa (*REFUSE*) à solicitação ou com uma mensagem de aceitação (*AGREE*) ao ato comunicativo. O *Responder* executa a ação e responde com um informe (*INFORM*) do resultado da ação ou com

uma falha (*FAILURE*), caso algo de errado tenha acontecido. A mensagem de aceitação pode ser omitida e o agente enviar diretamente mensagens de informe ou de falha.

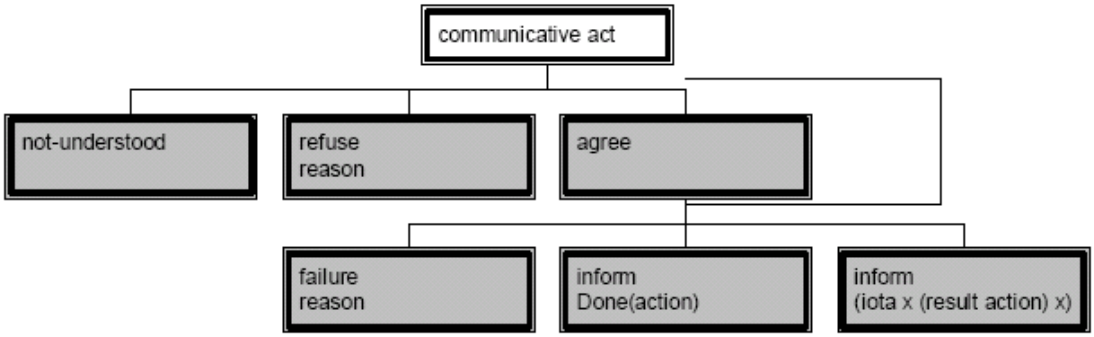


Figura 19 - Estrutura homogênea dos protocolos de interação em JADE [4]

JADE provê para implementação dos diferentes papéis as seguintes classes, de acordo com [61] e [4]:

- ***jade.proto.AchieveREInitiator***: utilizada para implementação de um papel de *Initiator* de conversação, fazendo uso de protocolos de comunicação. Instâncias desta classe são construídas, passando como argumento do seu construtor a mensagem usada para iniciar o protocolo;
- ***jade.proto.SimpleAchieveREInitiator***: trata-se de uma implementação simplificada do papel do *Initiator*. A principal diferença com relação à classe anterior é que esta versão do protocolo não permite ao programador registrar comportamentos específicos da aplicação para manipular os estados do protocolo;
- ***jade.proto.AchieveREResponder***: implementa o papel de *Responder*. De acordo com o modelo de mensagem fornecido como argumento para o construtor, define-se qual mensagem ACL recebida será tratada. O método ***createMessageTemplate*** pode ser usado para criar um modelo de mensagem para um determinado protocolo de interação;
- ***jade.proto.SimpleAchieveREResponder***: implementa de forma mais simplificada a classe ***AchiveREResponder***. A principal diferença é que esta versão não permite ao programador registrar outros comportamentos como manipuladores dos estados do protocolo.

5.7.2 *FIPA-Contract-Net*

De acordo com [61], trata-se de um dos protocolos de interação mais conhecidos no paradigma de orientação a agentes. Permite ao agente *Initiator* enviar uma chamada de propostas (*Call for Proposal* – CFP) para um conjunto de *Responders*. Estes, por sua vez, podem responder enviando uma mensagem de proposta (*PROPOSE*), incluindo as pré-condições que desejam para a ação solicitada. Podem também enviar uma mensagem de recusa (*REFUSE*) ou de não compreensão (*NOT-UNDERSTOOD*), para o caso de problemas de comunicação. Ao receberem a resposta, o agente *Initiator* avalia todas as propostas recebidas e aceita a que ele preferir. Pode também não aceitar nenhuma delas. Uma vez que ele aceitou uma delas, através da mensagem de aceite (*ACCEPT-PROPOSAL*), os *Responders* então responderão à ação solicitada com informe (*INFORM*) ou falha (*FAILURE*).

A Figura 20 ilustra uma visão do funcionamento deste protocolo. Nesta, o agente *Initiator* envia uma chamada de proposta com as respectivas pré-condições (*preconditions1*) associadas. O agente *Responder* pode responder que não entendeu (*not-understood*) a chamada, pode recusar a proposta (*refuse*) apresentando a razão (*reason*) para tal ou pode concordar em receber a proposta de fato (*propose*) e suas pré-condições (*preconditions2*). Neste momento, termina-se o prazo para as propostas. Ao receber a proposta, o agente *Responder* pode aceitá-la (*accept-proposal*) ou rejeitá-la (*reject-proposal*), apresentando a razão (*reason*) da rejeição. Ao aceitar a proposta, o agente inicia o processo de interação, podendo responder com:

- Um *inform*, que indica a completude da ação solicitada;
- Uma falha (*failure*), apresentando a razão (*reason*) que a causou;
- Um cancelamento (*cancel*), indicando a razão (*reason*) pela qual o agente *Responder* cancelou o contrato e interrompeu o processo de interação.

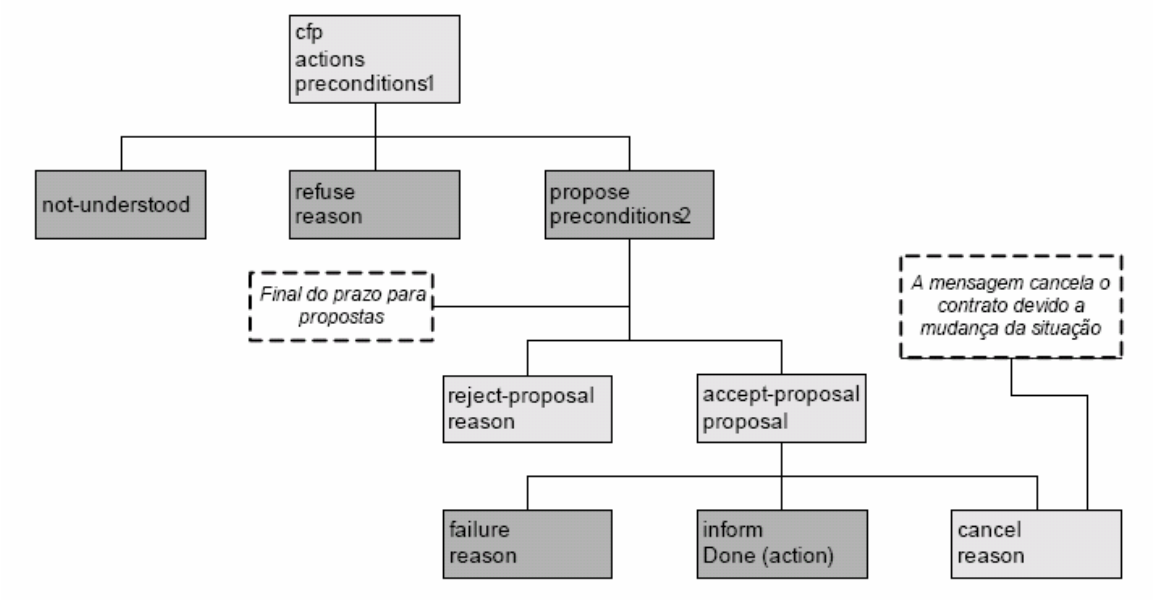


Figura 20 - Protocolo de interação *FIPA-Contract-Net* [61]

JADE oferece para implementação do protocolo *FIPA-Contract-Net* as seguintes classes:

- ***jade.proto.ContractNetInitiator***: implementa o comportamento do agente iniciando o protocolo, ou seja, o envio da mensagem de CFP;
- ***jade.proto.ContractNetResponder***: implementa o comportamento de responder a uma chamada de propostas.

5.8 Considerações Finais

Este capítulo apresentou a estrutura geral da plataforma de desenvolvimento de agentes JADE. A escolha de JADE justifica-se pelos seguintes motivos:

- JADE é um padrão aberto, considerado *software* livre, gratuito e sem data de expiração;
- JADE trabalha com Java, uma linguagem de programação portável e também um padrão aberto;

- JADE possui exigências de *hardware* mínimas e suas especificações são compatíveis com os padrões da FIPA [20];
- A implementação da plataforma de agentes já é provida por JADE, sem que o desenvolvedor tenha que se preocupar com tais conceitos;
- JADE trabalha com ontologia e fornece os mecanismos de transporte e verificação de mensagens trocadas entre os agentes construídos;
- Apesar de ser um padrão aberto, JADE provê suporte a usuários, através de fóruns de discussão, lista de problemas mais frequentes e documentação voltada para administração e programação usando este *framework*.

As principais desvantagens do JADE estão relacionadas ao uso de chamadas remotas para comunicação entre os agentes que pertençam a containers distintos e o fato de apresentar problemas ao trabalhar com a arquitetura web, mais especificamente com *servlets*. Um *servlet* é o elemento responsável por atender a uma requisição feita a uma aplicação que esteja desenvolvida na plataforma web.

A utilização de chamadas remotas por meio do RMI do Java, se usado de forma intensa, agrega um *overhead* de comunicação ao sistema podendo degradar o desempenho do mesmo, o que pode ser um eventual problema de acordo com as características que o sistema possua.

A dificuldade no trabalho com *servlets* está no fato de apresentar problemas relacionados à comunicação entre o servidor de aplicação, que processa o *servlet*, e a plataforma JADE. Porém este fato está sendo trabalhado pela comunidade JADE e as novas versões já devem apresentar evoluções neste assunto.

ONTOLOGIA: DE CONCEITOS AO MODELO PROPOSTO

Controlar e classificar informações são tarefas cada vez mais relevantes no mundo organizacional, principalmente em um cenário no qual tais informações são heterogêneas e distribuídas. De acordo com [44], organizar estes dados de modo a torná-los compartilháveis e acessíveis de maneira eficiente é uma meta de diversas áreas de trabalho e a criação de ontologias é um caminho para a busca desta interoperabilidade.

Ontologia pode ser definida, segundo [70] [71] citando [25], [30] e [31], como sendo uma especificação formal e explícita de uma conceituação, de modo a torná-la única dentro de um mesmo domínio de conhecimento, isto é, dentro do contexto estudado. Conforme [25], uma ontologia é explicita porque define conceitos, propriedades, relacionamentos, funções, axiomas e restrições que a compõem. É formal, porque pode ser compreendida e interpretada computacionalmente. É uma conceituação, porque representa um modelo abstrato e simplificado de uma realidade, composto por um domínio do conhecimento e um conjunto de relações envolvendo as entidades que formam o contexto modelado [30].

Para [48], uma ontologia define um vocabulário comum para pesquisadores que necessitam compartilhar informações dentro de um mesmo domínio de conhecimento, o que inclui o mapeamento dos conceitos básicos

existentes e os relacionamentos entre eles. Segundo [48], as principais razões para justificar a utilização de ontologias são:

- Possuir um entendimento único e comum da estrutura da informação compartilhada por pessoas ou agentes de *software*;Prover a possibilidade de reuso do domínio mapeado;
- Assumir de forma explícita afirmações sobre o domínio;
- Separar o domínio do conhecimento do domínio operacional;
- Analisar o domínio do conhecimento modelado.

De acordo com [75], o processo de construção de uma ontologia inicia-se pela definição do contexto do domínio a ser modelado e se segue pelo uso de uma determinada representação e linguagem para se gerar o modelo. A conclusão do mesmo se dará, segundo [75], através do uso de mapeamentos para conectar os conceitos às fontes de informação de acordo com uma metodologia focada na engenharia de software voltada para a integração e reutilização de ontologias. O objetivo final da aplicação de uma metodologia será principalmente na definição dos requisitos que esta ontologia deve satisfazer, bem como na captura dos conceitos, propriedades, restrições e axiomas que irão compor a ontologia construída, de maneira que esta estabeleça um vocabulário único dentro do domínio de conhecimento em que está inserida.

No contexto de gerência de redes, para se atingir um nível de qualidade de serviço estabelecido, por exemplo, precisam-se, na maioria das vezes, ter um controle do seu ambiente para se evitar problemas inesperados, como rotas bloqueadas devido a um equipamento parado. Este controle é proporcionado pelo conhecimento da estrutura do ambiente de rede no qual se está inserido. Tal conhecimento está relacionado com a gerência de configuração e o processo de *Discovery*, no qual são identificados os elementos que compõem a rede gerenciada.

Ao realizar o *Discovery* da rede, busca-se ter um domínio de ativos e serviços providos pela mesma, de forma a se criar um modelo que permita o mapeamento de tais informações. Devido a constantes mudanças na estrutura da rede e a necessidade de aproveitamento das informações evolutivas, aliada à necessidade crescente de se conhecer de modo homogêneo o ambiente

organizacional como um todo, independente da tecnologia e do protocolo de gerenciamento utilizado para se buscar estas informações, o uso de ontologia surge como uma alternativa natural para este problema [72] [73].

6.1 Análise de Domínio e Ontologias

Um dos principais objetivos ao se fazer uso de ontologias está na capacidade de reuso que um modelo ontológico pode proporcionar [37]. Ao se tratar de reutilização devem-se estabelecer os níveis em que ela pode ser aplicada. Quanto mais alto for o nível de abstração onde possa ocorrer, maior a economia gerada.

É possível realizar reutilização de domínios de informação. Tal abstração favoreceria a facilidade de comunicação. Conforme [33], modelos de domínio necessitam de uma forma de representar seus conceitos, os relacionamentos e restrições que moldam certa área de conhecimento. É nesse momento que o uso de ontologias se torna importante, pois vêm a definir um linguajar formal de representação para um dado contexto [33].

De acordo com [33], a análise de domínio visa identificar as informações relevantes para um dado contexto do mundo real e relacioná-las de forma a garantir a coesão de conceitos deste contexto. Nesta situação, uma série de passos podem ser definidos para suprir a necessidade de descrição de um domínio. Deve-se começar por um planejamento, no qual a viabilidade da análise de domínio é posta em prova, indicando as vantagens e desvantagens de tal atividade. Após a aprovação da análise, passa-se à aquisição de dados. Neste momento, torna-se importante não só dar atenção à bibliografia especializada, mas principalmente aos especialistas que convivem no contexto do domínio estudado e possuem informações muito mais difíceis de obter. Por fim, os dados são analisados em prol da formulação de uma modelagem que permita a identificação e a validação conceitual dos itens identificados como membros do domínio da informação.

Um dos mais importantes resultados do processo de análise de domínios é a definição de modelos léxicos que tentam manter as informações devidamente classificadas visando reduzir ambigüidades relativas aos conceitos pertinentes ao

domínio. Ou seja, uma informação ao ser categorizada deve possuir uma semântica única, de modo a minimizar eventuais conceitos com definições muito próximas, passíveis de causar algum tipo de ambigüidade que venha a prejudicar o entendimento do modelo criado para o domínio em questão.

Uma ontologia vem a ser a definição formal de conceitos inter-relacionados que possuem propriedades caracterizantes e são restringidos em um dado contexto. Como se apresentam como uma formalização de nível abstrato e independente de plataformas, linguagens ou mesmo paradigmas, o uso de ontologias para a definição de domínios mais concisos e reaproveitáveis tornou-se uma importante ferramenta. Com origem na Filosofia, o termo ontologia surgiu no contexto de Ciência da Computação inicialmente associado a conceitos de Bancos de Dados e, posteriormente, em estudos de Inteligência Artificial (IA). Desde então, as ontologias deixaram de ser utilizadas com o prisma filosófico e se fixaram nas vantagens principais que sua capacidade taxionológica provê: capacidade de isenção em relação a tecnologias e capacidade de expressão de informações facilmente identificáveis por conhecedores de um dado contexto [33].

Ou seja, as ontologias fornecem a possibilidade de estabelecer comunicação entre entes que conheçam suas definições. Elas mantêm um compromisso com a consistência de um domínio e não com a completude do mesmo. Desta forma, é dada maior ênfase no vocabulário de conceitos, em suas definições e possíveis propriedades, na representação de todas as possíveis relações entre os conceitos e em um conjunto de axiomas formais que restrinjam a interpretação dos conceitos e relações, representando de forma não ambígua o conhecimento do domínio e permitindo a inferência de novos conhecimentos baseados nos pré-existentes.

Há várias classificações diferentes de ontologias de acordo com o conteúdo que pretendem representar. A principal delas, conforme [30], é apresentada na Figura 21 e define quatro categorias de acordo o nível de especialização de cada ontologia. São elas:

- Ontologias Genéricas - Dizem respeito a conceitos muito abrangentes, como tempo, espaço, objeto, normalmente fora da possibilidade de restrição para um contexto em particular. São independentes de um problema específico ou

domínio. Dessa forma, podem ser utilizadas por um grande número de usuários e comunidades científicas.

- Ontologias de Domínio - Mais particulares com relação a contexto, mas suficientemente abrangentes para não implicar em restrições indevidas de informações. Elas representam um domínio de conhecimento específico, como medicina, contabilidade, gerência de redes, entre outros.
- Ontologias de Tarefa - Visam integrar o conhecimento de tarefas a serem realizadas e sua utilização em domínios. Tais ontologias se caracterizam por representar atividades que sejam especializações dos conceitos introduzidos na ontologia genérica.
- Ontologias de Aplicação – Descrevem conceitos dependentes de um domínio particular e das tarefas que compõem este mesmo domínio, tendo deste modo uma relação direta com ambas as categorias (domínio e de tarefa).

Por fim, outra categoria que também tem sua relevância são as ontologias de Representação, que mantém o formalismo representativo de conhecimento [33].

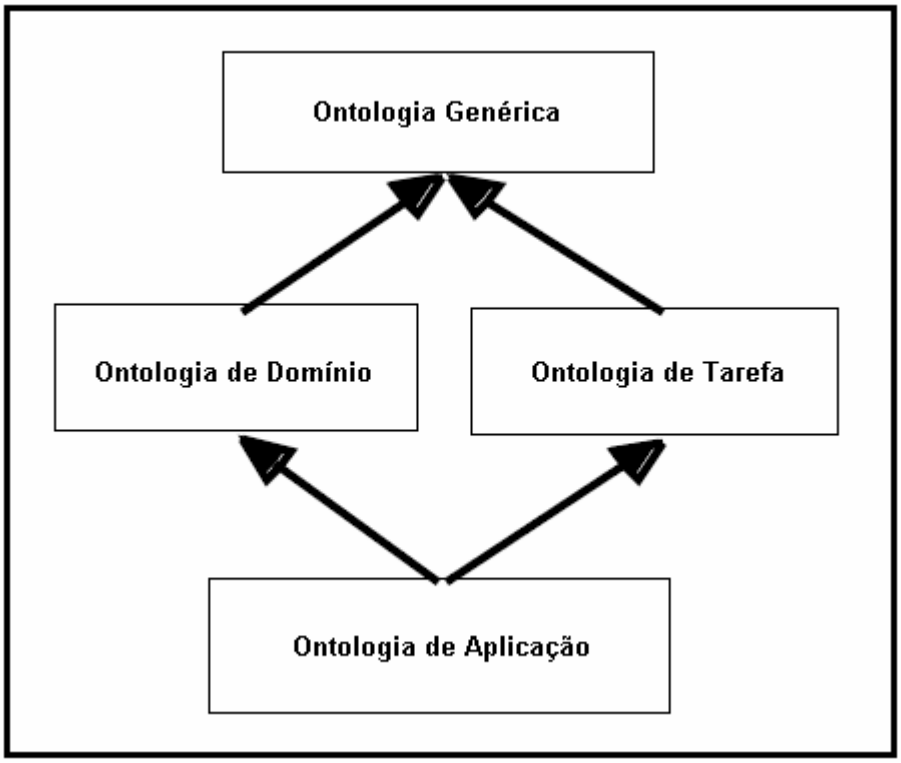


Figura 21 – Classificação de ontologias segundo seu conteúdo [30]

Dentre as benesses do uso de ontologias destacam-se o ganho de comunicação, a formalização que elimina as contradições e inconsistências envolvendo as restrições e uma representação de conhecimento com alto potencial de reuso. Em contrapartida, surgem como problemas o fato de que nenhuma ontologia pode atender a todos os indivíduos envolvidos; pouco enfoque na característica evolutiva das ontologias; a dificuldade de extensão; a interdependência ontológica dificultada por visões diferentes do mundo e a não padronização de uma metodologia de criação.

Por causa deste último problema, várias propostas de construção de ontologias surgiram, tendo como principais pontos em comum o enfoque em atividades que definam um ciclo iterativo que permita uma engenharia voltada para ontologias.

A Figura 22 apresenta uma representação do ciclo de vida de uma ontologia, no qual o domínio analisado é modelado em cenários que irão ser base para a definição dos conceitos que compõem o modelo ontológico. Este, por sua vez, será utilizado pelas aplicações inseridas no contexto modelado.

O ciclo de vida inicia-se pela etapa de criação (*Creating*), na qual são analisados os possíveis cenários e conceitos que envolvem os usuários chave no domínio de conhecimento que será modelado. Posteriormente, segue-se a fase de definição dos conceitos e suas propriedades (*Populating*), na qual fontes de conhecimento a respeito do contexto estudado são utilizadas. As etapas seguintes de validação (*Validating*) e publicação (*Deploying*) abrangem, respectivamente, a verificação do modelo ontológico para analisar se este contempla os requisitos definidos para o mesmo e a disponibilização da ontologia para ser utilizada pelas aplicações e pessoas que componham o domínio em questão. A última fase (*Evolving/Maintaining*) estabelece a análise do modelo implantado, a fim de identificar melhorias e evoluções neste dentro do contexto ao qual pertence.

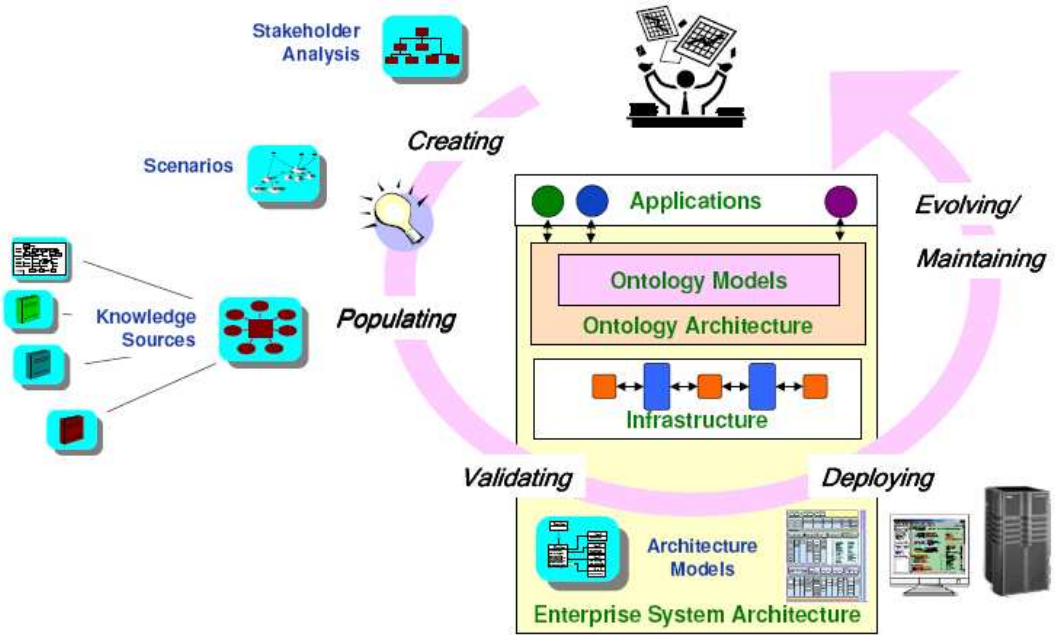


Figura 22 - Ciclo de vida de uma ontologia [48]

6.2 Construção de Ontologias

Projetos como *METHONTOLOGY* [26] [27] [28] [69], TOVE (*Toronto Virtual Enterprise*) [18] [24], *Enterprise Model Approach* [66] [67] [68], KBSI IDEF5 [40] destacam-se como propostas de metodologias voltadas para o desenvolvimento e manutenção de ontologias [39]. De acordo com [33], tais projetos não têm como ponto forte a construção de modelos ontológicos como parte de um processo de engenharia de ontologias.

Desta forma, segundo [13] [14], pode-se estabelecer uma abordagem para construção de ontologias que faz uso de características das metodologias citadas anteriormente e define atividades organizadas em um ciclo interativo. Estas atividades, ilustradas na Figura 23, são descritas a seguir:

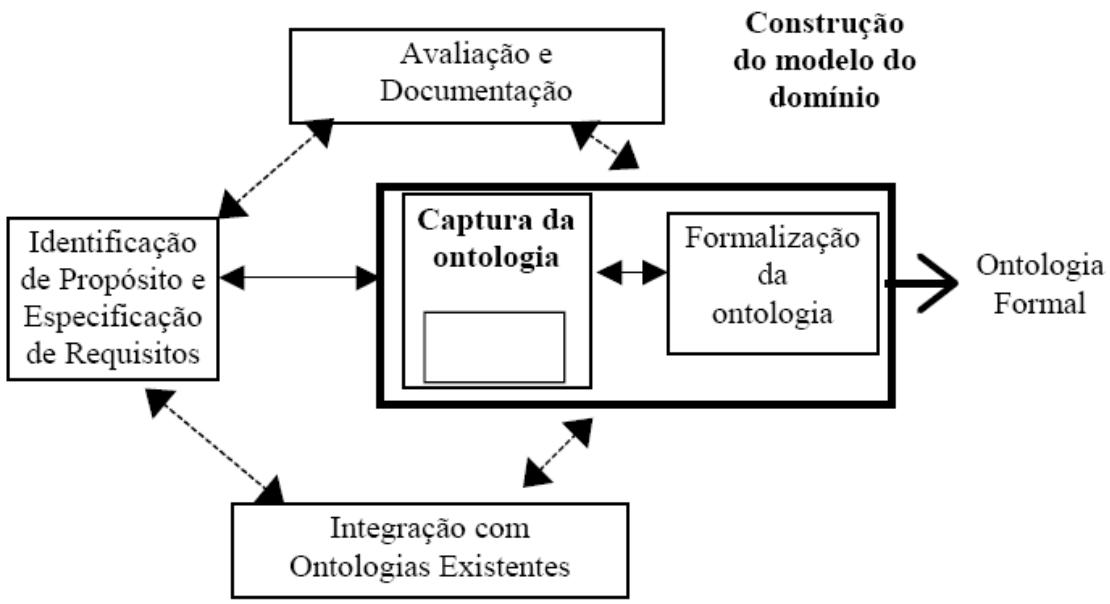


Figura 23 - Etapas do processo de construção de uma ontologia [33]

- **Identificação de Propósito e Especificação de Requisitos** - Esta atividade se caracteriza por delimitar o que é de fato relevante para a ontologia, sua competência. Uma vez definida, devem-se especificar os requisitos da mesma em termos de questões de competência, ou seja, as questões que a ontologia tem obrigação de responder [33] [15].

Nesta fase, o foco é identificar o domínio e o escopo a ser atendido pela ontologia, de modo a responder indagações do tipo quem irá manipular esta ontologia e que tipo de perguntas as informações providas pelo modelo proposto deve ser capaz de responder [48].

De acordo com [33] e [15], é importante também identificar potenciais usuários e eventuais cenários que motivaram o desenvolvimento da ontologia proposta. Os requisitos da ontologia devem ser expressos em termos de questões de competência: as questões que a ontologia deve ser capaz de responder. Segundo [33], as questões de competência desempenham para a engenharia de ontologias, um papel análogo aos modelos de casos de uso na engenharia de software orientada a objetos, uma vez que ambas atuam na etapa de elicitação de requisitos e podem ser utilizadas como casos de teste na fase de validação.

- **Captura da ontologia** - De acordo com [33] e [15], trata-se da etapa mais importante no desenvolvimento de uma ontologia. O objetivo é capturar os conceitos envolvidos no contexto com base na competência da ontologia, ou seja, o conjunto de elementos de um domínio que podem ser representados numa ontologia. Os conceitos e relações relevantes devem ser identificados e organizados. É importante, segundo [15], definir os conceitos primitivos e os derivados. Os primeiros referem-se àqueles que são definidos por linguagem natural e que não são passíveis de uma definição em termos de outros conceitos da ontologia. Conceitos derivados são aqueles definidos em termos de outros conceitos, de forma a possuírem um relacionamento formal com os mesmos [13].

Os conceitos e relações formam a base da ontologia. No entanto, uma característica essencial nessa fase é a definição de axiomas, que especificam definições de termos na ontologia e restrições sobre sua interpretação. Os axiomas constituem as propriedades (*slots*) do conceito mapeado, bem como eventuais restrições associadas ao mesmo (*facets*).

De acordo com [33], os axiomas em uma ontologia podem apresentar duas formas e propósitos diferentes: axiomas de derivação e axiomas de consolidação. Axiomas de derivação são aqueles que permitem definir informações a partir do conhecimento previamente existente e consolidado. Representam conseqüências lógicas no processo. Axiomas de consolidação são utilizados para descrever a coerência das informações existentes. Não representam conseqüências lógicas. Conforme [13], as categorias de axiomas são caracterizadas da seguinte forma:

- Axiomas de consolidação definem principalmente condições para a definição de um objeto como instância de um conceito;
- Axiomas de derivação podem ter origem no significado dos conceitos e relações da ontologia ou mesmo na forma como são estruturados;
- Os axiomas também podem ser classificados de acordo com o conteúdo descrito pelos mesmos. Neste caso, eles são ditos formais ou epistemológicos, quando são descritos para representar restrições

impostas pela forma de estruturação dos conceitos. São independentes de domínio;

- Caso o conteúdo descrito represente restrições de significação impostas no domínio, são categorizados como axiomas dependentes de domínio, ou seja, axiomas ontológicos [13].

De acordo com [33], os axiomas devem ser necessários e suficientes para expressar as questões de competência e para caracterizar suas soluções. Se os axiomas propostos não forem suficientes para esse propósito, então se devem adicionar conceitos, relações ou axiomas ao modelo ontológico proposto. Por outro lado, axiomas redundantes ou que não contribuem para responder a uma questão de competência devem ser eliminados [13].

- **Formalização da ontologia** - Nesta etapa, é necessário que se faça uso de um formalismo de representação das diversas categorias de conhecimento da ontologia mapeada [33] [15]. No entanto, modelos orientados a objetos, em que os conceitos são definidos e modelados como classes, de modo a se criar os relacionamentos entre os mesmos e inserir eventuais restrições, tem tornado o processo de especificação da ontologia mais voltado e integrado à Engenharia de Software. Um exemplo de ferramenta que faz uso da modelagem orientada a objeto para construir modelos de ontologias é o projeto *Protégé* [54].

- **Integração com ontologias Existentes** - Durante as fases de definição da ontologia, principalmente na captura e formalização, pode ser necessário ou possível a integração com modelos ontológicos já existentes. Exemplos de bibliotecas de ontologias públicas são *Ontolingua ontology library* [50] ou *DAML ontology library* [11].

- **Avaliação** - A ontologia deve ser avaliada para garantir que os requisitos definidos na especificação estão sendo satisfeitos [33]. Esta etapa deve ser realizada em paralelo com as etapas de captura e formalização. Segundo [13], os principais critérios para guiar tanto o desenvolvimento, quanto a avaliação da qualidade das ontologias construídas são: clareza, coerência, extensibilidade e compromissos ontológicos mínimos.

- **Documentação** - De acordo com [33], todo o processo de desenvolvimento da ontologia deve ser documentado. Esta documentação deve englobar propósitos, requisitos, cenários, as descrições textuais de cada conceito, a ontologia formal e os critérios de projeto adotados. Esta etapa deve ocorrer ao longo de todo o ciclo, em paralelo com as demais etapas.

As ontologias podem ser utilizadas de várias formas. Para que haja uma clara compreensão da aplicação das ontologias, foi definido um conjunto de cenários. Conforme [37], um cenário é um caso de uso abstrato para uma classe de aplicações similares e descreve uma situação específica na qual uma ontologia é utilizada para um determinado propósito. De acordo com [37], são identificadas quatro categorias principais para aplicações de ontologias: autoria neutra, na qual um artefato é produzido em linguagem única e convertido em diferentes formas de uso para atender múltiplos sistemas; ontologia como especificação, na qual é criada uma ontologia para um dado domínio e é utilizada como base para especificação e desenvolvimento de *softwares*; acesso comum, na qual a ontologia é desenvolvida a prover uma base de termos comuns a ser compartilhada; ontologia de pesquisa, na qual a ontologia é utilizada para melhorar o desempenho de pesquisas de informações em um dado repositório. Neste trabalho a categoria utilizada para o modelo proposto foi o de ontologia como especificação, ilustrada pela Figura 24, na qual os atores utilizam a ontologia assim como as aplicações construídas para o referido contexto.

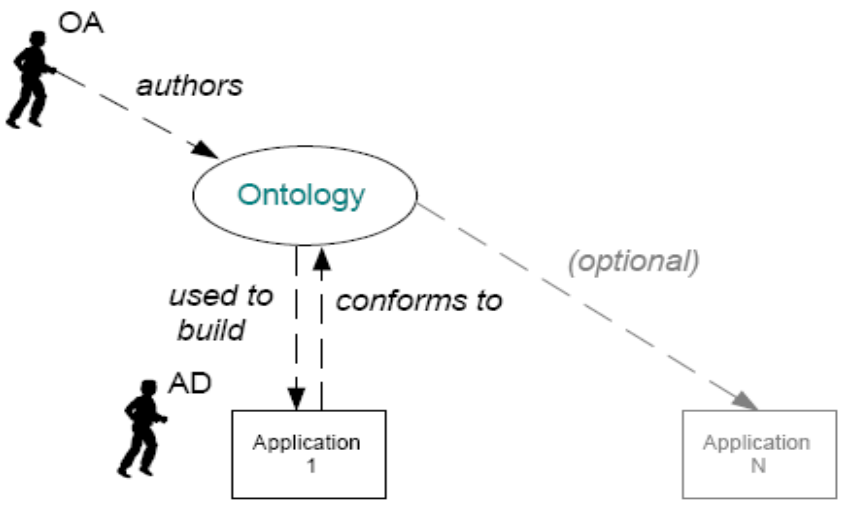


Figura 24 - Ontologia como especificação [37]

6.3 Ontologias em Gerência de Redes

As ontologias, de acordo com [70], podem ser classificadas em dois grandes grupos: as consideradas de peso leve (*lightweight*) e as que são peso pesado (*heavyweight*). As primeiras apresentam um modelo de domínio, sem a presença de axiomas e restrições. São estruturas simples de termos primitivos ou compostos associados às suas respectivas definições. Já as ontologias *heavyweight*, incluem todos os elementos, possibilitando a realização de inferências através da informação existente. São axiomatizadas e representam um modelo ontológico de modo explícito. Toda ontologia peso pesado contém necessariamente uma versão peso leve.

Ontologias, que modelam o contexto de gerência de redes, são, em boa parte, peso leve. Porém, uma ontologia *lightweight* não consegue garantir, por si mesma interoperabilidade semântica. Segundo [72] [73], pode se modelar um conhecimento neste contexto, sob a forma de ontologias, através de: redes semânticas, representação baseada em *frameworks* definidos, lógica de descrição e por modelos orientados a objetos. Este último usa os conceitos de classes, objetos e atributos para modelar a informação.

No modelo orientado a objeto, de acordo com [48], os conceitos são mapeados em classes e os atributos destas, chamados de *slots*, constituem as propriedades que cada conceito apresenta dentro do contexto em questão. As restrições associadas aos atributos, denominadas *facets*, definem características dos *slots*. Os relacionamentos entre as classes irão prover o conjunto de regras associadas ao conteúdo que está sendo modelado.

Para se aplicar ontologias em modelos de gestão de ambientes de rede, três passos devem ser realizados, conforme [72]. O primeiro deles trata-se de se estudar as possibilidades de se empregar uma linguagem de definição de ontologias na especificação de um modelo de informações de gerência, contemplando as restrições e peculiaridades típicas de cada linguagem de especificação de gestão, como o SMI (*Structured Information Management*) referente ao protocolo SNMP, MIF (*Model Information Format*) referente ao DMI, GDMO (*Guidelines for the*

Definition of Managed Objects) correspondente ao CMIP ou o CIM (*Common Information Model*) que se refere ao WBEM.

O segundo passo, de acordo com [72], seria, a partir da linguagem estabelecida para especificação da ontologia na fase anterior, definir um modelo de correspondência entre os diversos protocolos utilizados para extração de informações gerenciais. Neste ponto, a construção de um modelo baseado em uma abordagem *upper-level*, que se caracteriza pela identificação de conceitos que serão base para a compreensão do contexto modelado, traz benefícios como a possibilidade de se criar uma camada conceitual genérica suficiente para ser utilizada em um domínio amplo como o de gerência de redes.

Por fim, analisar um meio de se adicionar um comportamento às especificações das informações de gerência, utilizando para isso as características proporcionadas pela ontologia construída. Tais regras de comportamento estão relacionadas, principalmente, a restrições impostas pelos relacionamentos entre os conceitos modelados e às suas propriedades. Tais regras serão responsáveis por definir que o relacionamento entre conceitos como recurso e serviço, dentro do domínio de gerência de configuração, é de herança ou especificação, por exemplo. A Figura 25 ilustra os passos descritos.

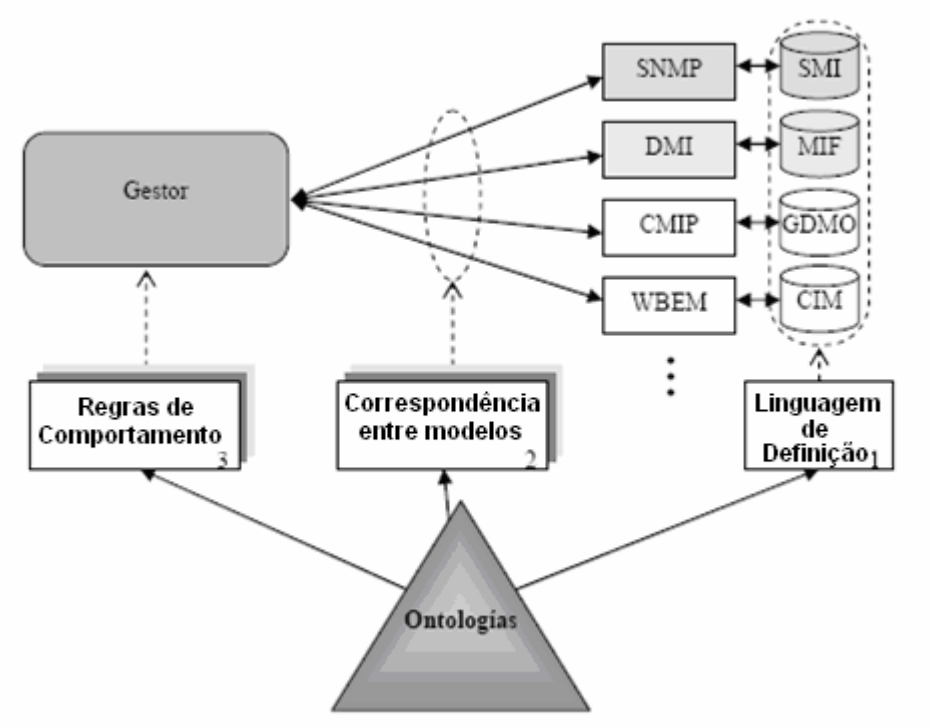


Figura 25 - Passos para se modelar uma ontologia em gerência de redes [72]

Para que se tenha um modelo ontológico capaz de atender o processo de gerência de redes, é necessária uma linguagem de definição capaz de expressar a semântica precisa dos conceitos, bem como as restrições associadas à informação [72]. Segundo [72], as linguagens de definição que são mais utilizadas no momento estão relacionadas à web semântica. Neste contexto, destaque para DAML+OIL (*DARPA Agent Markup Language + Ontology Inference Layer*), embora não seja uma linguagem voltada para gestão de redes.

De acordo com [72], DAML+OIL é uma linguagem de ontologias que permite definir classes, propriedades e restrições (*facets*) relacionadas às mesmas e pode utilizar o formato KIF (*Knowledge Interchange Format*) para realizar sua axiomatização. KIF utiliza lógica de primeira ordem para mapear os relacionamentos, de forma a ser logiamente equivalente às descrições realizadas em DAML+OIL.

Outra característica importante da linguagem DAML+OIL, é o fato de possibilitar a especificação de classes, além de outras características próprias do paradigma orientado a objeto. No entanto, não permite a definição de métodos e algumas restrições particulares relacionadas a informações de gerência. Tais características devem ser modeladas com o uso de definições em RDF (*Resource Description Language*). Um exemplo de uma restrição a ser inserida através do uso da linguagem RDF seria a definição do tipo de acesso para um recurso.

A Figura 26 mostra a definição de uma classe em DAML+OIL de uma informação extraída do modelo CIM. Nela é descrita a classe *CIM_ManagedSystemElement* e suas propriedades associadas.

```
<daml:Class rdf:ID=
    "CIM_ManagedSystemElement">
        <rdfs:comment>
CIM_ManagedSystemElement is the base class for
the System Element hierarchy. [...]
        </rdfs:comment>
        <rdfs:subClassOf rdf:resource=
            "#CIM_ManagedElement"/>
</daml:Class>

<daml:DatatypeProperty rdf:ID=
    "InstallDate">
    <rdfs:comment>
A datetime value indicating when the object was
installed. A lack of a value does not indicate that
the object is not installed.
    </rdfs:comment>
    <daml:domain rdf:resource=
        "#CIM_ManagedSystemElement"/>
    <daml:range rdf:resource=
        "http://www.w3.org/2000/10/XMLSchema
#dateTime"/>
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="Name">
    <rdfs:comment>
The Name property defines the label by which the
object is known. When subclassed, the Name
property can be overridden to be a Key property.
    </rdfs:comment>
    <daml:domain rdf:resource=
        "#CIM_ManagedSystemElement"/>
    <daml:range rdf:resource=
        "http://www.w3.org/2000/10/XMLSchema
#string"/>
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="Status">
    <rdfs:comment>
A string indicating the current status of the object.
Various operational and non-operational statuses
are defined. [...]
    </rdfs:comment>
    <daml:domain rdf:resource=
        "#CIM_ManagedSystemElement"/>
    <daml:range rdf:resource=
        "http://www.w3.org/2000/10/XMLSchema
#string"/>
</daml:DatatypeProperty>
```

Figura 26 - Script DAML+OIL modelando uma classe do modelo CIM [72]

6.4 Ontologia de gerenciamento de agentes FIPA

De acordo com as especificações da FIPA, os agentes DF e AMS comunicam-se utilizando a linguagem de conteúdo FIPA-SL (*Foundation for Intelligent Physical Agents - Standard Language*), a ontologia *fipa-agent-management* e o protocolo de interação *fipa-request* [61]. Ou seja, dois agentes para se comunicarem, transferem informação através de uma mensagem ACL. Conforme [61], dentro da mensagem ACL, a informação é representada como uma expressão de conteúdo, consistente com uma linguagem de conteúdo apropriada (FIPA-SL) e codificada em um formato *String*. No entanto, cada agente possui sua própria forma de representar a informação internamente. Em cada momento que um determinado agente enviar um pedaço de informação para outro agente, têm-se dois

problemas a serem resolvidos para que haja o estabelecimento de fato da comunicação entre ambos: conversão e checagem da mensagem enviada. O agente emissor precisa converter sua representação interna da informação para a representação da expressão de conteúdo ACL correspondente, e o agente receptor necessita realizar a conversão oposta. Além disso, este último, ao receber a informação, deve realizar a checagem semântica para verificar se esta é um pedaço de informação significativa, isto é, se está compatível com as regras da ontologia através da qual os dois agentes envolvidos atribuíram um significado apropriado para a informação [61].

O JADE está em conformidade com estas especificações através da implementação dos componentes:

- O FIPA-SL é implementado pela classe: *jade.content.lang.sl.SLCodec*, através da adição do método:
  - *getContentManager().registerLanguage(SLCodec(0))*;
- Os conceitos de ontologia são implementados pelas classes constantes no pacote: *jade.domain.FIPAAgentManagement*. Ou seja, as classes do JADE que serão estendidas pelas classes referentes aos conceitos identificados na ontologia, estão neste pacote;
- A classe *FIPAManagementOntology* define o vocabulário com os símbolos da ontologia. Para utilizá-la, o agente deve ser acrescido do seguinte código:
  - *getContentManager().registerOntology(FIPAManagementOntology.*

    getInstance());
- O protocolo de interação fipa-request é implementado como um comportamento (*behaviour*) no pacote: *jade.proto.*

O *framework* JADE provê um suporte para linguagens de conteúdo e ontologias projetado de forma a realizar automaticamente as conversões e checagem das operações. Conforme a Figura 27, a mensagem é transferida contendo uma seqüência de bytes e a conversão para objetos Java é realizada pelo JADE. Desta forma, JADE permite que os desenvolvedores manipulem informações dentro de seus agentes como objetos de classes Java, sem a necessidade de qualquer esforço extra. Desta forma, toda classe implementando

o conceito de ontologia de gerenciamento de agentes é uma simples coleção de atributos, com métodos públicos para ler e escrever nestes atributos.

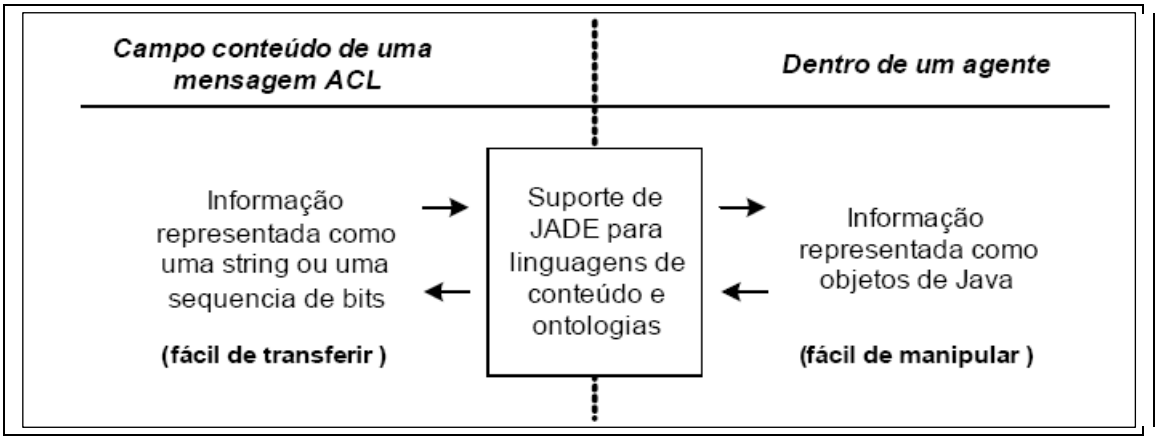


Figura 27 – Conversão do JADE para linguagens de conteúdo e ontologias [61]

6.5 Modelo de Referência JADE para Ontologias

O modelo de ontologia apresentado neste capítulo na seção 6.6 segue o padrão baseado no paradigma orientado a objeto [6], definido pelo JADE, *Java Agent Development Framework*, conforme a Figura 28.

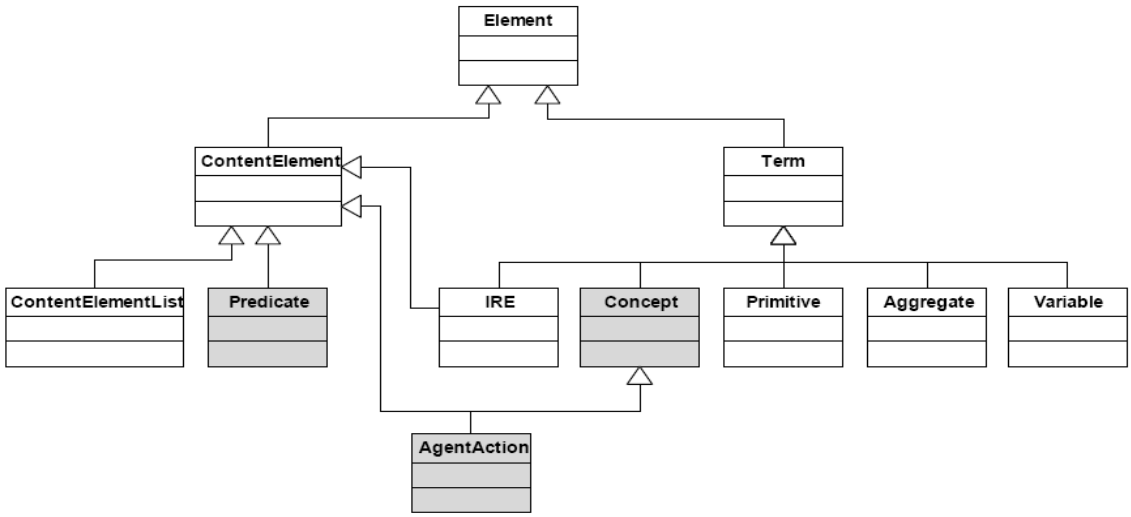


Figura 28 - Modelo de referência de conteúdo JADE [4]

A Figura 28 apresenta o modelo de referência de conteúdo de JADE para definição de elementos de uma ontologia. Os principais elementos que compõe este modelo, de acordo com [4], são:

- **Predicados** - São expressões que dizem algo a respeito do estado do mundo, podendo assumir um valor verdadeiro ou falso. Por exemplo: (Possui-rota (*Roteador:fabricante Cisco*) (*Rota :destino 200.241.1.30*)) está questionando se "o roteador de marca Cisco possui, em sua tabela de rotas, a rota destino 200.241.1.30". Os predicados podem ser usados significativamente como conteúdo de mensagens com as performativas INFORM ou QUERY-IF, enquanto que não faz muito sentido se for utilizado como conteúdo de uma mensagem REQUEST, uma vez que não se solicita um estado do mundo como "*Rota Inalcançável*", por exemplo, e sim alguma informação que possa refletir o mesmo, como a rota *200.241.1.30* que pode estar inalcançável em um dado momento.

- **Termos** - Expressões que identificam entidades (abstratas ou concretas) que "existem" no mundo e que os agentes podem conversar e raciocinar sobre elas. Os termos são classificados em conceitos, ações, primitivas, que são expressões que indicam entidades atômicas, como *strings* e inteiros, IREs (*Identifying Referential Expressions*), agregados e variáveis. IREs são expressões que identificam a entidade (ou entidades) para as quais um determinado predicado é verdadeiro. Agregados são expressões que indicam entidades como grupos de outras entidades. Variáveis são expressões utilizadas em consultas e referenciam elementos desconhecidos nestas.

- **Conceitos** - Expressões que indicam entidades com uma estrutura complexa que pode ser definida em termos de *slots*. Como exemplo tem-se a estrutura roteador, que pode ser definida, neste caso, com os *slots* fabricante e número de portas: (*Roteador :fabricante Cisco :numPortas 16*). Geralmente, não faz sentido usar conceitos diretamente como conteúdo de uma mensagem ACL. Em geral eles são referenciados dentro de predicados ou outros conceitos, pois desta forma agregam semântica trabalhada pelo modelo ontológico à mensagem, como por exemplo, em (*Dispositivo :camada "3" :ativo (Roteador :fabricante Cisco :numPortas 16)*), que indica que o conceito *Roteador* é categorizado como um outro conceito, o de *Dispositivo*.

- **Ações de agentes** - Conceitos especiais que indicam ações que podem ser realizadas por determinados agentes. Por exemplo, a ação de localizar um endereço destino na tabela de rotas de um roteador pode ser representada da seguinte forma: (*IdentificarRota (Rota :destino "200.241.1.30") (Roteador :fabricante Cisco :numPortas 16)*). Possuem conteúdos significativos para certos tipos de mensagens ACL, tais como aquelas com performativa do tipo REQUEST. Enviar e receber mensagens ACL são ações de agentes.

6.5.1 Estrutura de uma ontologia em JADE

De acordo com [4], uma ontologia em JADE é uma instância da classe *jade.content.onto.Ontology*. Nesta, a estrutura de predicados, ações de agentes e conceitos que são pertinentes ao domínio que está sendo modelado através da ontologia são definidos como conjuntos de esquemas declarados. Os esquemas são instâncias das classes *PredicateSchema*, *AgentActionSchema* e *ConceptSchema*, pertencentes ao pacote *jade.content.schema*. Tais classes provêem métodos para se declarar os campos que definem a estrutura de cada tipo de conceito, ação ou predicado modelado.

Ao definir a ontologia, esta deve ser declarada como um objeto *singleton*. Para tanto, será necessário construir uma classe que estenda *jade.content.onto.Ontology* com um método estático. Este fará o acesso a esse objeto *singleton*. O objetivo desta estratégia é compartilhar, entre os agentes contidos em uma mesma JVM, o mesmo objeto que define a ontologia e, dessa forma, todos os esquemas pertencentes a ele [61].

Cada elemento adicionado na ontologia é associado a uma classe Java. Ou seja, a informação modelada como o conceito Recurso, seu esquema para o JADE, é associado à classe *Recurso.java*. Ao fazer uso da ontologia definida, expressões representando informações modeladas com o conceito de recursos, são instanciadas como objetos da classe *Recurso*. Cada campo de um esquema tem um nome e um tipo, isto é, valores para este campo devem estar compatíveis com o esquema dado. Um campo pode ser declarado como opcional e possui uma cardinalidade própria. Por exemplo, o campo *protocolosSuportados* dentro do

esquema do conceito *Serviço* pode conter um ou mais elementos do tipo *Protocolo*. Um esquema pode conter outros super esquemas, o que permite definir relacionamentos de especialização ou extensão entre conceitos, como no caso do conceito *Dispositivo* como sendo uma especialização do conceito *Entidade*. Tais conceitos são ilustrados no diagrama de classes da Figura 29. Este representa, de forma parcial, a modelagem dos conceitos identificados no domínio estudado neste trabalho.

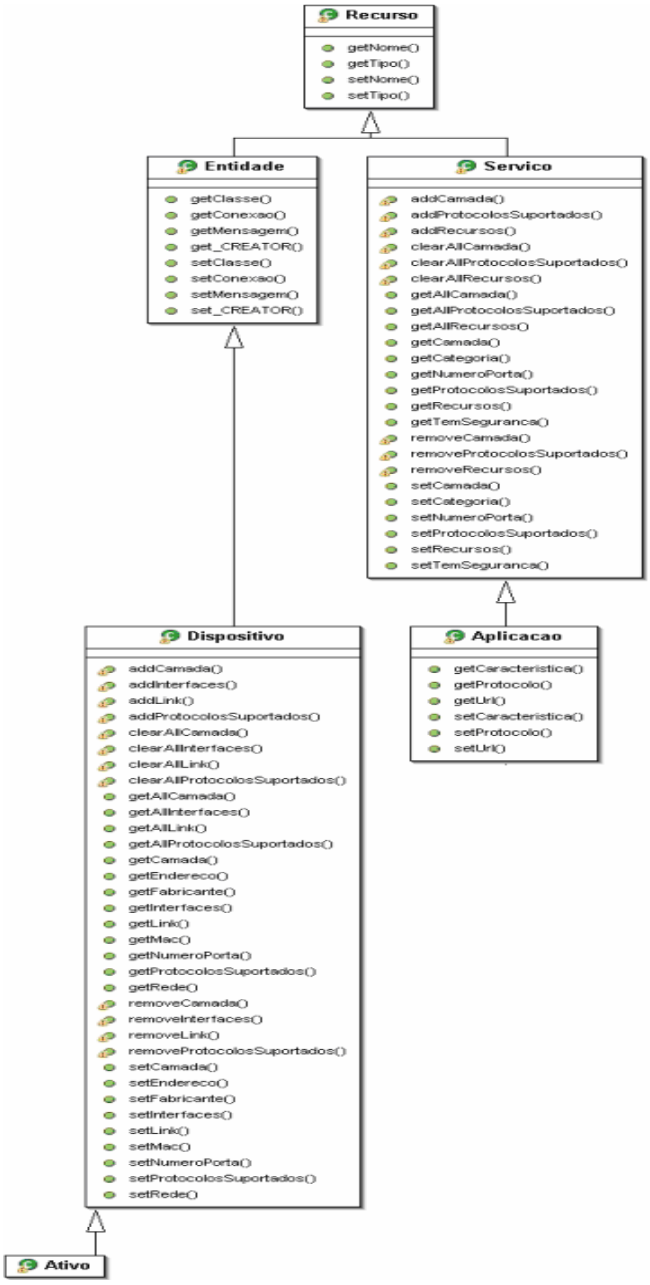


Figura 29 - Diagrama parcial de classes Conceitos mapeados na Ontologia

De acordo com [4], a conversão e checagem de operações é realizada pelo gerenciador de conteúdo da plataforma JADE. Este é uma instância da classe *jade.content.ContentManager*. Cada agente construído com este *framework*, possui um gerenciador de conteúdo que está acessível através do método *getContentManager()* da classe *Agent*.

A classe *ContentManager* provê todos os métodos para converter objetos Java em seqüências de bytes e para, posteriormente, inserir esta sequência no campo de conteúdo de instâncias da classe *ACLMessage* [61]. Ele fornece também os métodos para o procedimento inverso, objetivando extrair as informações da mensagem recebida e transformar seu conteúdo em objetos de classes Java. Esta transformação é realizada em duas fases. A primeira fase é referente à validação semântica do conteúdo da informação, ou seja, refere-se ao processo de verificar se a informação trazida na mensagem está instanciada em objetos que respeitem as regras semânticas do modelo de domínio ao qual referencia. É realizada pela própria ontologia, através de uma instância da classe *jade.content.onto.Ontology*. A segunda fase corresponde à tradução do conteúdo da mensagem para *strings* ou sequências de *bytes*. Esta é realizada por um codificador de linguagem, por meio de uma classe que estenda a interface *jade.content.lang.Codec*. Desta forma, o processo de validação e conversão de mensagens é transparente para o desenvolvedor, que mantém seu foco no desenvolvimento do modelo ontológico capaz de mapear as informações que trafegam no contexto estudado.

6.6 Proposta de Modelo de Ontologia

Neste trabalho, o processo de engenharia de domínio para desenvolvimento da ontologia foi dividido em duas fases: análise e projeto de domínio. Em referência à metodologia apresentada para construção de ontologias [33], a fase de análise de domínio para este modelo proposto envolveu a fase de Identificação de Propósito e Especificação de Requisitos, na qual se definiu o contexto a ser explorado (gerência de configuração de redes) e os principais pontos que a ontologia deveria atender; e parte da fase de Captura da Ontologia, uma vez que neste momento foi realizada a identificação e o mapeamento dos conceitos que

compõem o domínio do conhecimento em questão, bem como a definição semântica de cada um deles. O objetivo desta etapa de análise de domínio foi ter como artefato a lista de conceitos, com suas propriedades e relacionamentos definidos.

A segunda, referente ao projeto de domínio, abordou a parte final da fase de Captura da Ontologia, visto que nesta etapa os conceitos foram organizados dentro de um modelo conceitual com as restrições associadas às suas propriedades e aos seus relacionamentos. Caracterizou-se pela aplicação de uma modelagem orientada a objetos para a construção da ontologia, conforme o padrão adotado pelo JADE. Ou seja, fez-se uso da ferramenta *Protégé* [54] para se produzir um modelo compatível com o modelo de referência utilizado pelo JADE para se modelar ontologias. Como esta ferramenta permite tal modelagem através de conceitos próprios do paradigma orientado a objeto, como classes e atributos, por exemplo, e ao mesmo tempo, possibilita que tais informações sejam construídas segundo o padrão do JADE para ontologias, ao trabalhar com definições como conceitos, predicados e ações de agente, esta se mostrou adequada para utilização neste momento. Desta forma, foi possível construir um modelo ontológico de maneira mais simples e compreensível, como também produzir este modelo em compatibilidade com as exigências do *framework* que foi utilizado na construção do protótipo para o estudo de caso. Este modelo, posteriormente, pôde ser trabalhado como um diagrama de classes através de um processo de engenharia reversa, que permitiu a conversão do modelo criado na ferramenta *Protégé* [54] em um diagrama de classes, próprio de ferramentas CASE utilizados para modelagem UML[7].

Sendo assim, a etapa de projeto de domínio teve como insumo a lista de conceitos definida na fase anterior e produziu como artefato, o modelo ontológico proposto em forma de diagrama de classes, de acordo com a metodologia orientada a objeto [6] e o padrão UML [7].

6.6.1 Descrição dos Conceitos Mapeados e suas Propriedades

Os conceitos identificados no modelo de ontologia proposto são descritos a seguir com os seus respectivos atributos (*slots*). A Figura 30 ilustra a hieraquia de conceitos no que diz respeito aos recursos da rede.

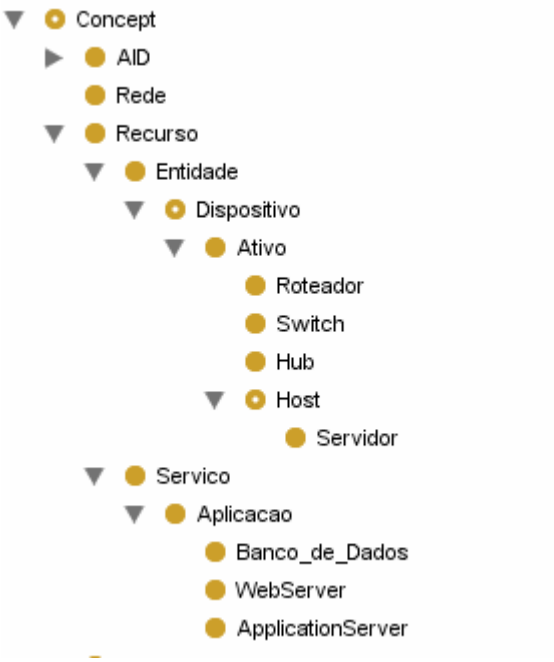


Figura 30 - Hieraquia de Conceitos relacionados a Recurso

Redes: Forma de organização de dispositivos de forma a permitir que estes se comuniquem dentro de um grupo. O que determina se dispositivos estão em uma mesma rede será o endereço IP e a máscara de cada um.

> ***Slots***: o endereco do gateway da rede, o local físico da rede, a máscara da classe da rede, o conjunto de recursos da mesma, onde estão inclusos os dispositivos físicos.

Recurso: Um serviço ou característica fornecida por uma rede e que é consumida por uma entidade ou dispositivo. Entende-se por característica uma propriedade fornecida pela rede, como, por exemplo, a largura de banda que a rede proveja, ou mesmo uma entidade da mesma. Um recurso pode ser especializado em Entidade e Serviço.

> ***Slots***: nome do recurso, tipo de recurso.

Entidade: Estrutura física ou lógica com um papel definido dentro do contexto trabalhado. Pode prover e/ou consumir recursos. Especializa-se em Dispositivo.

> ***Slots***: nome da entidade, tipo de entidade (se física ou lógica), classe a que pertence,conexão que possui, mensagem que está sendo recebida pela mesma.

Dispositivo: Objeto físico conectado à rede. Pode referenciar um objeto da rede que esteja, em um dado momento, inativo. Especializa-se em Ativo.

> ***Slots***: nome do dispositivo, tipo de entidade (se física ou lógica), classe a que pertence, rede que pertence o dispositivo, endereço IP, endereço MAC, camada em que atua, protocolos suportados, interfaces disponíveis, número de portas, links que possui com outros dispositivos, conexão que possui no momento, lista de serviços instalados no mesmo, conjunto de rotas que compõem sua respectiva tabela de encaminhamento, lista de processos que estejam ativos em memória, mensagem que está sendo recebida pelo dispositivo, nome do fabricante e modelo.

Ativo: Dispositivo conectado fisicamente à rede e que esteja em funcionamento.

Especializa-se em Host, Roteador, Switch e Hub.

> ***Slots***: Mesmos do conceito Dispositivo.

Host**:** Estação em funcionamento em uma rede. Pode ser um servidor ou uma estação cliente. Especializa-se em Servidor e *Workstation*.

> ***Slots***: Além dos herdados pelo conceito pai, tem-se o conjunto de dispositivos que possui, memória, disco, sistema operacional, número e modelo do(s) processador(es).

Servidor: Host equipado com *software*(s) que o permite prover serviços a demais estações que se comuniquem com o mesmo.

> ***Slots***: Além dos herdados pelo conceito pai, tem-se o conjunto de aplicações hospedadas pelo mesmo.

Workstation**:** Estação de trabalho que caracterize um *host* consumidor de serviços da rede através da troca de mensagens.

> ***Slots***: Os herdados pelo conceito pai.

Roteador: Dispositivo que interliga nós que podem pertencer a redes distintas e define a melhor rota para que se estabeleça a comunicação entre nós distintos.

> ***Slots***: Os herdados pelo conceito pai e tecnologia, número e modelo do(s) processador(es).

Switch: Dispositivo que interliga nós em uma rede local, porém somente encaminha dados do nó origem ao(s) nó(s) destino(s).

> ***Slots***: Os herdados pelo conceito pai além de largura de banda e um indicativo se possui ou não suporte à colisão.

Hub: Dispositivo que interliga nós em uma rede local encaminhando dados a todos estes.

> ***Slots***: Os herdados pelo conceito pai além de largura de banda.

Serviço: Qualquer demanda que se forneça ao requisitante mediante uma solicitação. Tem sempre um produto final associado. Especializa-se em Aplicação.

> ***Slots***: Os herdados pelo conceito pai além da camada em que o serviço atua, protocolos suportados, categoria a que pertence, conjunto de recursos que o provê e um indicativo se possui mecanismo de segurança.

Aplicação: Serviço a ser consumido por um cliente através da troca de mensagens. Especializa-se em Banco_de_Dados, WebServer e ApplicationServer.

> ***Slots***: Os herdados pelo conceito pai além de característica adicional que venha a possuir a aplicação.

Banco_de_Dados: Aplicação caracterizada por prover armazenamento e gerência de informações.

> ***Slots***: Os herdados pelo conceito pai além de tipo de banco e URL de acesso.

WebServer: Aplicação caracterizada por prover acesso ao cliente via interface *web*.

> ***Slots***: Os herdados pelo conceito pai além do fabricante do servidor e URL de acesso para a aplicação.

ApplicationServer: Aplicação caracterizada por hospedar e processar instâncias de aplicações.

> ***Slots***: Os herdados pelo conceito pai além do fabricante do servidor e URL de acesso para a aplicação.

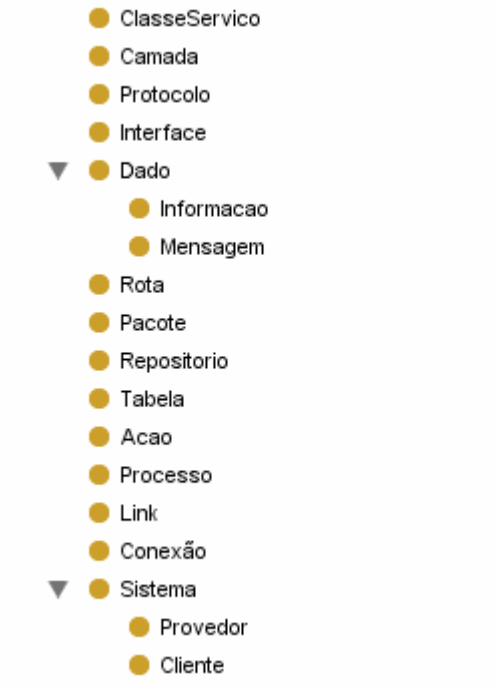


Figura 31 - Demais Conceitos modelados na Ontologia

A Figura 31 apresenta os outros conceitos que foram mapeados na ontologia em questão e são descritos a seguir.

ClasseServiço: Categoria de um determinado serviço que apresenta um conjunto de propriedades que atendem um determinado nível de qualidade de serviço.

Slots: Serviço a que se refere, largura de banda e tempo de resposta.

Camada: Nível de um modelo ocupado por entidades capazes de prover um determinado tipo de serviço.

Slots: Serviço a que se refere, protocolo suportado, número e nome da camada da arquitetura TCP-IP e número e nome da camada do modelo OSI.

Protocolo: Conjunto de regras que define a estrutura de mensagem a ser utilizada na comunicação entre duas entidades.

Slots: Nome, versão, camada que atua e tipo de mensagem suportada pelo mesmo.

Interface: Ponto ou área limite fornecida por um dispositivo ou por uma entidade para se comunicar com dispositivos ou entidades distintas.

Slots: Índice da interface, tipo, padrão, camada, endereço MAC, MTU, status e velocidade suportada.

Link: Conexão física entre dispositivos. Local onde trafega os dados trocados em um processo de comunicação.

Slots: Endereço IP origem, endereço IP destino e largura de banda.

Conexão: Estabelecimento de comunicação lógica entre duas entidades ou dois dispositivos.

Slots: Entidade origem, entidade destino.

Ação: Ato de realizar uma determinada atividade. Executada por qualquer entidade ou dispositivo.

Slots: Entidade que executa a ação, atividade executada.

Processo: Conjunto de regras que determinam a operacionalidade de determinada atividade fornecida por uma entidade. Ficam ativos em memória no dispositivo.

Slots: Entidade origem, ação.

Dado: Características das entidades e dispositivos que podem ser coletadas e registradas. São trocadas por meio de mensagens. Especializa-se em Informação e Mensagem.

Slots: Entidade origem, entidade destino, tipo de entidade, conteúdo.

Informação: Conjunto de dados após ser processados por um sistema. Possui alguma relevância para o contexto.

Slots: Os herdados pelo conceito pai e o conteúdo inferido.

Mensagem: Informação, trocada em um processo de comunicação, que segue um padrão definido pelo protocolo que está sendo utilizado.

Slots: Os herdados pelo conceito pai.

Rota: Caminho a ser percorrido por uma mensagem na rede para atingir seu destino.

Slots: Endereço IP da origem, endereço IP do destino, máscara referente a rede do emissor, gateway definido para a rota em questão, protocolo utilizado,

índice da interface para a qual a rota está mapeada, pacote, roteador dententor da rota .

Pacote: Agrupamento de dados a ser transmitido na rede.

Slots: Dado, tamanho do pacote.

Repositório: Local onde são armazenados os dados e informações organizados em tabelas.

Slots: Nome e tipo da entidade, fabricante do repositório, servidor que o hospeda.

Tabela: Conjunto de informação organizado em um agrupamento lógico de modo a facilitar o acesso a este bloco de dados.

Slots: Nome da tabela, atributos que possui, esquema da base de dados em que foi criada, repositório a que pertence

Sistema: Conjunto de elementos que se relacionam e interagem no desempenho de um serviço. Especializa-se em Cliente e Provedor.

Slots: Serviço a que se pretende consumir ou prover, entidade origem, entidade destino.

Provedor: Entidade com a capacidade de fornecer serviços a outra por meio de interfaces pré-definidas. Um recurso pode ser considerado um provedor de serviços ainda que seja um ativo como um roteador, por exemplo.

Slots: Os herdados pelo conceito pai e alguma restrição que venha a ser imposto ao serviço.

Cliente: Dispositivo consumidor de serviços da rede, podendo ser qualquer ativo da mesma, como por exemplo um switch ou um roteador.

Slots: Os herdados pelo conceito pai.

6.6.2 Descrição do Modelo Ontológico Proposto

Esta seção apresenta uma proposta de um modelo para uma ontologia cujo foco seja a gerência de configuração em redes de telecomunicações. São

descritos os conceitos abordados pelo modelo dentro do contexto pretendido. O objetivo final é prover uma base única e compartilhada de informações de configuração da rede.

Os conceitos modelados são utilizados por agentes específicos, preparados para a obtenção de informações e a conseqüente montagem do modelo. Faz parte da referida proposta as propriedades identificadas para cada conceito mapeado.

Os conceitos envolvidos no modelo e as relações entre eles são ilustrados na Figura 32, na qual é exibido o modelo na ferramenta *Protégé* [54].

Protégé [54] faz uso de uma abordagem de especificação voltado a orientação a objeto, o que torna a linguagem de definição da ontologia transparente para o desenvolvedor, embora ao avançar para especificação de restrições e pelo processo de axiomatização, seja importante o conhecimento apurado da linguagem que se irá utilizar. O *Protégé* utiliza a linguagem OWL (*Ontology Web Language*).

Nas janelas de título *Class Hierarchy*, são apresentadas as classes correspondentes aos conceitos mapeados. Na janela de título *Class Editor*, é exibida a lista de propriedades (*slots*) de uma das classes conceito selecionadas e sua definição. No caso do exemplo da Figura 32, a classe escolhida para visão das características é a classe *Recurso*.

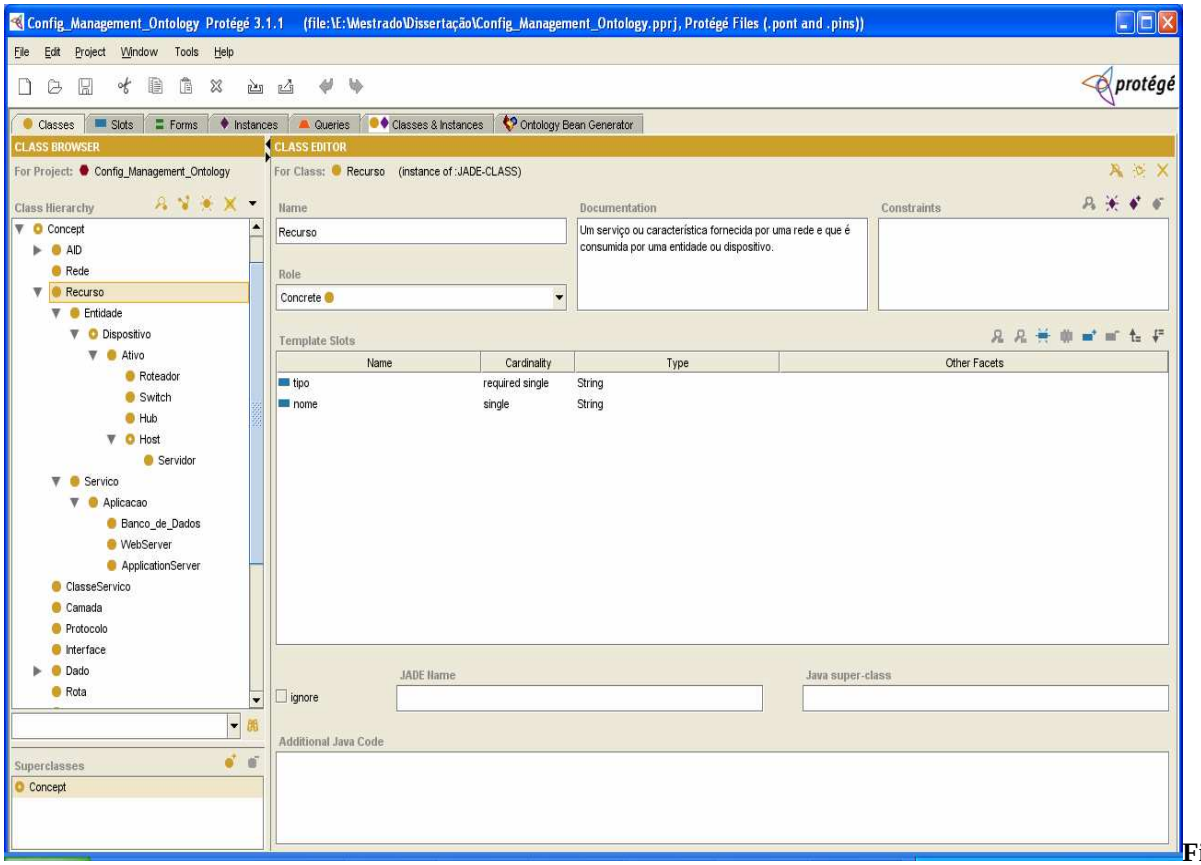


Figura 32 - Modelo da Ontologia no *Protégé*

A Figura 33 apresenta a definição das propriedades para cada classe modelada. Ou seja, para cada conceito identificado, definem-se os seus atributos, denominados *slots*.

Estes, por sua vez, podem ser especializados e atribuídos restrições, também conhecidas como *facets*, conforme a Figura 34.

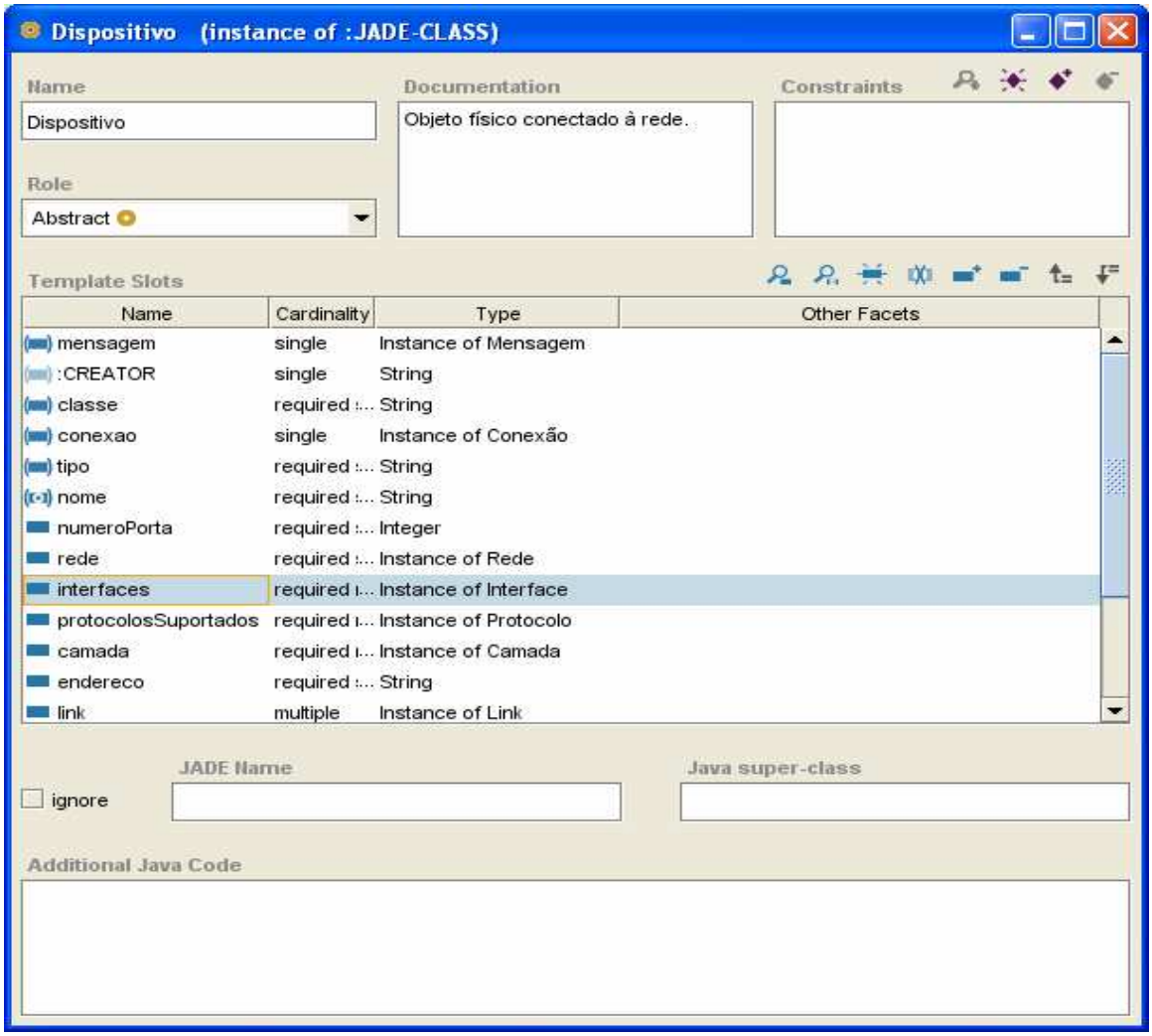


Figura 33 - Definição das propriedades das classes Conceitos no *Protégé*

Figura 34 - Definição de restrições e propriedades de Conceitos no *Protégé*

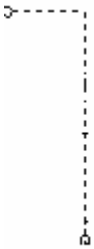
O modelo foi construído a partir da definição de conceitos, a serem contemplados no contexto de gerência de configuração de redes, e dos relacionamentos entre os mesmos, de forma a se criar uma proposta que consiga abranger uma base compartilhada de informações de gerência.

Os conceitos identificados se relacionam conforme os diagramas de classes apresentados no ANEXO C. Estes diagramas constituem a proposta e têm como ponto de partida o conceito de recurso. A proposta de ontologia, conforme a Figura 35, considera recurso como o conceito base a ser explorado. A partir deste, são definidos elementos importantes para o domínio de conhecimento estudado nesta proposta, como conceitos gerados diretamente a partir da especialização de recurso, como entidade e serviço. Tais conceitos possuem todas as propriedades definidas para o conceito genérico (recurso), além daquelas definidas especificamente para cada um deles.

Entidade, neste modelo, representa um objeto físico e concectado à rede. É caracterizada como um dispositivo, que ao ser definido como um ativo de rede, pode ser categorizado como sendo um roteador, um hub, um switch ou um host comum. Neste caso, de acordo com os dados extraídos do dispositivo, o host pode ser considerado um servidor.

O conceito de serviço, neste contexto, por estar associado a um recurso da rede pode ser especializado em aplicação. Esta é fornecida por entidades da rede e pode ser uma instância de um Servidor de Aplicação (*Application Servers*), Servidor Web (*Web Servers*) ou Bancos de Dados. A Figura 35 apresenta o modelo parcial de relacionamento entre os conceitos identificados no modelo proposto. Nesta, os símbolos utilizados têm a seguinte semântica:

Conceito identificado no Modelo Ontológico



Relação de Associação entre Conceitos

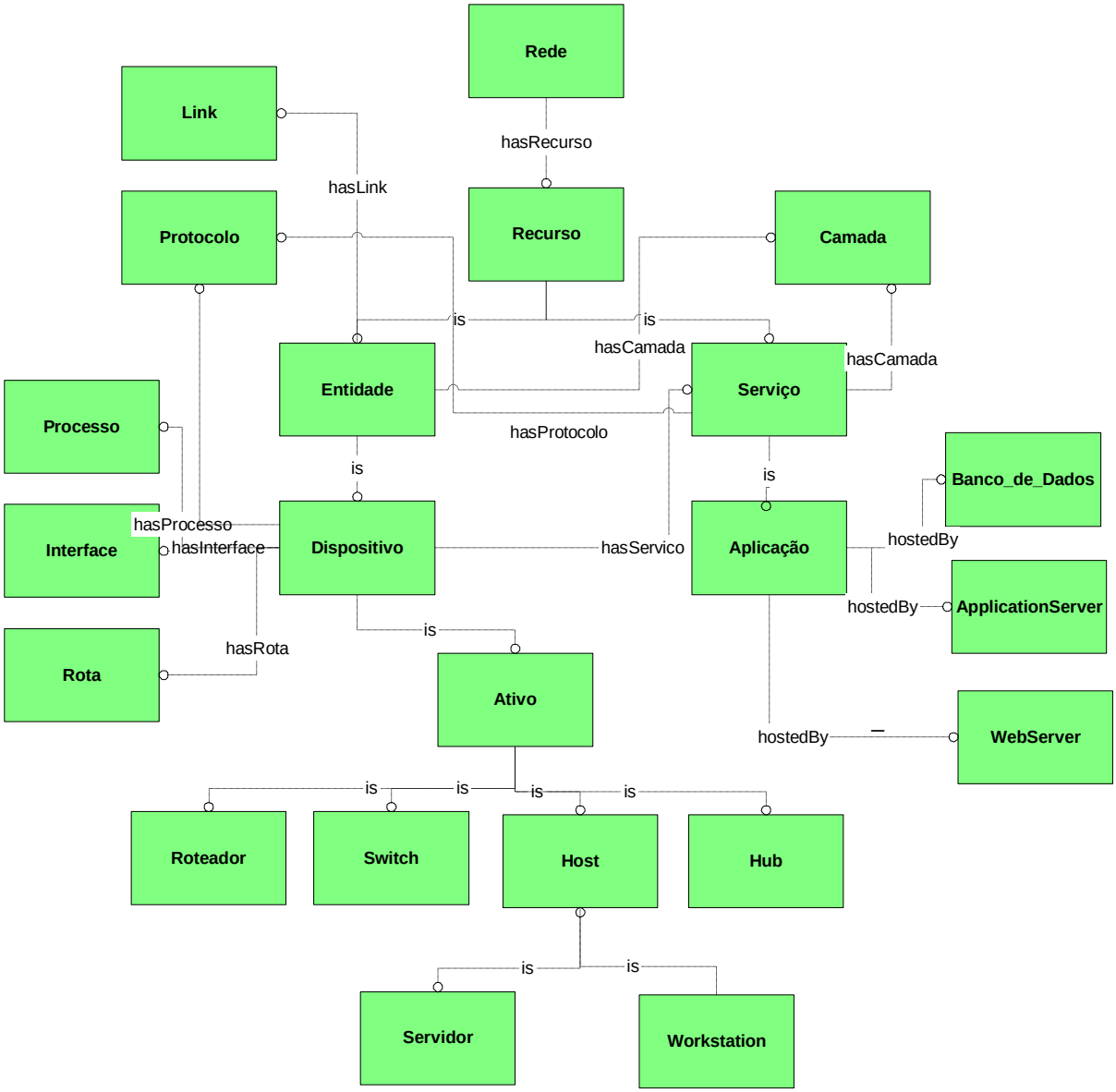


Figura 35 - Modelo parcial da Ontologia

O conceito de redes distintas é trabalhado neste modelo. Cada rede é composta por recursos que podem ser caracterizados como:

- Dispositivos físicos;
- Serviços, como por exemplo aplicações;
- Sistemas clientes de outros recursos da própria rede;
- Sistemas provedores de serviços.

A forma como cada sistema fará uso dos serviços existentes na rede irá definir seu comportamento, seja como cliente ou provedor.

Cada entidade ou dispositivo tem a capacidade de se comunicar com outra entidade ou dispositivo para consumo dos recursos e possuem, para tanto, interfaces que possibilitam tal atividade. Nesse contexto, cada dispositivo tem um conjunto de ações e processos que são capazes de executar, bem como mensagens e dados que podem ser recebidos ou extraídos dos mesmos. Um dado a ser processado passa a ser caracterizado como uma informação.

O tráfego desses dados ocorre via pacote e cada pacote possui uma rota específica, que indica a entidade origem e destino.

A comunicação entre as entidades do modelo acontece via conexão e faz uso de um link, definido para cada entidade emissora da informação. O processo de comunicação ocorre através de troca de mensagens que obedecem a um protocolo, específico do serviço que está sendo consumido, e da camada em que este atua.

Uma entidade pode ser especializada em um dispositivo que, por sua vez, especializa-se em ativo da rede. Como ativo, de acordo com suas características, este pode ser considerado um servidor. Neste caso, pode eventualmente possuir algum repositório, que, por sua vez, pode estar organizado em tabelas.

Os serviços são classificados em classes, de acordo com suas características, e são executados em uma camada específica do modelo OSI (*Open Systems Interconnection*), possuindo um protocolo de comunicação, como por exemplo HTTP (*HyperText Transfer Protocol*) .

O modelo completo do relacionamento entre as classes correspondentes aos conceitos modelados encontra-se no ANEXO C. A Tabela 3 retrata um trecho da construção de um conceito sob a forma de uma classe Java. Desta forma, observam-se como os relacionamentos e as propriedades do conceito são modelados como atributos da classe em questão.

```
package br.ufes.ontologia;

import jade.*;

/**

   * Objeto físico conectado à rede.

   * Protege name: Dispositivo

* @author ontology bean generator
```

```
*/
public class Dispositivo extends Entidade{
...
/**
    * Conjunto de serviços providos pelo dispositivo.
 * Protege name: servicos
    */
    private List servicos = new ArrayList();
    public void addServicos(Servico elem) {
      List oldList = this.servicos;
      servicos.add(elem);
    }
    public boolean removeServicos(Servico elem) {
      List oldList = this.servicos;
      boolean result = servicos.remove(elem);
      return result;
    }
    public void clearAllServicos() {
      List oldList = this.servicos;
      servicos.clear();
    }
    public Iterator getAllServicos() {return servicos.iterator(); }
    public List getServicos() {return servicos; }
    public void setServicos(List l) {servicos = l; }
   /**
   * Endereço IP do dispositivo ou do gateway da rede.
   * Protege name: endereco
   */
   private String endereco;
   public void setEndereco(String value) {
```

```
    this.endereco=value;
  }
  public String getEndereco() {
    return this.endereco;
  }
...
  /**
   * Local onde se encontra dispositivo.
 * Protege name: local
   */
   private String local;
   public void setLocal(String value) {
    this.local=value;
   }
   public String getLocal() {
     return this.local;
   }
  }
```

Tabela 3 - Trecho do código em Java da classe Conceito Dispositivo

6.7 Considerações Finais

Este capítulo apresentou a proposta do modelo de ontologia produzido para esta dissertação. Tratou-se de um modelo cujo foco foi o mapeamento e relacionamento dos principais conceitos no contexto de gerência de configuração de redes.

O processo de identificação dos conceitos foi dirigido por um mapeamento semântico dos principais termos envolvidos no domínio estudado. Tal abordagem facilitou o processo de análise de domínio.

O uso da ferramenta *Protégé* [54], proporcionou a modelagem dos conceitos, segundo o paradigma orientado a objeto e integrado com o *framework* JADE, definido para utilização no protótipo. Este fato possibilitou o desenvolvimento da fase de projeto de domínio de forma mais simples, fazendo com que o mapeamento dos conceitos fosse feito aos moldes de uma especificação orientada a objetos e, no mesmo momento, estivesse compatível com as definições específicas do modelo de referência do JADE.

A utilização do modelo ontológico proposto é realizada por agentes, cujo processo de modelagem e protótipo são apresentados no próximo capítulo.

PROPOSTA DE UM MODELO DE AGENTES

Agentes têm sido desenvolvidos para resolver diversos tipos de problemas dentro da área de gerência de redes. Um desses problemas diz respeito diretamente à gerência de configuração: o processo de *Discovery*. *Discovery* de rede é o procedimento de identificação de equipamentos, estações e servidores, ou seja, de elementos ativos instalados em um ambiente gerenciável. Tal função independe do protocolo a ser utilizado [35].

Várias tecnologias distintas, freqüentemente, são adotadas para construção de tais modelos. Dentre elas destacam-se: SNMP (*Simple Management Network Protocol*), CMIP (*Common Management Information Protocol*), DMI (*Desktop Management Interface*), WBEM (*Web Based Enterprise Management*) e CORBA (*Common Object Request Broker Architecture*). Porém como apresentado por [70] [71], cada um desses modelos possui sua própria linguagem de definição, o que dificulta um processo de integração entre tais modelos.

Dessa forma, uma solução que parta da definição de uma ontologia para este contexto, surge como um caminho factível para se obter uma interoperabilidade semântica em ambientes heterogêneos [70], no que diz respeito ao modelo conceitual definido.

A arquitetura escolhida para a construção dos agentes foi JADE por possuir como principal preocupação a conexão de agentes e sistemas multiagentes entre múltiplas plataformas, buscando como meta uma interoperabilidade entre meios heterogêneos.

7.1 Estudo de Caso

Esta seção apresenta uma proposta de um modelo de agentes para gerência de configuração de redes. O objetivo principal é obter um conjunto de entidades com a capacidade de atuar na construção de um modelo de informações da rede, bem como agir de modo reativo dentro do ambiente gerenciado. Para a obtenção do modelo supracitado, os agentes fazem uso de uma base compartilhada de informações de configuração da rede, modelo especificado no capítulo anterior [58].

O protótipo fez uso do protocolo SNMP para extração de dados, baseado em [55] [56]. Desta forma, foi necessário se fazer um estudo da MIB (*Management Information Base*) SNMP com o intuito de mapear as informações existentes nesta para o modelo ontológico utilizado. A MIB é um conjunto de objetos gerenciados dentro de um sistema aberto, na qual um objeto gerenciado é a visão abstrata de um recurso real dentro deste sistema [62]. Neste contexto, o maior esforço se concentrou em uma análise da estrutura da MIB e, consequentemente, a identificação da semântica associada a cada objeto existente na mesma, conforme trabalhado em [59]. Tal análise demandou um entendimento da estrutura SNMP, dos grupos que compõem a MIB, do significado da informação vinculada a cada OID (*Object Identifier*) válido da estrutura em questão para que, desta maneira, pudesse ser realizado a correspondência e o mapeamento dos dados extraídos, através de uma API Java que utiliza o protocolo SNMP, para o modelo de ontologia proposto nesta dissertação.

7.2 Modelo Arquitetural

Este modelo arquitetural utiliza a metodologia *Tropos* baseada no *framework* i* que possui os seguintes elementos de modelagem, conforme [77]:

- Ator: Entidade ativa que realiza ações para atingir metas como, por exemplo, um cliente que solicita um determinado serviço;

- Tarefa: Modo próprio de se fazer algo, como uma operação, um processo. Um exemplo é a ação de verificar a disponibilidade de um recurso na rede;
- Meta ou Objetivo: Condição ou estado do mundo que o ator necessita alcançar;
- Fronteira do Ator: Limite de atuação do ator no contexto em que está inserido;
- Recurso: Produto final de alguma ação, em um processo, que estará ou não disponível para o ator consumir;
- Meta-*Soft* ou Objetivo-*Soft*: Estado do mundo que o ator deseja atingir, porém sem um critério estabelecido para verificar se realmente a condição para se alcançar tal estado foi satisfeita;
- Agente: Ator que possui manifestações físicas concretas. Agentes podem ser pessoas, agentes de *software* ou *hardware*;
- Papel: Funções que podem ser exercidas por um agente dentro do domínio a que pertence. Refere-se aos comportamentos dos agentes;
- Posição: Entidade intermediária entre um agente e um papel, representando um local no contexto no qual o agente pode executar várias funções.

Estes elementos são apresentados graficamente na Figura 36.

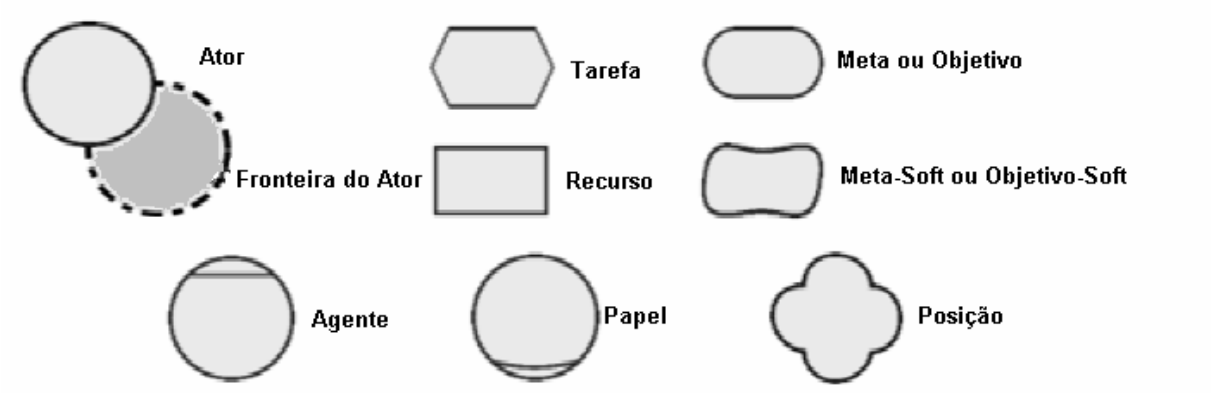


Figura 36 – Elementos para modelagem utilizados pelo *framework i [77]**

É proposto neste modelo, de acordo com a Figura 37, a utilização de quatro agentes que se comunicam entre si, no qual o relacionamento de dependência é representada pela conexão entre os mesmos e suas tarefas. O Agente de *Discovery* (*DiscoveryAgent*) tem o papel de realizar a descoberta dos recursos disponíveis e da construção do modelo de informações, segundo a

ontologia definida no capítulo anterior [58]. Para efetuar esta busca por recursos, foi utilizado o protocolo SNMP [17], conforme estudo realizado no trabalho [59].

Com o objetivo de gerenciar as alterações que venham a ocorrer na estrutura de rede em questão, tem-se o Agente de Configuração (*ConfigurationAgent*). Ele também se comunica com o Agente de *Discovery* para atualização do modelo de dados ontológico.

O Agente Provedor (*ProviderAgent*) tem a preocupação de publicar os recursos a serem disponibilizados para consumo na rede. Dentre tais recursos, estão os serviços e os dispositivos que compõem a rede. Estas informações são providas a este pelo Agente de *Discovery*.

O Agente Cliente (*ClientAgent*) tem a função de se comunicar com o agente Provedor para solicitar o consumo de algum recurso da rede.

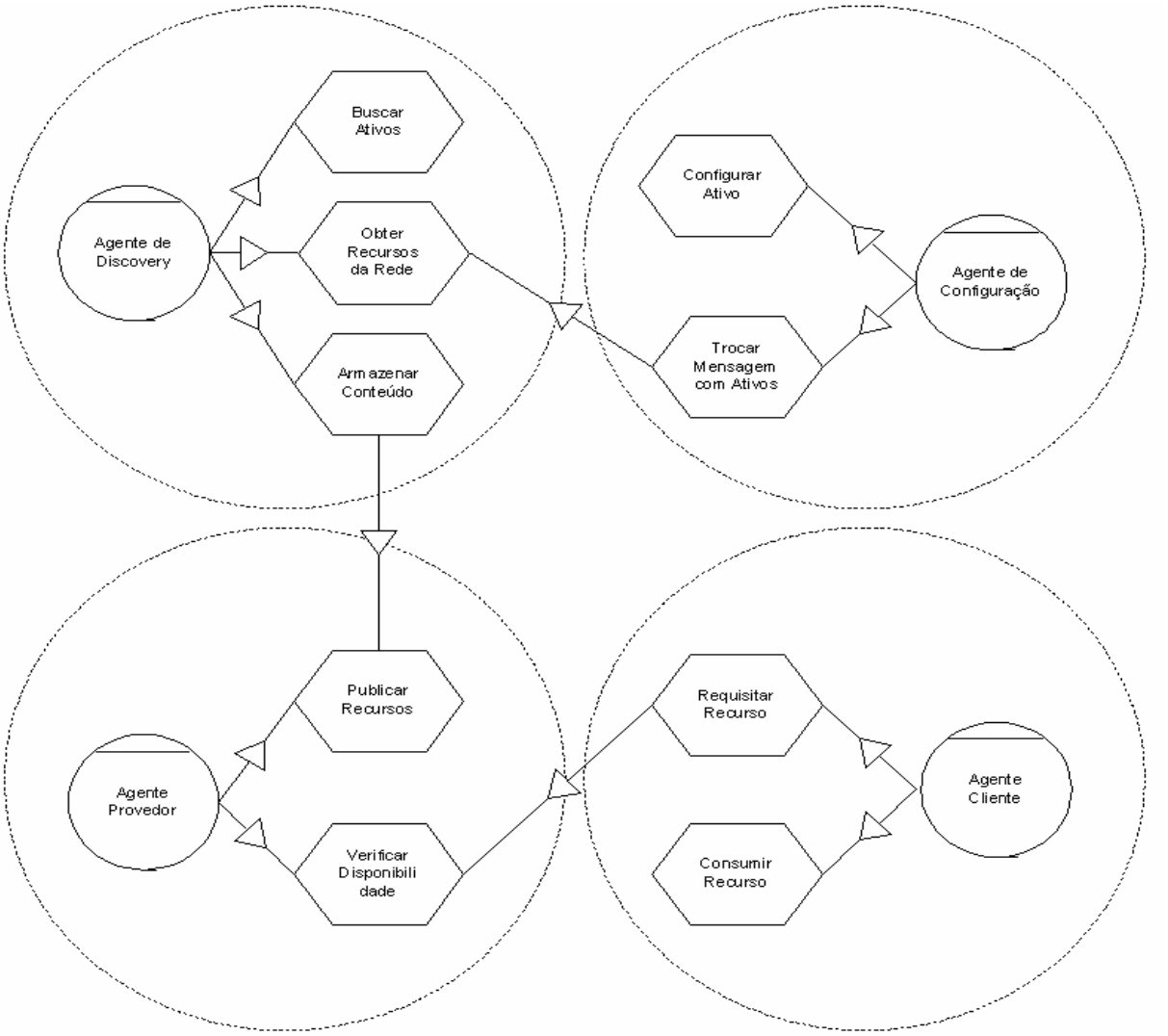


Figura 37 - Modelo Arquitetural dos Agentes Propostos

7.3 Padrões Sociais Utilizados

Quando se fala em modelagem de agentes, uma característica relevante é a definição de padrões sociais, que, conforme [41], são padrões de *design* orientados a agentes para descrever estruturas de SMAs, compostos por entidades autônomas que interagem entre si para atingir suas metas, assim como profissionais em organizações humanas.

Considerando os padrões sociais apresentados em [41] e as relações que há entre os agentes que compõem o sistema **O-bCNMS**, este tem no seu modelo a aplicação de alguns deles.

Os padrões supracitados são exibidos em uma representação que adota a metodologia *Tropos* para modelagem de agentes.

A Figura 38 ilustra os principais tipos de dependência representados no *framework i*[*], base do *Tropos*. De acordo com o *framework i**, dependência é um relacionamento que indica intenção, ou seja, uma ligação, um acordo entre dois atores: o *depender* e o *dependee*. Esta ligação é denominada *dependum*. O *dependee* é o ator que de fato depende de outro, o *depender*, para que a referida ligação, *dependum*, possa ser realizada. Quando o ator *dependee*, disponibilizar um recurso solicitado, executar uma tarefa da qual seja o responsável, atender a um objetivo do ator dependente ou realizar alguma tarefa para alcançar uma meta-*soft*, a relação de dependência estará satisfeita pelo *dependee* responsável pela mesma [55].

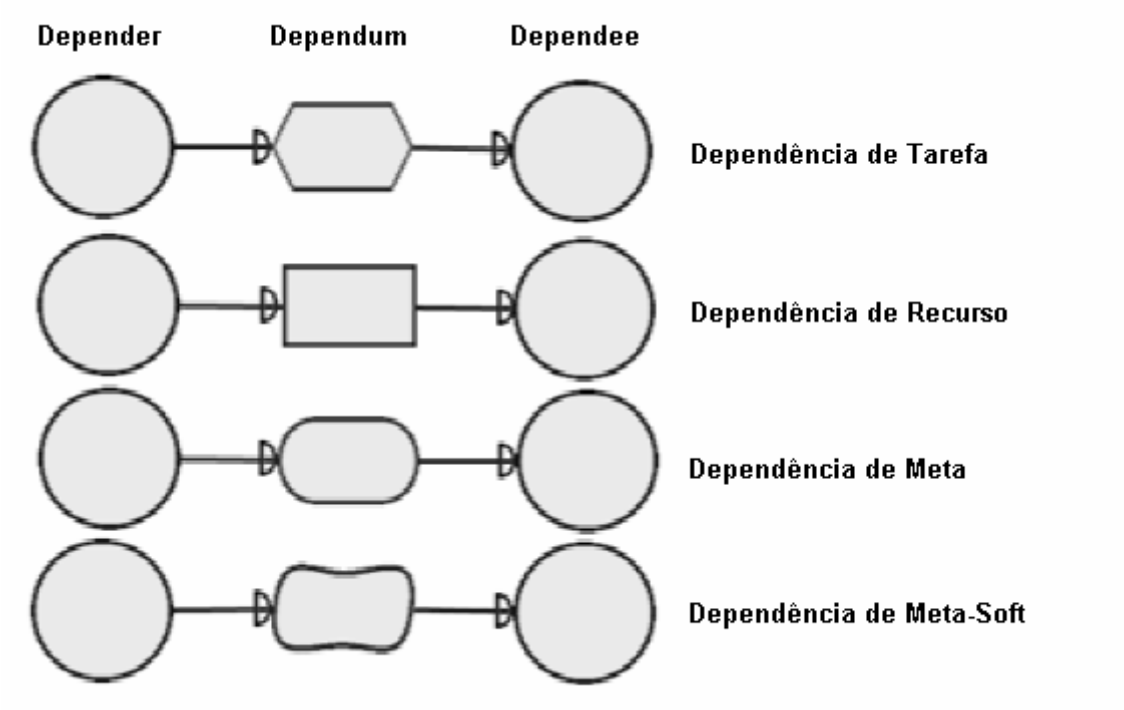


Figura 38 – Relacionamentos de dependência entre atores no *framework i [77]**

Segundo [77], os relacionamentos de dependência usados em *i** são de quatro tipos: tarefas, recursos, meta ou meta-*soft*. A dependência de tarefa caracteriza-se pelo ator *dependee* ser requisitado a executar uma dada atividade, sendo informado sobre o que deve ser feito, porém não como deve ser feito, nem o motivo pelo qual esta atividade deve ser realizada. Na dependência de recurso, o agente depende de haver a disponibilidade de uma entidade física ou de uma informação. A dependência de meta ocorre quando um ator depende de outro para que uma determinada condição seja satisfeita, independente da forma como esta será atendida. Na dependência meta-*soft*, um objetivo, que representa um desejo, deve ser satisfeito, porém não há um critério definido de verificação se a condição necessária para se alcançar o desejo foi de fato atingida.

O padrão *Booking*, ilustrado na Figura 39, envolve um cliente e um número determinado de provedores de serviço, que no caso do protótipo do **O-bCNMS** é um. O cliente requisita o uso de um recurso de um provedor que pode aceitar a solicitação, colocar a mesma em uma lista de espera ou rejeitar o pedido pelo recurso, caso este esteja indisponível no momento [41].

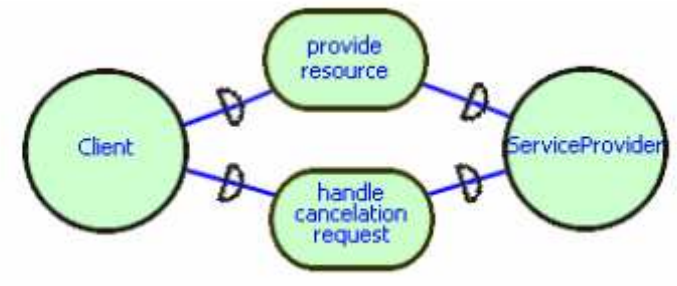


Figura 39 - Padrão Booking [41]

O padrão *Broker* localiza provedores correspondentes à solicitação do cliente por um serviço. Ele requisita e recebe o serviço dos provedores e então encaminha o mesmo para o cliente [41], conforme mostra a Figura 40 em *(b)*. Em *(a)*, o agente *Broker* atua com a função de registrar o serviço que provê, fornecendo em conjunto, o serviço de páginas amarelas, uma vez que cada serviço fornecido pelo mesmo será registrado no provedor. No protótipo, o Agente *Discovery* atua como *Broker* ao avaliar o tipo de solicitação recebida e, então, realizar a busca por ativos utilizando o protocolo SNMP.

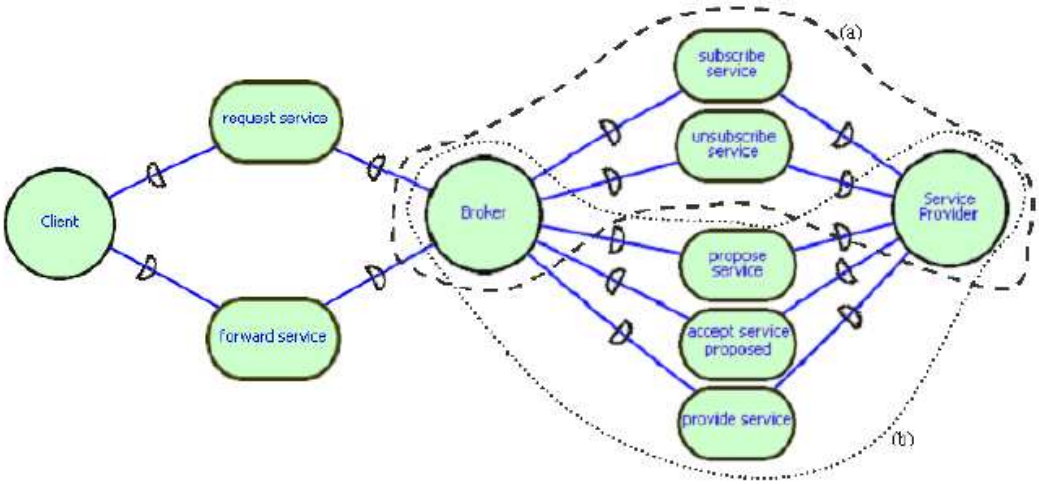


Figura 40 - Padrão Broker [41]

O padrão *Matchmaker* caracteriza-se pelo fato do agente localizar um provedor correspondente a uma solicitação de serviço de um cliente, e então encaminhar o requisitante para se comunicar diretamente com o provedor selecionado [41]. O agente DF de JADE atende a este padrão. A Figura 41 demonstra o conceito de um agente *Matchmaker*.

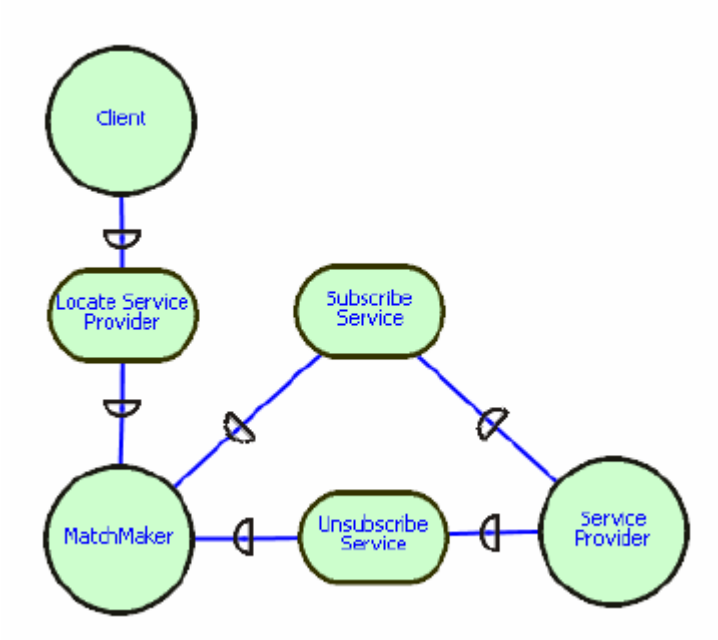


Figura 41 - Padrão Matchmaker [41]

7.4 Projeto Detalhado

O modelo de agentes e seus relacionamentos, apresentado na Figura 42, é o resultado da definição estabelecida no modelo arquitetural representado na Figura 37. As relações modeladas entre os agentes indicam a necessidade de comunicação entre eles para se alcançar as metas de cada um dentro do contexto definido para o estudo de caso.

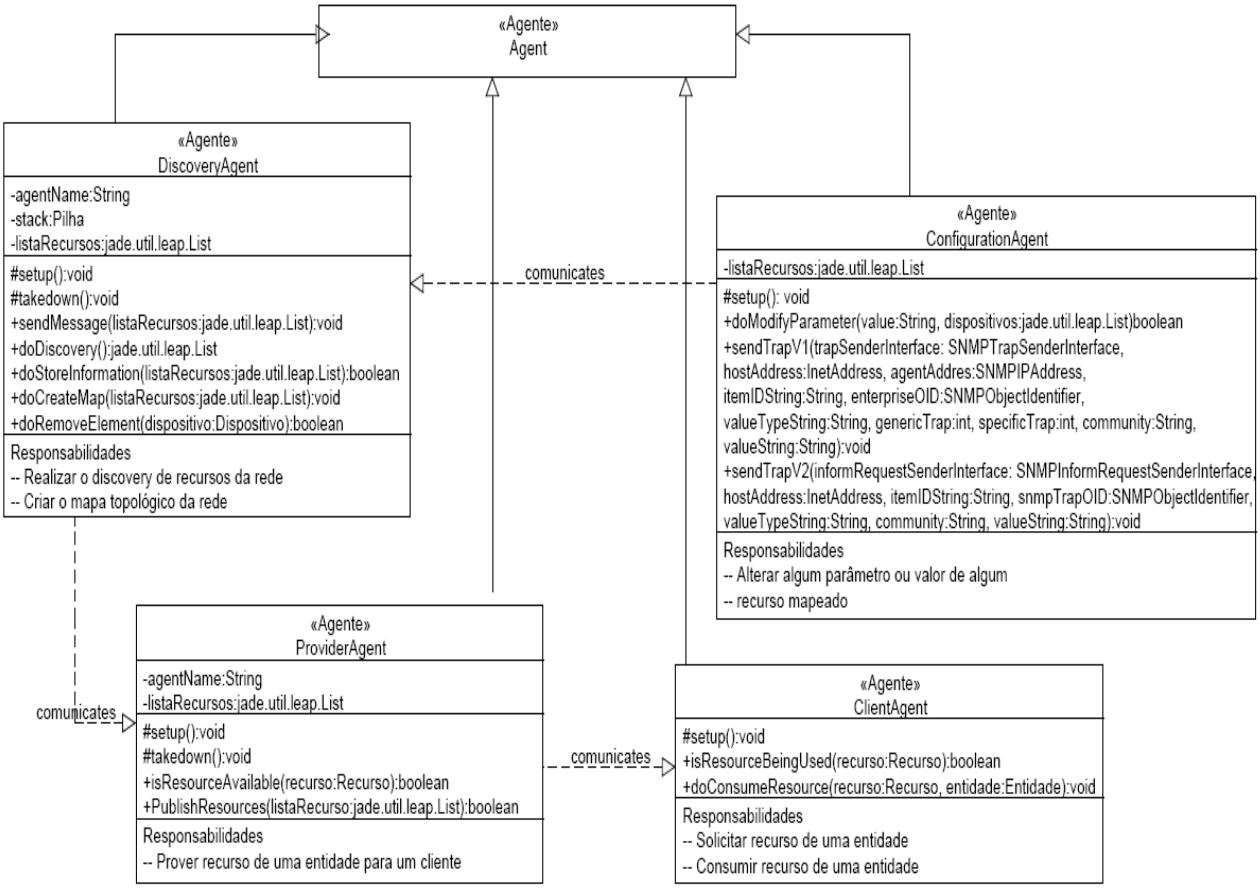


Figura 42 - Modelo de Agentes Proposto

Agentes em JADE, que utilizem ontologias, possuem suas ações mapeadas através do modelo conceitual definido na ferramenta de modelagem ontológica. Tais ações são derivadas da classe *jade.core.AgentAction* e representam uma determinada ação sobre algum conceito definido, que é realizada pelo agente. Por exemplo, considerando o modelo de ontologia definido em [58] e apresentado no Capítulo 5, a ação de localizar um endereço destino na tabela de rotas de um roteador pode ser representada da seguinte forma: (*DeterminarRota (Rota :destino "200.241.1.30") (Roteador :fabricante Cisco :numPortas 16)*). Neste caso, *DeterminarRota* constitui uma classe derivada de *AgentAction* e com métodos que permitam o acesso aos conceitos com os quais esta ação se relaciona. Nesse exemplo, os conceitos são *Rota* e *Roteador*.

As ações dos agentes, para serem executadas, dependem do comportamento definido para cada agente. São estas ações que irão manipular os conceitos modelados na ontologia de modo que o conteúdo das mensagens trocadas entre os agentes sejam objetos de tais ações. Estes, por sua vez, irão

conter os objetos referentes aos conceitos que cada ação manipula, como no exemplo da ação de *Discovery*. Tal ação poderá manipular os Conceitos de *Dispositivo*, *Ativo*, *Roteador*, *Host*, *Switch*, *Servidor*, *Serviço*, por exemplo. Desta forma, um objeto referente à classe da ação *Discovery* poderá conter objetos correspondentes a quaisquer desses conceitos. Tal característica permite ao agente trafegar mensagens com o conteúdo relacionado especificamente à ontologia utilizada.

A Figura 43 ilustra o diagrama de classes de ações mapeadas para o modelo de agentes proposto e de acordo com a ontologia definida em no capítulo 5 [58].

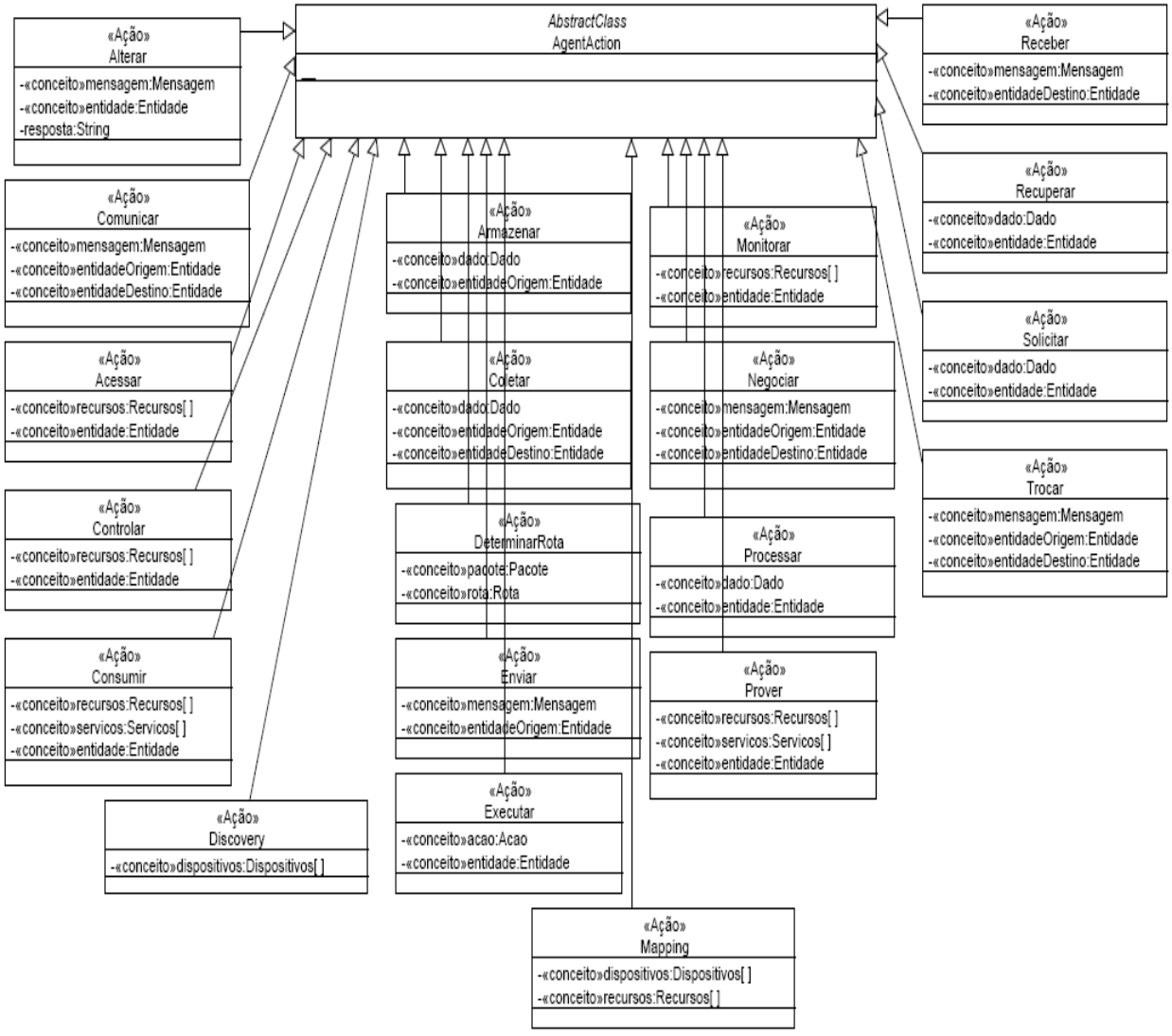


Figura 43 - Diagrama de Classes de Ações dos Agentes

Um comportamento em JADE corresponde à uma tarefa específica a ser desempenhada pelo agente. Tratam-se de subclasses de *jade.core.behaviours.Behaviour* que deverão ser instanciadas e adicionadas ao agente, através do método *public void addBehaviour(Behaviour comportamento)*.

Uma classe derivada de *Behaviour* possui dois métodos principais: *action()* e *done()*. O método *action()* é responsável por realizar ações que serão desempenhadas pelo agente através deste comportamento. O método *done()* tem a função de informar ao escalonador do agente se o comportamento executado terminou ou não. Caso o comportamento em questão não tenha sido encerrado, o método *action()* é novamente executado.

Em JADE, existem dois tipos de classes derivadas de *Behaviour* que definem um estilo comportamental: *SimpleBehaviour*, utilizada para modelar comportamentos únicos; e *CompositeBehaviour*, utilizada para modelar ações complexas. Na hierarquia de *SimpleBehaviour*, há a especialização em *OneShotBehavior*, no qual se modela uma tarefa que deve ser executada apenas uma vez e não pode ser bloqueada; e *CyclicBehaviour*, que define um comportamento simples que deve ser executado sempre. Os comportamentos associados às tarefas de *Discovery* (*HandleDiscoveryBehaviour*) e de configuração de algum parâmetro da rede (*HandleConfigureBehaviour*) constituem tarefas únicas e, deste modo, comportamentos do tipo *OneShotBehaviour*. Já os comportamentos relacionados à requisição (*HandleRequestResourceBehaviour*) e disponibilização (*HandleProvideResourceBehaviour*) de serviços são definidos como comportamentos simples e que se repetem, caracterizando desta forma comportamentos do tipo *CyclicBehaviour*.

A Figura 44 apresenta os comportamentos mapeados para o modelo proposto e a associação dos mesmos aos agentes definidos, conforme o parágrafo anterior.

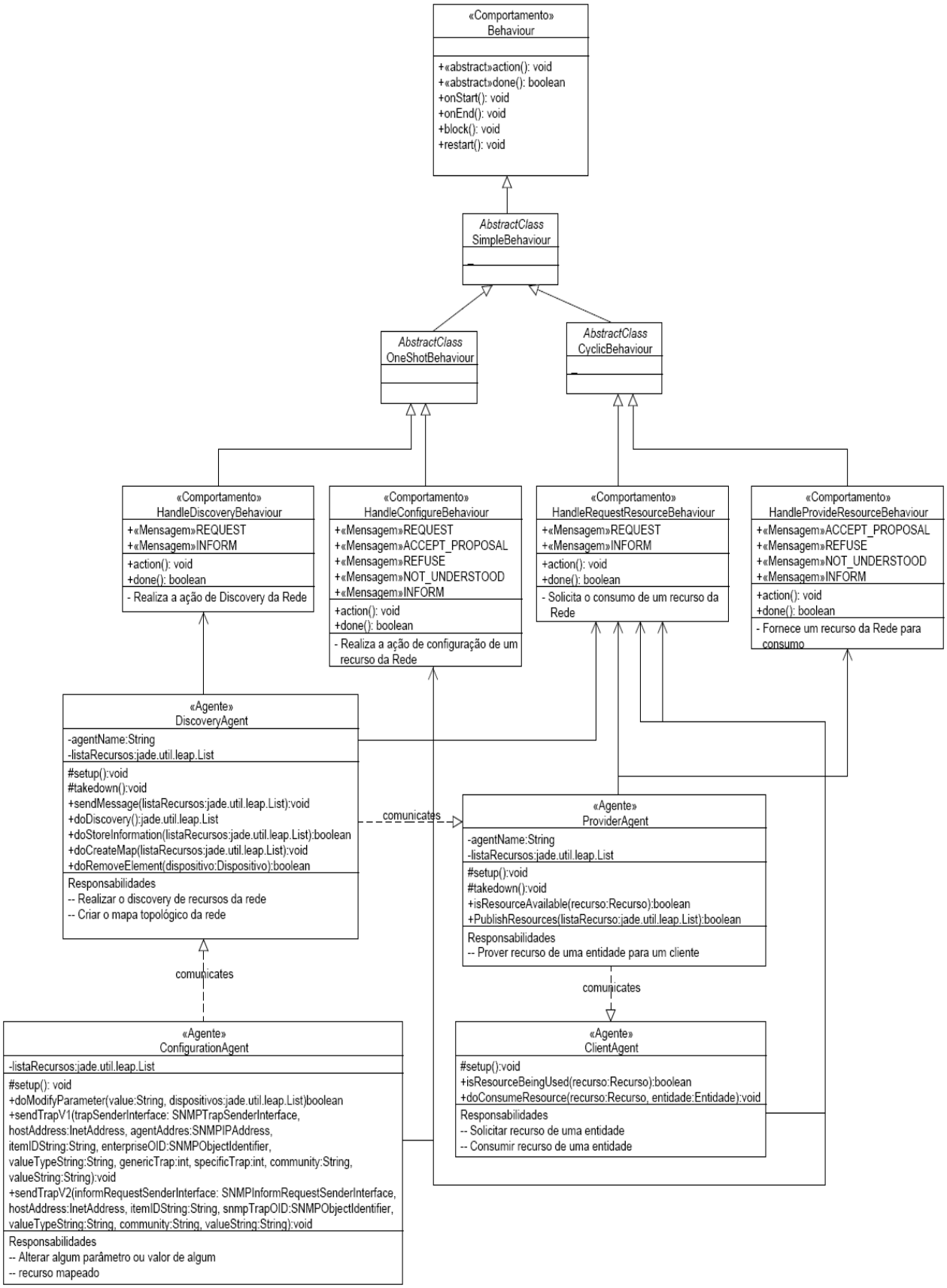


Figura 44 - Diagrama de Classes dos Comportamentos dos Agentes

7.5 Implementação em JADE

A implementação em JADE foi realizada a partir de um protótipo construído para o estudo de caso em questão. Para cada agente identificado no modelo arquitetural e especificado no projeto detalhado, foram criadas as classes correspondentes derivadas de *Agent* e cada um deles foi contruído em um pacote próprio. Desta forma, tem-se:

- ***DiscoveryAgent*** – Localizado no pacote *br.ufes.agentes.discovery*, juntamente com as classes *DataCollection* e *SearchControl*. Tais classes possuem métodos auxiliares às tarefas implementadas pelo agente de *Discovery*.
- ***ProviderAgent*** – Localizado no pacote *br.ufes.agentes.provider*, juntamente com a classe *PublishResource*. Tal classe também possui métodos auxiliares às tarefas implementadas pelo agente provedor.
- ***ConfigurationAgent*** – Localizado no pacote *br.ufes.agentes.configuration*.
- ***ClientAgent*** – Localizado no pacote *br.ufes.agentes.client*.

Os comportamentos modelados foram implementados em um pacote separado, *br.ufes.agentes.comportamentos*, de modo que possam ser reutilizados por agentes que tenham capacidades semelhantes.

As tabelas 4 e 5 apresentam, respectivamente, parte da estrutura da classe do agente *DiscoveryAgent* e de um de seus comportamentos, o *HandleDiscoveryBehaviour*.

```
package br.ufes.agentes.discovery;


import java.net.InetAddress;
import java.net.SocketTimeoutException;
import mestrado.SNMPConfig;


import br.ufes.pilhas.Pilha;
```

```
import snmp.SNMPVarBindList;
import snmp.SNMPv1CommunicationInterface;
import br.ufes.agentes.comportamentos.HandleDiscoveryBehaviour;
import br.ufes.agentes.provider.ProviderAgent;
import br.ufes.ontologia.*;
import jade.core.AID;
import jade.core.Agent;

import jade.content.lang.sl.SLCodec;
import jade.content.onto.basic.Action;
import jade.domain.DFService;
import jade.domain.FIPANames;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.lang.acl.ACLMessage;
import br.ufes.agentes.Constants;

 public class DiscoveryAgent extends Agent {

 // Atributos do agente
  public static String agentName;
  private Pilha stack = new Pilha();
  private jade.util.leap.List listaRecursos = new jade.util.leap.ArrayList();

// Métodos do agente
public void sendMessage(jade.util.leap.List lista) { ... }

public jade.util.leap.List doDiscovery () { ... }
protected void setup() {
            try {
                // Create the agent descrption of itself
```

```
            DFAgentDescription dfd = new DFAgentDescription();
            dfd.setName(new AID(Constants.AGENT_DISCOVERY_NAME,
AID.ISLOCALNAME));
            // Register the description with the DF
            DFService.register(this, dfd);
        } catch (Exception e) {
            e.printStackTrace();
        }
   agentName = this.getLocalName();
   try {
        SNMPConfig.container = (jade.wrapper.AgentContainer)getContainerController();
        this.getContentManager().registerLanguage(new SLCodec(),
FIPANames.ContentLanguage.FIPA_SL0);
        this.listaRecursos = this.doDiscovery();
        this.prepareRequestMessage(listaRecursos);
        addBehaviour(new HandleDiscoveryBehaviour(this));
        addBehaviour(new HandleRequestResourceBehaviour(this));
   } catch (Exception ex) {
      ex.printStackTrace();
   }
 }
```

Tabela 4 - Trecho da classe *DiscoveryAgent*

```
package br.ufes.agentes.comportamentos;


import br.ufes.agentes.provider.ProviderAgent;
import br.ufes.agentes.discovery.DiscoveryAgent;
import br.ufes.ontologia.ConexaoIndisponivel;
import br.ufes.ontologia.Discovery;
import br.ufes.ontologia.Dispositivo;
```

```
import jade.core.AID;
import jade.core.Agent;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.util.leap.Iterator;

import jade.core.behaviours.SimpleBehaviour;
import jade.content.ContentElementList;
import jade.content.onto.basic.Action;
import jade.content.onto.basic.TrueProposition;



public class HandleDiscoveryBehaviour extends OneShotBehaviour {

 private final static MessageTemplate mt =
                                    MessageTemplate.MatchPerformative(ACLMessage.REQUEST);

 public HandleDiscoveryBehaviour(Agent agent) {

  super(agent);
 }

 public ACLMessage prepareResponse(ACLMessage request, jade.util.leap.List lista) { ...}
 public void action() {
        while (true) {

           ACLMessage aclMessage = myAgent.receive(mt);
           if (aclMessage != null) {
              try {
                    this.prepareResponse(aclMessage,
```

```
                                                    (DiscoveryAgent)myAgent).getListaRecursos());
                    } catch (Exception ex) {


                         ex.printStackTrace();
                    }


                } else {
                    this.block();
                }
            }
 }
}
```

Tabela 5 - Trecho da classe *HandleDiscoveryBehaviour*

7.6 Arquitetura do Sistema

O protótipo foi construído utilizando-se a linguagem de programação **Java**. Para a camada de persistência, utilizou-se o banco de dados ***MySQL* 5.0**, com o *framework* ***Hibernate***, ambos pertencentes à comunidade de *softwares* livres.

O Sistema **O-bCNMS** foi implementado segundo o modelo de três camadas, conforme a Figura 45. Na primeira, tem-se a camada de apresentação, correspondente à interface com o usuário. A camada intermediária refere-se à plataforma de agentes, na qual os agentes construídos segundo o *framework* **JADE** se comunicam e trocam informações referentes à terceira camada. Esta corresponde à camada de negócios. Nesta camada está definido todo o modelo ontológico utilizado e os principais requisitos definidos para o sistema: *discovery*, publicação de recursos e configuração de ativos.

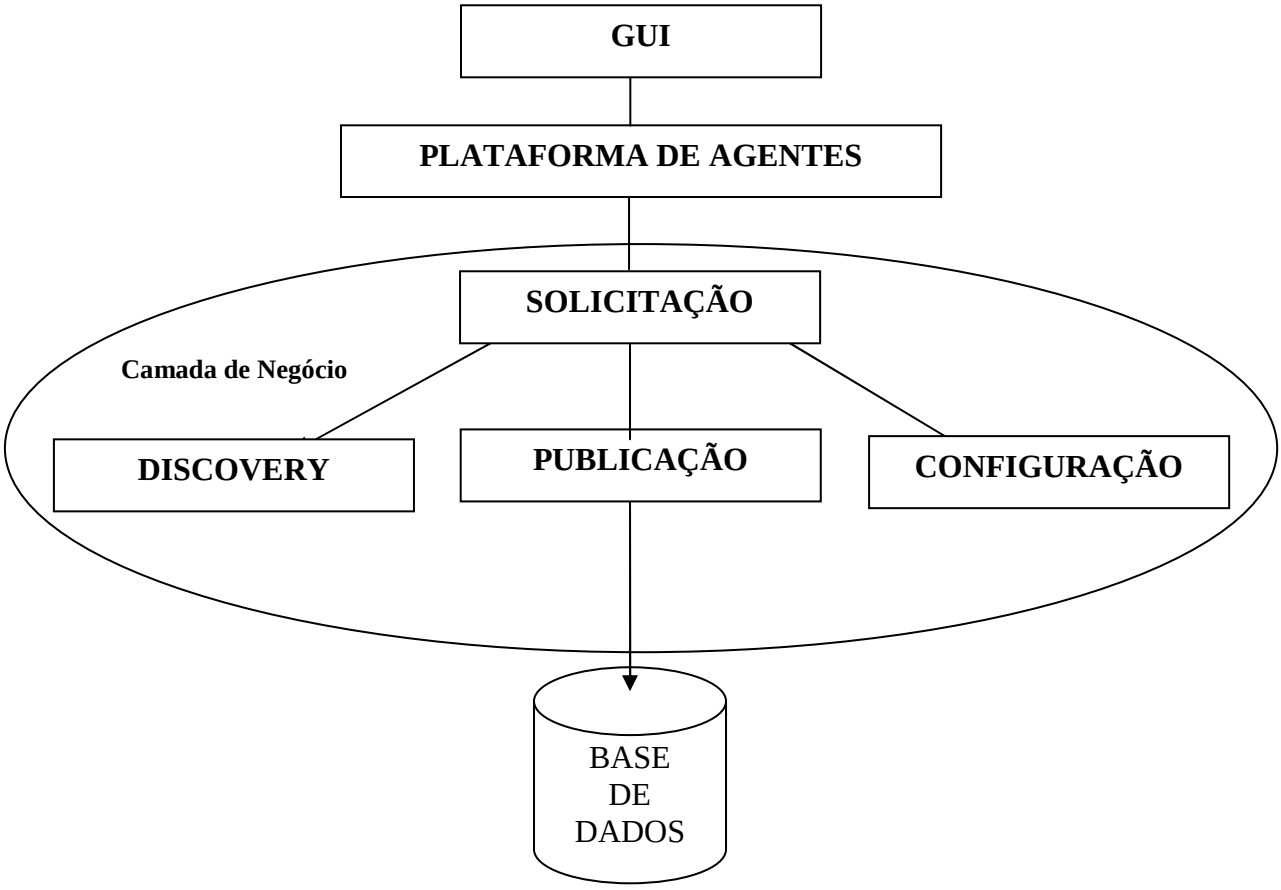


Figura 45 - Arquitetura do Sistema O-bCNMS

As operações de persistência ocorrem a partir da camada de negócio, através do agente responsável pela publicação dos recursos providos pela rede. Para tanto, o agente faz uso do *framework* ***Hibernate***, responsável por realizar a conexão com a base de dados como também por efetuar as atualizações e inserções de informações na mesma.

7.7 Modelo de Interface

O modelo da interface do protótipo do sistema **O-bCNMS** utiliza-se de abas para distinguir as funcionalidades definidas para o *software*.

A primeira aba refere-se ao Agente de *Discovery* da rede. Nessa tela, ao se clicar no botão **DISCOVERY**, o agente em questão é invocado e este inicia o

processo de busca por ativos. Cada dispositivo encontrado terá seu endereço armazenado na *list box*, cujo label é **ENDEREÇOS ENCONTRADOS**. Se desejar interromper o processo, basta pressionar o botão **INTERROMPER**, que irá fazer com que o referido agente seja finalizado. Na área de texto, o andamento do processo de busca vai sendo informado ao usuário. Os valores padrões para iniciar o procedimento são a comunidade ***public***, que é o valor padrão para o protocolo SNMP, e o endereço de ***loppback***. Ambos podem ser modificados pelo usuário.

Caso o usuário queira armazenar os dados dos ativos obtidos pelo Agente de *Discovery*, ele deve clicar no botão **ARMAZENAR**, conforme a Figura 46.

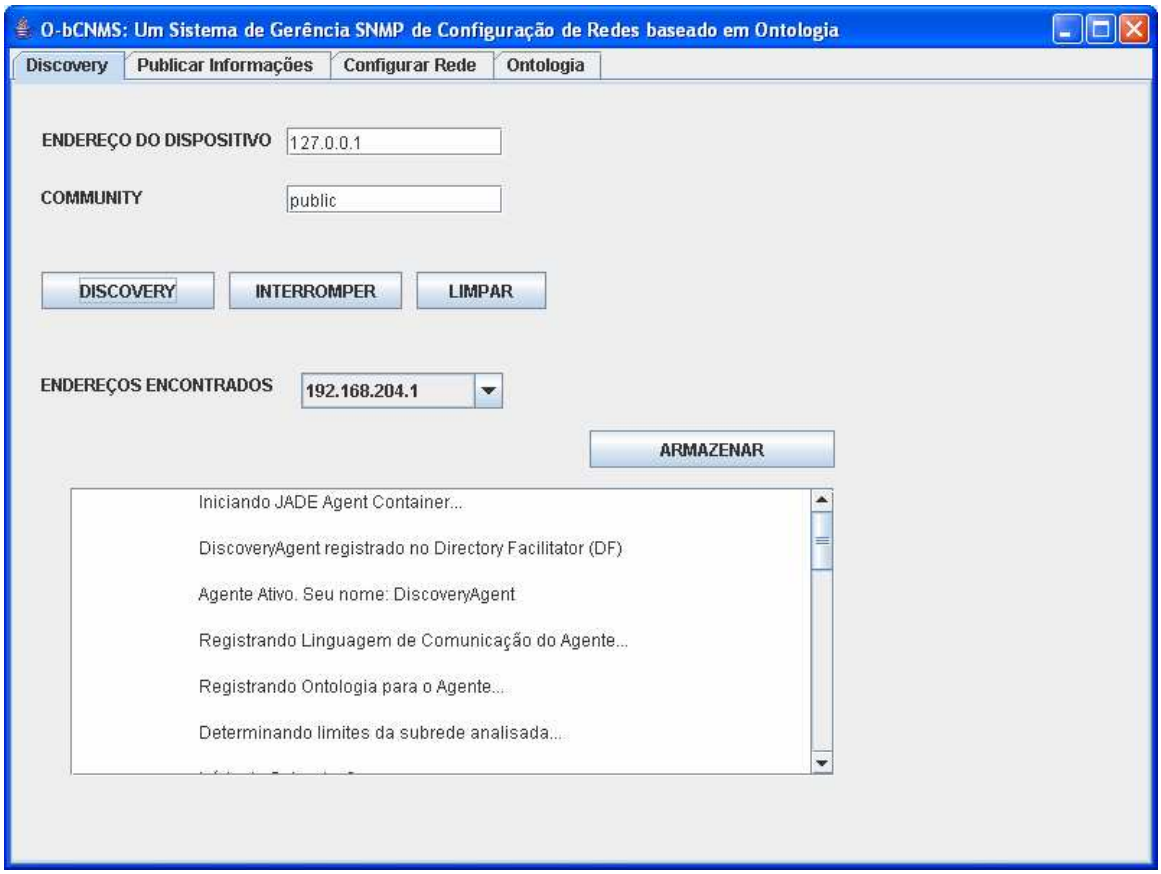


Figura 46 - Protótipo O-bCNMS - *Discovery*

A segunda aba é referente à publicação de informações de um ativo específico da rede. Dessa forma, o usuário deve selecionar no *list box* , o endereço do dispositivo que deseja visualizar seus dados e clicar no botão **EXIBE**. As informações serão apresentadas separadas por janelas.

Na primeira janela, aparecem as informações gerais do ativo selecionado. Na segunda, localizada abaixo da primeira, surgem os dados relaciondos à tabela de rotas do elemento. Na terceira, mais à direita, aparecem os dados das Interfaces do respectivo ativo. A Figura 47 ilustra tais afirmações.

Caso o usuário queira visualizar os serviços ativos no elemento em questão, basta clicar no botão **APLICAÇÕES**. Já se o objetivo for listar os processos ativos em memória no momento, o clique deverá ser no botão **PROCESSOS**. Em ambos os casos, os dados serão exibidos na janela mais à direita.

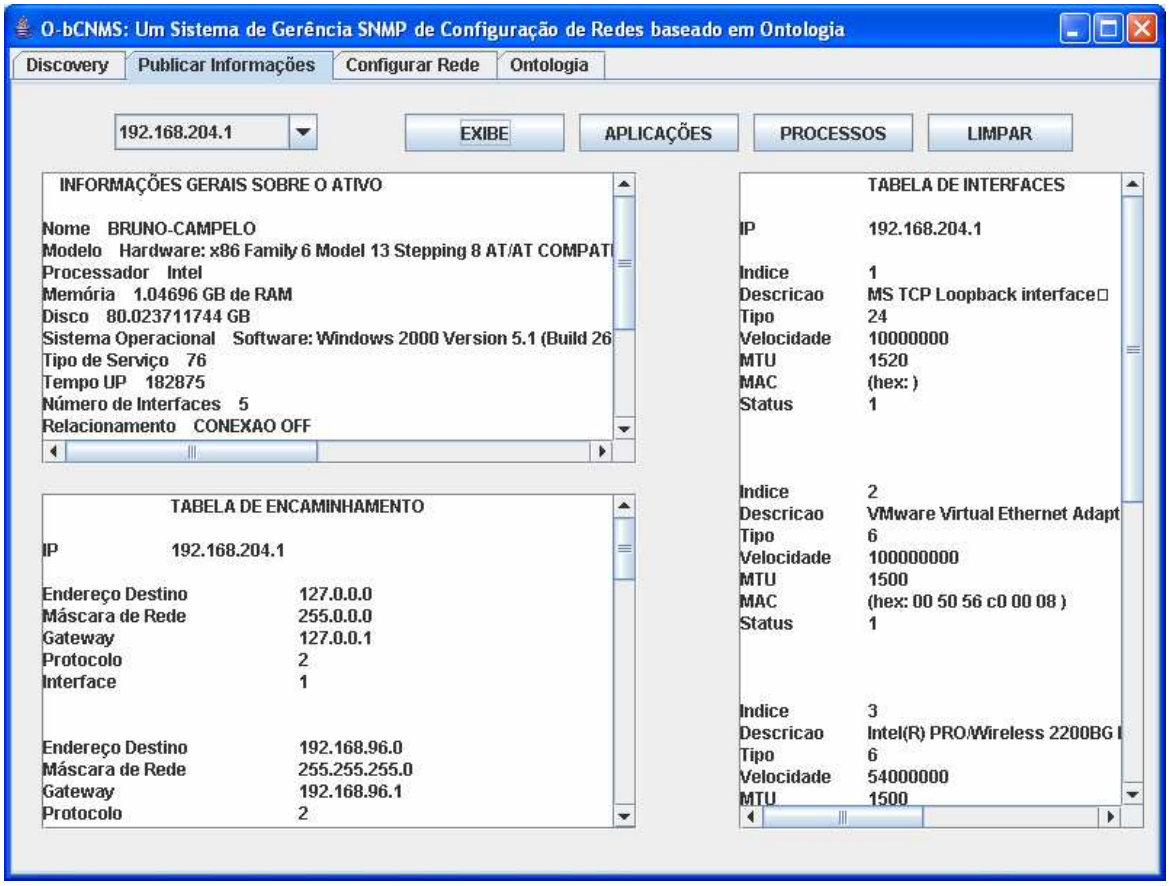


Figura 47 - Protótipo O-bCNMS – Publicar Informações

A terceira aba refere-se ao processo de configuração. O usuário deve informar o endereço do elemento que deseja alterar, bem como a comunidade SNMP que conceda tal permissão, de acordo com a Figura 48. Além disso, deve fornecer o novo valor a ser instanciado e o campo que deseja modificar.

Como o próprio protocolo SNMP possui restrições quanto aos objetos que podem ser submetidos à operação *SET*, o usuário deverá selecionar na *list box* **CAMPO**, uma das opções possíveis. Ao clicar no botão **CONFIGURAR**, o agente de Configuração é iniciado e o usuário é informado, na área de texto, do sucesso ou, no caso de alguma restrição imposta pelo protocolo SNMP ou pelo sistema operacional em uso, da impossibilidade de execução do processo.

Caso o usuário deseje fazer um teste em algum ativo da rede mediante um envio de uma mensagem ao mesmo, deverá clicar no botão **ENVIAR TRAP**. Esta ação irá abrir uma nova janela para que o usuário informe os dados do ativo e da mensagem *TRAP* a ser enviada. *TRAP* é uma operação do protocolo SNMP que se caracteriza por ser uma solicitação feita ao agente residente no ativo de rede, de forma que este deve respondê-la em detrimento à atividade que esteja executando naquele momento. Ao clicar no botão **ENVIAR TRAP SNMPv1** ou **ENVIAR SNMPv2 INFORM REQUEST**, o agente de configuração é invocado para realizar esta tarefa de enviar um *TRAP* ao ativo determinado. Os dados de envio e recebimento de tal mensagem são exibidos na área de texto **TRAPS RECEBIDOS**, conforme a Figura 49.

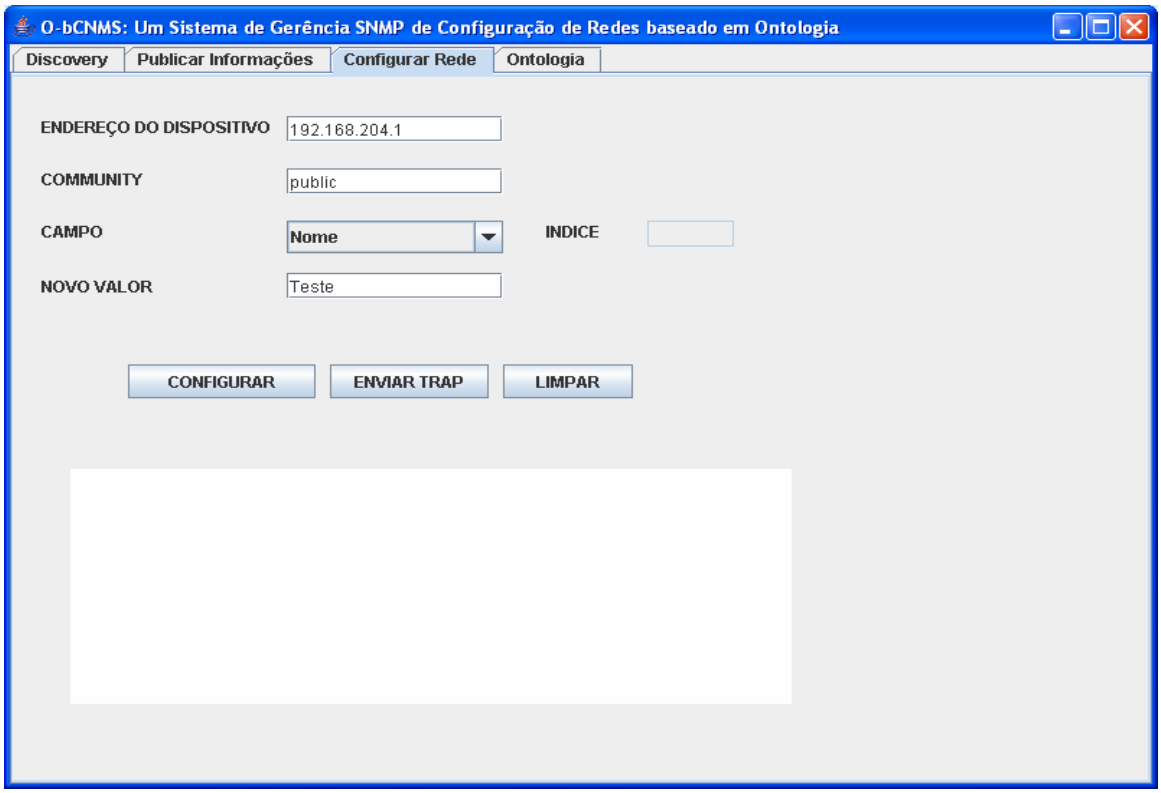


Figura 48 - Protótipo O-bCNMS - Configurar

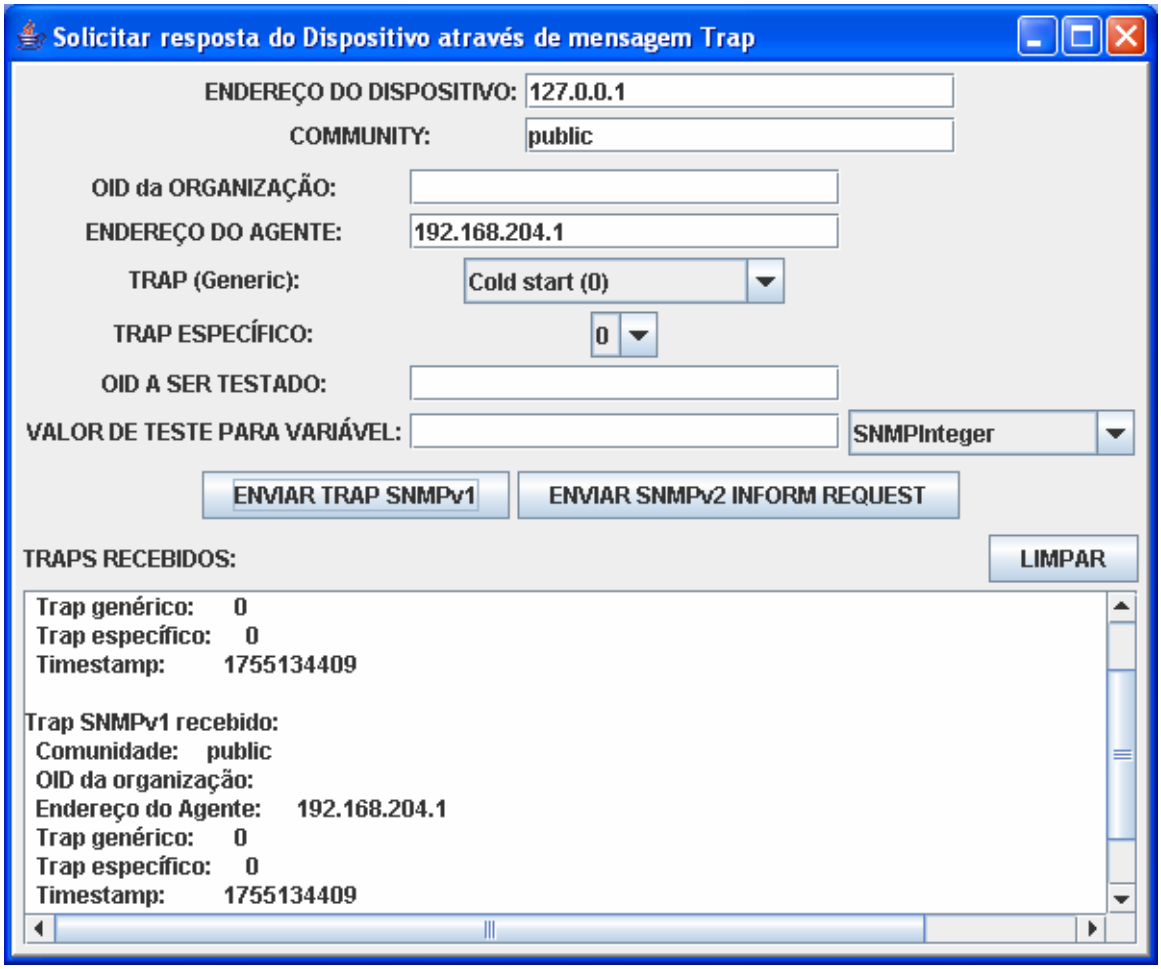


Figura 49 - Protótipo O-bCNMS – Configurar – Enviar TRAP ao Dispositivo

A última aba refere-se à exibição do conteúdo da mensagem trocada pelos agentes. O intuito é apresentar o uso do modelo da ontologia na camada de negócio, uma vez que encapsulada em uma ação de um agente, estarão os conceitos modelados na ontologia definida para o sistema **O-bCNMS**. Deste modo, a mensagem mostra as informações trocadas entre os agentes, mapeadas em dados que caracterizam objetos de classes referentes ao modelo ontológico definido para uso pelos agentes do protótipo.

Para visualizar o conteúdo da mensagem, o usuário deve clicar no botão **VER MENSAGEM**, conforme a Figura 50.

Figura 50 - Protótipo O-bCNMS – Ontologia

7.8 Considerações Finais

Este capítulo apresentou o estudo de caso produzido para esta dissertação, o sistema **O-bCNMS**.

A plataforma JADE mostrou-se, ao longo do desenvolvimento do protótipo, eficiente e de fácil aprendizagem para uso de agentes. Porém, o foco adotado foi definir inicialmente o modelo de ontologia e, em função deste, modelar um sistema de agentes que pudesse fazer uso de tal modelo. Diante deste quadro, o protótipo pode ser evoluído tanto na camada de apresentação que se refere à interface do *software*, como na criação de funcionalidades adicionais a serem agregadas ao sistema.

A escolha por se trabalhar o protótipo com agentes justifica-se pelo caráter dinâmico e independente que estes apresentam, principalmente no que

tange ao modelo de comunicação entre os mesmos, como também pela capacidade de reuso que agentes agregam ao sistema, por meio da definição de suas tarefas como comportamentos que podem ser reutilizados por outros posteriormente.

CONCLUSÕES E TRABALHOS FUTUROS

8.1 Considerações Finais

Esta dissertação tem como foco principal propor um modelo capaz de agregar informações que possibilite o controle do ambiente gerenciado, de modo a se conhecer a estrutura monitorada visando obter um nível de qualidade de serviço estabelecido. Diante da análise deste problema e de um estudo voltado para uma proposta de trabalho que tem como meta um controle dinâmico da rede a ser monitorada, chegou-se ao caminho de se buscar a constituição de um modelo de informações compartilhada e com um conteúdo semântico único dentro do contexto em que está inserido. Desta forma, foi definido o uso de ontologias para se contruir uma base voltada à gerência de configuração de redes. Uma vez definido o modelo ontológico e devido principalmente ao caráter dinâmico do ambiente, chegou-se à tecnologia de agentes para realizar as interações com o meio em busca das informações necessárias para se efetuar a gestão. Este trabalho faz uma associação das disciplinas de engenharia de software voltada a agentes, engenharia de ontologias e gerência de redes.

A engenharia de software apresenta fases, definidas para o processo de modelagem e construção de *software*, bem caracterizadas e com insumos e artefatos estabelecidos para cada uma delas, conforme [53]. No entanto, a engenharia de software voltada para agentes atua mais fortemente na modelagem

comportamental e na definição das interações entre os agentes envolvidos. Neste trabalho, utiliza-se um *framework* próprio para a construção de agentes, o JADE, que torna possível a aplicação de características fundamentais, como necessidade de comunicação entre os agentes, de forma mais transparente e permite, ao desenvolvedor, a definição de comportamentos através do uso de conceitos da orientação a objeto, como classes e especialização (herança), por exemplo. Tal abordagem facilita o processo de projeto e construção de agentes.

O uso de agentes tem sido algo crescente quando o objetivo envolve monitoração de ambientes e equipamentos. No entanto, apenas a utilização de agentes não é suficiente para se garantir um controle mais apurado da estrutura que se pretende gerenciar. É fundamental que, associado ao comportamento dinâmico e interativo destas entidades, haja uma lógica construída de modo a garantir o que, de fato, cada agente deve realizar no contexto em que está localizado. Neste trabalho, é destacada a forma como o JADE se apresenta para construção desses agentes bem como o projeto de agentes em que cada um possui um papel definido dentro do sistema voltado para o objetivo principal, que é proporcionar uma gerência de configuração da rede.

A gerência de configuração, que tem como uma de suas principais preocupações o controle dos recursos providos pelo ambiente, além de dados sobre suas relações lógicas, é explorada nesta dissertação como a principal meta a ser alcançada. Desta forma, pensou-se em um modelo que pudesse fornecer este controle por meio de uma base de informações única, independente de protocolo de gerenciamento, que possa ser dinâmica e confiável. A opção por ontologias justifica-se principalmente por esta apresentar tais características.

A engenharia de ontologias inicia-se pela construção de modelos de domínio de conhecimento. Entretanto, o desenvolvimento de tais estruturas em domínios como de gerência de redes, impõe obstáculos que demanda a aplicação de uma engenharia de domínio bem estruturada. Tais obstáculos estão relacionados principalmente às particularidades e restrições impostas por cada padrão de linguagem para extração de informações de gestão. Neste trabalho, adotou-se o método de divisão de engenharia de domínio em fases de análise e projeto do domínio estudado, sendo que esta última foi feita baseada na modelagem orientada a objetos para concepção da ontologia, conforme o padrão adotado pelo JADE.

O uso deste método se mostrou adequado e produtivo por permitir, em um primeiro momento, o mapeamento de requisitos a ser atendido pelo modelo proposto, e, em um segundo momento, a identificação de conceitos e sua definição semântica, o que proporcionou a criação de um modelo ontológico através do uso de diagramas de classes. Tal prática pôde permitir uma maior compreensão e definição dos relacionamentos entre os conceitos modelados.

A análise de domínio busca a produção de um modelo do contexto estudado que identifique e organize os conceitos referentes à representação do conhecimento a ser modelado. Neste trabalho, os principais conceitos voltados para gerência de configuração de redes são apresentados como classes. Estas por sua vez serão instanciadas pelos agentes, que serão responsáveis por transportar informações de acordo com o modelo ontológico proposto. Dessa forma, foi possível fazer com que o conteúdo das mensagens trocadas entre os agentes seja aderente ao modelo da ontologia proposta.

A ontologia proposta produz um modelo de domínio conceitual, composto por conceitos com uma semântica associada, suas propriedades e os relacionamentos entre os mesmos. Sendo assim, consegue-se produzir uma análise de domínio. Além disso, como conseqüência do modelo orientado a objeto gerado, tem-se o projeto de domínio focado em uma abordagem de acordo com os padrões da UML [7].

Como forma de representação e formalização da ontologia, é utilizada a ferramenta Protégé [54], que possibilita uma interface gráfica, com características de ferramenta CASE, para descrever a estrutura dos conceitos, suas propriedades (*slots*), seus relacionamentos e axiomas. A ferramenta se mostrou uma boa opção, pois se trata de um projeto aberto, permite a definição da ontologia em diferentes linguagens, como DAML+OIL por meio de OWL, embora OWL não seja uma linguagem muito representativa para formalizar um modelo de ontologia. No entanto, o uso do Protégé [54] no presente trabalho, justifica-se principalmente por este possuir uma integração com o *framework* JADE, de maneira que o modelo conceitual é gerado de acordo com o padrão deste, para uso de ontologias. Ou seja, os conceitos, suas propriedades, restrições e relacionamentos entre estes são construídas com os conceitos de diagrama de classes, próprios da UML [7].

A análise de domínio deve produzir uma ontologia possível de ser utilizada por outras aplicações que atuem no referido contexto. Para este trabalho, essa reutilização é provida através da definição de conceitos próprios do domínio de conhecimento Gerência de Configuração de Redes. A partir deste mapeamento, foi possível a criação de um modelo orientado a objetos que representasse esses mesmos conceitos e as relações entre eles. Desse modo, quaisquer agentes modelados com o *framework* JADE podem fazer uso de tal modelo ontológico proposto.

Com relação ao projeto de domínio dois são os objetivos: a escolha de que elementos do modelo conceitual deverão aparecer no modelo real de implementação e a efetiva execução desse processo através do mapeamento da estrutura da ontologia, ou seja, seus conceitos, relações e axiomas, em elementos do paradigma orientado a objeto, como classes, atributos e relacionamentos, de forma a se produzir como artefato o código referente à ontologia a ser utilizada. O uso da linguagem Java justifica-se por ser o padrão do *framework* JADE além de ser um padrão aberto e amplamente difundido no mercado.

O domínio de gerência de configuração é escolhido para ser o estudo de caso dessa dissertação com o intuito de se mostrar a aplicação do modelo proposto em uma situação de controle de um ambiente real e dinâmico. Foi definido o modelo ontológico a ser utilizado e, em seguida, passando pelas etapas de engenharia de software, o conjunto de agentes a ser construído de forma a prover informações da rede de maneira aderente à ontologia proposta. Esta abordagem se mostrou muito satisfatória, uma vez que possibilitou a aplicação do modelo proposto por meio de agentes, o que tornou o processo mais próximo da realidade do problema objeto de estudo deste trabalho.

8.2 - Trabalhos Futuros

Em função da linha de pesquisa adotada neste trabalho, as possibilidades de trabalhos futuros podem ser classificadas em três áreas:

- **Modelo de ontologia no contexto de gerência de configuração**: O modelo proposto é um projeto aberto e possui um caminho a evoluir. Uma opção de

evolução nesse sentido é a extensão desse modelo para que trate de questões relacionadas ao desempenho dos elementos do domínio, de modo a proporcionar um conjunto de informações que possibilite monitorar o funcionamento de cada ativo gerenciado. Outra evolução relevante está na axiomatização desta ontologia, isto é, na definição de axiomas inerentes ao modelo proposto. Outra linha de trabalho a ser desenvolvida está na ampliação desse modelo voltado para ontologia de aplicação, para abranger, além da ontologia de domínio, uma ontologia de tarefas. Com esta expansão, seria possível ter um modelo mais completo, focado não somente nos conceitos que compõem o domínio estudado, mas também nas ações que cada uma dessas entidades é capaz de realizar.

- **Expansão do modelo de agentes**: Os agentes modelados para o sistema **O-bCNMS** possuem um papel bem definido e com comportamentos modelados e implementados segundo a função que cada um deles deve desempenhar no contexto em questão. Dessa forma, surge a possibilidade de um crescimento deste modelo, de modo a englobar novas tarefas aos agentes criados bem como novos agentes com funções distintas das que foram então implementadas. Outra opção interessante de trabalho nessa linha está no desenvolvimento de formas alternativas de comunicação entre os agentes, como, por exemplo, agentes que invocam agentes “filhos” para a execução de subtarefas, ou mesmo agentes que agreguem comportamentos complexos relacionados. Estes, em JADE, referem-se a comportamentos que se subdividem de forma a serem executados em partes independentes. Um exemplo de tal situação seria a coleta de informações, sendo feita de modo paralelo, por meio de diferentes protocolos de gerenciamento. Tal contexto seria útil para um sistema que tivesse como objetivo obter todo o conjunto de informações gerenciais em tempo real e de forma simultânea, utilizando para isso protocolos distintos, como SNMP e CMIP por exemplo.

- **Desenvolvimento de agentes inteligentes**: A utilização de técnicas de Inteligência Artificial (IA) associada a agentes é algo cada vez mais comum nos dias de hoje. Dessa forma, uma evolução relevante desse trabalho poderia ser a criação de agentes inteligentes capazes de tomarem medidas pró-ativas no gerenciamento da rede. Para tanto, eles deverão possuir uma capacidade de

tomada de decisão. Esta deveria estar embasada no modelo ontológico, importante para que o agente reconheça cada dado a ser analisado por ele, e em uma condição de aprendizagem que o agente deverá possuir. Tal condição poderá ser adquirida gradativamente, à medida que o agente reage às situações que surgem no contexto em que está inserido. Trabalhos importantes na área de agentes desenvolvidos com técnicas de IA, como [22] [65], têm sido realizados e sua integração com ontologias aparece como um caminho promissor, como mostra [15] [16] [52] [60] [73].

REFERÊNCIAS BIBLIOGRÁFICAS

[1] BAKER, Albert. "**JAFMAS – A java-based agent framework for multiagent systems. Development and Implementation**". Doctoral thesis. Cincinnati: Department of Electrical & Computer Engineering and Computer Science University of Cincinnati, 1997.

[2] BAUER, B., MULLER, J. P., and ODELL, J.: "**Agent UML: A formalism for specifying multiagent software systems**". International Journal of Software Engineering and Knowledge Engineering, vol. 11, no. 3, pp. 207–230, 2001.

[3] BELLIFEMINE, Fabio, POGGI, Agostino & RIMASSA, Giovanni. "**JADE – A FIPA-compliant agent framework**". Itália: 1999.

[4] BELLIFEMINE, F., CAIRE, G., TRUCCO, T., RIMASSA, G.: **Jade Programmer's Guide**. JADE 3.3. Disponível em: http://jade.tilab.com/doc/programmersguide.pdf, 2005.

[5] BIESZCZAD, Andrzej; PAGUREK, Bernard; WHITE, Tony. "**Mobile Agents for Network Management**". IEEE Communication Survey, 1998.

[6] BOOCH., G.: "**Object-Oriented Analysis and Design**". Second edition. Addison-Wesley: Reading, MA, 1994.

[7] BOOCH, G., RUMBAUGH, J., JACOBSON, I.: "**The Unified Modeling Language User Guide**", The Addison Wesley Object Technology Series, Addison Wesley, 1999.

[8] BUCHANAN, Robert W. , Jr. **The Art of Testing Network Systems**. Wiley Computer Publishing. Canadá, 1996.

[9] COELHO, Diego Bueno. "**Gerenciamento e Manutenção de recursos em redes de computadores através de agentes de software**". Monografia do Projeto de Graduação para o curso de Engenharia de Computação. Universidade Católica Dom Bosco. Campo Grande, MS, maio de 2002.

[10] CIOFFI, Marco. "**Web services e Multi-Agent Systems**". Approfondimento per il corso di Ingegneria del Software 2. Politecnico di Milano, 2005.

[11] DAML Ontology Library. Disponível em: http://www.daml.org/ontologies, acessado em 13 de janeiro de 2007.

[12] DRAGO, Rodrigo B. "**Especificação Funcional de uma Ferramenta para Gerência de Ambientes de TI**". Projeto de Graduação em Engenharia de Computação. UFES. Vitória: 2004.

[13] FALBO, Ricardo A. "**Integração de Conhecimento em um Ambiente de Engenharia de Software**". Tese (Doutorado em Informática) - Engenharia de Sistemas e Computação, COPPE, Universidade Federal do Rio de Janeiro, 1998.

[14] FALBO, R. A.; MENEZES, C.S.; ROCHA, A.R.C.; "**A Systematic Approach for Building Ontologies**". Proceedings of the 6th Ibero-American Conference on Artificial Intelligence, Lisbon, Portugal, Lecture Notes in Computer Science, vol. 1484, 1998.

[15] FALBO, R.A.; GUIZZARDI, G.; DUARTE, K.C.; "**An Ontological Approach to Domain Engineering**".Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, SEKE'2002, pp. 351- 358, Ischia, Italy, 2002.

[16] FALBO, R.A.; NATALI, A.C.C.; MIAN, P.G.; BERTOLLO, G.; RUY, F.B.; "**ODE: Ontology-based software Development Environment**", IX Congreso Argentino de Ciencias de laComputación, p. 1124-1135, La Plata, Argentina, Outubro 2003.

[17] FEIT, Sidnie. "**SNMP: a guide to network Management"**. McGraw-Hill, Inc., USA, 1995.

[18] FERNANDEZ, M., GOMEZ-PEREZ, A. and JURISTO, N. (1997) "**METHONTOLOGY: From Ontological Art Towards Ontological Engineering**", AAAI-97 Spring Symposium on Ontological Engineering, Stanford University, 1997.

[19] FIPA specification XC00061: "**FIPA ACL Message Structure Specification**". Disponível em:http://www.fipa.org, acessado em 07 de dezembro de 2006.

[20] FIPA (The Foundation for Intelligent Agents), Disponível em: http://www.fipa.org, acessado em 21 de julho de 2006.

[21] FRANCESCHI, Analúcia S. Morales de. Exame de Qualificação de Doutorado – "**Desenvolvimento de agentes Autônomos para Gerência de Redes de Computadores"**. Universidade Federal de Santa Catarina. Florianópolis, SC, Dezembro de 1999.

[22] FRANCESCHI, Analúcia S. Morales de.; ROISENBERG, Mauro; BARRETO, Jorge M. "**Desenvolvendo agentes de Software para Gerência de Redes utilizando Técnicas de Inteligência Artificial"**. Universidade Federal de Santa Catarina. Florianópolis, SC, 2001.

[23] GIRARDI, Rosario & FARIA, Carla Gomes de. "**A Generic Ontology for the Specification of Domain Models**".UFMA. São Luís: 2003.

[24] GOMEZ-PEREZ, A., FERNANDEZ, M. and DE VICENTE, A.J. "**Towards a Method to Conceptualize Domain Ontologies**", ECAI-96 Workshop on Ontological Engineering, Budapest, 1996.

[25] GRUBER, Thomas R. "**Toward Principles for the Design of Ontologies used for Knowledge Sharing**". Int. J. Human-Computer Studies, v. 43, n. 5/6, 1995.

[26] GRUNINGER, M. and FOX, M.S. "**The Design and Evaluation of Ontologies for Enterprise Engineering**", Workshop on Implemented Ontologies, European Conference on Artificial Intelligence, Amsterdam, NL, 1994.

[27] GRUNINGER, M. and FOX, M.S. "**The Role of Competency Questions in Enterprise Engineering**", IFIP WG5.7 Workshop on Benchmarking - Theory and Practice, Trondheim, Norway, 1994.

[28] GRUNINGER, M. and FOX, M.S. "**Methodology for the Design and Evaluation of Ontologies**", IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing, Montreal, 1995.

[29] GUDWIN, Ricardo R. “**Introdução à Teoria de agentes”**. DCA-FEEC-UNICAMP. Disponível em: http://www.dca.fee.unicamp.br~/~gudwin/courses/IA009/ , acessado em 12 de julho de 2006.

[30] GUARINO, NICOLA. “**Formal Ontologies and Information System**”. In: FIRST

INTERNATIONAL CONFERENCE (FOIS), 1., 1998, Trento, Itália. Trento: IOS Press, 1998.

[31] GUARINO, NICOLA. “**Formal Ontology, Conceptual Analysis and Knowledge Representation**”. Itália, 1995.

[32] GUARINO, Nicola. “**Understanding, building and using ontologies**”. Int. Journal Human-Computer Studies, v. 45, n. 2/3, fev./mar. 1997.

[33] GUIZZARDI, Giancarlo. “**Desenvolvimento para e com reuso: Um estudo de caso no domínio de Vídeo sob Demanda**”, Dissertação de Mestrado. Mestrado em Informática, UFES, 2000.

[34] GÜRER, Denise; LAKSHMINARAYAN, Vinay; SASTRY, Ambatipudi. “**An Intelligent-Agent-Based Architecture for the Management of Heterogeneous Networks**”. Menlo Park, CA, USA, 1999.

[35] HEGERING, Heinz-Gerd; ABECK, Sebastian; NEUMAIR, Bernhard. “**Integrated Management of Networked Systems – Concepts, Architectures and Their Operational Application”**. Morgan Faufmann Publishers. USA, 1998.

[36] JADE: “**Java Agent Development Framework**”. Disponível em: http://jade.tilab.com/, acessado em 11 de novembro de 2006.

[37] JASPER, Robert; USCHOLD, Mike. “**A Framework for Understanding and Classifying Ontology Applications**”, Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management – KAW’99, Alberta, Canada, 1999.

[38] JENNINGS, Nicholas R., SYCARA, Katia. WOOLDRIDGE, Michael. “**A Roadmap of Agent Research and Development**”. Kluwer Academic Publishers, 1998.

[39] JONES, Dean; BENCH-CAPON, Trevor; VISSER, Pepijn. “**Methodologies for Ontology Development**”. Department of Computer Science, University of Liverpool. Liverpool, 1999.

[40] KBSI “**The IDEF5 Ontology Description Capture Method Overview**”, KBSI Report, Texas, 1994.

[41] KOLP, Manuel, TUNG DO, T., FAULKNER, Stéphane, HANG HOANG, T. T. . “**Introspecting Agent Oriented Design Patterns**”. In: S. K. Chang (Eds), Advances in Software Engineering and Knowledge Engineering, vol. III, World Publishing. Belgium, 2005.

[42] LEINWAND, Allan.; FANG, Karen. “**Network Management: A practical perspective**”. 2nd Ed. Massachusetts: Addison Wesley, 1995.

[43] LOPES, Raquel V. “**Melhores Práticas para a Gerência de Redes de Computadores**”. Dissertação de Mestrado. Coordenação de Pós-Graduação em Informática. UFPB. Campina Grande: 2002.

[44] MOHAN, Kannan & RAMESH, Balasubramaniam. “**Ontology-based Support for Variability Management in Product and Service Families**”. Department of Computer Information Systems, Georgia, USA. 2002.

[45] MONTEIRO, Maxwell E. “**Metodologia Para Gerência Pró-Ativa de Desempenho em Redes de Comunicação de Dados**”. Dissertação de Mestrado. UFES. Vitória: 2000.

[46] MORAIS E SILVA, Leonardo A. De. “**Estudo e Desenvolvimento de Sistemas Multiagentes usando JADE: Java Agent Development Framework**”. Monografia de Conclusão de Curso. Universidade de Fortaleza. Fortaleza: 2003.

[47] MOURA, Ralf Luis, GARCIA, Anilton Salles. “**Network Management - Architecture Based on Changeable Mobile Agents**”. UFES, Vitória. IWT ,2007 – International Workshop on Telecommunications.

[48] NOY, F. Natalya; MCGUINNESS, L. Deborah. “**Ontology Development 101: A Guide to Creating Your First Ontology**”. Stanford University, CA, USA, 2000.

[49] NWANA, Hyacinth S.; “**Software Agents: An Overview**”. Cambridge University Press, UK, 1996.

[50] Ontolingua Ontology Library. Disponível em: http://www.ksl.stanford.edu/software/ontolingua/, acessado em 12 de janeiro de 2007.

[51] PERROW, Graeme S.; HONG, James W.; LUTFIYYA, Hana L. "**The Abstraction and Modelling of Management Agents**". Department of Computer Science,University of Western Ontario, Canada, 1994.

[52] PEZZIN, J.; FALBO, R.A.; "**AgeODE: Uma Infra-estrutura para Apoiar a Construção de agentes para Atuarem no Ambiente de Desenvolvimento de Software ODE**", 4th Ibero-AmericanSymposium on Software Engineering and Knowledge Engineering, JIISIC'2004, Madrid,Spain, November 2004.

[53] PRESSMAN, R. "**Software Engineering: A Practioner's Approach**". McGraw-Hill, 2000.

[54] PROTÉGÉ. **The Protégé Project**. Disponível em: http://protege.stanford.edu, acessado em 10 de janeiro de 2007.

[55] RFC 1213 – "**Management Information Base for Network Management of TCP/IP-based internets: MIB-II"**. Disponível em: http://www.ietf.org, acessado em 16 de janeiro de 2007.

[56] RFC 2790 – "**Host Resources MIB"**. Disponível em: http://www.faqs.org/rfcs/rfc2790.html, acessado em 11 de fevereiro de 2007.

[57] ROISENBERG, M., BARRETO, J. M, de AZEVEDO, F. M. "**Uma Proposta de Modelização para Agentes Autônomos Baseada na Teoria de Sistemas**". In: Anais do 3o Simpósio Brasileiro de Automação Inteligente, Vitória, ES, 1997.

[58] SANTOS, Bruno Campelo L. dos; GARCIA, Anilton S. "**Uma Proposta de um Modelo de Ontologia para Gerência de Configuração de Redes**". Departamento de Informática. UFES. Vitória: 2007.

[59] SANTOS, Bruno Campelo L. dos. "**Proposta de um protótipo para controle de configuração de Rede**". Projeto de Graduação em Engenharia de Computação. UFES. Vitória: 2002.

[60] SANTOS, G.; VILLELA, K.; SCHNAIDER, L.; ROCHA, A.R.; TRAVASSOS, G.H.; "**Building Ontology-based Tools for a Software Development Environment**",

Proc. of the 6th International Workshopon Learning Software Organization, LSO'2004, pp. 19-30, Banff, Canada, July-2004.

[61] SILVA, Ismênia Galvão Lourenço da. "**Projeto e Implementação de Sistemas Multi-agentes: O Caso Tropos**". Dissertação de Mestrado. Universidade Federal de Pernambuco. Recife: 2005.

[62] STALLINGS, William. **SNMP, SNMPv2, SNMPv3, RMON1 e RMON2**. Addison Wesley. Third Edition. USA, 1999.

[63] The GNU Project. "**The GNU General Public License**". Disponível em: http://www.gnu.org/copyleft/lesser.html, , acessado em 20 de setembro de 2006.

[64] TILAB (Telecom Italia Lab), Disponível em: http://www.telecomitalialab.com/, acessado em 10 de outubro de 2006.

[65] TORSUN, I.S. "**Foundations of Intelligent Knowledge-based systems**". London: Academic Press, 1995.

[66] USCHOLD, M. "**Building Ontologies: Towards A Unified Methodology**", Proc. Expert Systems 96, Cambridge, 1996.

[67] USCHOLD, M. "**Converting an Informal Ontology into Ontolingua: Some Experiences**", ECAI-96 Workshop on Ontological Engineering, Budapest, 1996.

[68] USCHOLD, M. and KING, M. "**Towards A Methodology for Building Ontologies**", IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing, Montreal, Canada, 1995.

[69] USCHOLD, M. and GRUNINGER, M. "**Ontologies: Principles, Methods and Applications**", Knowledge Engineering Review, 11(2), 93-137, 1996.

[70] VERGARA, Jorge E. L.; VILLAGRÁ, Victor A.; ASENSIO, Juan I.; BERROCAL, Julio. "**Ontologies: Giving Semantics to Network Management Models**". Universidade Politécnica de Madrid, Spain, 2002.

[71] VERGARA, Jorge E. L.; VILLAGRÁ, Victor A.; BERROCAL, Julio. "**Semantic Management: advantages of using na ontology-based management information meta-model**". Proceedings of the HP Openview Unversity Association Ninth Plenary Workshop. Boblingen, Alemania, 2002.

[72] VERGARA, Jorge E. L.; VILLAGRÁ, Victor A.; ASENSIO, Juan I.; BERROCAL, Julio. "**Gestión Semântica: Aplicando las Ontologias a la Gestión de Red**". Universidad Politécnica de Madrid, España, 2003.

[73] VERGARA, Jorge E. L.; VILLAGRÁ, Victor A.; ASENSIO, Juan I.; BERROCAL, Julio; PIGNATON, Roney. "**Semantic Management: Applications of Ontologies for the Integration of Management Information Models**". Proceedings of the Eighth IFIP/IEEE International Symposium on Integrated Network Management (IM'2003), Colorado, Springs, USA, 2003.

[74] VILLAÇA, Rodolfo. "**Uma Abordagem para Gerência Pró-Ativa de Falhas e Desempenho em um Centro Remoto de Operações de Rede**". Dissertação de Mestrado em Engenharia Elétrica. UFES. Vitória: 2004.

[75] WACHE, H.; V¨OGELE, T.; VISSER, U. ; STUCKENSCHMIDT, H ; SCHUSTER, G.; NEUMANN, H. ; HUBNER, S. "**Ontology-Based Information Integration: A Survey**". University of Bremen, Germany: 2002.

[76] WEISS, Gerhard. "**Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence**", MIT Press, 1999.

[77] YU, E.: "**Modelling Strategic Relationships for Business Process Reengineering**", Ph.D. thesis. Dept. of Computer Science, University of Toronto, 1995.

ANEXO A

CLASSE EM JAVA REFERENTE À ONTOLOGIA

```
package br.ufes.ontologia;


import jade.content.onto.*;

import jade.content.schema.*;

import jade.util.leap.HashMap;

import jade.content.lang.Codec;

import jade.core.CaseInsensitiveString;


/** file: ConfigManagementOntology.java

 * @author Protègè

*/


public class ConfigManagementOntology extends jade.content.onto.Ontology  {


  //NAME


  public static final String ONTOLOGY_NAME = "ConfigManagement";


  // The singleton instance of this ontology


  private static ReflectiveIntrospector introspect = new ReflectiveIntrospector();


  private static Ontology theInstance = new ConfigManagementOntology();


  public static Ontology getInstance() {
```

```
    return theInstance;


  }


  // VOCABULARY


   public static final String APLICACAOINDISPONIVEL="AplicacaoIndisponivel";
   public static final String ERROINTERNO="ErroInterno";
   public static final String SERVICONAOSUPORTADO="ServicoNaoSuportado";
   public static final String VALORNAORECONHECIDO="ValorNaoReconhecido";
   public static final String REDEINALCANCAVEL="RedeInalcancavel";
   public static final String ROTAINDISPONIVEL="RotaIndisponivel";
   public static final String ARGUMENTOAUSENTE="ArgumentoAusente";
   public static final String INFORMACAOINDISPONIVEL="InformacaoIndisponivel";
   public static final String ERRODESCONHECIDO="ErroDesconhecido";
   public static final String CONEXAOINDISPONIVEL="ConexaoIndisponivel";
   public static final String ACAONAOSUPORTADA="AcaoNaoSuportada";
   public static final String NUMEROARGUMENTOSINSUFICIENTE="NumeroArgumentosInsuficiente";
   public static final String PARAMETROAUSENTE="ParametroAusente";
   public static final String NIVELSERVICOABAIXOPERMITIDO="NivelServicoAbaixoPermitido";
   public static final String ACAONEGADA="AcaoNegada";
   public static final String SIMPLEJADEABSTRACTONTOLOGY_CLASS_4="SimpleJADEAbstractOntology_Class_4";
   public static final String DADONAOENCONTRADO="DadoNaoEncontrado";
   public static final String COMUNICACAOINDISPONIVEL="ComunicacaoIndisponivel";
   public static final String ROTAINEXISTENTE="RotaInexistente";
   public static final String VALORNAOSUPORTADO="ValorNaoSuportado";
   public static final String ACESSONEGADO="AcessoNegado";
```

```
public static final String ERROCOMUNICACAO="ErroComunicacao";
public static final String MAPPINGAGENT="MappingAgent";
public static final String DISCOVERYAGENT="DiscoveryAgent";
public static final String COMUNICAR_MENSAGEM="mensagem";
public static final String COMUNICAR_ENTIDADEORIGEM="entidadeOrigem";
public static final String COMUNICAR_ENTIDADEDESTINO="entidadeDestino";
public static final String COMUNICAR="Comunicar";
public static final String MAPPING_DISPOSITIVOS="dispositivos";
public static final String MAPPING="Mapping";
public static final String ENVIAR_MENSAGEM="mensagem";
public static final String ENVIAR_ENTIDADEORIGEM="entidadeOrigem";
public static final String ENVIAR="Enviar";
public static final String EXECUTAR_ENTIDADE="entidade";
public static final String EXECUTAR_ACAO="acao";
public static final String EXECUTAR="Executar";
public static final String COLETAR_ENTIDADEORIGEM="entidadeOrigem";
public static final String COLETAR_DADO="dado";
public static final String COLETAR_ENTIDADEDESTINO="entidadeDestino";
public static final String COLETAR="Coletar";
public static final String PROVER_ENTIDADE="entidade";
public static final String PROVER_SERVICO="servico";
public static final String PROVER_RECURSOS="recursos";
public static final String PROVER="Prover";
public static final String PROCESSAR_ENTIDADE="entidade";
public static final String PROCESSAR_DADO="dado";
public static final String PROCESSAR="Processar";
public static final String TROCAR_MENSAGEM="mensagem";
public static final String TROCAR_ENTIDADEORIGEM="entidadeOrigem";
public static final String TROCAR_ENTIDADEDESTINO="entidadeDestino";
public static final String TROCAR="Trocar";
```

```
public static final String ACESSAR_ENTIDADE="entidade";
public static final String ACESSAR_RECURSOS="recursos";
public static final String ACESSAR="Acessar";
public static final String CONSUMIR_ENTIDADE="entidade";
public static final String CONSUMIR_SERVICO="servico";
public static final String CONSUMIR_RECURSOS="recursos";
public static final String CONSUMIR="Consumir";
public static final String RECEBER_MENSAGEM="mensagem";
public static final String RECEBER_ENTIDADEDESTINO="entidadeDestino";
public static final String RECEBER="Receber";
public static final String MONITORAR_ENTIDADE="entidade";
public static final String MONITORAR_RECURSOS="recursos";
public static final String MONITORAR="Monitorar";
public static final String ARMAZENAR_ENTIDADEORIGEM="entidadeOrigem";
public static final String ARMAZENAR_DADO="dado";
public static final String ARMAZENAR="Armazenar";
public static final String DETERMINARROTA_PACOTE="pacote";
public static final String DETERMINARROTA_ROTA="rota";
public static final String DETERMINARROTA="DeterminarRota";
public static final String NEGOCIAR_MENSAGEM="mensagem";
public static final String NEGOCIAR_ENTIDADEORIGEM="entidadeOrigem";
public static final String NEGOCIAR_ENTIDADEDESTINO="entidadeDestino";
public static final String NEGOCIAR="Negociar";
public static final String ALTERAR_MENSAGEM="mensagem";
public static final String ALTERAR_ENTIDADE="entidade";
public static final String ALTERAR_RESPOSTA="resposta";
public static final String ALTERAR="Alterar";
public static final String RECUPERAR_ENTIDADE="entidade";
public static final String RECUPERAR_DADO="dado";
public static final String RECUPERAR="Recuperar";
```

```
    public static final String DISCOVERY_DISPOSITIVOS="dispositivos";
    public static final String DISCOVERY="Discovery";
    public static final String CONTROLAR_ENTIDADE="entidade";
    public static final String CONTROLAR_RECURSOS="recursos";
    public static final String CONTROLAR="Controlar";
    public static final String SOLICITAR_MENSAGEM="mensagem";
    public static final String SOLICITAR_ENTIDADE="entidade";
    public static final String SOLICITAR="Solicitar";
    public static final String
SERVIDOR_APLICACOESHOSPEDADAS="aplicacoesHospedadas";
    public static final String SERVIDOR="Servidor";
    public static final String MENSAGEM="Mensagem";
    public static final String CLIENTE="Cliente";
    public static final String ROTEADOR_PROCESSADOR="processador";
    public static final String ROTEADOR_TECNOLOGIA="tecnologia";
    public static final String ROTEADOR_NUMPROCESSADOR="numProcessador";
    public static final String ROTEADOR="Roteador";
    public static final String INTERFACE_CAMADA="camada";
    public static final String INTERFACE_INDICE="indice";
    public static final String INTERFACE_VELOCIDADE="velocidade";
    public static final String INTERFACE_PADRAOINTERFACE="padraoInterface";
    public static final String INTERFACE_MAC="mac";
    public static final String INTERFACE_MTU="mtu";
    public static final String INTERFACE_STATUS="status";
    public static final String INTERFACE_TIPO="tipo";
    public static final String INTERFACE="Interface";
    public static final String ACAO_ATIVIDADE="atividade";
    public static final String ACAO_ENTIDADE="entidade";
    public static final String ACAO="Acao";
    public static final String DISPOSITIVO_INTERFACES="interfaces";
```

```
    public static final String DISPOSITIVO_REDE="rede";
    public static final String DISPOSITIVO_MAC="mac";
    public static final String DISPOSITIVO_FABRICANTE="fabricante";
    public static final String DISPOSITIVO_NUMEROPORTA="numeroPorta";
    public static final String DISPOSITIVO_ENDERECO="endereco";
    public static final String DISPOSITIVO_PROTOCOLOSSUPORTADOS="protocolosSuportados";
    public static final String DISPOSITIVO_SERVICOS="servicos";
    public static final String DISPOSITIVO_CAMADA="camada";
    public static final String DISPOSITIVO_LINK="link";
    public static final String DISPOSITIVO_MODELO="modelo";
    public static final String DISPOSITIVO_ROTAS="rotas";
    public static final String DISPOSITIVO="Dispositivo";
    public static final String HUB_LARGURABANDA="larguraBanda";
    public static final String HUB="Hub";
    public static final String ENTIDADE_MENSAGEM="mensagem";
    public static final String ENTIDADE_CLASSE="classe";
    public static final String ENTIDADE_CONEXAO="conexao";
    public static final String ENTIDADE__CREATOR="_CREATOR";
    public static final String ENTIDADE="Entidade";
    public static final String CONEXÃO_ENTIDADEORIGEM="entidadeOrigem";
    public static final String CONEXÃO_ENTIDADEDESTINO="entidadeDestino";
    public static final String CONEXÃO="Conexão";
    public static final String SISTEMA_ENTIDADEORIGEM="entidadeOrigem";
    public static final String SISTEMA_SERVICO="servico";
    public static final String SISTEMA_ENDERECODESTINO="enderecoDestino";
    public static final String SISTEMA="Sistema";
    public static final String HOST_SISTEMAOPERACIONAL="sistemaOperacional";
    public static final String HOST_DISCO="disco";
    public static final String HOST_DISPOSITIVOS="dispositivos";
```

```
public static final String HOST_PROCESSADOR="processador";
public static final String HOST_MEMORIA="memoria";
public static final String HOST_NUMPROCESSADOR="numProcessador";
public static final String HOST="Host";
public static final String PROCESSO_PADRAO="padrao";
public static final String PROCESSO_ENTIDADE="entidade";
public static final String PROCESSO_ACAO="acao";
public static final String PROCESSO="Processo";
public static final String APLICACAO_URL="url";
public static final String APLICACAO_PROTOCOLO="protocolo";
public static final String APLICACAO_CARACTERISTICA="caracteristica";
public static final String APLICACAO="Aplicacao";
public static final String DADO_ORIGEM="origem";
public static final String DADO_CONTEUDO="conteudo";
public static final String DADO_DESTINO="destino";
public static final String DADO_TIPO="tipo";
public static final String DADO="Dado";
public static final String INFORMACAO_CONTEUDOINFERIDO="conteudoInferido";
public static final String INFORMACAO="Informacao";
public static final String PACOTE_TAMANHO="tamanho";
public static final String PACOTE_DADO="dado";
public static final String PACOTE="Pacote";
public static final String REDE_LOCAL="local";
public static final String REDE_DISPOSITIVOS="dispositivos";
public static final String REDE_RECURSOS="recursos";
public static final String REDE_MASCARA="mascara";
public static final String REDE_ENDERECO="endereco";
public static final String REDE="Rede";
public static final String ROTA_PACOTE="pacote";
public static final String ROTA_PROTOCOLO="protocolo";
```

```
public static final String ROTA_GATEWAY="gateway";
public static final String ROTA_INDICE="indice";
public static final String ROTA_ENDERECODESTINO="enderecoDestino";
public static final String ROTA_ROTEADOR="roteador";
public static final String ROTA_ENDERECOORIGEM="enderecoOrigem";
public static final String ROTA_MASCARA="mascara";
public static final String ROTA="Rota";
public static final String WEBSERVER_TIPOSERVIDOR="tipoServidor";
public static final String WEBSERVER="WebServer";
public static final String RECURSO_NOME="nome";
public static final String RECURSO_TIPO="tipo";
public static final String RECURSO="Recurso";
public static final String APPLICATIONSERVER_TIPOSERVIDOR="tipoServidor";
public static final String APPLICATIONSERVER="ApplicationServer";
public static final String SERVICO_CATEGORIA="categoria";
public static final String SERVICO_TEMSEGURANCA="temSeguranca";
public static final String SERVICO_PROTOCOLOSSUPORTADOS="protocolosSuportados";
public static final String SERVICO_CAMADA="camada";
public static final String SERVICO_RECURSOS="recursos";
public static final String SERVICO_NUMEROPORTA="numeroPorta";
public static final String SERVICO="Servico";
public static final String CLASSESERVICO_TEMPORESPOSTA="tempoResposta";
public static final String CLASSESERVICO_NIVELBANDA="nivelBanda";
public static final String CLASSESERVICO_SERVICO="servico";
public static final String CLASSESERVICO="ClasseServico";
public static final String LINK_ENDERECODESTINO="enderecoDestino";
public static final String LINK_LARGURABANDA="larguraBanda";
public static final String LINK_ENDERECOORIGEM="enderecoOrigem";
public static final String LINK="Link";
public static final String CAMADA_NOMECAMADAOSI="nomeCamadaOSI";
```

```
public static final String CAMADA_NUMEROCAMADAOSI="numeroCamadaOSI";
public static final String CAMADA_NUMEROCAMADAIP="numeroCamadaIP";
public static final String CAMADA_NOMECAMADAIP="nomeCamadaIP";
public static final String CAMADA_PROTOCOLO="protocolo";
public static final String CAMADA_SERVICO="servico";
public static final String CAMADA="Camada";
public static final String BANCO_DE_DADOS_TIPOBANCO="tipoBanco";
public static final String BANCO_DE_DADOS="Banco_de_Dados";
public static final String TABELA_REPOSITORIO="repositorio";
public static final String TABELA_ATRIBUTOS="atributos";
public static final String TABELA_ESQUEMA="esquema";
public static final String TABELA_NOME="nome";
public static final String TABELA="Tabela";
public static final String SWITCH_LARGURABANDA="larguraBanda";
public static final String SWITCH_TEMSUPORTECOLISAO="temSuporteColisao";
public static final String SWITCH_NUMPROCESSADOR="numProcessador";
public static final String SWITCH="Switch";
public static final String PROVEDOR_RESTRICAO="restricao";
public static final String PROVEDOR="Provedor";
public static final String PROTOCOLO_MENSAGEM="mensagem";
public static final String PROTOCOLO_VERSÃO="versão";
public static final String PROTOCOLO_CAMADA="camada";
public static final String PROTOCOLO_NOME="nome";
public static final String PROTOCOLO="Protocolo";
public static final String ATIVO="Ativo";
public static final String REPOSITORIO_SERVIDOR="servidor";
public static final String REPOSITORIO_NOME="nome";
public static final String REPOSITORIO_FABRICANTE="fabricante";
public static final String REPOSITORIO_TIPO="tipo";
public static final String REPOSITORIO="Repositorio";
```

```
/**
 * Constructor
*/
private ConfigManagementOntology(){
  super(ONTOLOGY_NAME, BasicOntology.getInstance());
  try {

  // adding Concept(s)
  ConceptSchema repositorioSchema = new ConceptSchema(REPOSITORIO);
  add(repositorioSchema, br.ufes.ontologia.Repositorio.class);
  ConceptSchema ativoSchema = new ConceptSchema(ATIVO);
  add(ativoSchema, br.ufes.ontologia.Ativo.class);
  ConceptSchema protocoloSchema = new ConceptSchema(PROTOCOLO);
  add(protocoloSchema, br.ufes.ontologia.Protocolo.class);
  ConceptSchema provedorSchema = new ConceptSchema(PROVEDOR);
  add(provedorSchema, br.ufes.ontologia.Provedor.class);
  ConceptSchema switchSchema = new ConceptSchema(SWITCH);
  add(switchSchema, br.ufes.ontologia.Switch.class);
  ConceptSchema tabelaSchema = new ConceptSchema(TABELA);
  add(tabelaSchema, br.ufes.ontologia.Tabela.class);
  ConceptSchema banco_de_DadosSchema = new ConceptSchema(BANCO_DE_DADOS);
  add(banco_de_DadosSchema, br.ufes.ontologia.Banco_de_Dados.class);
  ConceptSchema camadaSchema = new ConceptSchema(CAMADA);
  add(camadaSchema, br.ufes.ontologia.Camada.class);
  ConceptSchema linkSchema = new ConceptSchema(LINK);
  add(linkSchema, br.ufes.ontologia.Link.class);
  ConceptSchema classeServicoSchema = new ConceptSchema(CLASSESERVICO);
  add(classeServicoSchema, br.ufes.ontologia.ClasseServico.class);
  ConceptSchema servicoSchema = new ConceptSchema(SERVICO);
```

```
add(servicoSchema, br.ufes.ontologia.Servico.class);
ConceptSchema applicationServerSchema = new ConceptSchema(APPLICATIONSERVER);
add(applicationServerSchema, br.ufes.ontologia.ApplicationServer.class);
ConceptSchema recursoSchema = new ConceptSchema(RECURSO);
add(recursoSchema, br.ufes.ontologia.Recurso.class);
ConceptSchema webServerSchema = new ConceptSchema(WEBSERVER);
add(webServerSchema, br.ufes.ontologia.WebServer.class);
ConceptSchema rotaSchema = new ConceptSchema(ROTA);
add(rotaSchema, br.ufes.ontologia.Rota.class);
ConceptSchema redeSchema = new ConceptSchema(REDE);
add(redeSchema, br.ufes.ontologia.Rede.class);
ConceptSchema pacoteSchema = new ConceptSchema(PACOTE);
add(pacoteSchema, br.ufes.ontologia.Pacote.class);
ConceptSchema informacaoSchema = new ConceptSchema(INFORMACAO);
add(informacaoSchema, br.ufes.ontologia.Informacao.class);
ConceptSchema dadoSchema = new ConceptSchema(DADO);
add(dadoSchema, br.ufes.ontologia.Dado.class);
ConceptSchema aplicacaoSchema = new ConceptSchema(APLICACAO);
add(aplicacaoSchema, br.ufes.ontologia.Aplicacao.class);
ConceptSchema processoSchema = new ConceptSchema(PROCESSO);
add(processoSchema, br.ufes.ontologia.Processo.class);
ConceptSchema hostSchema = new ConceptSchema(HOST);
add(hostSchema, br.ufes.ontologia.Host.class);
ConceptSchema sistemaSchema = new ConceptSchema(SISTEMA);
add(sistemaSchema, br.ufes.ontologia.Sistema.class);
ConceptSchema conexãoSchema = new ConceptSchema(CONEXÃO);
add(conexãoSchema, br.ufes.ontologia.Conexão.class);
ConceptSchema entidadeSchema = new ConceptSchema(ENTIDADE);
add(entidadeSchema, br.ufes.ontologia.Entidade.class);
ConceptSchema hubSchema = new ConceptSchema(HUB);
```

```
add(hubSchema, br.ufes.ontologia.Hub.class);
ConceptSchema dispositivoSchema = new ConceptSchema(DISPOSITIVO);
add(dispositivoSchema, br.ufes.ontologia.Dispositivo.class);
ConceptSchema acaoSchema = new ConceptSchema(ACAO);
add(acaoSchema, br.ufes.ontologia.Acao.class);
ConceptSchema interfaceSchema = new ConceptSchema(INTERFACE);
add(interfaceSchema, br.ufes.ontologia.Interface.class);
ConceptSchema roteadorSchema = new ConceptSchema(ROTEADOR);
add(roteadorSchema, br.ufes.ontologia.Roteador.class);
ConceptSchema clienteSchema = new ConceptSchema(CLIENTE);
add(clienteSchema, br.ufes.ontologia.Cliente.class);
ConceptSchema mensagemSchema = new ConceptSchema(MENSAGEM);
add(mensagemSchema, br.ufes.ontologia.Mensagem.class);
ConceptSchema servidorSchema = new ConceptSchema(SERVIDOR);
add(servidorSchema, br.ufes.ontologia.Servidor.class);

// adding AgentAction(s)
AgentActionSchema solicitarSchema = new AgentActionSchema(SOLICITAR);
add(solicitarSchema, br.ufes.ontologia.Solicitar.class);
AgentActionSchema controlarSchema = new AgentActionSchema(CONTROLAR);
add(controlarSchema, br.ufes.ontologia.Controlar.class);
AgentActionSchema discoverySchema = new AgentActionSchema(DISCOVERY);
add(discoverySchema, br.ufes.ontologia.Discovery.class);
AgentActionSchema recuperarSchema = new AgentActionSchema(RECUPERAR);
add(recuperarSchema, br.ufes.ontologia.Recuperar.class);
AgentActionSchema alterarSchema = new AgentActionSchema(ALTERAR);
add(alterarSchema, br.ufes.ontologia.Alterar.class);
AgentActionSchema negociarSchema = new AgentActionSchema(NEGOCIAR);
add(negociarSchema, br.ufes.ontologia.Negociar.class);
AgentActionSchema determinarRotaSchema = new
```

```
AgentActionSchema(DETERMINARROTA);
   add(determinarRotaSchema, br.ufes.ontologia.DeterminarRota.class);
   AgentActionSchema armazenarSchema = new AgentActionSchema(ARMAZENAR);
   add(armazenarSchema, br.ufes.ontologia.Armazenar.class);
   AgentActionSchema monitorarSchema = new AgentActionSchema(MONITORAR);
   add(monitorarSchema, br.ufes.ontologia.Monitorar.class);
   AgentActionSchema receberSchema = new AgentActionSchema(RECEBER);
   add(receberSchema, br.ufes.ontologia.Receber.class);
   AgentActionSchema consumirSchema = new AgentActionSchema(CONSUMIR);
   add(consumirSchema, br.ufes.ontologia.Consumir.class);
   AgentActionSchema acessarSchema = new AgentActionSchema(ACESSAR);
   add(acessarSchema, br.ufes.ontologia.Acessar.class);
   AgentActionSchema trocarSchema = new AgentActionSchema(TROCAR);
   add(trocarSchema, br.ufes.ontologia.Trocar.class);
   AgentActionSchema processarSchema = new AgentActionSchema(PROCESSAR);
   add(processarSchema, br.ufes.ontologia.Processar.class);
   AgentActionSchema proverSchema = new AgentActionSchema(PROVER);
   add(proverSchema, br.ufes.ontologia.Prover.class);
   AgentActionSchema coletarSchema = new AgentActionSchema(COLETAR);
   add(coletarSchema, br.ufes.ontologia.Coletar.class);
   AgentActionSchema executarSchema = new AgentActionSchema(EXECUTAR);
   add(executarSchema, br.ufes.ontologia.Executar.class);
   AgentActionSchema enviarSchema = new AgentActionSchema(ENVIAR);
   add(enviarSchema, br.ufes.ontologia.Enviar.class);
   AgentActionSchema mappingSchema = new AgentActionSchema(MAPPING);
   add(mappingSchema, br.ufes.ontologia.Mapping.class);
   AgentActionSchema comunicarSchema = new AgentActionSchema(COMUNICAR);
   add(comunicarSchema, br.ufes.ontologia.Comunicar.class);


   // adding AID(s)
```

```
ConceptSchema discoveryAgentSchema = new ConceptSchema(DISCOVERYAGENT);
add(discoveryAgentSchema, br.ufes.agentes.discovery.DiscoveryAgent.class);
ConceptSchema providerAgentSchema = new ConceptSchema(PROVIDERAGENT);
add(providerAgentSchema, br.ufes.agentes.provider.ProviderAgent.class);
ConceptSchema providerAgentSchema = new ConceptSchema(CONFIGURATIONAGENT);
add(configurationAgentSchema, br.ufes.agentes.configuration.ConfigurationAgent.class);
ConceptSchema providerAgentSchema = new ConceptSchema(CLIENTAGENT);
add(clientAgentSchema, br.ufes.agentes.client.ClientAgent.class);



// adding Predicate(s)
PredicateSchema erroComunicacaoSchema = new
PredicateSchema(ERROCOMUNICACAO);
add(erroComunicacaoSchema, br.ufes.ontologia.ErroComunicacao.class);
PredicateSchema acessoNegadoSchema = new PredicateSchema(ACESSONEGADO);
add(acessoNegadoSchema, br.ufes.ontologia.AcessoNegado.class);
PredicateSchema valorNaoSuportadoSchema = new
PredicateSchema(VALORNAOSUPORTADO);
add(valorNaoSuportadoSchema, br.ufes.ontologia.ValorNaoSuportado.class);
PredicateSchema rotaInexistenteSchema = new PredicateSchema(ROTAINEXISTENTE);
add(rotaInexistenteSchema, br.ufes.ontologia.RotaInexistente.class);
PredicateSchema comunicacaoIndisponivelSchema = new
PredicateSchema(COMUNICACAOINDISPONIVEL);
add(comunicacaoIndisponivelSchema, br.ufes.ontologia.ComunicacaoIndisponivel.class);
PredicateSchema dadoNaoEncontradoSchema = new
PredicateSchema(DADONAOENCONTRADO);
add(dadoNaoEncontradoSchema, br.ufes.ontologia.DadoNaoEncontrado.class);
PredicateSchema simpleJADEAbstractOntology_Class_4Schema = new
PredicateSchema(SIMPLEJADEABSTRACTONTOLOGY_CLASS_4);
add(simpleJADEAbstractOntology_Class_4Schema,
br.ufes.ontologia.SimpleJADEAbstractOntology_Class_4.class);
```

```
PredicateSchema acaoNegadaSchema = new PredicateSchema(ACAONEGADA);

add(acaoNegadaSchema, br.ufes.ontologia.AcaoNegada.class);

PredicateSchema nivelServicoAbaixoPermitidoSchema = new
PredicateSchema(NIVELSERVICOABAIXOPERMITIDO);

add(nivelServicoAbaixoPermitidoSchema,
br.ufes.ontologia.NivelServicoAbaixoPermitido.class);

PredicateSchema parametroAusenteSchema = new
PredicateSchema(PARAMETROAUSENTE);

add(parametroAusenteSchema, br.ufes.ontologia.ParametroAusente.class);

PredicateSchema numeroArgumentosInsuficienteSchema = new
PredicateSchema(NUMEROARGUMENTOSINSUFICIENTE);

add(numeroArgumentosInsuficienteSchema,
br.ufes.ontologia.NumeroArgumentosInsuficiente.class);

PredicateSchema acaoNaoSuportadaSchema = new
PredicateSchema(ACAONAOSUPORTADA);

add(acaoNaoSuportadaSchema, br.ufes.ontologia.AcaoNaoSuportada.class);

PredicateSchema conexaoIndisponivelSchema = new
PredicateSchema(CONEXAOINDISPONIVEL);

add(conexaoIndisponivelSchema, br.ufes.ontologia.ConexaoIndisponivel.class);

PredicateSchema erroDesconhecidoSchema = new
PredicateSchema(ERRODESCONHECIDO);

add(erroDesconhecidoSchema, br.ufes.ontologia.ErroDesconhecido.class);

PredicateSchema informacaoIndisponivelSchema = new
PredicateSchema(INFORMACAOINDISPONIVEL);

add(informacaoIndisponivelSchema, br.ufes.ontologia.InformacaoIndisponivel.class);

PredicateSchema argumentoAusenteSchema = new
PredicateSchema(ARGUMENTOAUSENTE);

add(argumentoAusenteSchema, br.ufes.ontologia.ArgumentoAusente.class);

PredicateSchema rotaIndisponivelSchema = new PredicateSchema(ROTAINDISPONIVEL);

add(rotaIndisponivelSchema, br.ufes.ontologia.RotaIndisponivel.class);

PredicateSchema redeInalcancavelSchema = new
PredicateSchema(REDEINALCANCAVEL);
```

```
    add(redeInalcancavelSchema, br.ufes.ontologia.RedeInalcancavel.class);
    PredicateSchema valorNaoReconhecidoSchema = new
PredicateSchema(VALORNAORECONHECIDO);
    add(valorNaoReconhecidoSchema, br.ufes.ontologia.ValorNaoReconhecido.class);
    PredicateSchema servicoNaoSuportadoSchema = new
PredicateSchema(SERVICONAOSUPORTADO);
    add(servicoNaoSuportadoSchema, br.ufes.ontologia.ServicoNaoSuportado.class);
    PredicateSchema erroInternoSchema = new PredicateSchema(ERROINTERNO);
    add(erroInternoSchema, br.ufes.ontologia.ErroInterno.class);
    PredicateSchema aplicacaoIndisponivelSchema = new
PredicateSchema(APLICACAOINDISPONIVEL);
    add(aplicacaoIndisponivelSchema, br.ufes.ontologia.AplicacaoIndisponivel.class);


    // Adding fields
    repositorioSchema.add(REPOSITORIO_TIPO,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);
    repositorioSchema.add(REPOSITORIO_FABRICANTE,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);
    repositorioSchema.add(REPOSITORIO_NOME,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);
    repositorioSchema.add(REPOSITORIO_SERVIDOR, servidorSchema,
ObjectSchema.MANDATORY);
    protocoloSchema.add(PROTOCOLO_NOME,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);
    protocoloSchema.add(PROTOCOLO_CAMADA, camadaSchema, 1,
ObjectSchema.UNLIMITED);
    protocoloSchema.add(PROTOCOLO_VERSÃO,
(TermSchema)getSchema(BasicOntology.FLOAT), ObjectSchema.MANDATORY);
    protocoloSchema.add(PROTOCOLO_MENSAGEM, mensagemSchema,
ObjectSchema.OPTIONAL);
    provedorSchema.add(PROVEDOR_RESTRICAO,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);
    switchSchema.add(SWITCH_NUMPROCESSADOR,
```

```
(TermSchema)getSchema(BasicOntology.INTEGER), ObjectSchema.OPTIONAL);

   switchSchema.add(SWITCH_TEMSUPORTECOLISAO,
(TermSchema)getSchema(BasicOntology.BOOLEAN), ObjectSchema.MANDATORY);

   switchSchema.add(SWITCH_LARGURABANDA,
(TermSchema)getSchema(BasicOntology.FLOAT), ObjectSchema.MANDATORY);

   tabelaSchema.add(TABELA_NOME, (TermSchema)getSchema(BasicOntology.STRING),
ObjectSchema.OPTIONAL);

   tabelaSchema.add(TABELA_ESQUEMA,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

   tabelaSchema.add(TABELA_ATRIBUTOS,
(TermSchema)getSchema(BasicOntology.STRING), 0, ObjectSchema.UNLIMITED);

   tabelaSchema.add(TABELA_REPOSITORIO, repositorioSchema,
ObjectSchema.MANDATORY);

   banco_de_DadosSchema.add(BANCO_DE_DADOS_TIPOBANCO,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

   camadaSchema.add(CAMADA_SERVICO, servicoSchema, ObjectSchema.MANDATORY);

   camadaSchema.add(CAMADA_PROTOCOLO, protocoloSchema,
ObjectSchema.OPTIONAL);

   camadaSchema.add(CAMADA_NOMECAMADAIP,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

   camadaSchema.add(CAMADA_NUMEROCAMADAIP,
(TermSchema)getSchema(BasicOntology.INTEGER), ObjectSchema.MANDATORY);

   camadaSchema.add(CAMADA_NUMEROCAMADAOSI,
(TermSchema)getSchema(BasicOntology.INTEGER), ObjectSchema.MANDATORY);

   camadaSchema.add(CAMADA_NOMECAMADAOSI,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

   linkSchema.add(LINK_ENDERECOORIGEM,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

   linkSchema.add(LINK_LARGURABANDA,
(TermSchema)getSchema(BasicOntology.FLOAT), ObjectSchema.MANDATORY);

   linkSchema.add(LINK_ENDERECODESTINO,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

   classeServicoSchema.add(CLASSESERVICO_SERVICO, servicoSchema,
ObjectSchema.MANDATORY);
```

```
classeServicoSchema.add(CLASSESERVICO_NIVELBANDA,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

classeServicoSchema.add(CLASSESERVICO_TEMPORESPOSTA,
(TermSchema)getSchema(BasicOntology.FLOAT), ObjectSchema.OPTIONAL);

servicoSchema.add(SERVICO_NUMEROPORTA,
(TermSchema)getSchema(BasicOntology.INTEGER), ObjectSchema.MANDATORY);

servicoSchema.add(SERVICO_RECURSOS, recursoSchema, 0,
ObjectSchema.UNLIMITED);

servicoSchema.add(SERVICO_CAMADA, camadaSchema, 1, ObjectSchema.UNLIMITED);

servicoSchema.add(SERVICO_PROTOCOLOSSUPORTADOS, protocoloSchema, 1,
ObjectSchema.UNLIMITED);

servicoSchema.add(SERVICO_TEMSEGURANCA,
(TermSchema)getSchema(BasicOntology.BOOLEAN), ObjectSchema.MANDATORY);

servicoSchema.add(SERVICO_CATEGORIA,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

applicationServerSchema.add(APPLICATIONSERVER_TIPOSERVIDOR,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

recursoSchema.add(RECURSO_TIPO, (TermSchema)getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);

recursoSchema.add(RECURSO_NOME,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

webServerSchema.add(WEBSERVER_TIPOSERVIDOR,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

rotaSchema.add(ROTA_MASCARA, (TermSchema)getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);

rotaSchema.add(ROTA_ENDERECOORIGEM,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

rotaSchema.add(ROTA_ROTEADOR, roteadorSchema, ObjectSchema.MANDATORY);

rotaSchema.add(ROTA_ENDERECODESTINO,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

rotaSchema.add(ROTA_INDICE, (TermSchema)getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);

rotaSchema.add(ROTA_GATEWAY, (TermSchema)getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);
```

```
rotaSchema.add(ROTA_PROTOCOLO, protocoloSchema, ObjectSchema.OPTIONAL);

rotaSchema.add(ROTA_PACOTE, pacoteSchema, ObjectSchema.OPTIONAL);

redeSchema.add(REDE_ENDERECO, (TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

redeSchema.add(REDE_MASCARA, (TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

redeSchema.add(REDE_RECURSOS, recursoSchema, 0, ObjectSchema.UNLIMITED);

redeSchema.add(REDE_DISPOSITIVOS, dispositivoSchema, 1, ObjectSchema.UNLIMITED);

redeSchema.add(REDE_LOCAL, (TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

pacoteSchema.add(PACOTE_DADO, dadoSchema, ObjectSchema.OPTIONAL);

pacoteSchema.add(PACOTE_TAMANHO, (TermSchema)getSchema(BasicOntology.FLOAT), ObjectSchema.MANDATORY);

informacaoSchema.add(INFORMACAO_CONTEUDOINFERIDO, (TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

dadoSchema.add(DADO_TIPO, (TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

dadoSchema.add(DADO_DESTINO, (TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

dadoSchema.add(DADO_CONTEUDO, (TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

dadoSchema.add(DADO_ORIGEM, (TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

aplicacaoSchema.add(APLICACAO_CARACTERISTICA, (TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

aplicacaoSchema.add(APLICACAO_PROTOCOLO, protocoloSchema, ObjectSchema.OPTIONAL);

aplicacaoSchema.add(APLICACAO_URL, (TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

processoSchema.add(PROCESSO_ACAO, acaoSchema, ObjectSchema.MANDATORY);

processoSchema.add(PROCESSO_ENTIDADE, entidadeSchema, ObjectSchema.MANDATORY);
```

```
processoSchema.add(PROCESSO_PADRAO,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

hostSchema.add(HOST_NUMPROCESSADOR,
(TermSchema)getSchema(BasicOntology.INTEGER), ObjectSchema.OPTIONAL);

hostSchema.add(HOST_MEMORIA, (TermSchema)getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);

hostSchema.add(HOST_PROCESSADOR,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

hostSchema.add(HOST_DISPOSITIVOS, dispositivoSchema, 1,
ObjectSchema.UNLIMITED);

hostSchema.add(HOST_DISCO, (TermSchema)getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);

hostSchema.add(HOST_SISTEMAOPERACIONAL,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

sistemaSchema.add(SISTEMA_ENDERECODESTINO,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

sistemaSchema.add(SISTEMA_SERVICO, servicoSchema, ObjectSchema.MANDATORY);

sistemaSchema.add(SISTEMA_ENTIDADEORIGEM, entidadeSchema,
ObjectSchema.MANDATORY);

conexãoSchema.add(CONEXÃO_ENTIDADEDESTINO, entidadeSchema,
ObjectSchema.OPTIONAL);

conexãoSchema.add(CONEXÃO_ENTIDADEORIGEM, entidadeSchema,
ObjectSchema.MANDATORY);

entidadeSchema.add(ENTIDADE__CREATOR,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

entidadeSchema.add(ENTIDADE_CONEXAO, conexãoSchema, ObjectSchema.OPTIONAL);

entidadeSchema.add(ENTIDADE_CLASSE,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

entidadeSchema.add(ENTIDADE_MENSAGEM, mensagemSchema,
ObjectSchema.OPTIONAL);

hubSchema.add(HUB_LARGURABANDA,
(TermSchema)getSchema(BasicOntology.FLOAT), ObjectSchema.MANDATORY);

dispositivoSchema.add(DISPOSITIVO_ROTAS, rotaSchema, 0, ObjectSchema.UNLIMITED);

dispositivoSchema.add(DISPOSITIVO_MODELO,
```

```
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

   dispositivoSchema.add(DISPOSITIVO_LINK, linkSchema, 0, ObjectSchema.UNLIMITED);

   dispositivoSchema.add(DISPOSITIVO_CAMADA, camadaSchema, 1,
ObjectSchema.UNLIMITED);

   dispositivoSchema.add(DISPOSITIVO_SERVICOS, servicoSchema, 0,
ObjectSchema.UNLIMITED);

   dispositivoSchema.add(DISPOSITIVO_PROTOCOLOSSUPORTADOS, protocoloSchema, 1,
ObjectSchema.UNLIMITED);

   dispositivoSchema.add(DISPOSITIVO_ENDERECO,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

   dispositivoSchema.add(DISPOSITIVO_NUMEROPORTA,
(TermSchema)getSchema(BasicOntology.INTEGER), ObjectSchema.MANDATORY);

   dispositivoSchema.add(DISPOSITIVO_FABRICANTE,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

   dispositivoSchema.add(DISPOSITIVO_MAC,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

   dispositivoSchema.add(DISPOSITIVO_REDE, redeSchema, ObjectSchema.MANDATORY);

   dispositivoSchema.add(DISPOSITIVO_INTERFACES, interfaceSchema, 1,
ObjectSchema.UNLIMITED);

   acaoSchema.add(ACAO_ENTIDADE, entidadeSchema, ObjectSchema.MANDATORY);

   acaoSchema.add(ACAO_ATIVIDADE, (TermSchema)getSchema(BasicOntology.STRING),
ObjectSchema.MANDATORY);

   interfaceSchema.add(INTERFACE_TIPO,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

   interfaceSchema.add(INTERFACE_STATUS,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

   interfaceSchema.add(INTERFACE_MTU,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

   interfaceSchema.add(INTERFACE_MAC,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

   interfaceSchema.add(INTERFACE_PADRAOINTERFACE,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

   interfaceSchema.add(INTERFACE_VELOCIDADE,
```

```
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

   interfaceSchema.add(INTERFACE_INDICE,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

   interfaceSchema.add(INTERFACE_CAMADA, camadaSchema, 1,
ObjectSchema.UNLIMITED);

   roteadorSchema.add(ROTEADOR_NUMPROCESSADOR,
(TermSchema)getSchema(BasicOntology.INTEGER), ObjectSchema.OPTIONAL);

   roteadorSchema.add(ROTEADOR_TECNOLOGIA,
(TermSchema)getSchema(BasicOntology.STRING), 0, ObjectSchema.UNLIMITED);

   roteadorSchema.add(ROTEADOR_PROCESSADOR,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

   servidorSchema.add(SERVIDOR_APLICACOESHOSPEDADAS, aplicacaoSchema, 1,
ObjectSchema.UNLIMITED);

   solicitarSchema.add(SOLICITAR_ENTIDADE, entidadeSchema,
ObjectSchema.MANDATORY);

   solicitarSchema.add(SOLICITAR_MENSAGEM, mensagemSchema,
ObjectSchema.OPTIONAL);

   controlarSchema.add(CONTROLAR_RECURSOS, recursoSchema, 0,
ObjectSchema.UNLIMITED);

   controlarSchema.add(CONTROLAR_ENTIDADE, entidadeSchema,
ObjectSchema.MANDATORY);

   discoverySchema.add(DISCOVERY_DISPOSITIVOS, dispositivoSchema, 1,
ObjectSchema.UNLIMITED);

   recuperarSchema.add(RECUPERAR_DADO, dadoSchema, ObjectSchema.OPTIONAL);

   recuperarSchema.add(RECUPERAR_ENTIDADE, entidadeSchema,
ObjectSchema.MANDATORY);

   alterarSchema.add(ALTERAR_RESPOSTA,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

   alterarSchema.add(ALTERAR_ENTIDADE, entidadeSchema, ObjectSchema.MANDATORY);

   alterarSchema.add(ALTERAR_MENSAGEM, mensagemSchema,
ObjectSchema.OPTIONAL);

   negociarSchema.add(NEGOCIAR_ENTIDADEDESTINO, entidadeSchema,
ObjectSchema.OPTIONAL);

   negociarSchema.add(NEGOCIAR_ENTIDADEORIGEM, entidadeSchema,
```

```
ObjectSchema.MANDATORY);
   negociarSchema.add(NEGOCIAR_MENSAGEM, mensagemSchema,
ObjectSchema.OPTIONAL);
   determinarRotaSchema.add(DETERMINARROTA_ROTA, rotaSchema,
ObjectSchema.MANDATORY);
   determinarRotaSchema.add(DETERMINARROTA_PACOTE, pacoteSchema,
ObjectSchema.OPTIONAL);
   armazenarSchema.add(ARMAZENAR_DADO, dadoSchema, ObjectSchema.OPTIONAL);
   armazenarSchema.add(ARMAZENAR_ENTIDADEORIGEM, entidadeSchema,
ObjectSchema.MANDATORY);
   monitorarSchema.add(MONITORAR_RECURSOS, recursoSchema, 0,
ObjectSchema.UNLIMITED);
   monitorarSchema.add(MONITORAR_ENTIDADE, entidadeSchema,
ObjectSchema.MANDATORY);
   receberSchema.add(RECEBER_ENTIDADEDESTINO, entidadeSchema,
ObjectSchema.OPTIONAL);
   receberSchema.add(RECEBER_MENSAGEM, mensagemSchema,
ObjectSchema.OPTIONAL);
   consumirSchema.add(CONSUMIR_RECURSOS, recursoSchema, 0,
ObjectSchema.UNLIMITED);
   consumirSchema.add(CONSUMIR_SERVICO, servicoSchema,
ObjectSchema.MANDATORY);
   consumirSchema.add(CONSUMIR_ENTIDADE, entidadeSchema,
ObjectSchema.MANDATORY);
   acessarSchema.add(ACESSAR_RECURSOS, recursoSchema, 0,
ObjectSchema.UNLIMITED);
   acessarSchema.add(ACESSAR_ENTIDADE, entidadeSchema,
ObjectSchema.MANDATORY);
   trocarSchema.add(TROCAR_ENTIDADEDESTINO, entidadeSchema,
ObjectSchema.OPTIONAL);
   trocarSchema.add(TROCAR_ENTIDADEORIGEM, entidadeSchema,
ObjectSchema.MANDATORY);
   trocarSchema.add(TROCAR_MENSAGEM, mensagemSchema, ObjectSchema.OPTIONAL);
```

```
processarSchema.add(PROCESSAR_DADO, dadoSchema, ObjectSchema.OPTIONAL);
processarSchema.add(PROCESSAR_ENTIDADE, entidadeSchema, ObjectSchema.MANDATORY);
proverSchema.add(PROVER_RECURSOS, recursoSchema, 0, ObjectSchema.UNLIMITED);
proverSchema.add(PROVER_SERVICO, servicoSchema, ObjectSchema.MANDATORY);
proverSchema.add(PROVER_ENTIDADE, entidadeSchema, ObjectSchema.MANDATORY);
coletarSchema.add(COLETAR_ENTIDADEDESTINO, entidadeSchema, ObjectSchema.OPTIONAL);
coletarSchema.add(COLETAR_DADO, dadoSchema, ObjectSchema.OPTIONAL);
coletarSchema.add(COLETAR_ENTIDADEORIGEM, entidadeSchema, ObjectSchema.MANDATORY);
executarSchema.add(EXECUTAR_ACAO, acaoSchema, ObjectSchema.MANDATORY);
executarSchema.add(EXECUTAR_ENTIDADE, entidadeSchema, ObjectSchema.MANDATORY);
enviarSchema.add(ENVIAR_ENTIDADEORIGEM, entidadeSchema, ObjectSchema.MANDATORY);
enviarSchema.add(ENVIAR_MENSAGEM, mensagemSchema, ObjectSchema.OPTIONAL);
mappingSchema.add(MAPPING_DISPOSITIVOS, dispositivoSchema, 1, ObjectSchema.UNLIMITED);
comunicarSchema.add(COMUNICAR_ENTIDADEDESTINO, entidadeSchema, ObjectSchema.OPTIONAL);
comunicarSchema.add(COMUNICAR_ENTIDADEORIGEM, entidadeSchema, ObjectSchema.MANDATORY);
comunicarSchema.add(COMUNICAR_MENSAGEM, mensagemSchema, ObjectSchema.OPTIONAL);


// Adding inheritance


ativoSchema.addSuperSchema(dispositivoSchema);
provedorSchema.addSuperSchema(sistemaSchema);
switchSchema.addSuperSchema(ativoSchema);
banco_de_DadosSchema.addSuperSchema(aplicacaoSchema);
```

```
servicoSchema.addSuperSchema(recursoSchema);
applicationServerSchema.addSuperSchema(aplicacaoSchema);
webServerSchema.addSuperSchema(aplicacaoSchema);
informacaoSchema.addSuperSchema(dadoSchema);
aplicacaoSchema.addSuperSchema(servicoSchema);
hostSchema.addSuperSchema(ativoSchema);
entidadeSchema.addSuperSchema(recursoSchema);
hubSchema.addSuperSchema(ativoSchema);
dispositivoSchema.addSuperSchema(entidadeSchema);
roteadorSchema.addSuperSchema(ativoSchema);
clienteSchema.addSuperSchema(sistemaSchema);
mensagemSchema.addSuperSchema(dadoSchema);
servidorSchema.addSuperSchema(hostSchema);



} catch (java.lang.Exception e) {
        e.printStackTrace();
}
}
}
```

Tabela 6 - Código em Java da classe *ConfigManagementOntology*

ANEXO B

CONCEITO MODELADO EM UMA CLASSE JAVA

```
package br.ufes.ontologia;


import jade.content.*;
import jade.util.leap.*;
import jade.core.*;


/**
   * Objeto físico conectado à rede.
   * Protege name: Dispositivo
* @author ontology bean generator
* @version 2007/03/17, 23:23:33
*/
public class Dispositivo extends Entidade{


   /**
   * Conjunto de protocolos suportados pelo dispositivo.
* Protege name: protocolosSuportados
   */
   private List protocolosSuportados = new ArrayList();
   public void addProtocolosSuportados(Protocolo elem) {
     List oldList = this.protocolosSuportados;
     protocolosSuportados.add(elem);
   }
   public boolean removeProtocolosSuportados(Protocolo elem) {
     List oldList = this.protocolosSuportados;
     boolean result = protocolosSuportados.remove(elem);
```

```
    return result;
  }
  public void clearAllProtocolosSuportados() {
    List oldList = this.protocolosSuportados;
    protocolosSuportados.clear();
  }
  public Iterator getAllProtocolosSuportados() {return protocolosSuportados.iterator(); }
  public List getProtocolosSuportados() {return protocolosSuportados; }
  public void setProtocolosSuportados(List l) {protocolosSuportados = l; }

  /**
   * Conjunto de serviços providos pelo dispositivo.
* Protege name: servicos
   */
   private List servicos = new ArrayList();
   public void addServicos(Servico elem) {
     List oldList = this.servicos;
     servicos.add(elem);
   }
   public boolean removeServicos(Servico elem) {
     List oldList = this.servicos;
     boolean result = servicos.remove(elem);
     return result;
   }
   public void clearAllServicos() {
     List oldList = this.servicos;
     servicos.clear();
   }
   public Iterator getAllServicos() {return servicos.iterator(); }
   public List getServicos() {return servicos; }
```

```
  public void setServicos(List l) {servicos = l; }


  /**
   * Entrada na tabela de encaminhamento do dispositivo.
* Protege name: rotas
   */
  private List rotas = new ArrayList();
  public void addRotas(Rota elem) {
    List oldList = this.rotas;
    rotas.add(elem);
  }
  public boolean removeRotas(Rota elem) {
    List oldList = this.rotas;
    boolean result = rotas.remove(elem);
    return result;
  }
  public void clearAllRotas() {
    List oldList = this.rotas;
    rotas.clear();
  }
  public Iterator getAllRotas() {return rotas.iterator(); }
  public List getRotas() {return rotas; }
  public void setRotas(List l) {rotas = l; }



  /**
   * Processos ativos no dispositivo.
* Protege name: processos
   */
  private List processos = new ArrayList();
```

```
  public void addProcessos(Processo elem) {
    List oldList = this.processos;
    processos.add(elem);
  }
  public boolean removeProcessos(Processo elem) {
    List oldList = this.processos;
    boolean result = processos.remove(elem);
    return result;
  }
  public void clearAllProcessos() {
    List oldList = this.processos;
    processos.clear();
  }
  public Iterator getAllProcessos() {return processos.iterator(); }
  public List getProcessos() {return processos; }
  public void setProcessos(List l) {processos = l; }

/**
* Links existentes com outros ativos.
* Protege name: link
*/
private List link = new ArrayList();
public void addLink(Link elem) {
  List oldList = this.link;
  link.add(elem);
}
public boolean removeLink(Link elem) {
  List oldList = this.link;
  boolean result = link.remove(elem);
  return result;
```

```
}
public void clearAllLink() {
  List oldList = this.link;
  link.clear();
}
public Iterator getAllLink() {return link.iterator(); }
public List getLink() {return link; }
public void setLink(List l) {link = l; }

/**
* Camada em que o dispositivo, serviço ou protocolo atua.
* Protege name: camada
*/
private List camada = new ArrayList();
public void addCamada(Camada elem) {
  List oldList = this.camada;
  camada.add(elem);
}
public boolean removeCamada(Camada elem) {
  List oldList = this.camada;
  boolean result = camada.remove(elem);
  return result;
}
public void clearAllCamada() {
  List oldList = this.camada;
  camada.clear();
}
public Iterator getAllCamada() {return camada.iterator(); }
public List getCamada() {return camada; }
public void setCamada(List l) {camada = l; }
```

```
/**
* Endereço IP do dispositivo ou do gateway da rede.
* Protege name: endereco
*/
private String endereco;
public void setEndereco(String value) {
 this.endereco=value;
}
public String getEndereco() {
  return this.endereco;
}

/**
* Endereço MAC do dispositivo.
* Protege name: mac
*/
private String mac;
public void setMac(String value) {
 this.mac=value;
}
public String getMac() {
  return this.mac;
}

/**
 * Local onde se encontra dispositivo.
* Protege name: local
 */
 private String local;
```

```
  public void setLocal(String value) {
   this.local=value;
  }
  public String getLocal() {
    return this.local;
  }

  /**
   * Tempo em ms que o dispositivo está ativo.
* Protege name: tempoUP
   */
   private String tempoUP;
   public void setTempoUP(String value) {
    this.tempoUP=value;
   }
   public String getTempoUP() {
     return this.tempoUP;
   }
 /**
 * Nome do fabricante do dispositivo.
 * Protege name: fabricante
 */
 private String fabricante;
 public void setFabricante(String value) {
  this.fabricante=value;
 }
 public String getFabricante() {
   return this.fabricante;
 }
```

```
  /**
   * Modelo do dispositivo.
* Protege name: modelo
   */
   private String modelo;
   public void setModelo(String value) {
    this.modelo=value;
   }
   public String getModelo() {
     return this.modelo;
   }

  /**
  * Rede a que pertence o dispositivo.
  * Protege name: rede
  */
  private Rede rede;
  public void setRede(Rede value) {
   this.rede=value;
  }
  public Rede getRede() {
    return this.rede;
  }

  /**
  * Número de portas do dispositivo.
  * Protege name: numeroPorta
  */
  private int numeroPorta;
  public void setNumeroPorta(int value) {
```

```
    this.numeroPorta=value;
  }
  public int getNumeroPorta() {
    return this.numeroPorta;
  }

  /**
  * Lista de Interfaces físicas ativas no dispositivo.
  * Protege name: interfaces
  */
  private List interfaces = new ArrayList();
  public void addInterfaces(Interface elem) {
    List oldList = this.interfaces;
    interfaces.add(elem);
  }
  public boolean removeInterfaces(Interface elem) {
    List oldList = this.interfaces;
    boolean result = interfaces.remove(elem);
    return result;
  }
  public void clearAllInterfaces() {
    List oldList = this.interfaces;
    interfaces.clear();
  }
  public Iterator getAllInterfaces() {return interfaces.iterator(); }
  public List getInterfaces() {return interfaces; }
  public void setInterfaces(List l) {interfaces = l; }

}
```

Tabela 7 - Código em Java da classe Conceito Dispositivo

ANEXO C

MODELO OO DA ONTOLOGIA PROPOSTA

Figura 51 - Modelo da Ontologia proposta – parte I

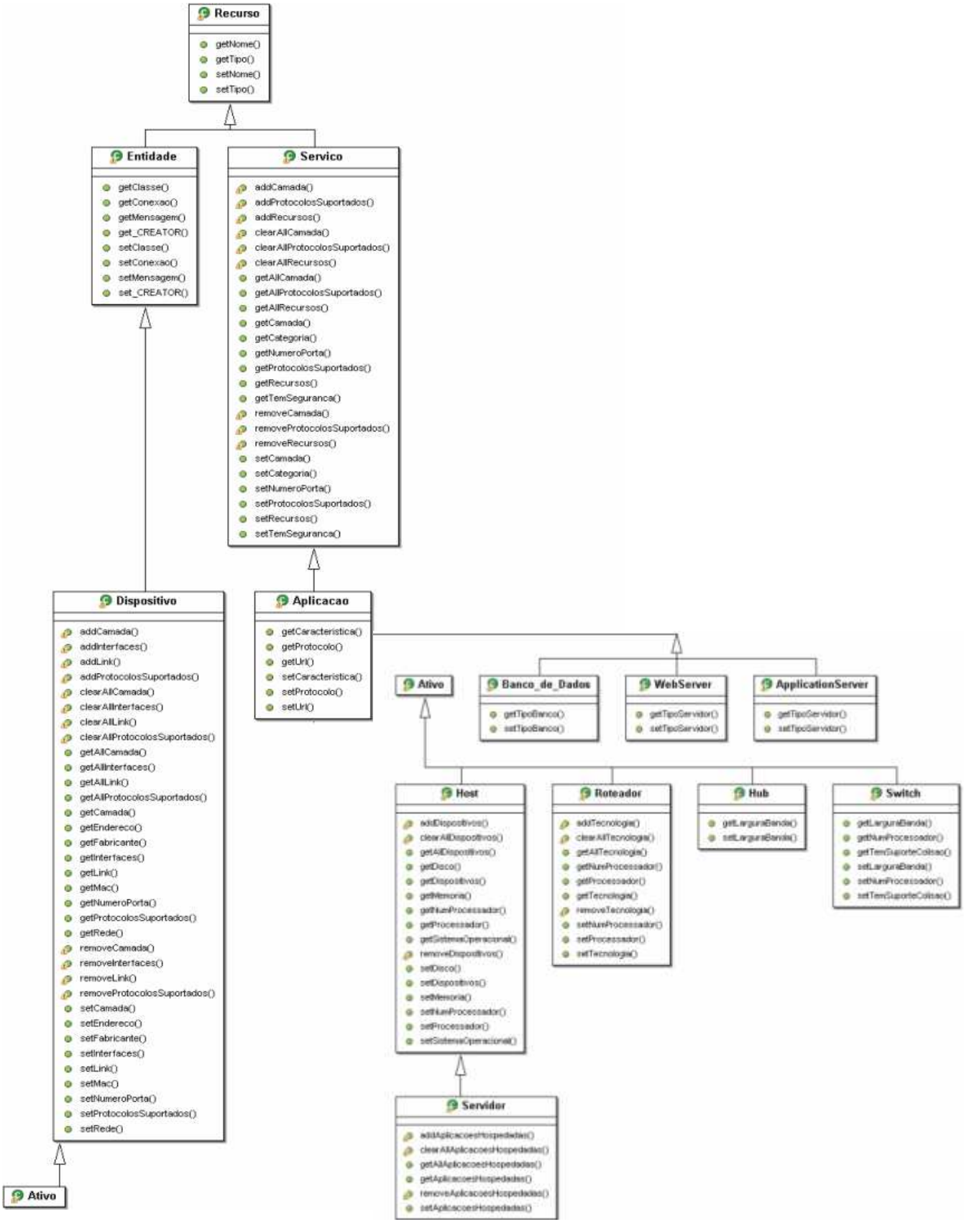


Figura 52 - Modelo da Ontologia proposta – parte II

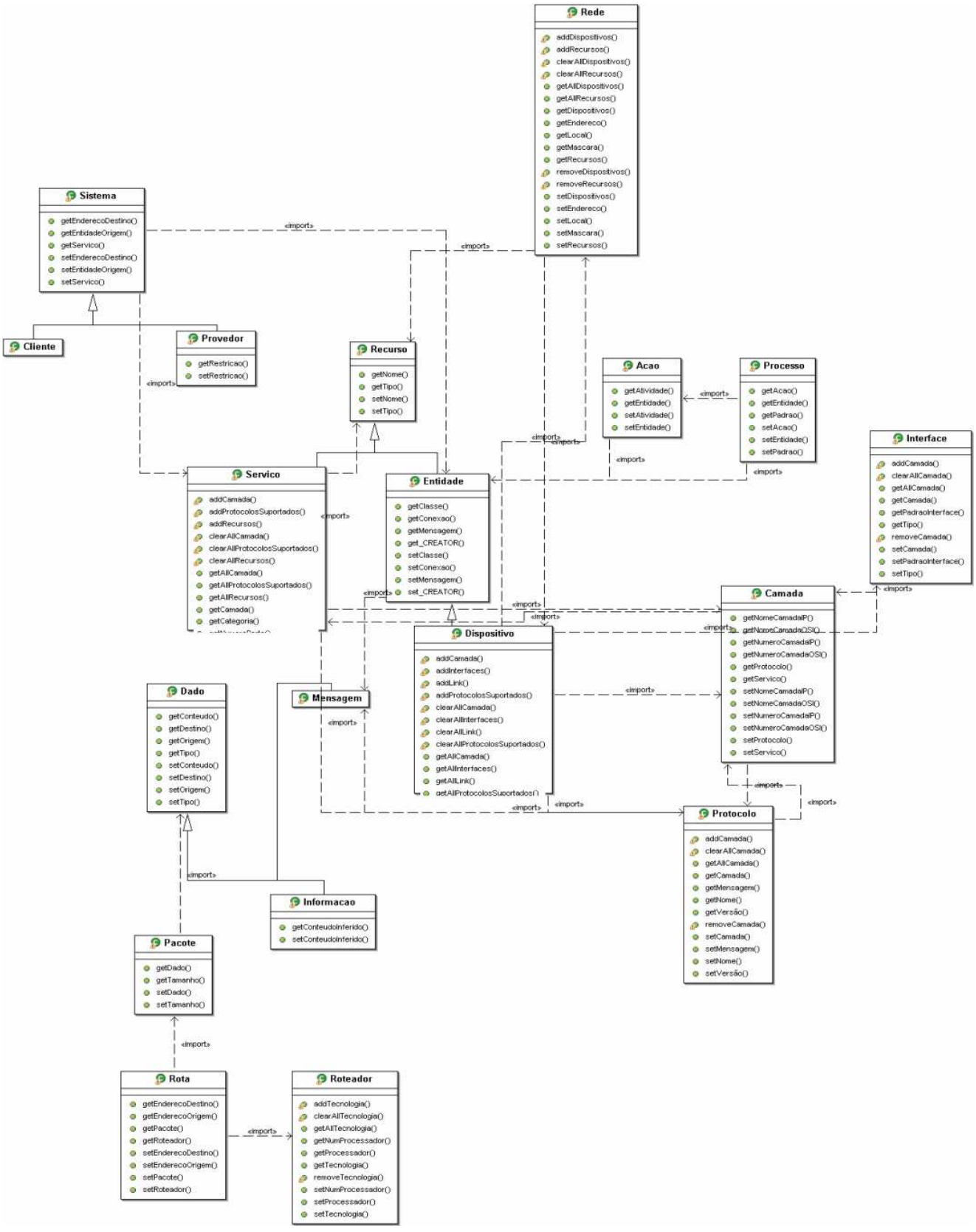
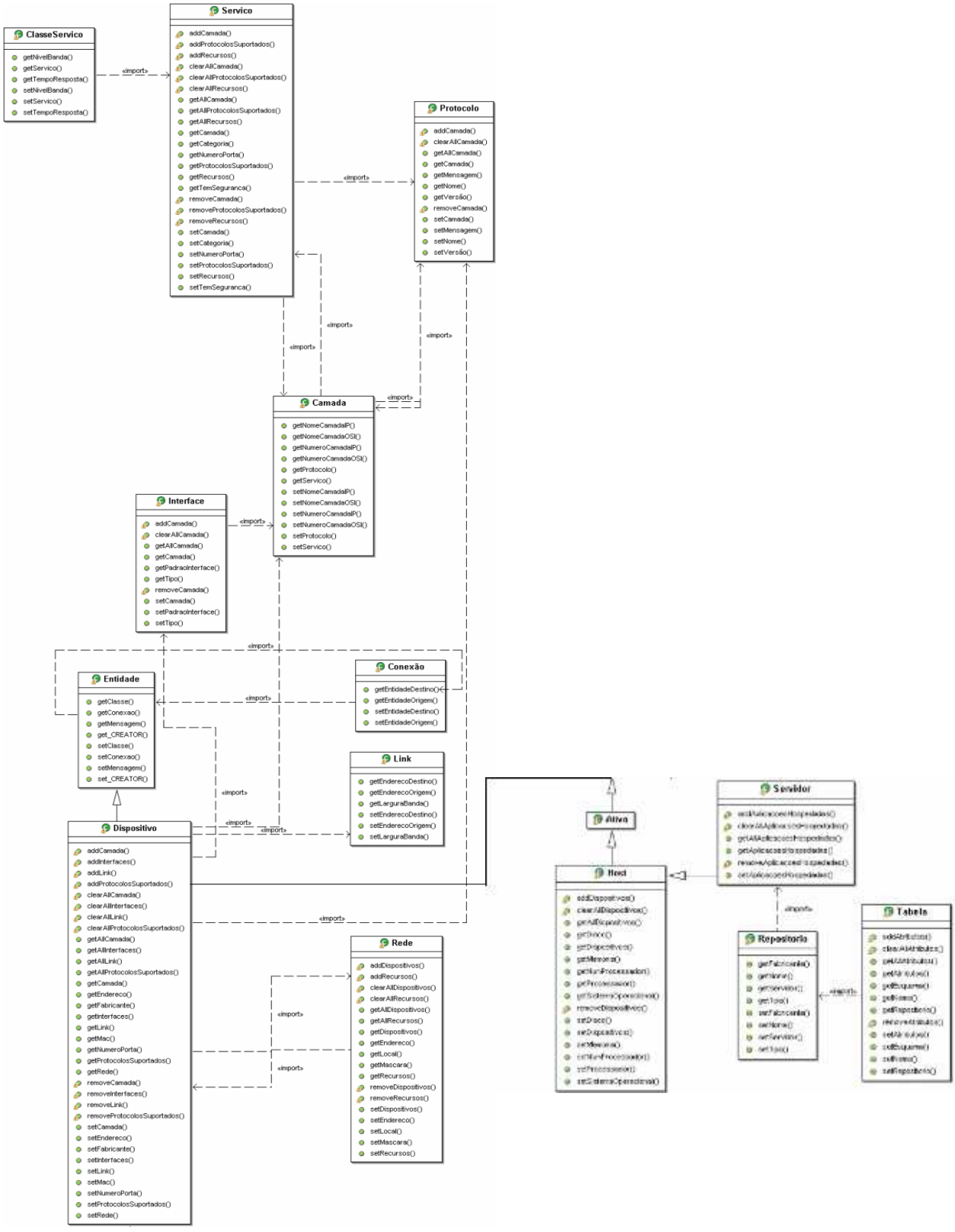


Figura 53 - Modelo da Ontologia proposta – parte III



Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo