

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA
COMPUTAÇÃO**

Marcos Dias de Assunção

**Implementação e Análise de uma Arquitetura de *Grids*
de Agentes para a Gerência de Redes e Sistemas**

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Carlos Becker Westphall

Co-orientador: M.Sc. Fernando Luiz Koch

Florianópolis, Março de 2004.

Implementação e Análise de uma Arquitetura de *Grids* de Agentes para a Gerência de Redes e Sistemas

Marcos Dias de Assunção

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação - Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Dr. Raul S. Wazlawick
Coordenador do Programa de Pós-Graduação em
Ciência da Computação

Banca Examinadora

Prof. Dr. Carlos Becker Westphall
Orientador

Prof. Dr. Joni da Silva Fraga

Prof. Dr. Mário Antônio Ribeiro Dantas

Prof. Dr. Vitorio Bruno Mazzola

Agradecimentos

Gostaria de agradecer ao meu professor orientador Carlos Becker Westphall pelo apoio, ajuda e principalmente pela paciência que ele teve comigo. Agradeço também ao meu amigo e doutorando Fernando Luiz Koch, pelo apoio, pela confiança, pelo tempo despendido, pelo conhecimento transmitido e pelas discussões sobre o assunto, pois sem elas este trabalho não teria sido desenvolvido. Estas duas pessoas me ajudaram muito durante o meu trabalho no LRG.

Agradeço aos meus pais, meus irmãos e minha irmã, por sempre terem me incentivado a seguir em frente, mesmo nos momentos em que as coisas pareciam difíceis. Devo muito a vocês.

Não poderia esquecer dos meus amigos do LRG: Silvia, Madalena, Mateus, Tadeu, Valério, Michel, Rafael Tavares, Rafael Plentz, Breno e Jean pela amizade, companhia e pelos momentos produtivos no LRG.

A todos aqueles que contribuíram, de forma direta ou indireta, para que este trabalho fosse realizado, meus sinceros agradecimentos.

Sumário

AGRADECIMENTOS	III
SUMÁRIO.....	IV
LISTA DE FIGURAS	VIII
LISTA DE TABELAS	XI
LISTA DE SIGLAS	XII
RESUMO	XIV
ABSTRACT	XV
1 INTRODUÇÃO.....	16
1.1 CARACTERIZAÇÃO DO PROBLEMA	18
1.2 OBJETIVOS DO TRABALHO	21
1.3 ESTADO DA ARTE	22
1.4 ESTADO DA ARTE EM <i>GRIDS</i> DE AGENTES	24
1.5 ORGANIZAÇÃO DO TRABALHO.....	28
2 AGENTES E SISTEMAS MULTI-AGENTES	30
2.1 DEFINIÇÕES	30
2.2 CARACTERÍSTICAS DOS AGENTES DE SOFTWARE.....	32
2.3 TIPOS DE AGENTES DE SOFTWARE.....	34
2.4 ARQUITETURA DE UM AGENTE	35
2.4.1 <i>Agentes Deliberativos</i>	36
2.4.2 <i>Agentes Reativos</i>	37
2.5 INTELIGÊNCIA ARTIFICIAL DISTRIBUÍDA (IAD).....	37
2.6 SISTEMAS MULTI-AGENTES (SMA).....	38
2.6.1 <i>Comunicação entre Agentes</i>	39
2.6.2 <i>O Conceito de Organização</i>	40
2.6.3 <i>Coordenação</i>	41
2.6.4 <i>Negociação e Protocolos de Leilão</i>	41
2.6.5 <i>Vantagens dos Sistemas Multi-Agentes</i>	42

2.7	ENGENHARIA BASEADA EM AGENTES PARA SISTEMAS COMPLEXOS	43
2.7.1	<i>Análise e Desenvolvimento Orientados por Agentes</i>	44
2.8	PADRÕES PARA CONSTRUÇÃO DE SISTEMAS DE AGENTES	46
2.8.1	<i>O Que Deve Ser Padronizado</i>	47
2.8.2	<i>A Arquitetura Abstrata da FIPA</i>	47
2.8.3	<i>A Plataforma de Agentes FIPA</i>	48
2.9	CONCLUSÕES	50
3	COMPUTAÇÃO EM GRID	51
3.1	FATORES MOTIVADORES	53
3.1.1	<i>Evolução dos Microprocessadores</i>	55
3.1.2	<i>Evolução das Tecnologias de Armazenamento</i>	57
3.2	EVOLUÇÃO DA COMPUTAÇÃO EM GRID.....	57
3.2.1	<i>Grid computacional</i>	58
3.2.2	<i>Grid de informação</i>	59
3.2.3	<i>Grid de conhecimento</i>	59
3.3	COMPUTAÇÃO DE ALTO DESEMPENHO	59
3.4	GRID VERSUS CLUSTER	60
3.5	GRID VERSUS P2P.....	62
3.6	ORGANIZAÇÕES VIRTUAIS	64
3.7	PROTOCOLOS E ARQUITETURAS.....	66
3.8	GRIDS ORIENTADOS POR SERVIÇOS	69
3.9	MIDDLEWARE PARA A COMPUTAÇÃO EM GRID.....	71
3.9.1	<i>O Globus Toolkit</i>	72
3.9.2	<i>O JINI</i>	74
3.10	CLASSIFICAÇÃO DOS TIPOS DE APLICAÇÕES GRID	75
3.10.1	<i>Supercomputação Distribuída</i>	75
3.10.2	<i>Computação de Alto Desempenho</i>	76
3.10.3	<i>Computação sob Demanda</i>	76
3.10.4	<i>Computação com Intensivo Uso de Dados</i>	76
3.10.5	<i>Computação Colaborativa</i>	77
3.11	CONCLUSÕES	77

4	GRIDS DE AGENTES	78
4.1	O QUE É O <i>GRID</i> DE AGENTES	78
4.2	SERVIÇOS NECESSÁRIOS EM UM <i>GRID</i> DE AGENTES	81
4.3	BENEFÍCIOS DE TECNOLOGIAS BASEADAS EM AGENTES PARA O <i>GRID</i>	83
4.3.1	<i>Composição de Sistemas</i>	83
4.3.2	<i>Desenvolvimento de Software</i>	84
4.3.3	<i>Orientação por Serviços</i>	85
4.4	PROJETOS EXISTENTES	85
4.4.1	<i>O Grid CoABS</i>	86
4.4.2	<i>O Grid Echelon</i>	89
4.5	CONCLUSÕES	89
5	O <i>GRID</i> DE AGENTES NA GERÊNCIA - PROPOSTA	91
5.1	ARQUITETURA PROPOSTA	92
5.1.1	<i>Grid de Agentes Coletores</i>	92
5.1.2	<i>Grid de Agentes Classificadores</i>	94
5.1.3	<i>Grid de Agentes Processadores</i>	95
5.1.4	<i>Grid de Agentes de Interface</i>	97
5.2	REQUISITOS DO <i>GRID</i> PARA A GERÊNCIA DE REDES E SISTEMAS.....	99
5.3	BALANCEAMENTO DE CARGA	100
5.4	NEGOCIAÇÃO NO <i>GRID</i> DE GERÊNCIA	102
5.5	GERÊNCIA DE REDES	103
5.5.1	<i>Modelo de Gerenciamento de Redes</i>	104
5.6	GERÊNCIA DE SISTEMAS.....	105
5.6.1	<i>Arquitetura WBEM</i>	106
5.6.2	<i>Utilitários Fornecidos pelos Sistemas</i>	108
5.7	CENÁRIOS POSSÍVEIS DE UTILIZAÇÃO	108
5.8	O <i>GRID</i> NA GERÊNCIA DE SISTEMAS	109
5.8.1	<i>O Grid Formado pela Rede Gerenciada</i>	110
5.9	CONCLUSÕES	111
6	AMBIENTE E RESULTADOS EXPERIMENTAIS	112
6.1	A PLATAFORMA <i>AGENTLIGHT</i>	112

6.1.1	<i>A Arquitetura do Agentlight</i>	114
6.2	O CONCEITO DE <i>SKILLS</i>	116
6.3	O NÚCLEO DO <i>GRID</i> DE AGENTES	118
6.4	INÍCIO DE UM NÓ <i>GRID</i>	119
6.5	SISTEMA DE COMUNICAÇÃO.....	122
6.6	ARQUIVOS XML DE DEFINIÇÃO	124
6.7	MEDIDORES DE DESEMPENHO	126
6.8	OS AGENTES COLETORES	128
6.8.1	<i>Agentes Coletores SNMP</i>	129
6.8.2	<i>Agentes Coletores de Linha de Comando</i>	134
6.8.3	<i>Agentes Coletores WMI</i>	137
6.9	OS AGENTES DO <i>GRID</i> DE ARMAZENAMENTO.....	139
6.9.1	<i>O Armazenamento de Dados</i>	142
6.10	A ETAPA DE ANÁLISE DOS DADOS.....	145
6.11	AS INTERFACES DO SISTEMA	147
6.12	APLICAÇÃO PARA INVENTÁRIO E CONFIGURAÇÃO – ESTUDO DE CASO 150	
6.12.1	<i>Criando Inventários de Hardware e Software</i>	151
6.12.2	<i>Ambiente de Experimentação</i>	153
6.12.3	<i>Inventários de Hardware e Software</i>	155
6.12.4	<i>Relatórios de Gerência</i>	157
6.12.5	<i>Informações Sobre o Grid</i>	159
6.12.6	<i>Análise de Informações</i>	160
6.12.7	<i>Distribuição das Atividades de Análise</i>	161
6.13	CONCLUSÕES	169
7	CONCLUSÕES	171
7.1	PRINCIPAIS CONTRIBUIÇÕES	172
7.2	TRABALHOS FUTUROS	172
	REFERÊNCIAS	175
	ANEXOS	188

Lista de Figuras

FIGURA 1 - CARACTERÍSTICAS DOS AGENTES INTELIGENTES [19].	32
FIGURA 2 - CLASSIFICAÇÃO COM BASE EM ALGUNS ATRIBUTOS PRIMÁRIOS [94].	34
FIGURA 3 - MODELO GENÉRICO DE UM AGENTE [19].	35
FIGURA 4 - ARQUITETURA DE UM AGENTE AUTÔNOMO [31].	36
FIGURA 5 - VISÃO GERAL DE UM SISTEMA COMPLEXO [69].	44
FIGURA 6 - REALIZAÇÕES DA ARQUITETURA ABSTRATA [43].	48
FIGURA 7 - PLATAFORMA FIPA NA VISÃO DO FIPA-OS [50].	49
FIGURA 8 - O <i>GRID</i> COMO UMA FONTE DE RECURSOS (ADAPTADO DE [20]).	51
FIGURA 9 – FATOS QUE CONTRIBUÍRAM PARA A COMPUTAÇÃO EM <i>GRID</i> [27].	53
FIGURA 10 - NÚMERO DE <i>HOSTS</i> DISPONÍVEIS NA INTERNET [66].	54
FIGURA 11 - EVOLUÇÃO DOS PROCESSADORES INTEL [92].	56
FIGURA 12 - ARQUITETURA DE UM CLUSTER DE COMPUTADORES [21].	61
FIGURA 13 - UM EXEMPLO DE ORGANIZAÇÃO VIRTUAL [56].	65
FIGURA 14 - ARQUITETURA <i>GRID</i> E COMPARAÇÃO COM PROTOCOLOS INTERNET [56].	67
FIGURA 15 - ARQUITETURA <i>GRID</i> DESCRITA POR BERMAN E OUTROS [16].	69
FIGURA 16 - INFRA-ESTRUTURA ORIENTADA POR SERVIÇOS PARA <i>E-SCIENCE</i> [32].	71
FIGURA 17 - ARQUITETURA GERAL DO GRAM DO GLOBUS TOOLKIT.	73
FIGURA 18 - VISÃO GERAL DO <i>GRID</i> CoABS [59].	87
FIGURA 19 - VISÃO ABSTRATA DA TECNOLOGIA JINI [29].	88
FIGURA 20- SISTEMA DE GERÊNCIA USANDO <i>GRIDS</i> DE AGENTES.	92
FIGURA 21 - CLASSIFICAÇÃO REALIZADA PELO <i>GRID</i> CLASSIFICADOR.	94
FIGURA 22 - DIVISÃO DAS TAREFAS DE ANÁLISE [75].	97
FIGURA 23 - EXEMPLO DE INTERFACES DO <i>GRID</i> .	98
FIGURA 24 - MODELO DE GERENCIAMENTO OSI [112].	104
FIGURA 25 - VISÃO ABSTRATA DO MODELO WBEM [112].	107
FIGURA 26 - CENÁRIO EXEMPLO DE GERENCIAMENTO.	110
FIGURA 27 - <i>GRID</i> É FORMADO PELA REDE GERENCIADA.	111
FIGURA 28 - VISÃO GERAL DOS COMPONENTES DO <i>AGENTLIGHT</i> (ADAPTADO DE [73]).	116
FIGURA 29 – INTERFACE JAVA <i>SKILLINTERFACE</i> .	117
FIGURA 30 – MAPEAMENTO DE REGRAS DE COLETA SNMP PARA A <i>SKILL</i> .	117
FIGURA 31 - SEQÜÊNCIA DE INICIAÇÃO DE UM NÓ DO <i>GRID</i> .	119

FIGURA 32 – EXEMPLO DE INICIAÇÃO DE UM NÓ GRID.	120
FIGURA 33 - LIGAÇÃO DE UM NÓ A OUTRO.....	121
FIGURA 34 - MENSAGEM FIPA TRANSPORTADA POR HTTP.	124
FIGURA 35 – REGRAS PARA UM AGENTE OBTER E EXIBIR INFORMAÇÕES DE ARMAZENAMENTO.	127
FIGURA 36 - COLETA E ARMAZENAMENTO DE DADOS.	129
FIGURA 37 – EXEMPLO DE REGRAS DE UM AGENTE COLETOR SNMP.	131
FIGURA 38 – REGRAS PARA UM AGENTE COLETOR OBTER INFORMAÇÕES DE INTERFACES.	132
FIGURA 39 – REGRAS E OBJETIVO A SER ATINGIDO POR UM AGENTE COLETOR SNMP... ..	132
FIGURA 40 – ÁRVORE RESULTANTE DE COLETA DE DADOS POR UM AGENTE SNMP.	133
FIGURA 41 – REGRAS GERAIS DE UM AGENTE DE LINHA DE COMANDO.	136
FIGURA 42 – REGRAS PARA COLETOR EXTRAIR INFORMAÇÕES USANDO ‘NETSTAT’	136
FIGURA 43 – RESULTADO DA COLETA DE UM AGENTE DE LINHA DE COMANDO.	137
FIGURA 44- FUNCIONALIDADE DE UM AGENTE COLETOR WMI.	138
FIGURA 45 – AGENTE WMI PARA A COLETA DE INFORMAÇÕES DO PROCESSADOR.	139
FIGURA 46- TRECHO USADAS ANTERIORMENTE PELOS AGENTES ARMAZENADORES.	141
FIGURA 47- ARQUIVO DE CONFIGURAÇÃO GERADO POR UM AGENTE COLETOR.	142
FIGURA 48- EXEMPLO DE ARQUIVO DE DADOS GERADO POR UM AGENTE COLETOR.	142
FIGURA 49- REGRAS COMUNS USADAS POR UM ARMAZENADOR.....	143
FIGURA 50 - PARTE DAS REGRAS PARA RECUPERAÇÃO DE INFORMAÇÕES.....	145
FIGURA 51 - SERVIDOR HTTP USADO PELO SISTEMA DE COMUNICAÇÃO E INTERFACE. .	148
FIGURA 52 - EXEMPLO DE FORMATAÇÃO USANDO XSL.....	149
FIGURA 53 - EXEMPLO DE GRÁFICO GERADO PELA <i>CHARTSKILL</i>	149
FIGURA 54 - EXEMPLO DE REGRAS PARA ENVIO DE NOTIFICAÇÃO.	150
FIGURA 55 - VISÃO GERAL DO SISTEMA.....	154
FIGURA 56 - EXEMPLO DE INTERFACE DO SISTEMA.....	155
FIGURA 57 - LISTA DE SOFTWARES INSTALADOS EM UM SISTEMA.....	156
FIGURA 58 - RELAÇÃO DE <i>SOFTWARES</i> INSTALADOS EM 150.162.63.10.	156
FIGURA 59 - DOCUMENTO XML DO RELATÓRIO DE SOFTWARES INSTALADOS.	157
FIGURA 60 - PARTE DA INTERFACE PARA EXIBIÇÃO DO <i>HARDWARE</i> INSTALADO.	157
FIGURA 61 - PARTE DAS REGRAS PARA ANÁLISE DA DISPONIBILIDADE.	158

FIGURA 62 - XML RESULTANTE DA ANÁLISE DE DISPONIBILIDADE.	158
FIGURA 63 - TELA COM ALERTAS GERADOS PELO SISTEMA.	159
FIGURA 64 - LISTA DE AGENTES COLETORES REGISTRADOS.	160
FIGURA 65 - INFORMAÇÕES SOBRE UM AGENTE ANALISADOR.	160
FIGURA 66 - RELATÓRIO EM XML CONTENDO NOTIFICAÇÕES.	161
FIGURA 67 - UTILIZAÇÃO DE RECURSOS SEM EXECUÇÃO DE TAREFAS.	163
FIGURA 68 - USO DE RECURSOS NA MÁQUINA A.	164
FIGURA 69 - DISTRIBUIÇÃO CÍCLICA DAS TAREFAS.	165
FIGURA 70 - DISTRIBUIÇÃO CÍCLICA - SEGUNDA TOMADA.	166
FIGURA 71 - DISTRIBUIÇÃO ALEATÓRIA DE TAREFAS.	166
FIGURA 72 - DISTRIBUIÇÃO BASEADA NO PROCESSADOR.	167
FIGURA 73 - DISTRIBUIÇÃO ALEATÓRIA, SEGUNDA TOMADA DE DADOS.	168
FIGURA 74 - DISTRIBUIÇÃO COM BASE NO USO/CAPACIDADE DO PROCESSADOR.	168
FIGURA 75 - TEMPO MÉDIO DE RESPOSTA NA EXECUÇÃO DAS TAREFAS.	169

Lista de Tabelas

TABELA 1 - AUMENTO DO NÚMERO DE TRANSISTORES ENTRE 1970 E 2000.	55
TABELA 2 - PARÂMETROS FORNECIDOS PARA A INICIAÇÃO DE UM NÓ <i>GRID</i>	119
TABELA 3 - INFORMAÇÕES OBTIDAS ATRAVÉS DO MEDIDOR DE DESEMPENHO.	122
TABELA 4 - AÇÕES EXECUTADAS PELA <i>SNMPSKILL</i>	129
TABELA 5 - AÇÕES EXECUTADAS PELA <i>CMDLINEEXECUTERSKILL</i>	134
TABELA 6 - AÇÕES EXECUTADAS PELA <i>REGEXPSKILL</i>	135
TABELA 7 – CONFIGURAÇÃO DOS EQUIPAMENTOS USADOS PARA OS TESTES.	162
TABELA 8 – PARÂMETROS DA SIMULAÇÃO COM APENAS UMA MÁQUINA.....	164

Lista de Siglas

API	<i>Application Programming Interface.</i>
CIM	<i>Common Information Model.</i>
CoABS	<i>Control of Agent-Based Systems.</i> Projeto do DARPA que investiga o uso da tecnologia de agentes para melhorar operações militares de comando, controle, comunicação, acúmulo e acesso a informações.
DAI	<i>Distributed Artificial Intelligence.</i> Ramo da inteligência artificial que tem investigado modelos de conhecimento e técnicas de comunicação e raciocínio que os agentes precisam para participar e agir nos sistemas de DAI.
DARPA	<i>Defense Advanced Research Projects Agency.</i> Agência de Projetos de Pesquisas Avançadas em Defesa, dos EUA. É uma agência responsável pelo desenvolvimento de novas tecnologias para o uso de operações militares.
DMTF	<i>Distributed Management Task Force.</i>
FAFNER	<i>Factoring via Network-Enabled Recursion.</i>
FIPA	<i>Foundation for Intelligent Physical Agents.</i> Organização que tem por objetivo estabelecer especificações que garantem a interoperabilidade entre diferentes plataformas de agentes.
FIPA ACL	Linguagem de comunicação entre agentes estabelecida pela FIPA .
GGF	<i>Global Grid Forum.</i>
GRAM	<i>Globus Resource Allocation Manager.</i> Componente do <i>Globus Toolkit</i> .
GSI	<i>Grid Security Infrastructure.</i>
IAD	Inteligência Artificial Distribuída.
I-WAY	<i>Information Wide Area Year.</i> Projeto norte americano idealizado em 1995 com o objetivo de ligar vários computadores de alto desempenho.
KQML	<i>Knowledge Query and Manipulation Language.</i> Linguagem utilizada para a comunicação entre agentes desenvolvida pelo DARPA .
LAN	<i>Local Area Network.</i>
MAGE	<i>Multi-AGent Environment.</i> Plataforma para construção de sistemas multi-agentes.

MAS	<i>Multi-Agent System.</i> Um dos ramos da inteligência artificial distribuída. Consiste de um sistema composto por vários agentes que têm uma visão limitada do problema e se apóiam em princípios como colaboração e coordenação para que os agentes possam atingir os seus objetivos.
MDS	<i>Monitoring and Discovery Service.</i> Serviço de informação do <i>Globus Toolkit</i> .
NSF	<i>National Science Foundation.</i> Agência independente do governo dos EUA. Tem o objetivo de promover o progresso da ciência; promover a saúde, prosperidade e bem-estar americano além de assegurar a defesa do país.
OGSA	<i>Open Grid Services Architecture.</i> Arquitetura que define um conjunto de interfaces para os serviços <i>grid</i> e tem como tecnologia de apoio os <i>Web Services</i> .
OV	Organizações Virtuais. Tradução do termo em inglês VO .
P2P	Peer-to-peer.
PSE	<i>Problem Solver Environment.</i>
SoFAR	<i>Southampton Framework for Agent Research.</i> Plataforma para construção de sistemas de agentes, desenvolvida pela Universidade de <i>Southampton</i> .
SSI	<i>Single System Image.</i>
VO	<i>Virtual Organizations.</i>
WBEM	<i>Web-Based Enterprise Management.</i>
WMI	<i>Windows Management Instrumentation.</i>

Resumo

O gerenciamento centralizado de redes de computadores, telecomunicações e de sistemas pode nos levar a situações em que grandes volumes de dados precisam ser manipulados e analisados por uma única estação de gerenciamento na rede. Além do alto custo, tal abordagem tem se apresentado pouco extensível à medida que cresce o número de dispositivos no ambiente gerenciado. Frente a este problema, este trabalho apresenta uma arquitetura baseada em *grids* de agentes aplicável ao gerenciamento de redes de computadores e de sistemas. Tal abordagem tem como objetivo básico aplicar princípios da computação em *grid*, usando agentes de *software* como infra-estrutura para construção do *grid*, com o objetivo de otimizar alguns problemas inerentes ao gerenciamento centralizado como a manipulação e análise de grande volume de informação de gerenciamento.

Também no escopo deste trabalho são descritos detalhes de implementação da arquitetura de *grid* proposta e a realização de experimentos em cenários de gerenciamento de sistemas usando um sistema baseado na proposta apresentada.

Palavras-Chaves: *grids* de agentes, computação em *grid*, gerência de redes, gerência de sistemas.

Abstract

The centralized computer and telecommunication network management and systems management can take us to situations where great data amounts have to be manipulated and analyzed by a single management station in the network. Beyond the prohibitive cost, such approach has presented to be no scalable when the number of managed elements in the management environment grows. Faced to this problem, this work presents an architecture based on grids of agents applicable to systems and computer network management. Such approach has as basic goal to apply grid-computing principles, using software agents as the infrastructure for the construction of the grid, with the objective of optimizing some inherent problems to the centralized management, such as the manipulation and analysis of great volume of management information.

This work also targets the description of the grid architecture implementation and the accomplishment of experiments in systems management scenarios using a management system based on the presented proposal.

Keywords: *grids of agents, grid computing, network management, systems management.*

1 Introdução

Alguns problemas computacionais, em função da complexidade e necessidade de uso de alto número de recursos de processamento, armazenamento e comunicação, não podem ser resolvidos em um ambiente computacional simples localizado sob uma mesma administração. Tais problemas são caracterizados por necessitarem de um grande número de recursos que não podem ser mantidos em um mesmo domínio por questões econômicas ou técnicas. Estes problemas estão frequentemente relacionados a áreas científicas e comerciais e são comumente chamadas de Aplicações de Grande Desafio [91]. Encontramos uma situação semelhante no gerenciamento de grandes redes de computadores com um enorme número de elementos e sistemas a serem gerenciados. A análise das informações de gerenciamento, com o objetivo de criar os relatórios de gerência apresentados ao gerente humano, em ambientes como este pode culminar em uma situação onde não existe, em um único servidor, por exemplo, uma capacidade computacional para realizar um gerenciamento eficiente.

A solução destes problemas necessita da utilização de *hardware* de alto desempenho, disponibilizado por supercomputadores ou *clusters* de computadores. Até alguns anos atrás, estas eram as únicas arquiteturas disponíveis para a execução destas aplicações. Porém, os custos de aquisição e manutenção destes recursos são geralmente elevados e nem sempre estas abordagens proporcionam recursos suficientes para a solução destes problemas. Antigamente esperava-se que todas as necessidades computacionais requeridas por estas aplicações pudessem ser supridas por computadores ou *clusters* [21] reunidos em um único domínio administrativo. Esta abordagem se mostrou impraticável e este cenário tem mudado com a grande disponibilidade de recursos computacionais e de rede a um custo muito menor, além da constante evolução das tecnologias de *software* [102].

Uma solução alternativa tem sido a utilização de recursos distribuídos com o intuito de agregá-los e assim formar um ambiente computacional distribuído em larga escala, cujo poder computacional possa ser utilizado na solução destes problemas complexos. Esta agregação e compartilhamento de recursos conectados em rede, formando um sistema distribuído em larga escala e possibilitando a resolução de problemas científicos e comerciais complexos tem sido chamada de computação em *grid*, ou *grid computing* [53]. Distribuindo a carga de trabalho de suas aplicações em um

grid, um usuário pode dispor de uma capacidade computacional e de armazenamento que não poderiam ser reunidas em um ambiente tradicional por questões técnicas ou econômicas. Este compartilhamento e agregação de recursos proporcionados pelas tecnologias *grid* se diferenciam das tecnologias atualmente disponíveis na Internet, pela forma como esta integração ocorre. O objetivo é proporcionar um acesso mais barato, eficiente, fácil, abrangente e, possivelmente, em larga escala [53]. As diferenças fundamentais entre um *grid* e um sistema distribuído tradicional estão na grande heterogeneidade de recursos, na dinamicidade do ambiente e na alta latência das redes que os interligam.

Recentemente as tecnologias *grid* têm sido mencionadas como uma forma de possibilitar uma integração de recursos e sistemas de uma forma mais flexível e rápida formando o que se pode chamar de organizações virtuais dinâmicas. A caracterização das tecnologias *grid* como forma de possibilitar o compartilhamento coordenado e controlado de recursos evidencia a necessidade de utilização de uma arquitetura baseada em serviços. Esta necessidade se deve ao fato das aplicações distribuídas estarem migrando de um cenário onde são fortemente acopladas para outro de fraco acoplamento, e onde os recursos podem ser agregados durante intervalos variáveis de tempo, de acordo com a necessidade do objeto de negócio das empresas e instituições. Neste cenário onde existem inúmeros recursos e serviços envolvidos o uso de computação baseada em agentes é recomendável [80], pois os agentes podem ser usados como uma forma de implementação deste paradigma de orientação por serviços. Neste caso os agentes podem definir as suas necessidades de recursos e serviços, procurar por outros agentes que oferecem os serviços e recursos requeridos e negociar com estes a utilização dos mesmos. Os agentes podem ser entidades em um *grid* e juntos com uma infra-estrutura que facilite a interação entre eles ou sistemas deles, a segurança no ambiente e um conjunto de mecanismos como *brokers*, *traders* e *matchmakers**, tem-se o que se pode chamar de *grid* de agentes [59].

* Estes elementos são conhecidos como *middle agents*, ou agentes intermediários, e têm a função de facilitar a busca de agentes que ofereçam os serviços requeridos. Um *matchmaker* conhece as características de serviços de vários agentes. Quando um serviço é procurado ele faz um casamento da necessidade com o agente que o oferece. O *broker* desempenha um papel parecido, porém, ele assume o papel de provedor de serviços para quem está requerendo. O *trader*, ao contrário do *matchmaker*, pode conhecer a função usada pelo requerente para julgar um serviço e pré-selecionar os agentes aptos.

O propósito de se utilizar *grids* de agentes na gerência de redes de computadores é usar uma arquitetura de *grid*, construída baseando-se nos princípios de tecnologias de agentes de *software*, para distribuir e otimizar as tarefas de gerenciamento, como coleta e análise de informações. Na gerência de redes de computadores e de sistemas encontramos alguns problemas que podem ser minimizados com a utilização de uma arquitetura de *grid* baseada em agentes. No fluxo de trabalho tradicional do gerenciamento de redes [111][112] temos uma situação onde dados são extraídos dos equipamentos e precisam ser transformados em informações que orientem o gerente na tomada de decisões e na execução de ações de gerenciamento. Para tais atividades de análise, sistemas baseados em regras de produção e inferência podem ser utilizados para extrair as informações necessárias e identificar possíveis problemas na rede. Porém, a consolidação destes dados em informações de gerência é uma tarefa intensiva e consome um grande poder de processamento, que por sua vez é caro e difícil de ser mantido por uma única organização. À medida que o ambiente cresce, a eficiência de um sistema centralizado diminui e aumenta o custo com hardware.

A proposta de uma abordagem de gerenciamento baseada em *grids* de agentes é a proposta deste trabalho, a qual foi apresentada e discutida durante o seu desenvolvimento [12],[13],[14]. A implementação e experimentação de toda a plataforma de gerenciamento proposta acarretam em um trabalho complexo e, portanto no escopo deste trabalho, são apresentados protótipos que apresentam o funcionamento da arquitetura proposta, proporcionando um conjunto de ferramentas e funcionalidades para a sua extensão futura, além de experimentos envolvendo os pontos implementados. Tal arquitetura também tem servido como linha para o projeto *AgentGrid* desenvolvido pelo Laboratório de Redes e Gerência da UFSC.

1.1 Caracterização do Problema

O gerenciamento centralizado apresenta sérias deficiências ao forçar que uma única estação de gerenciamento da rede tenha que analisar todas as informações de gerenciamento. Além disso, as tecnologias tradicionais - como o SNMP [111] - possuem um modelo organizacional baseado no princípio de “gerente e agente” e, neste caso, a estação de gerenciamento fica encarregada de requerer dos agentes os dados necessários ao gerenciamento. Neste tipo de interação a estação de gerenciamento é

responsável por requerer dos elementos gerenciados os dados necessários para que as tarefas de gerenciamento sejam realizadas. Os elementos gerenciados possuem um sistema denominado agente que é responsável por abastecer uma base de informações de gerenciamento localizada nestes elementos com informações que são coletadas através de sensores e outros dispositivos. Além disto, os agentes podem emitir um conjunto mínimo de alertas gerados sem a intervenção do gerente quando algum evento pré-determinado acontecer. Estas notificações são chamadas de *traps* em modelos de gerenciamento como SNMP.

Estas atividades de gerenciamento consomem largura de banda da rede e recursos da estação de gerenciamento para o seu processamento e análise. Com o crescimento do número de dispositivos e sistemas e a complexidade das redes de computadores e telecomunicações, a tarefa de gerenciamento se transforma em uma atividade complexa que requer cada vez mais recursos e tempo para que seja realizada com eficiência. Também é crescente o número de sistemas existentes nas redes atuais e para garantir a sua disponibilidade, cumprindo os contratos com os consumidores de serviços, vários itens devem ser verificados e gerenciados. Por conseqüência um grande volume de dados deve ser analisado com o intuito de verificar a configuração, as condições de operação dos serviços e prever e corrigir possíveis falhas que possam ocorrer.

Estes problemas de escalabilidade* podem impor sérias restrições quando o ambiente gerenciado é muito grande. Sem recursos financeiros para efetuar uma atualização constante do *hardware* usado nas atividades de gerência a organização corre o risco de não poder realizar um gerenciamento eficiente. Além disso, pode-se chegar a uma situação onde é financeiramente ou tecnicamente impraticável manter em uma única estação todo o poder de computação necessário para estas atividades.

Inicialmente, técnicas de inteligência artificial foram utilizadas com o objetivo de facilitar as atividades de gerenciamento e de livrar o gerente humano de atividades desgastantes como a análise de grande número de informações [101],[58]. Tais técnicas têm a intenção de automatizar as etapas de análise de dados realizadas pelo gerente, identificando falhas antes só identificadas pelo usuário do sistema de gerenciamento e tomando decisões, mantendo a concentração do gerente em atividades de gerência mais

* Optou-se por manter o jargão utilizado pelos profissionais da área de tecnologia como tradução da palavra *scalability*.

importantes. A vantagem da inclusão de elementos da inteligência artificial nos sistemas de gerenciamento é justamente a de possibilitar a automatização de determinadas atividades, de diminuir o trabalho do gerente humano ou de concentrá-lo em tarefas mais importantes e de mais alto nível.

Contudo, em muitas destas técnicas ainda persiste o problema da centralização e com ele o problema de escassez de recursos para a análise de dados. Propostas com o objetivo de proporcionar uma distribuição da gerência de redes, e de minimizar estes problemas de escassez de recursos através da distribuição, têm sido apresentadas. Dentre estas tecnologias pode-se citar o uso de agentes móveis [11]. As vantagens do uso de agentes móveis na gerência de redes [76] se devem ao fato destes poderem: reduzir o tráfego na rede; podem atuar de maneira autônoma e assíncrona; podem se adaptar ao ambiente de gerenciamento; e serem robustos e tolerantes a falhas.

Na gerência os agentes móveis apresentam bons resultados, pois podem migrar para dispositivos ou redes distantes, ligadas à estação de gerenciamento através de enlaces lentos, funcionar como compressores de dados e efetuar análises locais, fornecendo à estação de gerenciamento apenas informações já resumidas e compiladas. Alguns trabalhos importantes comparando o uso de agentes móveis com as tecnologias de gerenciamento tradicionais são apresentadas por Arantes e outros [8] e Costa [30]. Xavier [128] realiza experimentos no sentido de identificar a melhor configuração e disposição de comunidades de agentes móveis aplicáveis ao gerenciamento de redes.

Outra abordagem no sentido de proporcionar a distribuição do gerenciamento é o uso de elementos da inteligência artificial distribuída [74]. O uso de sistemas multi-agentes tem sido explorado e o seu uso se apóia em princípios como inteligência, comunicação, colaboração, sociabilidade e negociação para proporcionar a distribuição das atividades de gerenciamento. O uso de sistema multi-agentes em sistemas de gerenciamento baseia-se no fato de que os agentes, em um sistema como este, possuem uma visão limitada do problema e por sua vez não são capazes de realizarem o gerenciamento sozinhos. Assim sendo colaboram com outros agentes para a realização do gerenciamento da rede como um todo. Com este princípio e baseando-se no fluxo tradicional de um sistema de gerenciamento Koch e Westphall [74] apresentam um modelo de gerenciamento baseado em sistema multi-agentes, o qual dispõem os agentes

de acordo com as atividades que realizam, enquadrando-os em atividades como coleta de dados, análise e apresentação dos dados.

Além deste princípio, existe uma forte tendência nestes sistemas de levar as atividades de análise e decisão mais para perto de onde os dados de gerenciamento são extraídos. Isto significa que em um sistema multi-agentes os agentes envolvidos podem realizar tarefas de análise e tomarem decisões que dizem respeito a pequenas porções de uma rede pelas quais são responsáveis. Embora os sistemas multi-agentes proporcionem uma distribuição da gerência, dividindo o sistema de gerenciamento em módulos (agentes) que cooperam para a realização das atividades de gerenciamento, muitos deles ainda não apresentam uma forma eficiente de distribuição das atividades de armazenamento e análise de dados, um fator que consome muito dos recursos disponíveis em um sistema de gerenciamento.

O uso de *grids* de agentes no gerenciamento de redes de computadores e sistemas está baseado no princípio de que em determinadas situações é inevitável que tenhamos um grande volume de dados de gerenciamento para ser analisado. Em determinadas situações não existe um interesse de manter agentes de análise em *sites* de gerenciamento distantes, agentes estes que sejam responsáveis pelo gerenciamento daquele determinado *site*. Isto significa que em uma situação onde uma empresa de gerenciamento é responsável por prestar serviços de gerenciamento a outras empresas, por exemplo, é pouco provável que tenhamos algo além de coletores de dados nestes *sites* remotos. Neste caso, quanto maior a granularidade das informações coletadas, melhores são as chances de se realizar um gerenciamento mais eficiente. Contudo, este cenário recai sobre os problemas do gerenciamento centralizado citados anteriormente. É neste cenário que uma arquitetura de *grid* de agentes se torna aplicável. A divisão dos dados de gerenciamento a serem analisados em lotes, sua distribuição e o aproveitamento de recursos ociosos da rede para processamento destes dados se mostra uma boa alternativa.

1.2 Objetivos do Trabalho

Este trabalho tem como objetivo geral propor uma arquitetura de *grid* de agentes para a gerência de redes de computadores e sistemas.

Como objetivos específicos deste trabalho, podemos citar:

- Evidenciar o que são *grids* de agentes com o objetivo de criar um senso comum a respeito do que são e em que cenários tais modelos de *grids* podem ser utilizados.
- A implementação de um protótipo da arquitetura de *grid* de agentes e sua experimentação em um cenário de gerência de sistemas. A aplicação de gerência e o cenário descrito têm o objetivo básico de apresentar resultados, vantagens e desvantagens do seu uso no gerenciamento de sistemas e processamento de informações de gerência.
- Estudo de conceitos relacionados a agentes, de computação baseada em agentes e a contribuição de tais conceitos na especificação do *grid*.
- Identificar e testar mecanismos que proporcionem a distribuição das tarefas de análise de dados e o uso de recursos ociosos nas atividades de gerenciamento.

1.3 Estado da Arte

A computação em *grid* tem mostrado resultados em trabalhos científicos que requerem computação intensiva e um poder de computação que não pode ser encontrado em um ambiente tradicional. Este cenário apresenta o *grid* como uma infra-estrutura alternativa para as aplicações de grande desafio, que até alguns anos atrás tinham como plataforma de execução as plataformas de computação paralela e os *clusters* de computadores. O *grid* possibilitou a interligação destes *clusters*, de supercomputadores e centros de supercomputação. Este cenário de meta-computação é descrito por De Roure e outros [102] como primeira etapa de evolução das tecnologias de *grid*. Os projetos precursores foram o **I-WAY** [52] e o **FAFNER** [40].

Outro exemplo de aplicações, também citadas por alguns como primeiras tentativas de computação em *grid* são a análise de sinais de rádio na busca de vida extraterrestre inteligente (SETI@Home) [106], o superprocessador virtual proporcionado pela *United Devices* [106] e o *Distributed.Net* [35]. Estes três exemplos de aplicações utilizam o mesmo princípio, que consiste em decompor um conjunto de dados referente ao problema a ser resolvido em arquivos que podem ser baixados por uma aplicação instalada no computador do usuário, geralmente sob a forma de um protetor de tela, e após serem processados durante os períodos de ociosidade do computador do usuário, são devolvidos sob a forma de arquivos de resultados. Estas

aplicações em sua maioria, não são sensíveis à alta latência que possa existir na comunicação entre os elementos do ambiente, e suas sub-tarefas não requerem a passagem de um grande número de mensagens de sincronização. Uma vez fornecido o conjunto de dados à entidade de processamento, nenhuma comunicação acontece até que todo o conjunto de dados seja inteiramente processado.

Embora atualmente estas aplicações sejam consideradas aplicações **P2P** por alguns, elas são citadas por muitos como exemplos de *grid* e também como tecnologias precursoras. O SETI@Home possui algumas limitações, pois ele foi criado para executar um tipo de aplicação bem específico e não pode ser utilizado com outro propósito que não seja este. Este argumento é muito utilizado para desqualificá-lo como uma tecnologia de *grid*. Procurando resolver esta e outras limitações, foi criado o projeto *BOINC* [17] que proporciona um arcabouço que usa os mesmos princípios do SETI@Home, mas pode ser utilizado por outros tipos de aplicações.

Uma segunda corrente de desenvolvimento do *grid* iniciou com a tentativa de criar *middlewares* que facilitem a computação em larga escala, ou em *grid*, proporcionando gerenciamento de recursos e **APIs** para desenvolvimento de aplicações. Alguns trabalhos pioneiros são o *Legion* [77] e o *Globus* [54]. O *Globus* é um *middleware* que proporciona uma pilha de 4 camadas para controle de *hardware*, comunicação, compartilhamento de recursos e coordenação de tarefas. Outra tecnologia importante é o *JINI* [39] que proporciona ferramentas de registro e localização para construção de um ambiente dinâmico orientado por serviços. O JINI não tem como princípio primário ser um *middleware* para *grid*, porém é citado neste trabalho porque é usado com base para a infra-estrutura de *grid* de agentes do projeto **CoABS** [28].

Foster e outros [56] redefiniram o conceito de *grid* como sendo “o compartilhamento coordenado de recursos entre coleções dinâmicas de indivíduos, instituições ou recursos, as quais são referenciadas como organizações virtuais dinâmicas”. As organizações virtuais dinâmicas podem ser formadas, por exemplo, por pesquisadores, de diferentes instituições, compartilhando bases de dados, computadores, instrumentos científicos e outros recursos. O *grid* é a infra-estrutura que possibilita este compartilhamento de recursos. Este cenário apresenta uma arquitetura mais dinâmica e um pouco diferente das primeiras idéias iniciais de *grid*, ligadas aos conceitos de meta-computação. Juntamente com o artigo “*The Physiology*” [55], que apresenta a **OGSA**,

percebe-se que o *grid* não pode ser usado apenas com fins científicos e como plataforma de aplicações de grande desafio, mas como uma forma de suprir uma necessidade de integração das aplicações atuais, que estão migrando de um cenário onde elas são fortemente acopladas para um cenário de entidades fracamente acopladas.

A criação de padrões e especificações para esta arquitetura tem sido alvo de muitas discussões [25], e várias soluções de *middleware* para computação distribuída estão sendo desenvolvidas com base nestas especificações [54].

Muitos projetos de *grid* surgiram da necessidade de armazenamento e processamento de grande quantidade de dados requerida por aplicações da área de física, química e biologia. Os *grids* baseados neste princípio são comumente chamados de “*data grids*”. Em tais cenários, além do armazenamento de grande quantidade de informação, a busca por determinados padrões ou a análise destes dados requer que grandes conjuntos de dados sejam recuperados e processados. Além do armazenamento e recuperação destas informações, em muitos casos é importante poder adicionar uma semântica ou significado aos dados armazenados, podendo extrair novas informações ou conhecimento a partir destas informações. Esta necessidade é uma nova evolução do *grid*, conhecido como *grid* semântico [22],[117] e é considerado ainda um sonho por alguns. Estes cenários e etapas de evolução descrevem e reafirmam a arquitetura de camadas apresentada por Jeffery [68], que divide o *grid* em: dados, informação e conhecimento.

Os agentes de software podem fornecer funcionalidades importantes nas várias camadas e modalidades de *grid*. As características de pró-atividade, reatividade, mobilidade, comunicação, colaboração e cooperação são importantes em um cenário que envolve inúmeros recursos e serviços. Desta forma, é apresentado a seguir, o estado da arte em *grid* de agentes.

1.4 Estado da Arte em *Grids* de Agentes

Quando encontramos o termo *grid* de agentes relacionado em algum trabalho, percebemos que existem duas situações distintas, um *grid* de agentes como um sistema, e um *grid* como uma espécie de *middleware* que proporciona a interoperabilidade entre diferentes plataformas de agentes. No trabalho de Rana & Walker [97] a tecnologia de agentes é utilizada para generalizar e interligar ambientes de resolução de problemas

(PSE). Neste caso, os vários algoritmos usados pelos pesquisadores são encapsulados como componentes. Contudo, a integração e uso destes componentes são fatores complicados e é em função disso que uma arquitetura baseada em agentes, chamada de “*agent grid*” é apresentada [97]. Esta arquitetura apresenta como vantagens, as apresentadas quando se usa uma abordagem baseada em agentes para o desenvolvimento de *software* e integração de sistemas legados [59]. Os agentes podem ser vistos como uma extensão de objetos e componentes e provêm mecanismos melhores para a integração de sistemas e recursos, pois entre outras coisas não requerem que as aplicações ou componentes de software sejam conectadas em tempo de desenvolvimento.

Um dos trabalhos mais expressivos sobre *grids* de agentes é parte do projeto **CoABS** (*Control of Agent-Based Systems*) [28]. O **CoABS** é um programa do **DARPA** (*Defense Advanced Research Projects Agency*) e do *US Air Force Rome Labs*. O programa tem por objetivo investigar o uso da tecnologia de agentes para melhorar operações militares de comando, controle, comunicação, acúmulo e acesso a informações importantes no planejamento de ações militares. O *grid* **CoABS** também é responsável por facilitar a integração dinâmica dos vários sistemas desenvolvidos pelos pesquisadores do projeto **CoABS** no estabelecimento de coalizões construídas em campos de batalha. O **CoABS** é descrito com detalhes no Capítulo 4.

Ainda dentro do escopo do **CoABS**, existe o **eGents** [119], que é um sub-projeto do **Agility**, parte do programa **CoABS**. O **eGents** proporciona um barramento de comunicação baseado no serviço de e-mail para a comunicação entre agentes. As mensagens trocadas entre agentes (**ACL**) são codificadas em **XML** e aproveitam a ubiquidade do serviço de e-mail para o seu transporte. A utilização de mensagens de e-mail para transporte de mensagens pode apresentar características importantes no *grid* por aproveitar a possibilidade deste recurso estar disponível em muitas das plataformas atuais.

O Echelon [38] é uma infra-estrutura de *grid* baseada em agentes onde o usuário pode desenvolver soluções para problemas complexos, alocar os recursos de que precisa para resolver o problema e analisar os resultados finais. As tarefas são representadas por agentes móveis que migram para os recursos disponíveis no ambiente, processam e apresentam os resultados. Os resultados podem ser compartilhados entre os usuários do

sistema. O ambiente proporcionado pelo Echelon procura estender a área de trabalho do usuário proporcionando o acesso a recursos computacionais remotos (*hardware*, *software* e dados) abstraindo-o da complexidade, heterogeneidade e distribuição dos recursos do ambiente.

Zhongzhi e outros [108] apresentam um modelo de *grid* baseado em agentes, fazendo uso da plataforma **MAGE** (*Multi-AGent Environment*), usada para a construção dos agentes. O modelo de computação em *grid* baseada em agentes apresentada neste trabalho se baseia nos padrões **FIPA** para garantir a interoperabilidade com outras plataformas. O **MAGE** é citado como a primeira plataforma de agentes desenvolvida por um grupo asiático a se conectar ao projeto Agentcities [124]. Além da conformidade com os padrões **FIPA** que estabelecem critérios para os mecanismos de diretório de agentes e serviços, comunicação e interação entre agentes e plataformas, o modelo ainda apresenta como funcionam alguns serviços necessários como gerenciamento de recursos e de dados, ambos baseados em mecanismos providos pelo *middleware* Globus [54]. São citados dois exemplos de aplicações usando o modelo de *grid* proposto: uma aplicação de *e-Business* e uma segunda aplicação envolvendo uma cadeia de fornecimento de petróleo.

Moreau [86] apresenta algumas vantagens da utilização de agentes em um cenário de *grid*. Discorrendo sobre o desenvolvimento da plataforma de agentes **SoFAR** ele propõe a integração de agentes com *Web Services* [26]. No artigo “**Part I: Transport Layer**” [86] ele mantém o foco na camada de transporte das mensagens propondo meios para que as mensagens de comunicação entre agentes sejam representadas em XML e propõem uma **API** para a comunicação entre agentes em um cenário de *grid*. O uso de computação baseada em agentes e o modelo de comunicação proposto foram implementados na plataforma **SoFAR** que serve de base para o projeto de *grid* MYGRID [89].

Rana e Moreau [98] apresentam os serviços necessários em um *grid* computacional e uma abordagem baseada em agentes para a camada computacional do *grid*, baseada no princípio de serviços para o gerenciamento de recursos. Ressaltam que os agentes, por poderem se adaptar ao seu ambiente, podem prover funcionalidades importantes nas diversas camadas ou modalidades de *grid*. Segundo eles, um dos aprimoramentos mais comuns que os agentes podem fornecer é no que se refere à

descoberta e gerenciamento de recursos. É proposto um modelo baseado em agentes para proporcionar, além da descoberta de recursos, também a descoberta de serviços, onde os serviços podem variar desde chamadas a procedimentos remotos até interações complexas. Outra sugestão é feita com relação a forma como os serviços são especificados. Ontologias são necessárias para a descrição dos serviços e para que, além de consumidores e fornecedores de serviços estabeleçam um cenário comum, possa existir um cenário de negociação entre estes no *grid* computacional. Os agentes móveis podem proporcionar a migração de serviços e balanceamento de carga no *grid* computacional.

O *Agentcities* [124], mencionado anteriormente, é um projeto bastante amplo e tem por finalidade possibilitar a criação de um ambiente de agentes em larga escala, onde os pesquisadores poderão conectar suas plataformas de agentes e fazer uso dos serviços oferecidos. Esta integração será possível com a conformidade com os padrões estabelecidos pela **FIPA**. O *Agentcities* faz uso de tecnologia de agentes e de *grid* para a criação deste ambiente.

A exemplo do projeto *Agentcities* percebe-se um interesse constante em fazer com que os agentes possam ser empregados em ambientes de larga escala. Muitos grupos de pesquisa europeus e americanos acreditam que nos próximos anos, a tecnologia de agentes terá um papel importante nas aplicações de larga escala, como apresenta Luck em seu *roadmap* [80]. O sonho de que a Internet será habitada por agentes de software [94] tende a se concretizar. Passos importantes no sentido de garantir a interoperabilidade entre as plataformas de agentes existentes têm sido dados por instituições como a **FIPA**. Neste cenário de larga escala, assim como em um *grid*, um conjunto de protocolos padrões que garantam a interoperabilidade e a formação das organizações virtuais [56] é extremamente importante.

O estudo no sentido de adicionar características e funcionalidades que permitam que os sistemas de agentes sejam aplicados em cenários de larga escala também tem a sua relação com o *grid* de agentes. Congressos para a discussão e apresentação de idéias que facilitem a aplicação de agentes em cenários como a Internet têm sido realizados nos últimos anos [1],[105],[104]. Com isto tecnologias que permitam a criação de sistemas com inúmeros agentes e que atendam as características de larga escala exigidas atualmente têm sido apresentadas. Wijngaards e outros [123] apresentam o *AgentScape*

com o objetivo de facilitar a construção de sistemas baseados em agentes. Eles apresentam uma plataforma para criação de um grande número de agentes, suporte a múltiplas bases de código e sistemas operacionais, além de possibilitar a interoperabilidade entre diferentes plataformas. O *AgentScape* é também um *middleware* que proporciona a interoperabilidade entre diferentes plataformas de agentes.

O conceito de agentes também vem sendo utilizado durante vários anos para proporcionar o balanceamento de carga em ambientes de *cluster* e ultimamente em *grid*. Alguns trabalhos propõem o uso de agentes móveis para proporcionar o balanceamento de carga adequado [33]. Neste caso os agentes podem proporcionar a migração de serviços e tarefas para recursos com uma melhor capacidade de executá-los. Desic e Huljenic [33] apresentam uma revisão dos resultados obtidos na área de balanceamento de carga, descrevem algumas simulações e fazem uma comparação dos métodos de balanceamento de carga baseados em agentes.

Alguns conceitos dos trabalhos citados acima auxiliaram na definição da arquitetura para a construção de *grids* de agentes voltados à gerência de sistemas, como: as definições dos itens e serviços que compõem o *grid*, relacionados pelo CoABS e a conformidade com os as especificações **FIPA** [42] presente no *Agentcities*.

1.5 Organização do Trabalho

Para apresentar os resultados atingidos pela pesquisa e implementação neste trabalho, ele foi dividido em 7 capítulos assim distribuídos:

- Este primeiro capítulo apresenta uma breve introdução sobre o trabalho, caracterização do problema, estado da arte e trabalhos relacionados. Esta introdução tem o intuito de situar o leitor e contextualizar o tema da pesquisa.
- O segundo capítulo apresenta alguns conceitos relacionados à área de inteligência artificial distribuída e sistemas multi-agentes. Este capítulo tem como objetivo facilitar o entendimento dos capítulos posteriores e descrever a nomenclatura sobre agentes utilizada neste trabalho.
- O terceiro capítulo apresenta a computação em *grid*, origens e trabalhos que vem sendo realizados nesta área.

- Os *grids* de agentes são abordados no Capítulo 4, bem como as suas vantagens e fatores que levaram a adoção de agentes de software para a criação de ambientes de *grid*.
- No quinto capítulo são apresentados cenários de gerenciamento nos quais os *grids* de agentes podem ser aplicados. O gerenciamento de sistemas é um dos possíveis cenários descritos neste capítulo. É apresentada a proposta de arquitetura de *grid* de agentes para a gerência de sistemas.
- No sexto capítulo são relatados os experimentos realizados com a arquitetura de *grids* de agentes.
- As considerações finais e trabalhos futuros são apresentados no sétimo e último capítulo.

2 Agentes e Sistemas Multi-Agentes

Os agentes de *software* são componentes úteis na construção de sistemas distribuídos, constituem um paradigma para a engenharia de sistemas e para o desenvolvimento de software, e são vistos como extensões do modelo orientado a objetos e dos objetos componentes [110]. Além destas funcionalidades, as tecnologias de agentes possibilitam a integração de sistemas, facilitando a criação de interfaces entre sistemas existentes e possibilitando a integração de sistemas desenvolvidos por diferentes grupos de desenvolvimento. Em função destas características eles podem ser considerados elementos chaves na construção de ambientes orientados por serviços, como um *grid* de agentes.

Além disso, uma abordagem baseada em agentes proporciona meios para a composição de sistemas complexos ou a formação de coalizões, fatores importantes no estabelecimento das organizações virtuais dinâmicas, identificadas por Foster e outros [56].

Com o princípio de introduzir o leitor na tecnologia de agentes, este capítulo apresenta algumas definições, conceitos e aplicações das tecnologias relacionadas com este tema. Ele tem como meta possibilitar ao leitor um entendimento básico a respeito das tecnologias relacionadas a agentes utilizadas neste trabalho.

2.1 Definições

Definir o que é um agente é tão complicado quanto a definição de inteligência é para os pesquisadores da área de Inteligência Artificial. O termo “agente” tem sido usado em diversas áreas e pode ter significados diferentes [57]. Isto se deve ao fato do termo ser usado de maneira técnica e muitas vezes como uma metáfora [94],[18]. Uma definição bastante utilizada é a de que “*um agente é uma entidade que percebe o mundo através de seus sensores e atua sobre ele por meio de seus atuadores*”. Desta forma, um agente pode ser qualquer entidade humana, de *hardware* ou *software* capaz de perceber o ambiente no qual está inserido e atuar sobre ele. No caso de um agente de software, os sensores seriam as entradas e os atuadores as saídas do sistema ou a forma que o agente age sobre o meio.

Uma outra definição bastante abrangente e muito usada também é a de que *‘um agente é um componente de software e/ou hardware que é capaz de agir com o objetivo de realizar tarefas em nome de seu usuário’* [93]. Este exemplo está geralmente associado ao agente como um assistente pessoal do usuário e é comum encontrá-la associada ao exemplo do assistente de viagens que atende as solicitações do cliente e para isso analisa várias fontes de informação para encontrar as melhores opções de viagem para o seu cliente.

Alguns autores preferem definir o termo agente de maneira isolada, enquanto outros vêem os agentes como entidades que atuam de maneira colaborativa com outros agentes, o que leva ao conceito de sistemas multi-agentes ou solucionadores de problemas distribuídos. Neste caso, o agente é tido como o elemento chave da Inteligência Artificial Distribuída (IAD).

Para a realização deste trabalho é adotado que um agente deve possuir como atributos mínimos descritos no conceito fraco de agente definido por Wooldridge e Jennings [126]. Wooldridge & Jennings afirmam que os agentes são entidades que apresentam um comportamento resultante de um processo de raciocínio baseado na representação de suas atitudes, tais como: crenças, desejos e intenções. Neste caso, um sistema pode ser visto como um agente se ele possuir as seguintes características:

- **Autonomia:** deve ser capaz de funcionar sem intervenção humana, baseando suas ações no conhecimento que ele tem a respeito do ambiente;
- **Sociabilidade:** o agente interage com outros agentes através de uma linguagem;
- **Reatividade:** o agente deve ser capaz de perceber mudanças em seu ambiente e atuar de acordo com estas mudanças;
- **Pró-atividade:** não deve apenas perceber o ambiente, mas tentar alcançar um objetivo, apresentando a iniciativa.

Apesar de não se ter uma definição aceita por toda a comunidade sobre o que é um agente, muitas definições possuem características em comum, como autonomia, aprendizado, capacidade de responder a estímulos, entre outros, que ajudam a compreender o que é um agente. Contudo, esta indefinição não tem atrapalhado o desenvolvimento de software e aplicações baseados em agentes. Os agentes prometem ser uma nova e eficiente forma de modelagem de software, de desenvolvimento de

aplicações complexas e de solução de problemas, fatores importantes em um ambiente de *grid*.

2.2 Características dos Agentes de Software

Na tentativa de descrever o que são os agentes de software e diferenciá-los de softwares tradicionais, algumas características desejáveis são enumeradas por Brenner e outros [19]. Elas são divididas em propriedades internas e externas, conforme mostra a Figura 1. As propriedades internas são aquelas que determinam as ações internas do agente. As propriedades externas são todas aquelas características que afetam a interação dos agentes ou a comunicação com os agentes humanos.

Cada agente deveria possuir estas características. Porém, é óbvio que podem existir agentes que nem sempre possuem todas elas, o que não os desqualifica como sendo um agente de *software*. Agentes com estruturas mais complexas poderão ter boa parte ou todas as propriedades, mesmo que de maneira bastante geral.

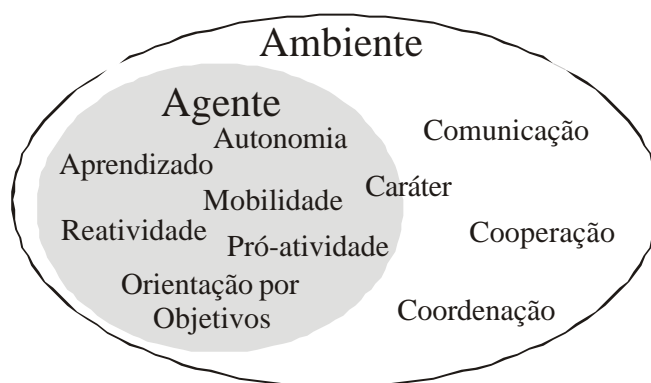


Figura 1 - Características dos agentes inteligentes [19].

As características desejáveis nos agentes de software são:

Reatividade. A reatividade significa que o agente é capaz de reagir adequadamente às influências ou informações do seu ambiente. O ambiente pode consistir de outros agentes, usuários, fontes de informação externas ou objetos físicos. A reatividade é uma das características essenciais de um agente inteligente e deveria ser suportada, de alguma maneira, por todos os agentes de software.

Pró-atividade e orientação por objetivos. Neste caso um agente não apenas reage às mudanças no seu ambiente, mas também toma a iniciativa em determinadas circunstâncias, o que pode ser chamado de comportamento pró-ativo.

Raciocínio e aprendizado. Para a realização de suas tarefas, um agente deve possuir um grau de inteligência. Esta inteligência pode variar desde mecanismos simples até mecanismos complexos que fornecem um alto grau de inteligência aos agentes. De maneira geral a inteligência de um agente pode ser formada basicamente por três componentes básicos: sua base de conhecimento, os mecanismos e a capacidade de raciocínio sobre a sua base de conhecimento e a habilidade de aprender e se adaptar ao ambiente.

Autonomia. Uma diferença entre os agentes e os programas de software tradicionais é a capacidade dos agentes de perseguirem os seus objetivos de maneira autônoma, ou seja, sem interações ou comandos vindos do ambiente ou usuário. A orientação por objetivos e a capacidade de aprender são características importantes para que exista um comportamento totalmente autônomo. Para que estas características sejam alcançadas é necessário que o agente seja orientado por objetivos e tenha um comportamento pró-ativo [126].

Mobilidade. A mobilidade pode ser descrita como a capacidade de um agente migrar de um *host* para outro.

Comunicação e cooperação. Frequentemente um agente pode requerer uma interação com o seu ambiente para realizar suas tarefas. Duas propriedades complementares podem ser diferenciadas como parte de uma interação: comunicação e cooperação. Um agente pode usar as capacidades de comunicação para fazer contato com o seu ambiente. Os mecanismos de comunicação permitem apenas que os agentes se comuniquem e troquem informações entre si. Uma forma melhorada é necessária quando na interação entre vários agentes para a resolução de um problema complexo. A cooperação entre vários agentes permite uma melhor e mais rápida solução de problemas complexos que excedem a capacidade individual de cada agente. A cooperação acarreta na necessidade de compartilhamento de conhecimento, objetivos e preferências.

Caráter. Também é desejável que um agente demonstre um comportamento externo assim como os humanos fazem. As características mais importantes que um agente pode apresentar são: honestidade, confiança e segurança.

Aqui foram apresentadas apenas algumas características desejáveis dos agentes de *software*. Vale lembrar que outros pesquisadores apresentam outras características ou

mencionam que algumas não são tão importantes. Portanto, estas definições podem apresentar variações.

2.3 Tipos de Agentes de Software

Muitos tentaram criar uma taxonomia para classificar os agentes de software. Porém, a exemplo das divergências existentes na definição de um “agente de *software*”, muitas classificações foram apresentadas usando critérios diferentes.

Alguns pesquisadores tentam classificar os agentes levando em consideração as noções *fraca e forte* de agente [126]. Agentes pertencentes à segunda categoria possuem qualidades mentais e emocionais.

Nwana & Ndumu [93] propõem alguns critérios para classificar os agentes. Primeiramente eles podem ser classificados de acordo com a capacidade de se mover pela rede. Este critério dá origem a classificação em agentes **móveis** e **estáticos**.

Em uma segunda classificação os agentes podem ser classificados em **reativos** e **deliberativos**. Os agentes deliberativos possuem um modelo de raciocínio simbólico do ambiente, os chamados estados mentais. Os agentes reativos não possuem nenhuma representação do seu ambiente e funcionam com o conceito de estímulo e resposta.

Uma terceira classificação é feita levando em consideração alguns atributos primários que os agentes deveriam apresentar. Alguns dos atributos listados são: autonomia, cooperação e aprendizado. Conforme apresentado na Figura 2 são identificados três tipos de agentes de acordo com esta classificação: **agentes colaborativos**, **agentes de interface** e **agentes inteligentes**.

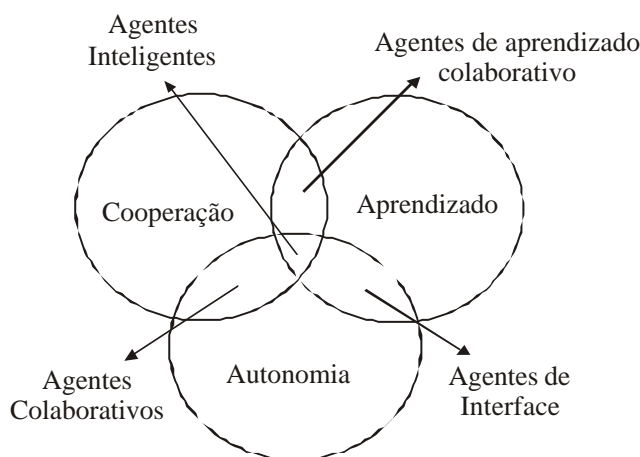


Figura 2 - Classificação com base em alguns atributos primários [94].

A quarta classificação leva em consideração as funções realizadas pelos agentes. Neste caso, como exemplo, temos os **agentes de informação**. Cabe ainda ressaltar que os agentes de informação podem ser estáticos ou móveis, deliberativos ou reativos. Os agentes também podem ser híbridos e combinar duas ou mais das filosofias listadas acima.

Com base nesta classificação, Nwana e Ndumu [93] identificaram os seguintes tipos de agentes: **agentes colaborativos, agentes de interface, agentes móveis, agentes de informação, agentes reativos, agentes híbridos e agentes inteligentes**.

2.4 Arquitetura de um Agente

Na visão de Brenner e outros [19] a forma mais simplista de representar a arquitetura de um agente é através do modelo de caixa-preta, conforme descrito na Figura 3. Neste caso, o agente recebe um conjunto de entradas através de seu mecanismo de percepção. Ele usa a sua inteligência para processar este conjunto de entradas que resulta em saídas que são exteriorizadas em forma de ações que devem ser executadas.

Embora aceita e utilizada, esta abordagem é bastante superficial e não apresenta todos os elementos arquiteturais que podem existir. Outra arquitetura mais detalhada é apresentada por Davidson [31], que descreve que todos os agentes autônomos apresentam uma arquitetura básica que é semelhante à apresentada na Figura 4. As partes componentes, assim como o funcionamento da arquitetura, são assim descritas:

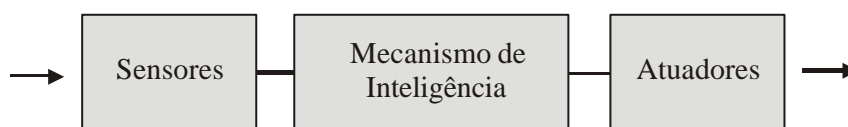


Figura 3 - Modelo genérico de um agente [19].

- **setas**: simbolizam o fluxo de dados;
- **sensores**: recebem informações do ambiente e providenciam dados para o mecanismo de inferência;
- **mecanismo de inferência**: é o cérebro do agente inteligente. Quando notificado de algum evento, o mecanismo de inferência opera sobre conjuntos

de regras e execução de raciocínio simbólico complexo para determinar como reagir ao evento e qual ação executar;

- **base de conhecimento:** é o local onde o agente armazena seu conhecimento;
- **atuadores:** são responsáveis pela execução das ações do agente sobre o ambiente.

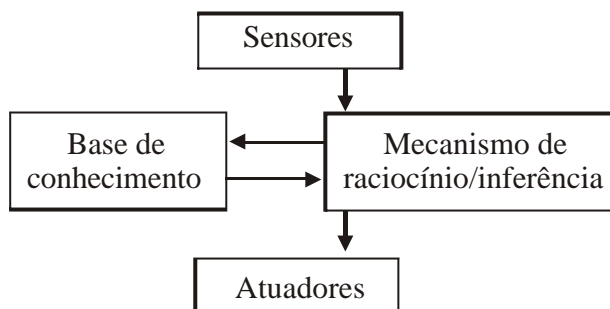


Figura 4 - Arquitetura de um agente autônomo [31].

A maioria das arquiteturas apresenta os agentes como elementos reativos, pois o fluxo de dados geralmente se dá no sentido de perceber o ambiente e agir sobre ele. Os agentes também podem manter um estado sobre as condições do ambiente, ter uma representação simbólica do mesmo e agir de forma pró-ativa, perseguindo seus próprios objetivos. Estas características podem levar a algumas diferenciações arquiteturais dos agentes. Percebendo estas diferenças Wooldridge & Jennings [126] afirmam que as arquiteturas de agentes podem ser divididas em:

- arquitetura de agentes deliberativos;
- arquitetura de agentes reativos;
- arquitetura híbrida.

Um agente reativo age de uma maneira baseada em ‘situação e reação’ ou ‘evento e reação’. Um agente deliberativo age de acordo com as suas decisões tomadas depois de um processo de raciocínio sobre o ambiente, processo este chamado de deliberação.

2.4.1 Agentes Deliberativos

Os agentes deliberativos também conhecidos como agentes BDI - estrutura composta por crença, desejo e intenção (do inglês: *belief*, *desire* e *intention*) - possuem um modelo simbólico do ambiente em sua base de conhecimento e a capacidade de raciocínio lógico como base das ações inteligentes. Normalmente a representação do

ambiente é previamente modelada e constitui a parte mais importante da base de conhecimento de um agente deliberativo [19]. O processo de raciocínio sobre as percepções sentidas nos sensores do agente é chamado de deliberação.

Além da representação do ambiente, os agentes podem tomar decisões lógicas. Como parte do processo de decisão o agente usa sua base de conhecimento para alterar o seu estado interno. Este estado interno é geralmente chamado de estado mental do agente e é composto pelos três componentes básicos descritos anteriormente: crença, desejo e intenção.

2.4.2 Agentes Reativos

Os agentes reativos não possuem uma representação simbólica interna do seu ambiente [19] e a capacidade de raciocínio complexo também é geralmente omitida. O motivo para estas restrições é poder criar agentes compactos, simples e flexíveis. No modelo reativo, a inteligência não é fruto de modelos internos complexos contidos nos agentes, mas, sobretudo, da interação do agente com outros agentes e com o ambiente.

Um agente reativo observa o ambiente, reconhece as mudanças que ocorrem e atua sobre ele. Este esquema segue um modelo de estímulo-resposta [126]. Este modelo permite a criação de agentes que monitoram continuamente um ambiente e atuam sobre ele quando necessário ou alguma situação ocorrer. Os agentes reativos oferecem uma boa abordagem no desenvolvimento de agentes de monitoração ou de prova no ambiente de gerenciamento.

2.5 Inteligência Artificial Distribuída (IAD)

As pesquisas da Inteligência Artificial (IA) têm como meta o desenvolvimento de softwares que simulem as capacidades inteligentes dos seres humanos, como raciocínio, comunicação e aprendizado. Inicialmente a IA procurava desenvolver sistemas que possuíssem todo o conhecimento necessário sobre o problema a ser resolvido, o que muitas vezes impõe restrições e problemas de escala. O surgimento das redes de computadores possibilitou a organização de computadores ou “sociedades”, onde a colaboração entre os indivíduos requer que conexões sejam estabelecidas e usadas.

A Inteligência Artificial Distribuída (IAD) é uma sub-área da Inteligência Artificial e tem investigado modelos de conhecimento e técnicas de comunicação e raciocínio que os agentes podem precisar para que possam participar das “sociedades” compostas de computadores e pessoas. A IAD está preocupada com a construção de sociedades de solucionadores de problemas, ou agentes, interagindo para resolver um problema.

As duas principais áreas dentro da IAD são: a Resolução Distribuída de Problemas (RDP) e os Sistemas Multi-Agentes (SMA). Trataremos apenas do segundo item por ser relevante a este trabalho.

2.6 Sistemas Multi-Agentes (SMA)

Conforme descrito anteriormente, os estudos da Inteligência Artificial estavam preocupados inicialmente com o desenvolvimento de agentes como se fossem entidades distintas, isoladas e tendo todo o conhecimento necessário a respeito do domínio do problema que estavam encarregados de resolver. Com o tempo constatou-se que esta abordagem é muito limitada, pois temos restrições em relação à quantidade de conhecimento que o agente consegue manter e manipular, além dos fatores referentes à capacidade computacional dos recursos que o agente está utilizando.

Neste cenário podemos perceber uma clara necessidade de uma estrutura modular e de abstração. Os sistemas multi-agentes (SMA) podem proporcionar esta estrutura modular necessária [113], pois se o problema a ser resolvido é muito grande e complexo, podemos resolvê-lo dividindo e desenvolvendo componentes modulares (agentes) que são especialistas apenas em uma parte específica do problema. Um sistema multi-agentes pode ser definido como *“uma rede de solucionadores de problemas fracamente acoplados que interagem uns com os outros para a solução de problemas que estão além das capacidades ou conhecimento individuais de cada um dos solucionadores”* [113].

Com o aparecimento dos sistemas multi-agentes, a comunidade de Inteligência Artificial Distribuída (IAD), teve um grande crescimento. Neste momento percebe-se a necessidade de ter sistemas com vários agentes autônomos interagindo entre si e com o ambiente, ao invés de termos apenas um. Quando se fala em múltiplos agentes, percebemos que não estão envolvidos apenas os aspectos relativos a inteligência de cada

agente, mas também os aspectos sociais, ou seja, as interações e colaboração entre os vários agentes do sistema.

Algumas características dos sistemas multi-agentes são:

- Cada agente tem informação ou capacidades incompletas para solucionar o problema geral e possuem uma visão limitada sobre ele.
- Não existe um controle global do sistema.
- Os dados e informações são descentralizados.
- A computação é assíncrona.

2.6.1 Comunicação entre Agentes

A comunicação é um fator importante em qualquer sistema de computação. Nos sistemas de agentes a comunicação possibilita a colaboração entre eles. Wooldridge [125] dá um exemplo de como a comunicação em sistema multi-agentes funciona, comparando-a com a comunicação entre objetos em um sistema orientado por objetos e demonstrando algumas diferenças. Na orientação por objetos, se um objeto i deseja se comunicar com um objeto j , ele pode fazer isso através da invocação de um método que i expõe, como por exemplo, o método $m1$. Neste caso dizemos que o objeto i irá executar as instruções de $m1$. Em um sistema de agentes, os agentes realizam atos ou ações, no caso da comunicação são atos comunicativos. Neste caso um agente não pode simplesmente obrigar outro agente a executar uma ação, porque ele tem controle sobre seu estado interno e sobre as suas ações. Pode não ser vantajoso para ele executar a ação que outro agente está solicitando. Neste caso o agente que está solicitando pode persuadí-lo a executar a ação através de atos comunicativos que expressem a sua intenção.

Em um SMA a comunicação acontece de uma maneira mais complexa do que em sistemas distribuídos tradicionais, onde as mensagens são definidas através de uma sintaxe e de um protocolo. Por isso é mais comum falar de interações [49] do que simplesmente comunicação. Uma abordagem mais semântica precisa ser adotada, e uma das alternativas é o uso da teoria de atos de fala [125] *apud* [103]. Em tal abordagem a interação entre os agentes acontece em pelo menos dois níveis: um correspondendo ao conteúdo informacional da mensagem e outro relacionado à intenção da mensagem

comunicada. Se a interação entre agentes é realizada através da passagem de mensagens, cada agente deve ser apto a deduzir a intenção do agente que enviou a mensagem. Mais detalhes sobre comunicação e a teoria dos atos de fala podem ser obtidos em [125].

Como exemplos de linguagens usadas na interação, usando passagem de mensagens, temos: o **KQML** (*Knowledge Query and Manipulation Language*) desenvolvida pelo **DARPA** e a **FIPA ACL**. Wooldridge apresenta uma explanação a respeito do funcionamento da KQML [125]. Detalhes a respeito da linguagem **FIPA ACL** podem ser obtidos na especificação fornecida pela FIPA [42].

2.6.2 O Conceito de Organização

Para Ferber [41], uma organização é um padrão que descreve como os seus membros interagem para atingir um objetivo comum. Este padrão pode ser estático, definido em tempo de projeto, ou dinâmico como no caso de sistemas abertos. Pode ser também descrito como o padrão de coordenação das tomadas de decisão e comunicação entre um conjunto de agentes que desempenham tarefas para alcançar objetivos com o sentido de atingir um estado global coerente.

Estruturas simples e outras mais complexas de organização foram propostas ao longo dos anos, como os *grupos*, *times* e *grupos de interesses*. Um grupo permite a colaboração entre os seus membros para atingir um objetivo comum. Neste caso a atividade geral é dividida em tarefas menores que são alocadas aos membros do grupo. No caso dos times, existem agentes agindo em um ambiente e se comunicando com outros agentes com o objetivo de dividirem suas tarefas e com isso resolver as possíveis inconsistências. Os grupos de interesse são compostos de agentes que possuem interesses comuns e podem cooperar para atingir os seus objetivos.

Um outro modelo um pouco melhor é o hierárquico, baseado no paradigma de mestre-escravo. Neste modelo existe o gerente responsável pela coordenação da tarefa e ela é dividida em tarefas menores que são alocadas aos escravos. Esta arquitetura pode ser replicada em vários níveis, formando uma espécie de hierarquia.

Para um ambiente descentralizado ou um sistema aberto, uma abordagem mais interessante é a baseada em modelos de mercado - do inglês (*market-based*). Neste tipo

de modelo temos agentes fornecedores de serviços ou mercadorias que possuem os recursos para produzi-las, e agentes compradores que requerem estas mercadorias e serviços para a realização de suas tarefas. Vale lembrar que neste caso, fornecedor e comprador são apenas papéis, o que não impede que um agente ora exerça um papel ora outro. Neste caso os agentes competem por interesses próprios e negociam com outros agentes a realização de um determinado serviço ou tarefa. Apesar de ser um modelo de organização interessante, um dos pontos negativos é o elevado número de interações que ele pode apresentar.

2.6.3 Coordenação

Para que possa existir coordenação em uma aplicação, um número de problemas precisa ser tratado. Primeiramente é necessário definir uma linguagem e um protocolo para a interação. O primeiro especifica a sintaxe e semântica da comunicação. O segundo especifica as regras que dizem quem pode dizer o que durante a interação. É necessário especificar a forma como os agentes irão raciocinar sobre as ações, planos e conhecimento sobre eles mesmos e outros. Terceiro, é a necessidade de identificar quão coerente o comportamento do sistema pode estar relacionado aos agentes que têm uma visão parcial do problema.

2.6.4 Negociação e Protocolos de Leilão

Em sistemas compostos de múltiplos agentes autônomos, negociação é uma forma chave de interação pois possibilita que grupos de agentes cheguem a um acordo mútuo com respeito a alguma crença, objetivo ou plano [15]. As técnicas de negociação são oriundas de teorias da área de economia e teoria dos jogos [80] e são importantes porque em um cenário onde os agentes são autônomos e não são benevolentes, uma forma de interação, onde os agentes podem convencer outros agentes a realizarem certas ações, é extremamente importante.

O processo de negociação pode variar de diferentes formas, como leilões, protocolos como *Contract-Net* e argumentação. Algumas estratégias se baseiam em técnicas de negociação seguindo o modelo “muitos-para-muitos” ou de várias ocorrências “um-para-muitos” simultaneamente, usando um único parâmetro – preço -

como base para a negociação. Os contratos negociados entre os agentes no cenário “um-para-muitos” também podem envolver múltiplos parâmetros. Alguns trabalhos teóricos vêm sendo realizados com a intenção de provar a efetividade de tal abordagem em ambientes complexos. Um estudo envolvendo um modelo baseado na teoria dos jogos para o balanceamento de carga em *grids* de agentes é apresentado por Shen e outros [107].

Jennings identificou três áreas de pesquisa relacionadas ao tema negociação: os protocolos de negociação, que são as regras que regem as interações; os objetos da negociação, que são os objetivos do contrato, como preço, tempo, etc.; os modelos de raciocínio dos agentes que proporcionam os mecanismos para os agentes raciocinarem e atingirem os seus objetivos. Beer e outros [15] sugerem que os agentes devem possuir um mecanismo de raciocínio deliberativo, que lhe permite ter uma visão de como o ambiente poderá estar, para que possam negociar de maneira efetiva.

2.6.5 Vantagens dos Sistemas Multi-Agentes

A abordagem de sistemas multi-agentes como desenvolvimento de aplicações pode apresentar uma série de vantagens sobre modelos centralizados. Algumas destas vantagens são apresentadas por *Sycara* [113]:

- Um SMA distribui os recursos computacionais e as capacidades em uma rede de agentes interconectados. Enquanto um sistema centralizado pode encontrar limitações de recursos, gargalos de desempenho ou falhas críticas, um SMA é descentralizado e não enfrenta o problema de "único ponto de falha" associado com os sistemas centralizados.
- Um SMA permite a interconexão e interoperabilidade de múltiplos sistemas legados existentes. Através da construção de um invólucro (*wrapper*) para estes sistemas, eles podem ser integrados a sociedade de agentes.
- Um SMA modela problemas em termos de interação autônoma entre agentes componentes, e são a maneira mais natural de representar a alocação de tarefas, planejamento de equipes, preferências de usuários, ambientes abertos e assim por diante.

- Um SMA eficientemente recupera, filtra e coordena informação de maneira global de fontes que estão esparsamente distribuídas.
- Um SMA provê soluções em situações onde a experiência ou inteligência está espalhada e temporariamente distribuída.
- Um SMA melhora o desempenho geral do sistema, especialmente com relação às dimensões de eficiência computacional, confiabilidade, extensibilidade, robustez, sustentabilidade, correspondência, flexibilidade e reusabilidade.

2.7 Engenharia Baseada em Agentes para Sistemas Complexos

Alguns problemas geralmente são difíceis de serem modelados e o desenvolvimento de software para estas atividades é extremamente complexo, como é o caso de um *grid*. É uma tarefa que diz respeito à engenharia de software prover meios que facilitem o desenvolvimento de tais sistemas. Embora sejam complexos geralmente os sistemas possuem algumas regularidades, conforme apresenta Jennings [69]:

- Eles podem ser decompostos hierarquicamente de forma que percebemos que um sistema complexo é composto por sistemas menores inter-relacionados;
- A identificação do que é primitivo no sistema cabe inteiramente aos objetivos do programador e depende dos seus objetivos;
- Sistemas hierárquicos se desenvolvem mais rapidamente que sistemas não hierárquicos;
- É possível distinguir as atividades e interações entre componentes dos subsistemas e as existentes entre os subsistemas. Como geralmente o primeiro tipo de interação é mais freqüente, é evidente que os sistemas complexos podem ser decompostos: sistemas menores podem ser tratados de forma separada, embora as interações entre eles devam ser consideradas. Algumas destas interações podem ser constatadas durante o desenvolvimento, enquanto que outras não.

Uma visão canônica abstrata de um sistema complexo pode ser vista na Figura 5. A hierarquia natural do sistema é expressa através das ligações de relação, componentes dentro de um subsistema são conectados através de interações freqüentes, e interações entre componentes são expressas através de ligações não-freqüentes.

Feitas estas observações, os engenheiros de software têm em mãos um número de ferramentas que são úteis no gerenciamento da complexidade:

Decomposição: A técnica mais básica para tratar problemas complexos é proceder a sua divisão em tarefas menores e mais gerenciáveis e que podem ser tratadas de uma forma mais isolada. Isto ajuda a reduzir o escopo do desenvolvimento.

Abstração: O processo de definir um modelo simplificado do sistema enfatiza alguns dos detalhes e propriedades enquanto que outros são ocultados. Esta abordagem também é funcional porque novamente limita o escopo do trabalho.

Organização: O processo de definição e gerenciamento das inter-relações entre os vários componentes do sistema. A habilidade em especificar e representar as relações organizacionais ajuda os engenheiros de software a tratar a complexidade: Possibilitando que os componentes básicos sejam agrupados e tratados como unidades de alto nível da análise, e prover meios de descrever as relações de alto nível entre as várias unidades.

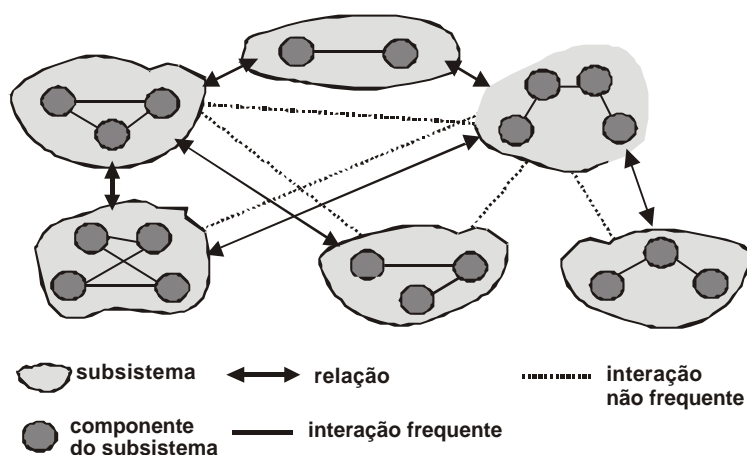


Figura 5 - Visão geral de um sistema complexo [69].

2.7.1 Análise e Desenvolvimento Orientados por Agentes

Uma metodologia de análise e desenvolvimento tem o objetivo de auxiliar no entendimento e refinamento do conhecimento acerca de um sistema a ser construído e posteriormente como desenvolvê-lo. Metodologias geralmente consistem de uma coleção de modelos e de um conjunto de diretivas associadas com estes modelos. Os modelos têm o objetivo de formalizar o entendimento do sistema que está sendo

considerado. Tipicamente estes modelos de análise iniciam com tentativas bastante abstratas e vão sendo refinados até serem bastante concretos e bem próximos à implementação.

Para Jennings [69], a adoção de uma abordagem de desenvolvimento de software orientada por agentes consiste em decompor o problema em múltiplos componentes autônomos que podem atuar e interagir de uma maneira flexível na busca de um objetivo conjunto.

Quando adotamos uma abordagem baseada em agentes para o desenvolvimento de *software*, fica claro que os sistemas irão requerer múltiplos agentes com atividades distintas, de forma que a natureza do sistema possa ser distribuída. Os agentes precisam interagir entre si no intuito de atingir os seus objetivos. Estas interações podem variar, desde simples interações até interações mais complexas como negociações e interações sociais.

Segundo Wooldridge [125] existem alguns fatores que tornam uma abordagem baseada em agentes mais adequada a alguns tipos de sistemas. Alguns dos fatores que podem justificar tal escolha são:

O ambiente é aberto, ou pelo menos dinâmico, incerto ou complexo. Em tais ambientes, sistemas capazes de ações autônomas constituem normalmente a única solução.

Os agentes são uma metáfora natural. Muitos ambientes (incluindo organizações, ambientes comerciais e competitivos) são naturalmente modelados como sociedades de agentes, que podem cooperar uns com os outros para resolver problemas complexos ou competir uns com os outros.

Distribuição de dados, controle ou conhecimento. Em alguns ambientes a distribuição de dados, controle ou conhecimento significam que uma solução centralizada é extremamente difícil ou impossível.

Sistemas legados. Um problema crescente com o qual os engenheiros de software têm que lidar é o dos sistemas legados: são *softwares* que estão tecnologicamente obsoletos, mas a sua funcionalidade é essencial à organização. Tais softwares não podem ser descartados devido ao elevado custo em reescrevê-los em curto prazo. Em alguns casos eles têm que ser integrados com outros sistemas de uma forma que nunca havia sido prevista pelos engenheiros. Uma forma é envolver os componentes legados, o

que é chamado de *wrapper* pela comunidade, provendo funcionalidades de agente que possibilite que eles possam se comunicar e cooperar com outros componentes de software.

Outros fatores em favor da engenharia de software baseada em agentes são [69]: as decomposições baseadas em agentes como uma forma efetiva de divisão do problema; as abstrações das intenções e objetivos orientados por agentes são uma forma de modelar sistemas complexos e; a filosofia orientada por agentes para modelagem e gerenciamento das relações organizacionais é apropriada para tratar as dependência e interações que existem em sistemas complexos.

2.8 Padrões para Construção de Sistemas de Agentes

A inexistência de padrões para o desenvolvimento de sistemas baseados em agentes pode levar a existência de inúmeras plataformas diferentes entre si e sem nenhum grau de interoperabilidade. A exemplo do que acontece com qualquer outra tecnologia, a falta de padronização pode levar a um cenário onde cada grupo de desenvolvimento possui a sua plataforma isolada impossibilitando que as tecnologias de agentes sejam aplicadas em ambientes abertos e de larga escala.

Uma necessidade em um cenário de *grids* de agentes é a existência de padrões e especificações abertas que garantam a interoperabilidade entre diferentes sistemas de agentes. Desta forma os agentes podem fazer uso de serviços oferecidos por outros sistemas de uma maneira fácil e uniforme.

Com este objetivo é que a **FIPA** (*The Foundation for Intelligent Physical Agents*) foi criada. A **FIPA** é uma organização sem fins lucrativos, criada em 1996 por um consórcio de empresas com o objetivo de criar especificações que garantam a interoperabilidade em sistemas de agentes, facilitando a sua aceitação pela indústria e sua aplicação em ambientes reais e abertos.

A **FIPA** tem criado especificações para pontos chaves e críticos que merecem ser padronizados para garantir a comunicação e interoperabilidade entre plataformas. A **FIPA** não define o comportamento interno dos agentes, mas como devem funcionar os mecanismos externos, como comunicação. A conformidade com os padrões **FIPA** garante que diferentes agentes, escritos em diferentes linguagens ou construídos em diferentes plataformas podem existir e interagir em um mesmo ambiente.

As especificações de agentes da **FIPA** consistem basicamente de:

- Modelos formais de desenvolvimento que podem ser verificados usando lógicas de prova, mas cuja complexidade de implementação não pode ser verificada formalmente.
- Modelos descritivos que definem um mapeamento bem feito entre o *design* e implementação e que verificam a implementação de pontos específicos.

2.8.1 O Que Deve Ser Padronizado

Alguns pontos críticos devem ser padronizados quando se quer atingir um grau coerente de interoperabilidade. A padronização destes serviços garante a interoperabilidade que se deseja em um sistema de agentes. Alguns destes pontos são:

- **Comunicação:** diálogos, primitivas de comunicação ou atos de fala, conteúdo, ações e ontologias.
- **Papéis de comunicação:** especifica a escolha de um ato de fala e o diálogo. Também envolve os modelos de comunicação, como ponto-a-ponto, cliente-servidor, gerente-contratado.
- **Serviços de suporte à comunicação:** transporte (codificação das mensagens), serviços de diretório e nomeação.
- **Organização e arquitetura:** plataformas para sistemas de agentes, domínios e arquiteturas abstratas.

2.8.2 A Arquitetura Abstrata da FIPA

A arquitetura da **FIPA** [43] apresenta um arcabouço geral para promover a interoperabilidade e reusabilidade. Ela identifica elementos comuns às plataformas e sistemas de agentes e provê especificações para a padronização destes elementos.

A arquitetura **FIPA** não pode ser simplesmente implementada, mas serve de referência para que sejam criadas outras especificações mais concretas e por fim, implementações que são chamadas de “realizações” da arquitetura abstrata. A arquitetura descreve como os sistemas devem ser construídos, que serviços devem ter, como deve ser a comunicação, os protocolos de rede que podem ser utilizados para

transporte, as codificações e outros fatores. Para uma “realização” ou arquitetura concreta ser compatível com a **FIPA**, ela deve ter ao menos um serviço de registro e descoberta de agentes e um serviço de transferência de mensagens. Estes são os elementos mínimos para que uma arquitetura seja compatível com a **FIPA**.

Vale ressaltar que nem todos os mecanismos descritos na arquitetura **FIPA** precisam ser implementados para que uma arquitetura concreta seja compatível com a **FIPA**. Uma concretização da arquitetura pode ter apenas um meio de codificação de mensagens e fazer uso de apenas um protocolo de transporte. Neste caso dizemos que a arquitetura concreta implementa apenas parte da arquitetura abstrata, o que não descaracteriza a sua compatibilidade. Vale ressaltar ainda, que a arquitetura não limita a introdução de elementos adicionais em uma realização. A escolha dos elementos como linguagem e ferramentas usadas na concretização da arquitetura também ficam a critério dos projetistas, sendo que a **FIPA** não impõe qualquer restrição a respeito disso. Um exemplo desta realização é apresentado na Figura 6.

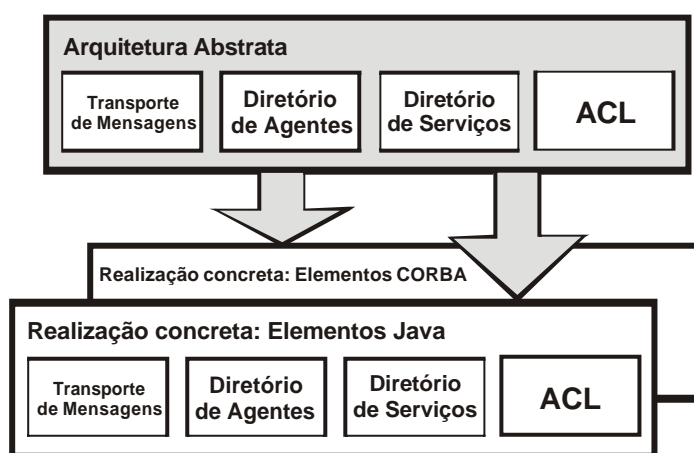


Figura 6 - Realizações da arquitetura abstrata [43].

2.8.3 A Plataforma de Agentes FIPA

A plataforma da **FIPA** é um refinamento da arquitetura abstrata e define um conjunto de elementos que uma plataforma de agentes deve possuir para que seja compatível com a plataforma **FIPA**.

A especificação **FIPA Agent Management Specification** [45] define com detalhes alguns elementos que na arquitetura abstrata não são abordados com profundidade. Esta

especificação provê normativas para uma plataforma de agentes na qual os agentes **FIPA** existem e interagem. Ela estabelece uma referência lógica para a criação, registro, localização, comunicação, migração e retirada de agentes.

Esta arquitetura define alguns elementos que uma plataforma deve ter para ser compatível como uma plataforma **FIPA**. Alguns dos elementos definidos são o Sistema de Gerenciamento de Agentes (AMS), o Facilitador de Diretório (DF) e um Sistema de Transporte de Mensagens (MTS).

Embora estes elementos não sejam todos obrigatórios para se garantir a interoperabilidade, eles nos fornecem uma idéia dos elementos e atributos dos agentes que devem ser registrados no diretório para que eles possam ser descobertos, baseados em seus atributos e nos atributos dos serviços por eles oferecidos.

Na plataforma da **FIPA**, o AMS é responsável por manter um serviço de páginas brancas que é utilizado pelos agentes dentro da plataforma, além de outras funções de gerenciamento referentes ao ciclo de vida e atividades dos agentes da plataforma. O DF fornece um serviço de páginas amarelas e permite a descoberta de agentes e de serviços baseados em seus atributos. Ambos os serviços são aprimoramentos do serviço de diretórios especificado na arquitetura abstrata da **FIPA** [43].

Para compreender os elementos que são registrados em um serviço de diretórios, é importante conhecer o conceito de “Identificador de Agente” (AID). Um AID é um elemento utilizado para a identificação de um agente. Esta identificação é registrada no DF e AMS. Um AID é um aprimoramento do conceito de entrada de diretório apresentado na arquitetura abstrata. Um AID permite que atributos adicionais sejam inseridos de uma maneira fácil e extensível. Na Figura 7 são apresentados os elementos da plataforma **FIPA**.

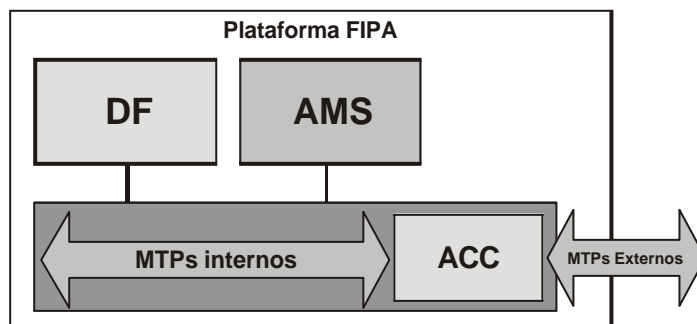


Figura 7 - Plataforma FIPA na visão do **FIPA-OS** [50].

Uma plataforma de agentes que siga as especificações da **FIPA** para o transporte de mensagens pode ter sistemas de transporte de mensagens internos, que são utilizados pelos agentes da mesma plataforma, e sistemas de transporte externos utilizados para a comunicação dos agentes com agentes de outras plataformas. Neste caso, para a comunicação interna podem ser utilizados mecanismos como RMI e CORBA (**MTPs internos**), enquanto que para a comunicação externa a plataforma, mecanismos mais ubíquos - como o protocolo HTTP - se mostram mais atraentes (**MTPs externos**).

2.9 Conclusões

Neste capítulo foram apresentados alguns fundamentos e aspectos dos agentes de software relevantes a este trabalho. No Capítulo 4, que apresenta os *grids* de agentes, percebe-se que o *grid* se assemelha mais a um sistema multi-agentes do que um ambiente de resolução de problemas distribuído. Conceitos como comunicação entre agentes, colaboração, negociação e distribuição de tarefas são essenciais em um ambiente de *grids* de agentes.

O capítulo procurou inicialmente mostrar uma visão geral do que é um agente, as classificações e arquiteturas utilizadas. Em uma segunda parte os agentes foram apresentados como elementos chave da inteligência artificial distribuída. Foram apresentadas algumas teorias que colocam os agentes como elementos para engenharia de sistemas e como modelos de programação. Por fim, o capítulo é finalizado com algumas especificações FIPA para sistemas baseados em agentes.

3 Computação em *Grid*

A última década foi marcada por uma melhoria significativa das tecnologias de rede, pelo aumento no desempenho e disponibilidade dos computadores e pela sofisticação dos recursos de software [20]. Com estes fatores, veio à tona a idéia de agregar recursos computacionais possibilitando o seu uso compartilhado na execução de aplicações comuns. O termo *grid* foi criado no final dos anos 90 para descrever uma arquitetura que possibilitava que um conjunto de recursos geograficamente distribuídos pudesse ser usado para dar suporte às aplicações de larga escala.

O *grid* tem a sua origem como uma analogia ao sistema de distribuição de energia elétrica, no inglês chamado de *Electrical Power Grid*, o qual é capaz de proporcionar aos seus usuários um acesso transparente, fácil, barato, onipresente e seguro ao recurso ‘energia elétrica’ através de um emaranhado de pontos de geração e linhas de transmissão, escondendo do usuário detalhes a respeito de como a energia é gerada e distribuída. Da mesma forma, acredita-se que em um futuro próximo as tecnologias de *grid* serão capazes de proporcionar o acesso pelos usuários a recursos computacionais da mesma forma que o sistema de distribuição de energia, bastando o usuário se conectar ao *grid* e obter os recursos de que precisa. Este exemplo faz com que o usuário veja o *grid* como uma fonte inesgotável de recursos. A Figura 8 ilustra abstratamente esta idéia.

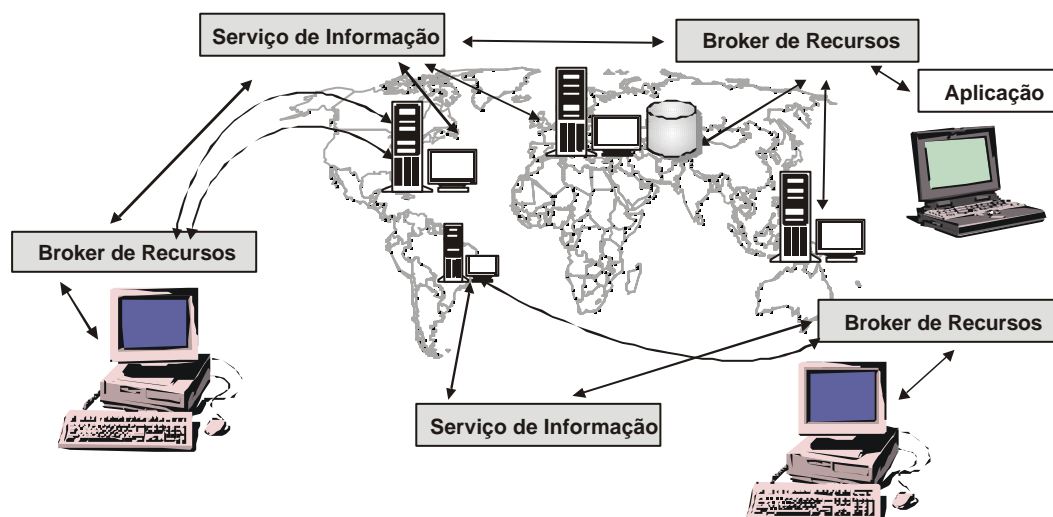


Figura 8 - O *grid* como uma fonte de recursos (Adaptado de [20]).

Um *grid* pode possibilitar a agregação e compartilhamento de uma variedade de recursos que podem incluir supercomputadores, *clusters*, dispositivos de armazenamento e fontes de informação, que estejam distribuídos, e sob a administração de diferentes organizações. Estes recursos podem ser usados na solução de problemas complexos e que requerem um grande poder computacional, que são as chamadas aplicações de grande desafio [91], típicas nas áreas de ciência, comércio e engenharia. Tais aplicações são extremamente exigentes em termos de consumo de recursos de processamento e capacidade de armazenamento que requerem [53].

Inicialmente o *grid* foi caracterizado como a infra-estrutura que possibilita a agregação de recursos distribuídos na resolução de problemas complexos e de larga escala nas áreas de ciência, comércio e engenharia [53]. Neste caso, a complexidade e tamanho destes problemas requerem uma quantidade de recursos que não estão disponíveis em computadores especializados como supercomputadores e *clusters* atuais e que não podem ser reunidos em um mesmo domínio administrativo. Segundo Berman e outros [16] a infra-estrutura de *grid* irá nos proporcionar a habilidade de ligar dinamicamente recursos, formando agrupamentos para suportar a execução de aplicações de larga escala e distribuídas, intensivas no consumo de recursos.

Segundo Buyya [20], alguns aspectos caracterizam um *grid*, diferenciando-o das tecnologias distribuídas e paralelas tradicionais:

- **Múltiplos Domínios Administrativos e Autonomia** – Os recursos do *grid* estão geograficamente distribuídos em múltiplos domínios administrativos e em poder de diferentes organizações. A autonomia dos proprietários dos recursos precisa ser respeitada juntamente com as suas políticas de gerenciamento e uso.
- **Heterogeneidade** – Um *grid* envolve uma multiplicidade de recursos que são heterogêneos e compostos por uma diferente gama de tecnologias.
- **Escalabilidade** – Um *grid* pode crescer de uma quantidade pequena de recursos até a faixa de milhões.
- **Dinamicidade e Adaptabilidade** – Em um *grid*, a falha de um recurso não é uma exceção e sim uma regra. De fato, com inúmeros recursos em um *grid* a probabilidade de algum recurso falhar é muito alta. Os gerenciadores de recursos ou as aplicações devem saber como adaptar o seu comportamento

dinamicamente e fazer uso dos serviços e recursos de maneira eficiente e efetiva.

Para entender as iniciativas de computação em *grid* é necessário levar em consideração alguns fatores históricos [16], analisando um pouco do passado que demonstra o amadurecimento de uma série de tecnologias. O *grid* pode ser visto como um estágio evolucionário de uma porção de tecnologias já existentes, como por exemplo, intensos estudos em computação paralela e distribuída nas décadas de 80 e 90, seguidas por iniciativas de construir infra-estruturas de computação paralela em larga-escala e altamente distribuídas. Na próxima seção são analisados alguns dos fatores que contribuíram para o surgimento das tecnologias de *grid* existentes atualmente.

3.1 Fatores Motivadores

Uma série de fatores e a evolução de uma porção de tecnologias culminaram com o surgimento das infra-estruturas de *grid* propostas e discutidas atualmente. Chetty e Buyaa [27], Berman e outros [16] e Foster e Kesselman [53] destacam alguns fatores que contribuíram para o surgimento da computação em *grid*. Na Figura 9 são apresentados alguns fatores que contribuíram para o surgimento da computação em *grid*, na visão de Chetty e Buyaa.

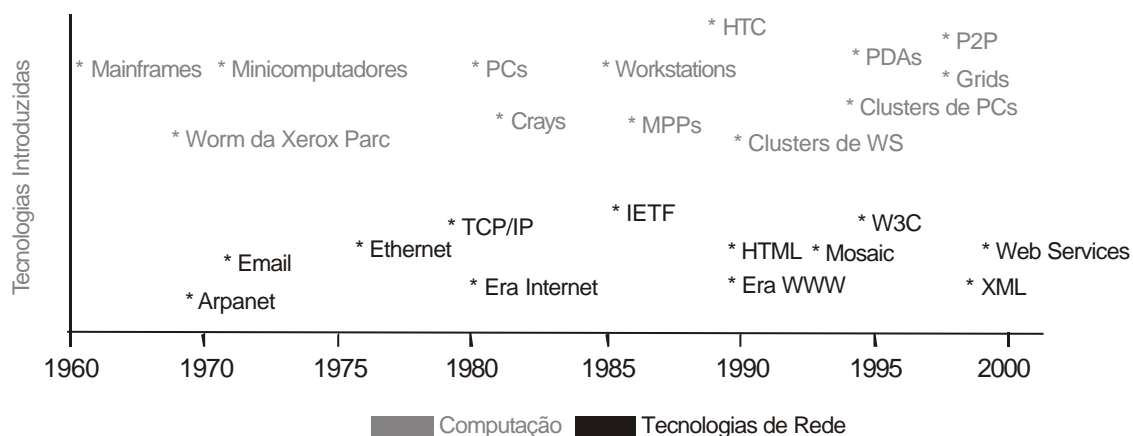


Figura 9 – Fatos que contribuíram para a computação em *grid* [27].

Buyya divide os fatores em computacionais e de comunicação. Com relação a comunicação, os principais marcos históricos relatados são a Internet, que surgiu como um projeto modesto do **DARPA** em 1969: a ARPANET, com apenas 4 nós na época. Mais tarde, em 1970, a ARPANET já contava com mais de 30 universidades

interligadas. Em 1973, a tese de PhD de Bob Metcalfe [83] apresentava a Ethernet como uma idéia para interligar computadores e periféricos. A Ethernet se tornou uma realidade em 1976 e contribuiu em muito no sucesso dos *clusters* e redes de computadores atuais.

Em 1983 a ARPANET ainda era uma rede com algumas centenas de nós. O que possibilitou a sua expansão foi a iniciativa do **NSF** - *National Science Foundation* - de utilizá-la como suporte a colaboração de centros de supercomputação e mais tarde a transferência da responsabilidade da ARPANET dos interesses militares para os acadêmicos. Isso possibilitou uma expansão surpreendente, conforme destaca a Figura 10 [66], que apresenta o crescimento no número de *hosts* disponíveis na Internet.

Embora a infra-estrutura da Internet tenha sido muito importante e tenha possibilitado a interligação de centros de computação, o que possibilitou a expansão e sua utilização por muitos projetos de *grid* e computação distribuída foram o surgimento do e-mail e da *Web*, conforme destaca [16]. Estas foram as aplicações de maior impacto na popularização e expansão da Internet.

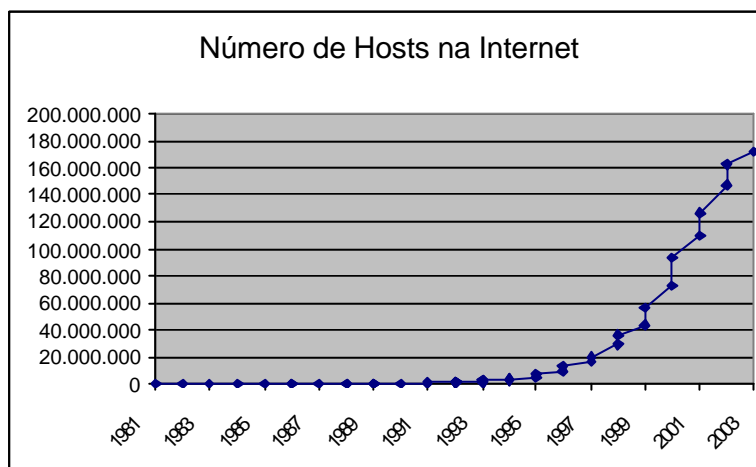


Figura 10 - Número de *hosts* disponíveis na Internet [66].

São ainda destacados os fatores computacionais. Os fatores relacionados à evolução dos recursos de comunicação também tiveram uma grande influência no aspecto computacional. Na década de 80 os pesquisadores trabalharam intensivamente no desenvolvimento e aprimoramento de algoritmos paralelos para o processamento de informações. A medida que estes códigos foram crescendo e se tornando mais complexos, o seu processamento se mostrou impraticável nas arquiteturas paralelas da época [102]. A existência de tecnologias de rede como a Ethernet e a disponibilidade de

computadores do tipo PC possibilitaram o surgimento de arquiteturas como *clusters* para o processamento de alto desempenho. A existência destes *clusters* em várias instituições e a existência de uma tecnologia como a ARPANET despertaram o interesse de interligar estes centros e *clusters* e utilizá-los no processamento de aplicações de interesse mútuo.

Inicialmente acreditava-se que o *grid* era até então uma possibilidade de estender aplicações paralelas com um alto grau de acoplamento que faziam uso de *clusters*. Porém na prática o *grid* tem se mostrado uma solução onde as aplicações utilizadas são fracamente acopladas, embora arquiteturas com um alto grau de acoplamento possam ser elementos do *grid*.

3.1.1 Evolução dos Microprocessadores

Nas últimas décadas o desenvolvimento dos processadores tem apresentado um avanço significativo. Hoje encontramos em um computador pessoal, processadores dezenas ou até centenas de vezes mais poderosos do que os de supercomputadores de uma ou duas décadas atrás. Em 1965, Moore [85] previu que o número de componentes, ou transistores, existentes nos processadores iria seguir um crescimento exponencial, dobrando em quantidade a cada 18 meses. Atualmente presenciamos com um certo assombro a concretização de sua previsão. Um exemplo disso pode ser percebido na Tabela 1, que apresenta o número de transistores contidos nos processadores Intel fabricados entre 1970 e 2000. Na Figura 11 é apresentado o crescimento da capacidade de processamento dos processadores, também da empresa Intel. Embora este cenário de crescimento possa ser mantido até 2010 [92], Berman e outros [16] ressaltam que as aplicações científicas se tornarão cada vez mais exigentes em termos de poder de processamento de que necessitam.

Tabela 1 - Aumento do número de transistores entre 1970 e 2000.

Processador	Ano de Introdução	Número de Transistores
4004	1971	2.250
8008	1972	2.500
8080	1974	5.000
8086	1978	29.000
286	1982	120.000

386™	1985	275.000
486™ DX	1989	1.180.000
Pentium®	1993	3.100.000
Pentium II	1997	7.500.000
Pentium III	1999	24.000.000
Pentium 4	2000	42.000.000

O barateamento dos componentes como transistores, possibilitou que processadores e computadores fossem construídos a um custo muito mais atrativo e, junto com o avanço nas tecnologias de comunicação, possibilitou o agrupamento de vários processadores, o que deu origem às arquiteturas de processamento paralelo e distribuído atuais. Netravali [92] apresenta informações e previsões interessantes a respeito do desenvolvimento dos processadores, redução de custo de produção dos transistores e o reflexo de tudo isto na evolução das tecnologias de rede, processamento e armazenamento.

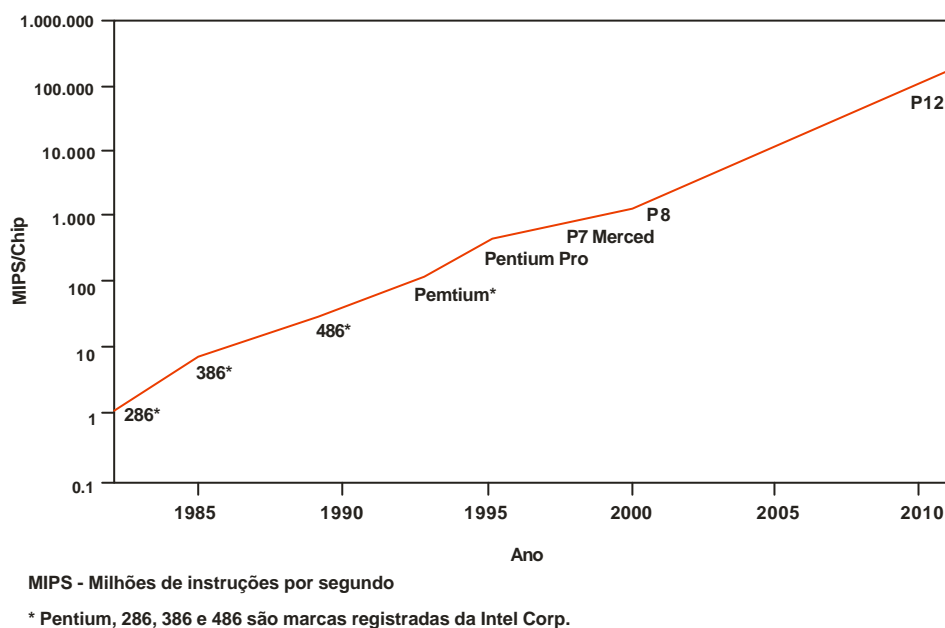


Figura 11 - Evolução dos processadores Intel [92]

Porém, junto com a evolução dos processadores existe um crescimento na complexidade das aplicações distribuídas. Antigamente, cientistas ficavam felizes quando conseguiam processar um pequeno conjunto de informação que atualmente, com o poder computacional instalado, chega a ser ridículo. Um exemplo demonstrando a evolução de uma aplicação de previsão de tempo é apresentado por von Laszewski [120]. Inicialmente as previsões meteorológicas eram extremamente simples e levavam

em consideração apenas um pequeno número de variáveis. Atualmente, inúmeros fatores climáticos e ambientais são levados em conta e as aplicações meteorológicas são cada vez mais intensivas em termos da quantidade de dados que precisam manipular e do poder de processamento requerido. Assim como o estudo de fatores climáticos, existem inúmeras outras aplicações na área de física, química e biologia que requerem um grande poder computacional para processamento [60].

3.1.2 Evolução das Tecnologias de Armazenamento

Embora os processadores tenham tido um significativo avanço nas últimas décadas, eles estão ficando para trás dos dispositivos de armazenamento [23]. De acordo com Foster e Kesselman [53], a capacidade destes dispositivos dobra a cada 12 meses e este aumento tem profunda importância na computação em *grid*. Esta melhoria se reflete tanto na quantidade de informação que conseguem armazenar, quanto na velocidade de acesso. Estas melhorias se devem principalmente a fatores como aprimoramentos físicos das mídias de armazenamento, nas cabeças de leitura e no posicionamento e velocidade das cabeças de leitura, conforme descreve Netravali [92].

Conforme Ruettgers previu, previsão que ficou conhecida como lei de Ruettgers, a demanda por armazenamento tende a dobrar a cada sete meses e de acordo com a sua previsão, em 2003 esta demanda seria da ordem de *Petabytes*. Embora os dispositivos tenham avançado de maneira significativa, como pode ser constatado em [96], as previsões de Licklider [65], de que no futuro a quantidade de dados que as aplicações científicas irão gerar e manipular serão de dimensões galácticas, tendem a se concretizar. Tem-se exemplos claros quando nos deparamos com projetos que mencionam uma capacidade de armazenamento na ordem de *Petabytes*. Tão difícil quanto armazenar estes dados, é prover mecanismos para uma recuperação eficiente.

3.2 Evolução da Computação em *Grid*

Conforme destacam De Roure e outros [102] as primeiras iniciativas de computação em *Grid* tinham o objetivo de interligar supercomputadores e centros de supercomputação dispersos geograficamente. Tais projetos ficaram conhecidos como precursores do *grid*. Como pioneiros neste contexto pode ser citado o projeto I-WAY

[52] que mais tarde deu origem ao projeto Globus [54], atualmente um dos projetos mais expressivos em *middleware* e ferramentas para computação em *grid* e usado por vários projetos americanos e europeus, e o FAFNER [40]. Em seguida o *grid* passou a ser usado como uma plataforma para a resolução de aplicações de grande desafio [91], que até então tinham como plataforma de execução os *clusters* e outras arquiteturas paralelas. Esta abordagem permitia reunir uma quantidade de recursos muito maior e a um custo muito menor do que as tecnologias de *cluster* e computação paralela existentes. Em função desta necessidade, mais tarde começaram a aparecer vários projetos propondo *middlewares*, sistemas operacionais e ferramentas para proporcionar esta computação em larga-escala.

Um pouco distante do escopo original, percebemos várias modalidades de *grids*, desde *grids* computacionais [53] e de dados, que compartilham ciclos de processamento e dispositivos de armazenamento, até *grids* de informação e conhecimento [68] que se preocupam com a integração de diferentes bases de informação e com a representação e distribuição mais semântica destes dados armazenados. O *grid* também tem possibilitado o compartilhamento de instrumentos científicos em um ambiente colaborativo.

Atualmente encontramos o *grid* em várias modalidades de aplicações com diferentes finalidades e com um escopo e conceitos diferentes. Neste contexto, a representação do *grid* em camadas, definindo três modalidades possíveis de *grid* é apropriada. Nesta classificação, apresentada por Jeferry [68], temos as três camadas descritas a seguir.

3.2.1 *Grid* computacional

Este é nível mais baixo, preocupado com a reunião em larga escala de recursos computacionais e de armazenamento. Ou seja, como os recursos computacionais são alocados, como o escalonamento das tarefas é feito e como elas são realizadas por estes recursos. Esta reunião de recursos pode ser utilizada com o objetivo de agregar um poder de processamento equivalente ou superior ao de um supercomputador e utilizá-lo no processamento de grandes volumes de dados. O recurso neste caso pode não ser apenas poder de processamento, mas também bases de dados ou instrumentos

científicos. Um *grid* computacional é definido com detalhes em [53] e [20]. Uma infraestrutura compartilhada para possibilitar esta agregação de recursos é necessária para possibilitar o monitoramento e controle dos recursos.

Esta camada também é referenciada como camada de dados, pois em alguns projetos ela é usada para designar o agrupamento de várias fontes e dispositivos de armazenamento de dados. Neste caso, os dados são apenas dados ‘*crus*’ sem nenhuma interpretação, ou seja, seqüências de bits e bytes.

3.2.2 *Grid* de informação

Camada intermediária, permitindo o acesso uniforme às diferentes fontes de informação e tornando possível que os serviços possam ser executados em recursos computacionais distribuídos. Esta camada procura localizar, integrar e gerenciar as diferentes fontes de informação. Trata de como a informação é representada, armazenada, acessada, compartilhada e mantida.

3.2.3 *Grid* de conhecimento

É a camada mais alta do modelo e proporciona serviços especializados, os quais podem procurar por padrões nos repositórios de dados existentes e gerenciar os serviços de informação. O *grid* de conhecimento pode auxiliar na tomada de decisões e na interpretação das informações manipuladas no *grid* de informação. Esta camada se preocupa em como o conhecimento é adquirido, usado, recuperado, publicado e mantido, para que possa orientar os cientistas ou usuários do *grid* no cumprimento de suas metas e objetivos. Nesta camada do *grid* o conhecimento é usado como uma informação que é compreendida e pode ser aplicada para que um objetivo seja alcançado, para resolver um problema ou na tomada de uma decisão.

3.3 Computação de Alto Desempenho

Um ambiente de computação de alto desempenho geralmente se caracteriza por ser um sistema "mais bem comportado" do que um *grid*: sem falhas, requisitos mínimos de segurança, consistência do estado entre aplicações componentes, disponibilidade de

informação global e políticas simples de compartilhamento de recursos [23]. À medida que as aplicações se tornam mais distribuídas, as características dos sistemas fortemente acoplados tendem a serem quebradas, pois tais cenários não apresentam as mesmas condições de ambientes pequenos ou restritos a uma mesma área administrativa.

3.4 Grid Versus Cluster

De acordo com Buyaa [21], “um cluster é um tipo de sistema de processamento paralelo ou distribuído, o qual consiste de uma coleção de computadores interconectados que trabalham juntos como se fossem um único recurso integrado”.

Algumas características de um *cluster* são apresentadas por [95] *apud* [5]:

- Um *cluster* pertence a um domínio administrativo único, e por isso é utilizado para resolver problemas de um limitado grupo de pesquisa ou organização;
- ele é visto externamente como um recurso computacional único.
- a latência nas comunicações é pequena;
- a taxa de transmissão disponível é elevada, pois normalmente as entidades do *cluster* estão ligadas por uma rede local;
- pode se beneficiar de protocolos de comunicação mais eficientes entre as suas entidades;
- a segurança na comunicação entre as entidades não tem muita importância, chegando mesmo a ser desnecessária, caso a rede de interconexão dos elementos do *cluster* não esteja conectada a rede de acesso ao *cluster*;
- pode-se ter homogeneidade de *hardware* e *software* nas suas entidades, facilitando o desenvolvimento de aplicações;
- sua escala é limitada, da ordem de dezenas de entidades.

Normalmente um *cluster* é composto de duas ou mais entidades de processamento interligadas por uma rede de comunicação que pode ser uma **LAN**. Tais sistemas podem apresentar uma relação de custo/benefício muito mais efetiva do que as arquiteturas tradicionais de processamento paralelo. A arquitetura típica de um cluster é apresentada na Figura 12 [21].

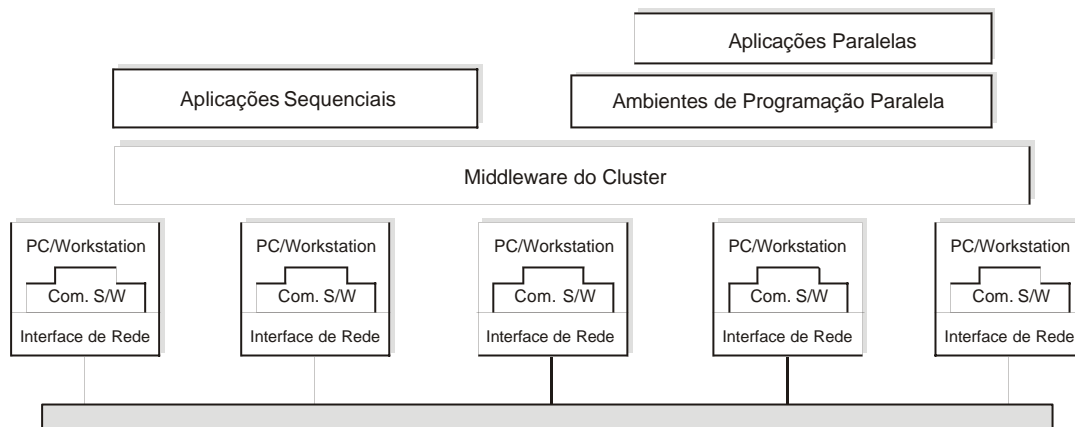


Figura 12 - Arquitetura de um cluster de computadores [21].

Um cluster é composto basicamente pelos seguintes componentes [21]:

- Múltiplos computadores de auto desempenho;
- Sistemas operacionais (Windows, Unix, Linux);
- Redes e *switches* de alto desempenho;
- Interfaces de rede;
- Protocolos e serviços para uma comunicação rápida;
- *Middleware* do cluster;
- Ambientes para programação paralela e ferramentas;
- As aplicações do *cluster*.

Alguns fatores diferenciam os *clusters* de um *grid*. Primeiramente o fato de o *cluster* estar geralmente restrito a um único domínio administrativo e seus recursos não serem tão heterogêneos quanto os recursos existentes em um *grid*. Muitas vezes, por questões estratégicas, é comum adotar um mesmo padrão de entidades de processamento em um *cluster*.

Outro ponto importante é que os recursos de um *cluster* estão concentrados em uma mesma área geográfica, ao contrário do que pode acontecer com um *grid*, onde podem existir recursos que estejam geograficamente distribuídos. Além disso, no *cluster* os recursos estão, geralmente a disposição única e dedicada do *cluster*, ao contrário do que acontece em um *grid* onde os recursos podem estar sendo utilizados pelos usuários, que são os reais proprietários dos recursos.

Ainda a escalabilidade das arquiteturas e pacotes de *middleware* empregados nos *clusters* permitem a utilização de apenas uma quantidade de entidades de processamento

na ordem de algumas dezenas ou centenas, o que impossibilita a sua utilização como plataforma de processamento das chamadas aplicações de grande desafio.

3.5 Grid Versus P2P

P2P são classes de aplicações que tiram vantagens de recursos - armazenamento, ciclos, conteúdo e presença humana - disponíveis nas bordas da Internet [109]. Por bordas da Internet entende-se os recursos oferecidos pelos usuários finais. O termo é usado como referência a um modelo de rede que permite a um grupo de usuários com um mesmo programa de rede conectarem uns com os outros e compartilharem arquivos ou ciclos de CPU. O *Gnutella* é um exemplo clássico de *software* que utiliza o conceito **P2P**. O que caracteriza este modelo é o acesso direto às entidades que compõem o sistema e o fato de não precisarem de servidores centralizadores para controlar a rede [100].

Os sistemas **P2P** provém uma infra-estrutura para as comunidades que compartilham ciclos de CPU (como SETI@Home, Entropia) espaço de armazenamento (como Napster, Freenet, Gnutella), ou suporte a ambientes colaborativos (Groove) [100]. Dois fatores têm favorecido a explosão recente de tais sistemas: primeiro é o baixo custo e a alta disponibilidade de um grande número de recursos computacionais e de armazenamento e segundo, a conectividade crescente de rede.

Diferente dos sistemas distribuídos tradicionais, as redes P2P tentam agregar um grande número de computadores que se conectam e desconectam da rede freqüentemente e podem ter endereços de rede (IP) permanentes ou não. Em redes puramente P2P os usuários se comunicam diretamente uns com os outros e compartilham informação sem a necessidade de servidores dedicados. Uma característica interessante destes sistemas é que eles constroem em nível de aplicação, uma rede virtual com mecanismos de roteamento próprios. A topologia desta rede virtual e seus mecanismos de roteamento têm um impacto significativo nas propriedades da aplicação, como desempenho, confiança, escalabilidade e, em alguns casos o anonimato do usuário. A natureza descentralizada dos sistemas puramente P2P faz com que as entidades da rede sejam independentes e auto-organizadas.

P2P e *grid* se assemelham e parecem ter um mesmo objetivo final - a agregação e utilização coordenada de grandes conjuntos de recursos distribuídos - mas as

comunidades de pesquisa são distintas e o foco de cada uma é em muitos casos diferente.

Segundo [51] *grid* e **P2P** compartilham de algumas características em comum:

- Ambas as tecnologias se preocupam com o problema geral que é a organização do compartilhamento de recursos dentro de comunidades virtuais;
- As duas tecnologias tentam resolver este problema através da criação de estruturas adicionais que façam uso das estruturas organizacionais existentes;
- Cada tecnologia tem seus méritos e suas deficiências. Um exemplo é o fato de que o *grid* se preocupa mais com a questão de infra-estrutura e não com as falhas, enquanto que **P2P** está mais preocupada com as falhas do que com a estrutura;

Também são destacados alguns pontos que diferem as duas tecnologias:

- **P2P** está menos preocupada com as questões de confiança sendo que usuários de redes **P2P** podem, em alguns *softwares*, permanecerem no anonimato;
- O *grid* está mais preocupado com os aspectos de qualidade de serviço, enquanto que as redes **P2P** nem sempre levam este fator em consideração;
- Um *grid* está mais preocupado em fornecer uma série de serviços sofisticados, enquanto que as redes **P2P** geralmente têm limitações quanto ao tipo de recurso que é compartilhado.
- Geralmente o *grid* conecta recursos que são mais sofisticados e poderosos, mais diversos e melhor acoplados que os recursos tipicamente interligados pelas redes **P2P**. Um recurso no *grid* pode ser, por exemplo, um *cluster* de computadores, um sistema de armazenamento, uma base de dados ou instrumentos científicos. Em contraste, as redes **P2P** geralmente integram recursos mais simples e o tempo de conexão destes à rede é baixo.
- As aplicações que podem fazer uso de um *grid* podem variar em escopo e tipo, além do tipo de recursos que requerem, enquanto que as aplicações **P2P** se concentram em problemas de compartilhamento bastante específicos, onde a variação se dá apenas na arquitetura e escala.
- O *grid* a princípio não se preocupa com as características de escala, mesmo sendo que a maioria dos projetos envolve um número grande. Mesmo assim este número não é tão grande quanto nas aplicações **P2P**. Outrossim, não tem

a filosofia de auto-gerenciamento e auto-organização que as redes **P2P** possuem.

- O *grid* se preocupa com a criação de uma infra-estrutura de múltiplo propósito, enquanto que **P2P** tende a focar na integração de recursos mais simples (computadores individuais) via um conjunto de protocolos desenvolvidos para prover funcionalidades para esta integração.

3.6 Organizações Virtuais

O problema real e específico que orienta o conceito de *grid* segundo Foster e outros [56], é “*o compartilhamento coordenado de recursos e a resolução de problemas em organizações virtuais dinâmicas e multi-institucionais*”. Por compartilhamento, neste caso, não se entende apenas a troca de arquivos, mas o acesso direto a computadores, *softwares*, dados e outros recursos, conforme a necessidade e estratégias dos solucionadores de problemas e *brokers* de empresas, indústrias e instituições. O conceito de organizações virtuais (OV) foi apresentado por Foster e outros [56] no artigo “*The Anatomy of the Grid*” e o *grid* é redefinido e apresentado como uma forma de viabilizar o estabelecimento destas organizações virtuais.

Casanova [23], comentando o artigo de Foster, identifica algumas propriedades de uma organização virtual:

- O compartilhamento "coordenado" significa que provedores e consumidores dos recursos devem ser claramente definidos. Os recursos compartilhados podem ser organizados de uma maneira integrada para alcançar vários níveis de qualidade de serviço.
- A habilidade de negociar acordos de compartilhamento de recursos de maneira dinâmica e flexível possibilita uma grande variedade de metodologias de resolução de problemas, indo desde engenharia colaborativa até mineração de dados distribuída.
- A sociedade em uma OV é dinâmica, de forma que os participantes podem entrar e sair em qualquer momento. Uma organização virtual é então um conjunto de indivíduos e instituições definidos pelas mesmas regras de compartilhamento.

- Muitos exemplos de organizações virtuais existentes mostram que elas podem variar em propósito, tamanho, estrutura e duração.

Uma observação feita é de que as tecnologias atuais não são o suficiente para se construir tais organizações virtuais. As tecnologias não dão suporte a grande variedade de serviços e recursos, ou elas apresentam uma falta de flexibilidade e controle, os quais são necessários para possibilitar o tipo de compartilhamento de recursos necessário nas OV's. É necessário definir um conjunto de tecnologias para dar suporte às organizações virtuais, ou seja, criar uma infra-estrutura ou arquitetura para o *grid*. Algumas destas características e arquiteturas são descritas em uma seção específica deste trabalho.

Os seguintes são exemplos de OV's: servidores de aplicações, de armazenamento, de ciclos de processamento e consultores engajados na avaliação de um cenário durante o desenvolvimento de um novo veículo; membros de um consórcio de indústrias ligados no desenvolvimento de uma nova aeronave; etc. Cada um destes cenários demonstra uma abordagem para solução de problemas, baseada em colaboração, ambientes computacionais e acesso intensivo a dados.

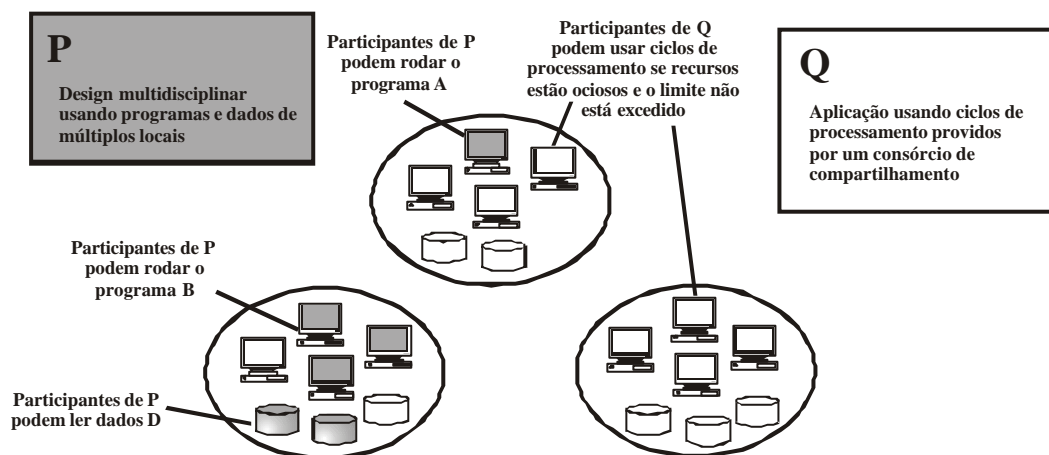


Figura 13 - Um exemplo de organização virtual [56].

Um exemplo de organização virtual é apresentado na Figura 13. Uma organização, representada na figura pelas formas ovais, pode participar de uma ou mais organizações virtuais, compartilhando alguns ou vários de seus recursos. Neste exemplo existem três organizações e duas organizações virtuais. A OV **P** envolve participantes engajados no desenvolvimento de uma aeronave. A OV **Q** envolve participantes que estão compartilhando ciclos de processamento. A organização à esquerda participa da

OV **P**, a da direita participa de **Q**, e a do centro participa de ambas organizações virtuais.

3.7 Protocolos e Arquiteturas

Observando as várias definições de *grid*, algumas o vendo como um grande sistema paralelo e distribuído, fica claro que um sistema assim, a exemplo de outros sistemas como *clusters*, precisa de uma infra-estrutura que ofereça serviços e trate aspectos de: segurança; autonomia; heterogeneidade nas interfaces de acesso a recursos, de políticas, capacidades dos recursos e custo de acesso; localização de dados; a variação na disponibilidade de recursos e a complexidade na criação de aplicações. As arquiteturas que tentam estabelecer um conjunto de protocolos a serem utilizados no *grid* apresentam-no como uma arquitetura em camadas [27].

Foster e outros [56] apresentam o conceito de organizações virtuais e comentam que o estabelecimento, gerenciamento e exploração de relacionamentos dinâmicos entre os membros de tais organizações virtuais que englobam mais de uma organização requerem novas tecnologias. Para isso definem uma arquitetura em termos de componentes necessários, o propósito, objetivos destes componentes e como eles interagem uns com os outros. A interoperabilidade é um ponto central a ser resolvido. E em um ambiente de rede, interoperabilidade significa protocolos comuns.

No artigo “The Anatomy” [56], Foster e outros definem estas propriedades essenciais que um *grid* deve ter e apresentam um conjunto de protocolos, separando-os em camadas e justificando cada uma delas. Os protocolos são importantes para se atingir a interoperabilidade porque eles especificam como os elementos de sistemas distribuídos interagem uns com os outros de maneira a atingir um determinado comportamento, e a estrutura de informação trocada durante estas interações. A arquitetura de protocolos do *grid* pode ser vista na Figura 14.

Nesta arquitetura, a camada **Ambiente** do *grid* provê recursos para os quais o acesso compartilhado é mediado pelos demais protocolos do *grid*: como por exemplo, recursos computacionais, sistemas de armazenamento, catálogos, recursos de rede e sensores. Um recurso pode ser uma entidade física ou lógica, como um sistema distribuído, um *cluster* de computadores ou um grupo de computadores distribuídos. Os componentes da camada **Ambiente** implementam operações locais e específicas dos

recursos como um resultado das operações de compartilhamento nas camadas superiores.

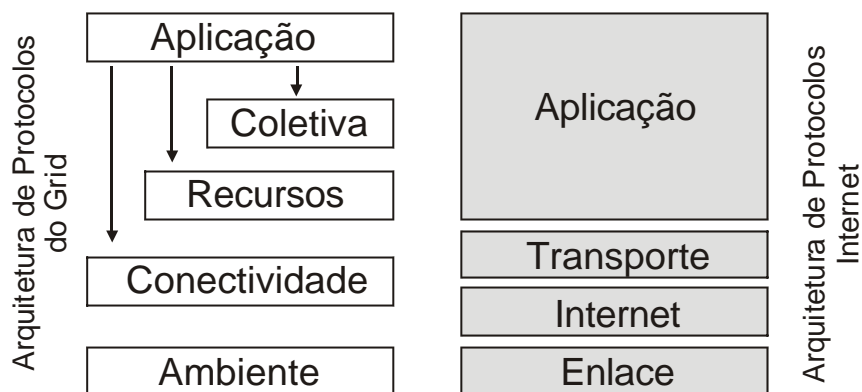


Figura 14 - Arquitetura *grid* e comparação com protocolos Internet [56].

A camada de conectividade define protocolos vitais de comunicação e autorização. Protocolos de comunicação permitem a troca de dados entre os recursos da camada **Ambiente**. Protocolos de autenticação construídos sobre os serviços de comunicação provém mecanismos de criptografia e segurança para verificar a identidade dos usuários e recursos.

A camada de conectividade fornece os protocolos de comunicação e autenticação requeridos pelas transações de redes específicas do *grid*. Os protocolos de comunicação permitem a troca de dados entre os recursos da camada **Ambiente**. Os requisitos de comunicação incluem transporte, roteamento e nomeação.

A camada de recursos é construída sobre a camada dos protocolos de comunicação e autorização para definir protocolos (**APIs** e **SDKs**) para a negociação segura, iniciação, monitoramento, controle, contabilização e pagamento de operações de compartilhamento sobre recursos individuais. Implementações da camada de recursos chamam funções da camada **Ambiente** para acessar e controlar recursos locais. Protocolos da camada de recursos estão preocupados inteiramente com recursos individuais e ignoram as características do estado global e ações atômicas entre coleções de recursos distribuídos.

Nesta camada podem ser identificadas duas classes distintas de protocolos:

- Protocolos de informação, que são usados para obter informações sobre a estrutura e estado dos recursos, como configuração, carga, políticas de uso, etc.
- Protocolos de gerenciamento, os quais são usados para negociar o acesso a recursos, especificando, por exemplo, requisitos de recursos e operações a serem executadas, como processo de criação e acesso a dados.

Enquanto a camada de recursos está focada nas interações com recursos simples, a próxima camada na arquitetura contém protocolos e serviços que não estão associados a qualquer recurso específico e são de natureza global e capturam interações entre coleções de recursos.

Esta camada implementa uma série de comportamentos de compartilhamento sem necessitar de novos requisitos nos recursos sendo compartilhados. Por exemplo:

- Serviços de diretório.
- Co-alocação, escalonamento e serviços de *broker*.
- Serviços de monitoração e diagnósticos.
- Sistemas de programação para o *grid*.
- Serviços de replicação de dados.
- Serviços de descoberta de software.
- Servidores de autorização.
- Sistemas de gerenciamento de carga de trabalho e *frameworks* de colaboração.
- Acesso aos recursos.
- Serviços de pagamento e contabilização.
- Serviços de colaboração.

Por fim, a última camada da arquitetura é a camada de aplicação e compreende as aplicações do usuário que operam no ambiente da organização virtual. Elas podem ser desenvolvidas chamando serviços definidos pelas outras camadas.

Uma evolução desta arquitetura, descrita por Foster e outros [56], pode ser encontrada em [55] descreve uma fusão da arquitetura com os serviços *Web*. A **OGSA** define um conjunto de interfaces que os serviços *grid* devem ter e para isso se apóia na tecnologia de *Web Services*. Atualmente o desenvolvimento da OGSA faz parte de um grupo de trabalho do *Global Grid Forum* (GGF).



Figura 15 - Arquitetura *grid* descrita por Berman e outros [16].

Contudo, apesar das tentativas de padronização propostas pelo **GGF**, não existe uma arquitetura totalmente aceita. À medida que novas tecnologias vão surgindo, uma arquitetura sofre alterações. Berman e outros [16] apresentam uma modificação da arquitetura descrita por Foster, classificando vários serviços em camadas horizontais e verticais englobando outros aspectos importantes como computação móvel. Uma visão abstrata desta arquitetura é apresentada na Figura 15.

3.8 *Grids* Orientados por Serviços

Uma grande tendência é a de que os *grids* estão migrando de um cenário de onde as aplicações são fortemente acopladas para outro, onde estas são fracamente acopladas. Neste cenário, o modelo de composição proporcionado pelos ambientes orientados por serviços apresenta algumas vantagens perante as abordagens tradicionais. Uma visão que se tem de um *grid* deste tipo, é a idéia é de que tais *grids* poderiam proporcionar serviços com fins científicos e comerciais para qualquer usuário, a qualquer momento e em qualquer lugar, usando qualquer dispositivo. A estruturação dos *grids* neste cenário de orientação por serviços fornece funcionalidades importantes para que seja possível a formação das organizações virtuais dinâmicas apresentadas por Foster [56].

Um ambiente orientado por serviços apresenta uma situação onde entidades fornecem serviços que são utilizados por outras entidades mediante alguma forma de contrato, que pode ser um SLA (*Service Level Agreement*). A visão de orientação por serviços pode ser aplicada em qualquer uma das camadas do *grid*, seja no nível computacional, informação ou conhecimento.

Arquiteturas como a **OGSA** têm favorecido a aceitação deste modelo. Na **OGSA**, os serviços *grid* são estruturados em termos de um conjunto de interfaces que os serviços devem possuir e a implementação destes está relacionada com os Web Services [26]. Além de proporcionar uma forma neutra com relação à implementação do serviço, estas arquiteturas evidenciam a adoção de um cenário baseado em serviços. Esta visão de orientação por serviços tem sido bastante adotada por projetos na área de *e-Science*, onde os serviços do *grid* são oferecidos por pesquisadores que os disponibilizam para serem utilizados por outros pesquisadores e assim por diante.

A justificativa para uma adoção de um cenário baseado em serviços está no fato de que ele fornece o modelo necessário para as aplicações em larga escala, fornecendo o nível de abstração necessário, como os componentes devem ser especificados e como eles se relacionam. Sem um modelo assim, as tecnologias de *grid* tendem a ser um conjunto de iniciativas isoladas que apresentam soluções para problemas específicos. É crescente a forma de ver sistemas de larga escala em termos de serviços que eles oferecem. Neste caso, um serviço pode ser caracterizado como uma visão abstrata de uma funcionalidade, provimento de algum conteúdo ou alguma capacidade de processamento.

Um serviço possui sempre um proprietário, ou seja, é oferecido por um indivíduo ou instituição, e está hospedado em algum lugar. O proprietário oferece um serviço para ser consumido por alguém. As políticas para disponibilizar os serviços pode ser diferentes para cada proprietário. Alguns podem disponibilizar seus serviços livremente na Internet, e podem ser utilizados sem nenhum custo, enquanto outros podem requerer algum valor ou pagamento para que seus serviços sejam utilizados.

Os consumidores dos serviços são entidades que invocam a execução daqueles e os critérios para utilização ou se eles alocam estes serviços para oferecê-los a outros consumidores fica a critério da estratégia adotada pelos consumidores.

De acordo com De Roure e outros [32] os componentes chaves de uma arquitetura orientada por serviços são apresentados na Figura 16: proprietários dos serviços (envolvidos pelos retângulos) oferecem serviços (círculos com preenchimento) para consumidores (triângulos com preenchimento) sob certos contratos (linhas sólidas entre produtores e consumidores). Cada interação com o dono do recurso ocorre em um ambiente baseado em mercado (denotado pelas formas ovais) nas quais as regras são

estabelecidas pelo dono do comércio (forma em cruz). O dono do mercado pode ser uma das entidades no ambiente de mercado (um produtor ou consumidor) ou pode ser uma terceira parte neutra.

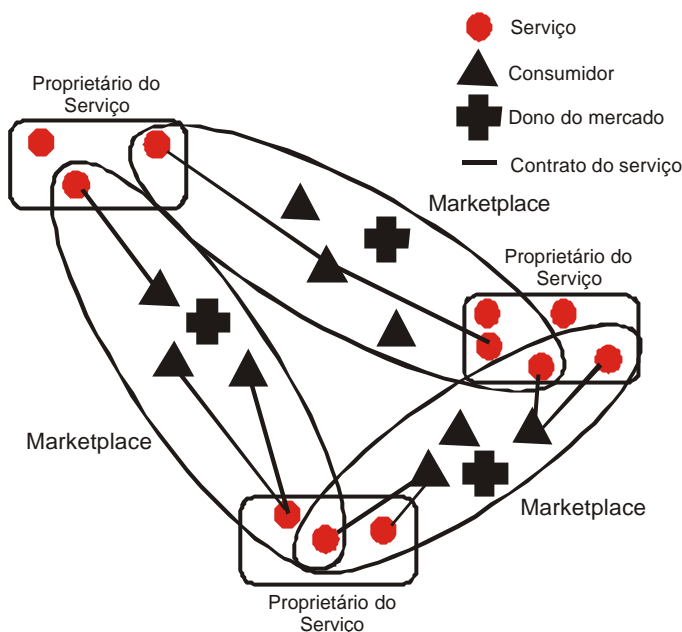


Figura 16 - Infra-estrutura orientada por serviços para *e-Science* [32].

Esta visão de negociação de serviços adota uma visão de comércio (*marketplace*). Nem sempre esta visão precisa ser seguida. Existem casos onde a entrada de entidades pode ser controlada e restrita a determinados grupos ou indivíduos.

3.9 *Middleware* para a Computação em *Grid*

O *middleware* é a camada de software colocada entre o sistema operacional e as aplicações, proporcionando uma série de serviços requeridos pelas aplicações como um conjunto de **APIs**, protocolos e *softwares*, permitindo o desenvolvimento de um sistema distribuído, como por exemplo, um *grid*. O *middleware* tem sido largamente usado em ambientes de computação distribuída e, em um ambiente de *grid*, ele pode ser usado para ocultar da aplicação, a heterogeneidade de recursos existentes no ambiente, proporcionando um acesso uniforme e seguro aos recursos. Ele compõe a camada que está em um nível mais baixo que as aplicações dos usuários e acima dos serviços e meios de transporte proporcionados pela infra-estrutura de rede.

Existe uma série de *middlewares* disponíveis para serem usados na construção de *grids*, dos quais analisaremos alguns dos softwares mais conhecidos.

3.9.1 O Globus Toolkit

O projeto Globus oferece o *middleware* mais utilizado pelos projetos de computação em *grid* [54]. O Globus proporciona um pacote de ferramentas que facilita a construção e uso de *grids*, possibilitando um agrupamento fácil de pessoas, computadores, bases de dados e instrumentos. Possibilita que cientistas possam tratar de grandes conjuntos de dados e colaborar uns com os outros remotamente. Ele tem sido utilizado para computação distribuída em larga escala, instrumentação remota, transferência remota de dados e computação colaborativa.

O Globus possui uma arquitetura modular e oferece serviços e ferramentas que constituem algumas das camadas da arquitetura descrita por Foster e outros [56]. Na segunda camada da arquitetura, por exemplo, o Globus oferece alguns serviços, como **GRAM**, **MDS** e **GSI**. Estes serviços *grid*, oferecidos pelo Globus, são responsáveis por integrar recursos oferecidos pela primeira camada da arquitetura (**Ambiente**).

O *Globus Resource Allocation Manager*, (**GRAM**), é um serviço básico que proporciona funcionalidades que permitem a realização da submissão remota de tarefas. Ele unifica as máquinas do *grid*, provendo uma interface uniforme para o usuário, para que ele possa submeter uma tarefa para múltiplas máquinas da camada **Ambiente** do *grid*. Além disso, o **GRAM** informa o **MDS** sobre o estado dos recursos. A Figura 17 apresenta a arquitetura geral do módulo **GRAM**, onde são apresentados os seus três componentes (*Gatekeeper*, *Job Manager* e *GRAM Reporter*) e os componentes que interagem com o **GRAM**. O cliente **GRAM** é usado para submeter e controlar a execução de tarefas. A manipulação é uniforme, pois não importa qual é o escalonador local do recurso. As requisições enviadas ao **GRAM** são sempre escritas em uma linguagem especial chamada de RSL (*Resource Specification Language*). O *Job Manager* converte a requisição em um formato que o escalonador entenda. O *Gatekeeper* recebe as requisições enviadas pelo cliente, identifica o usuário junto ao **GSI** e suas permissões. Se ele tiver permissão, um *Job Manager* é criado para controlar a tarefa. O **GRAM Reporter** obtém informações do recurso e as repassa ao **MDS** que disponibiliza estas informações para outros usuários do sistema.

O *Monitoring and Discovery Service*, (MDS), também chamado de *Grid Information Service* (GIS) ou *Metacomputing Directory Service*, ou ainda serviço de informação do *grid*, proporciona serviços de informação. O usuário consulta o MDS para descobrir as propriedades das máquinas, computadores e redes que deseja utilizar. O MDS utiliza um servidor LDAP (*Lightweight Directory Access Protocol*) e provê um *middleware* de informações com uma interface comum com o objetivo de unificar as informações dos diferentes tipos de equipamentos do *grid*.

O GSI (*Grid Security Infrastructure*) provê programas para facilitar o *login* do usuário a uma série de *sites*, enquanto que cada *site* tem suas próprias políticas e medidas de segurança. Isso significa que as várias máquinas da camada **Ambiente** podem estar regidas por políticas de segurança diferentes e o GSI provê meios de simplificar estes múltiplos *logins* remotos.

Na terceira camada, os pacotes de ferramentas disponibilizados pelo Globus, permitem a utilização dos serviços *grid* para possibilitarem funcionalidades de mais alto nível oferecidas para as aplicações.

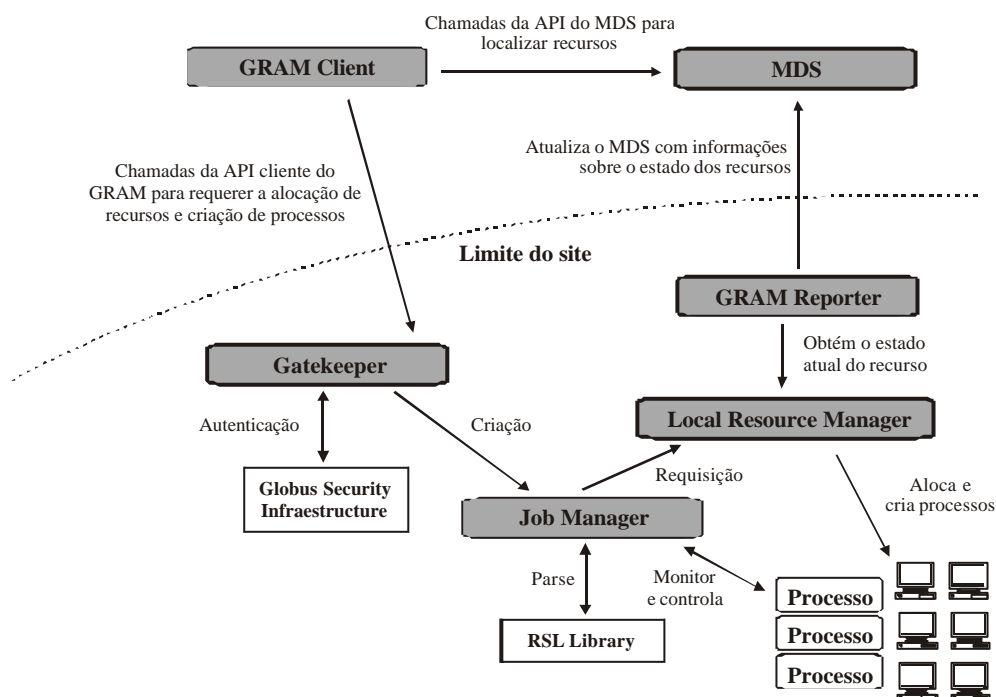


Figura 17 - Arquitetura geral do GRAM do Globus Toolkit.

O projeto possui um grupo de desenvolvimento preocupado com a criação de ferramentas para permitir a distribuição remota de tarefas. Estas ferramentas incluem comandos para a submissão remota de tarefas, construídos sobre o serviço GRAM, e a

MPICH-G2, uma implementação de MPI (*Message Passing Interface*) para *grid*. Outros grupos também se preocupam com o desenvolvimento de pacotes de ferramentas para suporte ao gerenciamento distribuído de grandes conjuntos de dados, visualização colaborativa e instrumentação.

3.9.2 O JINI

O JINI [39] tem favorecido o desenvolvimento de sistemas baseados em agentes. Basicamente, o JINI tenta resolver o problema de administração de uma rede provendo interfaces onde diferentes componentes da rede podem entrar e sair ao mesmo tempo.

Coleções destas entidades JINI são chamadas de federações JINI e estas entidades podem representar consumidores ou fornecedores de serviços. O conceito introduzido pelo JINI é interessante e um serviço pode ser qualquer coisa, um dispositivo de armazenamento, processamento, um software, etc. Um serviço contém uma série de atributos que são usados para descrevê-lo quando ele é adicionado ao sistema. Estes atributos são usados para que usuários ou aplicações procurem os serviços de que precisam.

O JINI pode proporcionar o ambiente dinâmico que muitos sistemas de agentes precisam e pode servir como cola para vários sistemas de agentes e sistemas legados, economizando tempo e trabalho.

Os Cinco Conceitos Chave do JINI

O JINI [39] se apóia em cinco conceitos que são implementados como um conjunto de bibliotecas e convenções que são usadas pelo código participante das comunidades JINI:

Descobrimeto – processo de encontrar comunidades existentes na rede e juntar-se a elas. Esta parte é responsável por criar espontaneamente as propriedades das comunidades do sistema.

Lookup – governa como o código, que precisa usar um serviço, encontra um meio de expressar aos participantes da comunidade que ele deseja utilizá-lo. Desempenha o papel de serviço de diretórios dentro de cada comunidade JINI e proporciona facilidades para procura e localização de serviços que são conhecidos dentro de uma comunidade.

Leasing – é um dos conceitos mais importantes do JINI, simplesmente porque ele é usado extensivamente. É a técnica que proporciona a natureza adaptativa e de auto-recuperação do JINI. Ele garante que uma comunidade irá se recuperar, após um determinado tempo, da perda de um serviço chave que fazia parte da comunidade.

Eventos Remotos – é o paradigma que o JINI usa para possibilitar que serviços notifiquem outros serviços sobre mudanças em seu estado. O serviço de *lookup*, por exemplo, pode usar eventos remotos para notificar as partes interessadas quando um conjunto de serviços em uma comunidade teve o seu estado alterado.

Transações – são os mecanismos que permitem que atividades que envolvem a utilização de múltiplos serviços atinjam um estado seguro. Isto garante que todas as atividades serão realizadas em caso de sucesso, ou que nenhuma será realizada em caso de fracasso, com o sistema retornando ao estado anterior à execução da transação. Este mecanismo impede que ocorram falhas parciais na realização de uma tarefa.

Alguns projetos de *grid* têm se apoiado nas funcionalidades proporcionadas pelo JINI para prover a infra-estrutura necessária ao *grid*. Como exemplos de projetos que fazem uso do JINI como infra-estrutura temos o CoABS [28], o JiniGrid [116] e o Jini Metacomputing [70].

3.10 Classificação dos Tipos de Aplicações Grid

Foster e outros [53] identificaram cinco classes de aplicações em *grid* e criaram uma taxonomia para classificá-las. Embora esta classificação seja abrangente, uma aplicação *grid* pode se enquadrar em mais de uma classe.

3.10.1 Supercomputação Distribuída

Nesta classe de aplicação se enquadram aquelas que necessitam da reunião de um grande número de recursos para tratar de problemas que não podem ser resolvidos em sistemas tradicionais. Neste caso, as aplicações podem agregar supercomputadores ou recursos que sejam caros e impossíveis de serem obtidos em um único sistema.

Como exemplos deste tipo de aplicações, temos: grandes simulações que envolvem centenas de milhares de entidades com um padrão de comportamento

complexo; e simulações de processos complexos em áreas como física, biologia, química e meteorologia.

3.10.2 Computação de Alto Desempenho

Neste caso, o *grid* é utilizado com o objetivo de executar o maior número possível de tarefas independentes e fracamente acopladas, com a intenção de fazer uso de recursos ociosos. Os resultados podem ser os mesmos proporcionados pela supercomputação distribuída, porém o foco destas aplicações consiste em maximizar o uso dos recursos que não estejam sendo usados.

Como exemplo deste tipo de aplicações temos a análise de sinais de rádio em busca de vida extraterrestre inteligente [106], o projeto Distributed.Net [35] e o superprocessador virtual proporcionado pela United Devices [118].

3.10.3 Computação sob Demanda

A computação sob demanda tem o objetivo de satisfazer necessidades de curta duração das aplicações. As aplicações fazem uso de recursos que por questões de custos não podem ser mantidos no ambiente local de execução da aplicação. Ao contrário da supercomputação distribuída e da computação de alto desempenho, a computação sob demanda é orientada por fatores financeiros como o custo do desempenho desejado.

3.10.4 Computação com Intensivo Uso de Dados

Nas aplicações com acesso intensivo a dados o foco está no acesso e extração de informações oriundas de dados mantidos em diversos repositórios geograficamente distribuídos, bibliotecas digitais e bases de dados. Esta extração de informação e síntese de dados consome um grande número de recursos computacionais e de comunicação.

Como exemplos desta classe de informações, temos: experimentos de física e previsões meteorológicas que são capazes de produzir *terabytes* de informações em um único dia.

3.10.5 Computação Colaborativa

Aplicações colaborativas se preocupam em possibilitar e melhorar as interações entre humanos e destes com recursos computacionais. Muitas aplicações colaborativas buscam a melhora no uso compartilhado de recursos computacionais, como arquivos de dados e simulações. Estas aplicações também podem ter características das outras classes de aplicações.

3.11 Conclusões

A computação em *grid* é o resultado da evolução de uma série de tecnologias. Através do conteúdo apresentado neste capítulo, percebemos que ela é o resultado de mais de uma década de pesquisas na área de computação paralela e de alto desempenho e de várias décadas de estudos e melhoramentos das tecnologias de computação e de rede.

Os *grids* surgiram como a possibilidade de plataforma para a execução de aplicações de grande desafio em meados da década de 90. Este foi o fator chave para o seu progresso. Hoje encontramos vários projetos com os mais variados fins e usando das mais diferentes tecnologias de software para a criação de *grids*, inclusive o uso de agentes.

Percebemos uma inclinação do *grid* para um cenário de orientação por serviços que favorece o estabelecimento das organizações virtuais, tidas como o ponto chave das tecnologias de *grid* atuais.

Este capítulo apresentou os conceitos relacionados a área de computação em *grid*, apresentando as raízes de sua origem, arquiteturas, tipos de aplicações e fazendo algumas comparações com os modelos de computação baseadas em *cluster* e P2P.

4 *Grids* de Agentes

Este capítulo apresenta alguns conceitos sobre *grids* de agentes. Porque usar a computação baseada em agentes e quais os motivos que levam a crer que os agentes são peças importantes na construção de um ambiente de *grid*. O capítulo inicia com algumas definições sobre *grids* de agentes, tentando estabelecer alguns conceitos que ajudarão a identificar o que eles são. Embora não se tenha uma definição clara, procurou-se montar uma definição própria que possa orientar o trabalho baseando-se em algumas das possíveis definições de *grids* de agentes. Em seguida é apresentada a arquitetura de *grid* sugerida para a gerência de redes e sistemas e que será detalhada em uma seção específica.

4.1 O Que é o *Grid* de Agentes

Esta pergunta foi feita durante todo o desenvolvimento do trabalho e em muitos momentos foi difícil encontrar uma definição clara e uma forma de diferenciar o “*grid* de agentes” de sistemas “multi-agentes” tradicionais. Para esclarecer um pouco o que seria um *grid* de agentes vamos analisar algumas definições apresentadas por Manola e Thompson [82]. Segundo eles um *grid* de agentes pode ser visto sob três aspectos:

- Uma coleção de mecanismos baseados em agentes - como *brokers*, linguagens de comunicação, regras, planos e outros - que aprimoram as tecnologias de objetos ou outras tecnologias existentes para dar aos projetistas a habilidade de modelar e dinamicamente compor sistemas. Muitos destes mecanismos podem ser independentemente estudados e podem ser padronizados para que comunidades que usarem estes mecanismos possam garantir a interoperabilidade entre sistemas de maneira mais fácil.
- Um *framework* para conectar agentes e sistemas de agentes. O *framework* poderia ser construído usando mecanismos padrões, bem como tecnologias existentes não baseadas em agentes (como um *middleware*). O *grid* como um *framework* é responsável por prover serviços e alocar recursos entre seus membros. Nesta visão, o *grid* de agentes é visto como um *middleware* que conecta sistemas de agentes, sistemas baseados em objetos e sistemas legados.

- Uma forma de modelar entidades que poderiam ser usadas para representar entidades em um organograma, um time ou outro conjunto. Esta visão assume que existem muitos *grids*, mas eles interoperam de acordo com os protocolos padrões. Nesta visão, um *grid* poderia representar a união de federações de agentes.

Outra maneira de visualizar o *grid* de agentes é considerar que o mesmo é uma junção dos três pontos apresentados acima. Estas definições foram criadas no início do desenvolvimento do projeto **CoABS** [28],[81] e foram posteriormente refinadas. O *Grid CoABS* dá ênfase ao segundo ponto, apresentando um *middleware* para conexão de sistemas de agentes e outros sistemas baseados em objetos, que são desenvolvidos pelos pesquisadores que integram o **CoABS**. O **CoABS** é um programa de pesquisa do **DARPA** - *Defense Advanced Research Projects Agency* e do *US Air Force Rome Labs*. Alguns experimentos com relação à escalabilidade do sistema de registro e descoberta de agentes no **CoABS** podem ser encontrados em [71]. O objetivo chave que orienta o **CoABS** e que pode explicar as definições acima é que ele é uma forma de facilitar a composição de sistemas e a formação de coalizões em ambientes de batalha. Neste cenário, onde podem existir diversos sistemas legados existentes, uma abordagem baseada em agentes e mecanismos que proporcionem a integração destes agentes pode representar um ganho considerável de tempo e dinheiro.

Embora estas definições sejam usadas para caracterizar um *grid* de agentes, elas não deixam claro quais as diferenças em relação a sistemas de agentes tradicionais. O *grid* de agentes não deixa de ser um sistema de agentes, ou um sistema multi-agentes. A computação baseada em agentes é uma forma de possibilitar o uso compartilhado de recursos requerido em um *grid*. Ou seja, o *grid* é também uma coesão de vários conceitos relacionados a agentes, pois eles são um meio de se compor sistemas mais facilmente [69] e de possibilitar a interoperabilidade e integração necessárias em um ambiente de *grid* [126].

Os agentes podem ser vistos como entidades que representam recursos e ou outros sistemas da primeira camada, **Ambiente**, descrita por Foster e outros [56] em sua arquitetura. Estes agentes oferecem serviços que podem ser usados por outros agentes, constituindo a segunda camada. Eles podem ser detentores de recursos da segunda camada, conhecendo como acessar e escalonar tarefas localmente a estes recursos. Os

agentes podem descrever suas necessidades, procurar por outros agentes que proporcionam os serviços necessários e fazer uso destes serviços, proporcionando o seu uso coletivo. Os agentes podem usar protocolos padrões, mencionados na arquitetura de Foster como meios de possibilitar o compartilhamento dos recursos em organizações virtuais dinâmicas, e a conformidade com padrões como **FIPA** podem garantir a interoperabilidade entre diferentes plataformas de agentes e a composição de sistemas que transcendem os limites de uma organização.

Em trabalhos anteriores [10] afirmamos que um *grid* de agentes se difere de um sistema multi-agentes pelas suas características de escalabilidade, distribuição de carga, cooperação e gerenciamento de recursos. Isto não deixa de ser uma afirmação verdadeira, uma vez que nem sempre estas funcionalidades estão inclusas em um sistema multi-agentes, e sendo que este está normalmente ligado a um problema específico no qual os agentes envolvidos trabalham para a sua solução. No *grid* de agentes os agentes são usados para prover serviços que são utilizados por outras entidades que também são representadas por agentes, sendo que os agentes são um meio utilizado para implementação deste paradigma orientado por serviços. Porém, não é possível enumerar uma lista de atributos para qualificar ou desqualificar uma arquitetura de agentes como sendo um *grid* ou não.

Para a integração de recursos e de sistemas que um *grid* de agentes deve proporcionar, é necessário que exista um conjunto de mecanismos que envolvem dentre outras coisas, serviços de diretórios, funcionalidades para anúncio e descoberta de serviços, *gateways* entre diferentes plataformas, mecanismos de negociação de recursos, balanceamento de carga, além de serviços de segurança como autenticação e controle de acesso. Estes mecanismos devem garantir a interoperabilidade, o crescimento e aplicação de agentes em ambientes dinâmicos e multi-institucionais.

No caso do projeto **CoABS** [28] esta infra-estrutura é um *middleware* desenvolvido sobre o Java JINI [39]. O **CoABS** usa funcionalidades como o conceito de aluguel (*leasing*), transações, eventos, além do serviço de diretórios fornecidos pelo JINI. Outros projetos se apóiam na conformidade com padrões existentes [124], como as especificações **FIPA**, e deixam em aberto detalhes de implementação como linguagem ou ferramentas utilizadas.

As funcionalidades apresentadas anteriormente no Capítulo 2, que descreve as tecnologias de agentes e sistemas multi-agentes, fazem dos agentes uma boa abstração de entidades do *grid*. A visão dos agentes como uma evolução dos objetos componentes de software, juntamente com uma poderosa linguagem de comunicação (ACL), fazem deles uma boa alternativa para modelar e implementar sistemas complexos como é o caso de um *grid*. Geralmente as interfaces entre sistemas diferentes são difíceis de serem construídas e geralmente elas devem ser conectadas em tempo de desenvolvimento [29]. As aplicações atuais estão migrando para um ambiente mais orientado por serviços, onde as entidades e aplicações são fracamente acopladas e, portanto as interfaces entre sistemas heterogêneos precisam ser alteradas durante a sua execução. Com sua linguagem de comunicação e em conjunto com ontologias podem proporcionar mecanismos muito mais flexíveis para possibilitar esta integração.

Os agentes, por serem entidades capazes de descrever as suas necessidades, por poderem procurar, colaborar e negociar com outros agentes aptos a prover os serviços que eles precisam para a realização de suas tarefas podem ser vistos como entidades do *grid*. Percebemos que este cenário se assemelha aos ambientes orientados por serviços, onde uma entidade ou aplicação faz uso de serviços oferecidos por outras entidades ou organizações. Os agentes podem adicionar uma funcionalidade a mais, através dos mecanismos da inteligência artificial distribuída, além de poderem se adaptar ao ambiente.

4.2 Serviços Necessários em um *Grid* de Agentes

Para a construção de um *grid* de agentes existe a necessidade de um conjunto de serviços básicos que constituem a infra-estrutura para possibilitar a integração de agentes e sistemas necessária no *grid*. A exemplo de outras infra-estruturas de *grid*, se faz necessário, dentre outras coisas, um sistema de informação que possibilite que os agentes localizem outros agentes e os serviços proporcionados por estes. Alguns serviços necessários ao *grid* de agentes são:

Serviço de informação: consiste em um serviço de diretório onde são registradas informações sobre os agentes, e recursos existentes no *grid*, assim como suas habilidades e serviços oferecidos. Este diretório deve prover serviços de páginas brancas e amarelas aos agentes registrados. O serviço de informação compreende a parte mais

importante da infra-estrutura, possibilitando a procura e monitoramento dos agentes. Através deste mecanismo de diretórios os agentes podem se registrar e disponibilizar seus próprios serviços e habilidades, além de poderem procurar outros agentes que sejam capazes de executar tarefas que eles próprios não podem, devido a sua limitação de recursos ou por questões estratégicas. O diretório deve ser transparente e escalável o suficiente para que um grande número de agentes, talvez na ordem de milhares, possa se registrar ao *grid*. A **FIPA** em sua arquitetura abstrata [43] apresenta como um mecanismo de diretórios deve funcionar nas arquiteturas que desejam ser compatíveis com o padrão **FIPA**. Esta arquitetura abstrata é refinada na especificação que apresenta o sistema de gerenciamento **FIPA** [45], o qual descreve como uma plataforma de agentes deve ser e como os padrões garantem a integração de diferentes plataformas. O modelo de plataforma de agentes da **FIPA** define que devem existir um facilitador de diretórios e um gerenciador em cada plataforma. A interligação entre diferentes sistemas deve acontecer através da interligação de seus facilitadores.

Além do serviço de diretório de agentes e serviços, pode ser mantido um serviço de informações sobre os recursos existentes no *grid*. Este serviço pode estar separado do diretório de agentes e pode se apoiar em soluções já existentes e bem definidas, como o MDS-2 do **Globus Toolkit** [78]. O gerenciamento dos recursos disponíveis no *grid* de agentes é importante para que o escalonamento seja realizado de tal forma que as tarefas possam ser mapeadas para os recursos adequados. Neste caso os agentes podem ter um módulo responsável por recuperar informações de um serviço de informação de recursos como o MDS.

Serviço de nomeação: o *grid* deve adotar algum mecanismo de nomeação dos agentes, com o objetivo de evitar que dois agentes pertencentes ao *grid* tenham um mesmo nome. O *grid* **CoABS** usa os identificadores de serviços JINI para atribuir identificadores únicos aos agentes registrados no *grid* de agentes.

Serviços de visualização: os serviços de visualização são ferramentas que, através de interfaces gráficas e relatórios permitem verificar o que está acontecendo no *grid*. As informações que as ferramentas de visualização podem apresentar dizem respeito as interações que acontecem entre os agentes do *grid*, quais os agentes que estão registrados, os serviços e funcionalidades que eles têm a oferecer, assim como os seus interesses que podem ser declarados ao *grid*.

Serviços de segurança: outro fator importante, quando falamos de integração de recursos entre diferentes organizações, é a segurança. Mecanismos de autenticação e confidencialidade devem existir no *grid* para garantir a confiança e seu funcionamento. No *grid* de agentes, além dos critérios referentes a criptografia das mensagens trocadas, são necessários mecanismos de autenticação das mensagens trocadas. A **FIPA** estabelece alguns requisitos mínimos de segurança que podem ser seguidos para se garantir a segurança entre diferentes plataformas de agentes [43]*.

4.3 Benefícios de Tecnologias Baseadas em Agentes Para o *Grid*

O uso de tecnologias baseadas em agentes para a construção de *grids* pode apresentar uma série de aprimoramentos as tecnologias atuais e, além disso, adicionando elementos da inteligência artificial que podem ser úteis em ambientes complexos como um *grid*. Nesta seção são apresentados alguns benefícios e vantagens da utilização de agentes na construção de *grids*.

Os benefícios da utilização de *grids* de agentes estão diretamente relacionados a alguns dos fatores descritos no capítulo que trata de agentes e sistemas multi-agentes. Sob o ponto de vista de engenharia de software os sistemas baseados em agentes podem ser vistos como uma forma de modelagem e desenvolvimento de sistemas complexos. Conforme apresentado no Capítulo 2, os sistemas complexos podem ser divididos em sistemas menores e a interações entre estes sistemas podem ser modeladas como mensagens entre os agentes de software.

A utilização de técnicas de inteligência artificial constitui outra vantagem da utilização de um *grid* de agentes. Os agentes têm sido utilizados e existem vários projetos na área.

4.3.1 Composição de Sistemas

Uma das principais vantagens do uso da tecnologia de agentes, conforme já foi mencionado, é a facilidade proporcionada na composição de sistemas. Os agentes provêm mecanismos que podem facilitar a integração de software e de sistemas legados.

* O “Anexo D” da especificação que define a arquitetura abstrata da FIPA trata de alguns aspectos de segurança desejáveis em implementações da arquitetura abstrata.

Um exemplo disso é dado pela formação de coalizões em ambiente de batalha proporcionada pelos *grid* de agentes CoABS [4]. Os sistemas legados podem ser integrados ganhando um invólucro (chamado *wrapper*) que envolve o software existente e antigo e lhe dá um comportamento de agente. A **FIPA** possui especificações que definem como devem ser construídos tais *wrappers* para que sejam compatíveis com o padrão **FIPA** [47].

Outra característica importante é que os agentes podem ser vistos como evolução dos objetos componentes. Neste caso, os agentes se diferem por poderem encapsular um mecanismo de raciocínio que lhes dá a funcionalidade de poder descrever suas necessidades e procurar outros agentes que possam ajudá-los a cumprir com os seus objetivos. A comunicação e cooperação são fatores importantes neste cenário. Ao contrário da comunicação em sistemas baseados em objetos, os agentes utilizam uma linguagem de comunicação mais poderosa e que em conjunto com ontologias, que descrevem o domínio do diálogo, podem trocar informações com um conteúdo mais semântico. Nem todas as interações entre os agentes precisam ser previstas em tempo de desenvolvimento, ao contrário do que acontece com os sistemas orientados por objetos. Estas características, entre outras, fornecem uma facilidade na construção de interfaces entre diferentes softwares, o que seria complicado usando outras tecnologias tradicionais.

Estes fatores mencionados são importantes em um ambiente de gerenciamento onde softwares e ferramentas existentes precisam ser utilizadas e não podem ser abandonadas porque fornecem funcionalidades vitais à atividade de gerenciamento da rede e sistemas da empresa. Um sistema de gerência existente pode fornecer dados de gerenciamento interessantes e cuja análise pode ser realizada pelos agentes do *grid*. Ainda, o *grid* pode usar de funcionalidades de gerência e análise providas por tais softwares, desde que eles sejam encapsulados sob a forma de agentes de software.

4.3.2 Desenvolvimento de Software

Além das facilidades referentes à composição de sistemas, de integração de sistemas legados e da formação de coalizões, os agentes podem ser vistos como uma evolução no desenvolvimento de sistemas. Conforme apresentado no Capítulo 2, os

agentes fornecem uma abordagem interessante para a modelagem e construção de sistemas complexos. Em ambientes onde não se pode prever todas as interações necessárias entre seus componentes, os agentes apresentam funcionalidades no mínimo interessantes para a solução destes problemas, e se diferenciam dos objetos, em um sistema orientado a objetos ou componentes pelo seu mecanismo de raciocínio. É justamente este mecanismo de raciocínio que pode fazer com que os agentes ajam com interesses próprios e que permite que eles cooperem ou negociem com outros agentes com o objetivo de atingir o equilíbrio do sistema.

A adoção de modelos de organização - como os orientados por mercado - onde os agentes agem com o objetivo de perseguir seus objetivos, abre novas portas no desenvolvimento de *software*. A aplicação destas teorias e modelos no *grid* de agentes e por consequência na gerência de sistemas é interessante. Vários tipos de aplicações de larga escala estão adotando tais modelos e a gerência de grandes redes e sistemas é um tipo especial de aplicação no qual estes modelos podem ser empregados com sucesso.

4.3.3 Orientação por Serviços

A orientação por serviços é uma tendência nos ambientes de *grid* [55]. A arquitetura baseada em serviços para ambientes de agentes é reforçada pela arquitetura abstrata da **FIPA**. O conceito de serviços é adotado porque adiciona uma certa flexibilidade ao ambiente, fazendo com que os agentes expressem as suas habilidades e capacidades em termos de serviços.

Realizar uma busca por um serviço em um serviço de informação, composto por um serviço de diretório, por exemplo, como acontece na arquitetura da **FIPA**, é mais fácil porque os agentes podem procurar por características específicas do serviço, ontologias utilizadas, dentre outras informações. O modelo de orientação por serviços facilita a composição de sistemas [26] e permite a utilização de mecanismos orientados por serviços para a negociação entre agentes.

4.4 Projetos Existentes

No capítulo introdutório foram apresentados alguns trabalhos relacionados envolvendo a utilização de agentes na construção de *grids* e de plataformas de agentes

que procuram atender as novas características de larga escala. Nesta seção são descritos com um número maior de detalhes os projetos mais relevantes relacionados com o tema ‘*grid* de agentes’. Este trabalho tem um certo relacionamento e utiliza conceitos dos trabalhos citados no desenvolvimento da arquitetura proposta.

4.4.1 O *Grid* CoABS

O *grid* CoABS [28], descrito anteriormente, é um *middleware* que integra sistemas heterogêneos baseados em agentes, aplicações baseadas em objetos, e sistemas legados. Inclui uma API com métodos para registrar os agentes, anunciar as suas capacidades, descobrir agentes com base nas suas habilidades, e enviar mensagens entre os agentes do *grid*. O *grid* também provê um serviço de *log*, para registrar o tráfego de mensagens e outras informações; um serviço de segurança que provê autenticação, criptografia e comunicação segura; e notificação de eventos quando os agentes se registram, desregistram, ou alteram os atributos anunciados. A primeira versão do CoABS *Grid* foi liberada em julho de 1999.

O CoABS está sendo usado para produzir uma integração de sistemas nos experimentos militares onde existem sistemas legados e sistemas multi-agentes desenvolvidos por diferentes grupos de pesquisa do CoABS. O CoABS possibilita o registro dinâmico e descoberta de participantes. É adaptável e robusto, e o sistema evolui para satisfazer as exigências variáveis sem necessitar de uma re-configuração do sistema.

O CoABS *Grid* é somente parte do programa CoABS – é o *middleware* que conecta os componentes desenvolvidos por todos os pesquisadores do CoABS para a solução de problemas diversos. O *grid* neste caso pode ser visto não apenas como o *middleware* que conecta os sistemas de agentes, mas também todos os agentes que habitam o ambiente e fazem uso do *middleware* para se conectarem.

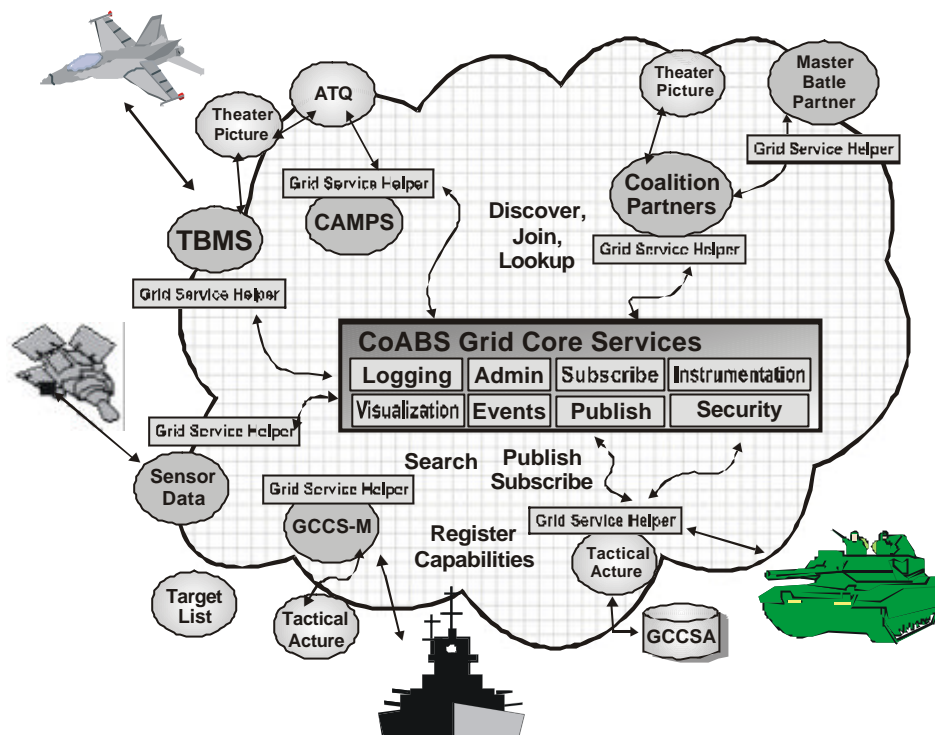


Figura 18 - Visão geral do *grid* CoABS [59].

Em linhas gerais, o *grid* CoABS proporciona as seguintes funcionalidades:

- Um princípio para construir invólucros (*wrappers*) para sistemas legados.
- Uma rede para descoberta distribuída e integração de sistemas em tempo de execução.
- Funções de registro.
- Anúncio e modificação das capacidades dos agentes.
- Funções de busca sofisticadas.
- Descoberta de serviços e de agentes, baseada em modelos e predicados.
- Uma infra-estrutura de comunicação entre agentes por meio de uma interface de entrega de mensagem.
- Notificação de eventos de registro e modificações.
- *Logging* automático de mensagens dos agentes.
- Funções de segurança.
- Criptografia de mensagem e autenticação de serviço.
- Ferramentas de administração e visualização.
- Serviços de publicação e subscrição.

Na Figura 18 [59] temos uma visão geral de como funciona o *grid* **CoABS**. A figura mostra a relação entre as aplicações no ambiente **CoABS**, os componentes chamados de *grid-aware components* (formas ovais), os serviços *grid* representados pelos retângulos e a infra-estrutura de *grid*. As aplicações são representadas pelas formas ovais maiores. Muitas destas podem ser sistemas legados e outras compostas por agentes. Cada aplicação possui um módulo *proxy GridServiceHelper* para interoperar com os outros componentes do *grid* e com os serviços do *grid*.

Os serviços *grid* proporcionam as funcionalidades necessárias para que os componentes do *grid* possam interagir e formar as aplicações dinâmicas. Eles também provêm serviços como instrumentação, teste, visualização e gerenciamento das aplicações. Os componentes *grid-aware* podem acessar os serviços do *grid* através de mecanismos de acessos como protocolos, *wrappers* e outros.

O **CoABS** tira vantagem de alguns componentes JINI para proporcionar a infra-estrutura de *grid* necessária. Uma visão dos elementos básicos da tecnologia JINI são descritos na Figura 19. O **CoABS** utiliza os componentes da seguinte forma:

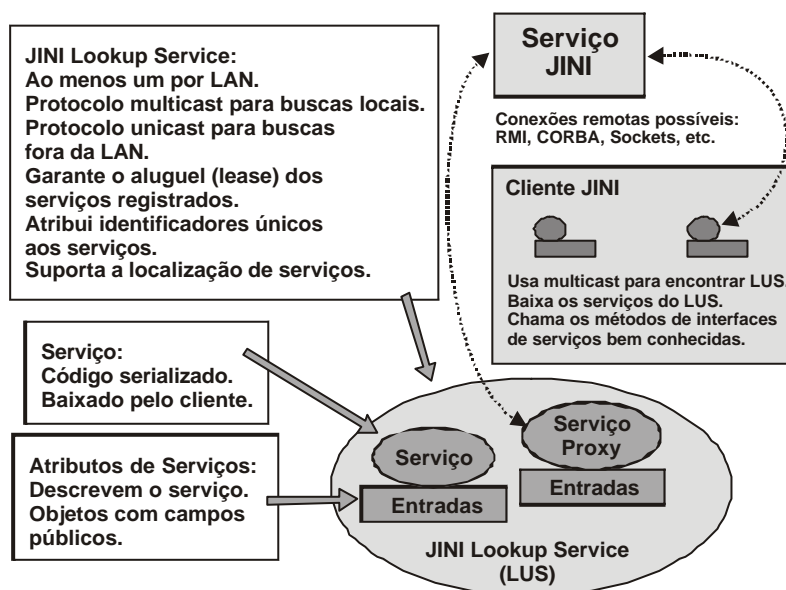


Figura 19 - Visão abstrata da tecnologia JINI [29].

- O conceito de serviço é utilizado para representar um agente.
- O serviço de registro e localização (LUS) é usado para registrar e descobrir agentes e outros serviços.
- As entradas JINI são utilizadas para anunciar as capacidades dos agentes que estão registrados no *grid*.

Maiores detalhes sobre como o **CoABS** utiliza a infra-estrutura JINI como *middleware* para a interligação de sistemas de agentes podem ser obtidas em [29], onde é apresentado um manual de usuário para a operação do *grid* **CoABS**.

4.4.2 O *Grid* Echelon

O Echelon [38] é uma infra-estrutura de *grid* baseada em agentes onde o usuário pode programar problemas complexos, alocar os recursos de que precisa para resolver o problema, e analisar os resultados finais. As definições de problemas, métodos de resolução, e as soluções são persistentemente armazenadas no sistema, podem ser vistos e ser reusados posteriormente, podem ser compartilhados entre os pesquisadores e engenheiros que os buscam. O ambiente Echelon estende a área de trabalho do usuário proporcionando o acesso a recursos computacionais remotos (hardware, software e dados) e escondem do usuário a complexidade, heterogeneidade e distribuição dos recursos.

Os serviços providos pelo *grid* Echelon são:

- Comunicação e mensagens cifradas.
- Submissão remota de trabalhos.
- Transferência remota de arquivos.
- Composição de recursos no *grid*.
- Super escalonamento.
- Visualizações distribuídas.
- Disseminação de trabalhos dinamicamente.
- Novos nós podem entrar e sair do *grid* em qualquer momento.

4.5 Conclusões

Os *grids* de agentes podem ser vistos como uma coesão de uma série de tecnologias. São uma maneira de compor *software* mais facilmente e de possibilitar a conexão dinâmica de sistemas. Estas características fazem dos agentes entidades de extrema importância nos ambientes de *grid* e no estabelecimento das organizações virtuais dinâmicas. O *grid* de agentes pode ser também, uma infra-estrutura de software baseada em agentes construída sobre as funcionalidades providas por um *grid*

computacional. Eles podem prover serviços e funcionalidades para as camadas superiores de informação e conhecimento.

Alguns trabalhos importantes relacionados ao tema *grid* de agentes foram citados e comentados com um nível maior de detalhes. Neste capítulo procurou-se evidenciar o que são os *grids* de agentes, tentando facilitar o entendimento a respeito do que eles podem ser. Também foram citadas algumas vantagens da utilização dos *grids* de agentes.

5 O *Grid* de Agentes na Gerência - Proposta

Este capítulo apresenta a arquitetura de *grid* de agentes proposta. Para isso este capítulo apresenta alguns problemas inerentes ao gerenciamento centralizado de redes e sistemas e em quais casos a utilização de um *grid* de agentes pode ser útil. A arquitetura apresentada tenta oferecer uma forma de lidar com a grande quantidade de informação de gerência que deve ser manipulada e tratada pelas aplicações de gerenciamento atuais. Presenciamos ao longo dos últimos anos um grande crescimento na variedade e número de dispositivos nas redes. Em um cenário onde os serviços oferecidos por infraestruturas de rede são cada vez mais importantes, e em muitos casos vitais à existência da organização, é importante realizar um gerenciamento adequado dos mecanismos que proporcionam estes serviços.

Neste cenário de serviços, a gerência de sistemas é tão importante quanto a gerência da própria rede. Ao detectar a falha de um serviço, um usuário leigo não será capaz de identificar se esta falha aconteceu em virtude de uma falha de rede ou de um sistema que é necessário ao provimento de tal serviço.

Contudo, para o gerenciamento são necessárias a coleta e análise de dados extraídos dos sistemas e de dispositivos da rede. Critérios que dizem respeito ao que coletar e ao que analisar podem ser orientados por um SLA (*Service Level Agreement*). Além do monitoramento de itens que sejam considerados vitais ao provimento de qualquer serviço, o monitoramento pode ser orientado pelo SLA. Neste caso, a análise de informações para verificar se os itens constantes no acordo estão sendo cumpridos e o que coletar para verificar isso são importantes.

Em um grande ambiente o volume de informação manipulada e analisada também é muito grande. Além de poder consumir muito tempo para verificar todos os itens que precisam ser analisados, as tarefas de análise podem consumir muitos recursos computacionais para a atividade de gerenciamento.

Neste capítulo primeiramente descrevemos a arquitetura e seus blocos componentes. Em seguida são apresentados cenários onde ela pode ser aplicada e de que maneira isso pode ser realizado.

5.1 Arquitetura Proposta

Durante o desenvolvimento do trabalho, uma primeira visão de uma possível arquitetura para o *grid* de agentes foi sugerida em [12]. Esta arquitetura está baseada no fluxo tradicional do gerenciamento de redes de computadores apresentado por Koch & Westphall [74]. Levando em consideração este fluxo tradicional de gerenciamento a arquitetura foi dividida em componentes que descrevem cada uma das etapas do gerenciamento: a coleta de dados dos dispositivos gerenciados da rede; o tratamento e armazenamento destes dados coletados; a análise destas informações e; a apresentação ao gerente humano ou a outro sistema de gerência através de interfaces. Uma visão abstrata do sistema de gerência idealizado pode ser visto na Figura 20.

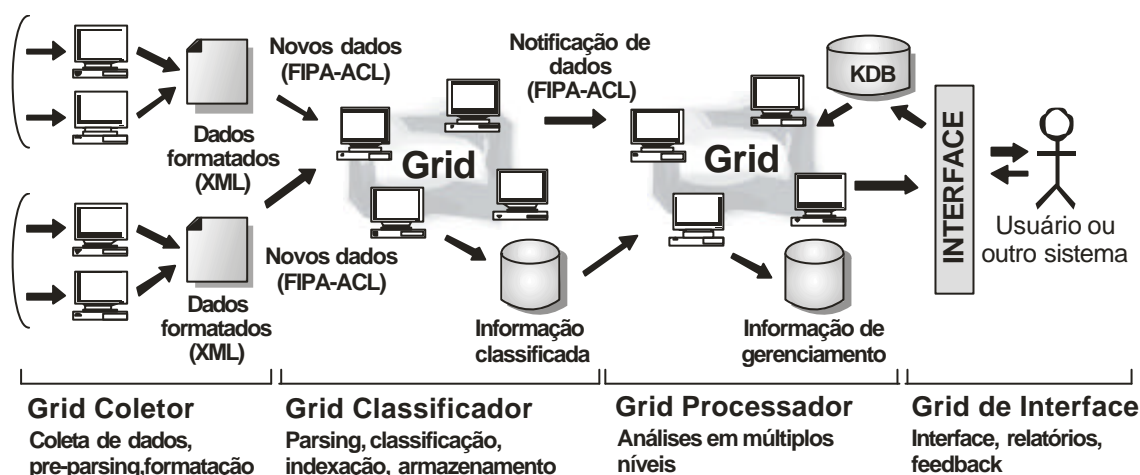


Figura 20- Sistema de gerência usando *grids* de agentes.

Este modelo é composto dos seguintes módulos que também são chamados de *grids* e que serão descritos com maiores detalhes nas sub-seções a seguir: *grids* de agentes coletores; *grids* de agentes classificadores; *grids* de agentes processadores e *grids* de agentes de interface.

5.1.1 Grid de Agentes Coletores

A responsabilidade dos agentes do *grid* coletor é coletar informações dos equipamentos da rede gerenciada usando um protocolo de gerenciamento ou outro meio. Chamaremos de “interface” esta habilidade dos agentes coletores de utilizar um protocolo ou outro. Um agente coletor pode ter uma interface SNMP (*Simple Network*

Management Protocol) ou utilizar um utilitário de linha de comando, por exemplo. No caso do *Agentlight* [3], ferramenta que está sendo utilizada para validar a idéia proposta, esta interface é feita através de uma *Skill* que nos agentes coletores é uma implementação para cada protocolo de gerenciamento de uma interface Java chamada *SkillInterface* que é responsável por fazer a ligação do que é lógico na base de conhecimento do agente com coisas que não são, como a chamada de um método de um objeto Java.

Uma vez que as informações são extraídas usando várias interfaces ou ferramentas, estas podem apresentar formatos bastante heterogêneos e por isso é necessário criar uma representação comum para estes dados. Esta representação é feita usando XML [121], em conjunto com ontologias [62] que garantem a correta interpretação de dados pelos agentes armazenadores. Neste caso, garantimos que as informações coletadas da rede poderão ser interpretadas corretamente pelo *grid* de agentes que irá recebê-las.

Como foi citado, um agente coletor é dotado de uma base de conhecimento com regras que permitem coletar dados usando o protocolo requerido. Estes agentes podem ter um ou mais objetivos que consistem em extrair valores de objetos gerenciados de um ou mais equipamentos da rede em intervalos determinados de tempo. Assim como um agente pode interagir com um ou mais dispositivos da rede, pode ocorrer de vários agentes extraírem dados de um único dispositivo. O *grid* coletor pode conter agentes que executam algumas análises locais destas informações. Os agentes coletores são agentes reativos que podem conter em suas bases de conhecimento regras que fazem eles reagirem a algum evento ou situação específicos. Eles não precisam de um mecanismo de raciocínio muito complexo.

As informações coletadas pelo *grid* coletor são enviadas para um *grid* classificador ou de armazenamento já num formato padrão, através de um protocolo de transporte como SMTP ou HTTP. Os agentes coletores realizam tarefas de *parser* dos dados extraindo apenas o que é relevante antes de enviá-las aos armazenadores de dados. Aspectos específicos da implementação destes agentes são descritos em uma seção específica.

5.1.2 Grid de Agentes Classificadores

Em uma rede com um número grande de elementos gerenciados podemos ter inúmeros coletores de dados. Por consequência teremos dados coletados de vários tipos de equipamentos chegando ao *grid* de agentes armazenadores. O *grid* de classificação é responsável por classificar estas informações e armazená-las de uma forma estruturada e mais fácil de ser recuperada. Um arquivo de dados enviado por um *grid* coletor pode conter valores coletados de vários objetos gerenciados de equipamentos heterogêneos. É interessante organizar estas informações antes de realizar qualquer análise. Desta forma, o *grid* classificador desempenha algumas tarefas de *parsing*, classificação, indexação e armazenamento destes dados.

Na Figura 21 temos um exemplo de um *grid* classificador recebendo dados coletados dos objetos gerenciados de três equipamentos diferentes. Os agentes do *grid* classificador possuem objetivos que consistem em classificar estas informações e armazená-las da forma correta, separando-as de uma forma que facilite a distribuição e análise (*data-clustering*).

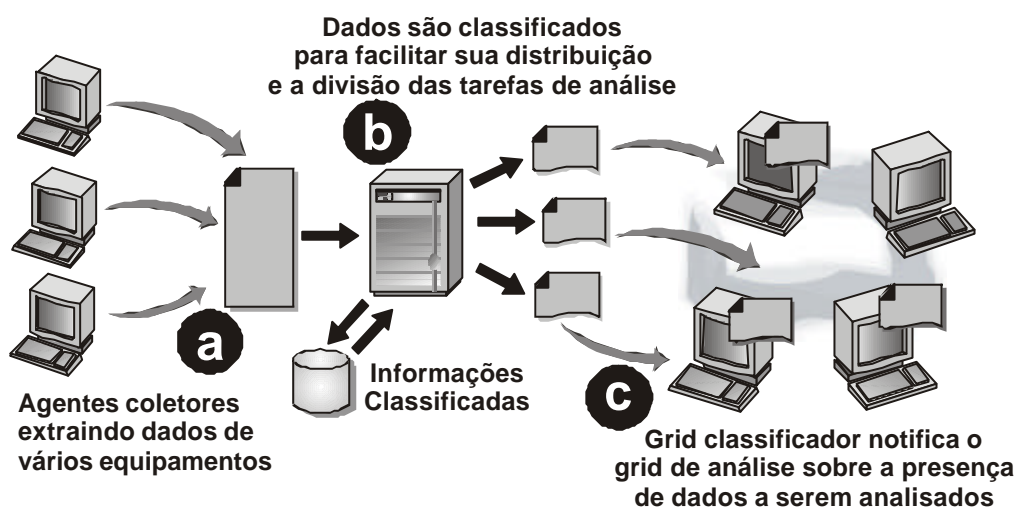


Figura 21 - Classificação realizada pelo *grid* classificador.

Este *grid* é responsável por proporcionar a agregação de recursos de armazenamento para serem utilizados pelas aplicações de gerência. As formas de armazenamento podem variar de bancos de dados relacionais a arquivos com armazenamento de documentos XML que representam informações de gerenciamento ou relatórios que precisam ser recuperados pelos agentes de interface.

5.1.3 *Grid* de Agentes Processadores

O *grid* de processamento ou análise é a parte mais importante da arquitetura, e onde estão localizados os maiores desafios de desenvolvimento. É responsável pela consolidação dos dados coletados em relatórios de gerência. Os principais problemas que podem surgir dizem respeito à divisão das atividades de análise, controle de recursos, balanceamento de carga e tolerância à falhas. O resultado da análise é composto por informações de gerência que darão origem aos relatórios de gerência e de alertas enviados aos agentes de interface.

Este *grid* é composto por *containers* [88] de agentes distribuídos em vários computadores. Temos desta forma um ambiente dinâmico em que novos recursos podem ser adicionados ou removidos do *grid*. Se houver a necessidade de aumentar o poder de processamento deste *grid*, podemos adicionar novos recursos ao mesmo. É apresentado o conceito de *container* de agentes porque a plataforma utilizada para os testes e validação da idéia usa deste conceito. Porém, o *grid* processador poderia ser composto por vários agentes, e não necessariamente *containers*, localizados em vários recursos. Um *container* nada mais é do que uma forma de possibilitar a execução de vários agentes em uma mesma aplicação. Para efeito externo, ele se comporta como se fosse um agente.

O *grid* de processamento pode realizar uma série de análises em múltiplos níveis. Exemplos destas análises e como elas ocorrem são descritas a seguir:

- Ao receber uma mensagem do *grid* classificador, indicando a presença de dados, o *grid* processador faz com que um número de agentes ou *containers*, baseados em suas regras, procurem por problemas nestes dados. Esta primeira análise não leva em consideração informações armazenadas anteriormente na base de dados e não procura por nenhuma relação entre os fatos. Neste caso percebemos que a presença de dados para análise faz com que o *grid* armazenador avise o analisador de que os dados existem. Em um cenário real os agentes analisadores podem ter como objetivos analisar dados e solicitar as informações de que precisam ao *grid* classificador, e este se responsabiliza por encontrar no *grid* a melhor cópia destes dados e a mais rápida para ser recuperada.

- Os agentes deste *grid* podem consolidar os dados, extraindo informações armazenadas anteriormente. Com isso podem procurar por problemas e agrupar dados, constituindo algumas informações que servirão para a construção de relatórios de gerência.
- No terceiro nível de análise, os agentes do *grid* podem procurar por relações entre os fatos na base de dados. Tendo uma visão de mais alto nível das informações. Podem desta forma, identificar problemas que são possíveis de serem identificados através do cruzamento de informações de um conjunto de equipamentos e não apenas de dados isolados.

Dentro deste contexto, a principal dificuldade é proporcionar uma divisão das atividades de análise no *grid*. Distribuir esta análise sem que as informações percam seu significado é uma tarefa complicada. A divisão do conjunto de dados “problema” de aplicações distribuídas tem sido alvo de vários estudos e, na gerência de redes, esta divisão pode ser um pouco complicada, devendo ser tratada para não proporcionar uma perda de significado das informações. Daí a existência do *grid* classificador. O *grid* classificador prepara as informações para que as tarefas de análise possam ser distribuídas mais facilmente.

A raiz do *grid* de análise recebe uma mensagem do *grid* classificador, indicando que existem dados a serem analisados e que esta análise precisa ser distribuída entre os *containers* pertencentes ao *grid*. Desta forma, é necessário que exista um registro dos *containers* que compõem este *grid* de processamento e suas habilidades, para que seja possível esta distribuição. Por isso a existência do serviço de informação descrito anteriormente. Na Figura 22 é apresentada uma visão abstrata da divisão das atividades de análise.

Na arquitetura original tinha-se a visão de que existiria uma raiz do *grid* responsável pelo escalonamento das atividades de análise. Inclusive alguns experimentos foram realizados tendo em mente esta visão. Porém, tal arquitetura não se mostra escalável por apresentar um único ponto de falha e um único ponto de escalonamento das tarefas. Uma alternativa é a de que os agentes analisadores têm objetivos próprios e para a realização destas tarefas eles podem usar de serviços ou delegar responsabilidades a outros agentes. Para isso eles consultam no serviço de informação, ou *brokers* e *traders* para conhecer os agentes que são capazes de atender

as suas necessidades. Neste caso a existência de uma raiz não seria extremamente necessária.

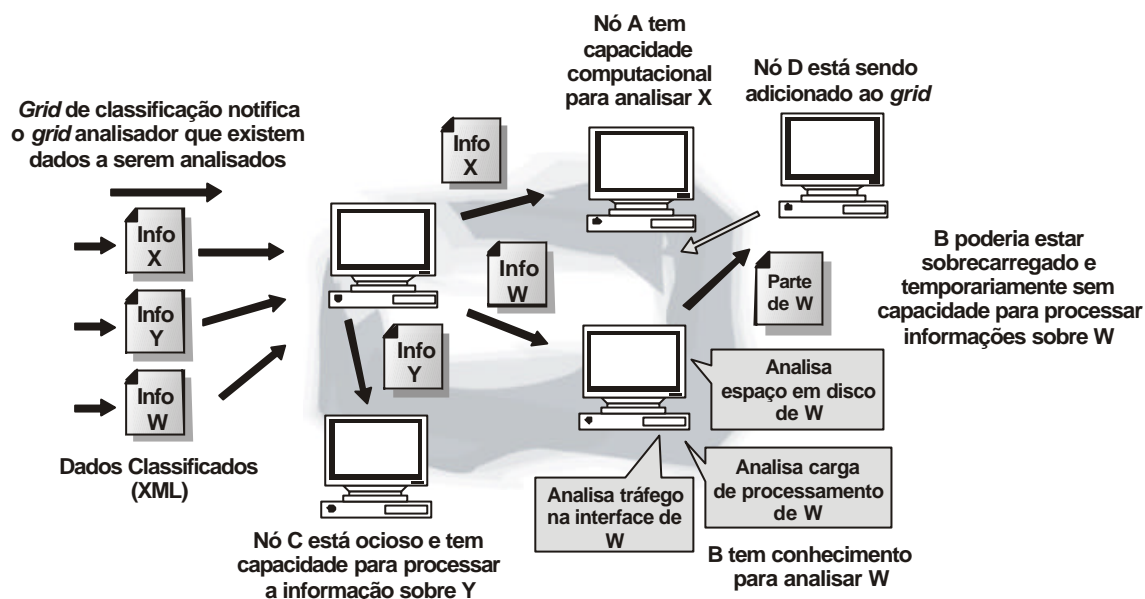


Figura 22 - Divisão das tarefas de análise [75].

Como foi citado, podemos adicionar um novo *container* ao *grid* de processamento, com regras para identificar alguns problemas. Quando o *container* é adicionado, ele anuncia seus serviços ao *grid* e passa a analisar as informações que lhe são designadas.

5.1.4 Grid de Agentes de Interface

Os agentes de interface são o canal de comunicação entre o *grid* e o gerente da rede. É também uma forma de receber o *feedback* do usuário e fornecê-lo ao sistema. A interface é tanto um canal de saída quanto de entrada para o sistema. Através de uma interface o usuário pode receber as informações processadas pelo *grid* de processamento e as mensagens de alerta. É possível programar novos relatórios e interagir com a base de conhecimento, definindo novas regras e objetivos e alterando objetivos existentes.

Esta interface deve ser abrangente, flexível e pode fazer uso de múltiplos protocolos, permitindo várias formas de interação conforme a necessidade do usuário – por exemplo, páginas HTML, e-mail, *chat*, XML/HTTP – e o retorno do usuário também pode se dar sob estas formas. Esta interface, além de ser o canal com o usuário, pode servir de alimentação para um outro sistema, ou mesmo ser realimentada por este

sistema, do qual pode aprender novas regras de gerência e transmiti-las ao *grid*. O *grid* de interface pode aprender novas regras através da análise das preferências e necessidades do usuário, ajustando o sistema.

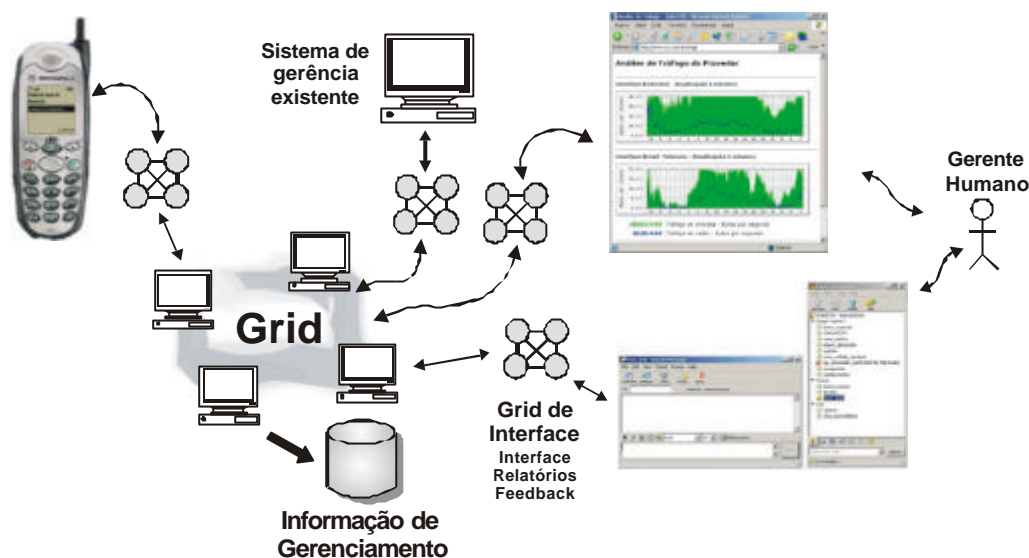


Figura 23 - Exemplo de interfaces do *grid*.

Na Figura 23 temos um exemplo de possíveis interfaces que o *grid* pode ter. O usuário pode ter uma interface HTTP e ter acesso a páginas em HTML onde são apresentados relatórios de gerência. Mensagens de alerta avisando possíveis anormalidades que o sistema não conseguiu tratar, ou que merecem a atenção do usuário, poderiam ser enviadas para o seu celular ou para um *software* de mensagens instantâneas como um *Yahoo Messenger* ou um *ICQ*. Ressaltando que além de apresentar informações, as interfaces podem receber informações do usuário ou de outros sistemas, o usuário poderia ter uma interface na qual os acordos de nível de serviço e políticas de gerenciamento são inseridos, e com isso, pode forçar o *grid* de análise e coleta a modificar seu comportamento e parâmetros, forçando a análise dos itens constantes nos acordos de níveis de serviço.

5.2 Requisitos do *Grid* para a Gerência de Redes e Sistemas

Com base na arquitetura de *grid* proposta para a gerência podem ser identificados alguns requisitos operacionais desejáveis a tal solução. São enumerados a seguir alguns requisitos necessários:

O sistema deve ser escalável: o sistema deve permitir uma escalabilidade não somente em número de agentes no sistema, mas também em relação às dimensões que ele pode tomar. Por dimensões, neste caso, nos referimos ao fato do *grid* de agentes poder agregar recursos de mais de uma organização para a realização de tarefas de gerência.

O sistema não deve ser intrusivo: os recursos que serão utilizados pela aplicação de gerenciamento baseada em *grid* poderão estar sendo utilizados pelos seus usuários, neste caso, chamados de donos dos recursos. O sistema não deve atrapalhar ou degradar as atividades dos usuários, usando o recurso para uma atividade de gerenciamento que seja exaustiva e que consuma uma grande quantidade do seu poder computacional. A ociosidade dos recursos pertencentes ao *grid* deve ser explorada ao máximo neste caso e o sistema deve ser o mais oportunista possível.

Deve ser adaptável e tolerante a falhas: o sistema deve poder se adaptar as condições atuais do ambiente. O ambiente pode ser aberto e dinâmico, com novos agentes entrando e outros deixando o *grid*. Neste caso, o sistema deve adaptar as suas atividades de gerenciamento para as condições de recursos existentes. Recursos ou agentes pertencentes ao *grid* podem falhar durante a execução de uma atividade de gerenciamento. O sistema deve se comportar de tal forma que os efeitos destas falhas sejam o menos devastadores possíveis.

A existência de aplicações legadas: uma das vantagens do uso de agentes de *software* no desenvolvimento de sistemas é a possibilidade de envolver sistemas legados e fazer com que estes possam ser vistos como agentes. Este ponto não pode ser esquecido porque no *grid* de agentes para a gerência poderão existir sistemas de gerenciamento legados que são adicionados ao *grid* por meio de um invólucro (*wrapper*) agente. Estes sistemas podem ser utilizados para prover funcionalidades de análises específicas, podem ser alimentados com dados ou informações de gerenciamento provenientes do *grid*, ou podem ainda, de forma inversa, prover dados de gerenciamento ao *grid* de gerenciamento.

Crítérios de segurança: dentre os critérios de segurança básicos no *grid*, os agentes que são adicionados ao *grid* e que irão realizar tarefas de gerenciamento devem ter certeza de que estas tarefas que podem lhe ser delegadas vêm de uma fonte segura e confiável. Algum certificado digital ou outro mecanismo que garanta esta segurança deve estar presente. Ele também deve garantir que a realização destas atividades não causará um consumo excessivo de recursos que pode causar o seu travamento, por exemplo. Ele deve ter algum conceito de *sandbox* para evitar este consumo excessivo ou que a atividade realize alguma operação não desejada. Deve ainda existir mecanismos de criptografia que garantam a confidencialidade das mensagens trocadas entre os agentes do sistema [43]^{*}, uma vez que eles podem usar de uma infra-estrutura de rede pública para trocar mensagens.

5.3 Balanceamento de Carga

O potencial de se combinar sistemas de processamento é óbvio: permite o escalonamento de tarefas, e que estas sejam realizadas de maneira paralela, em vários recursos fortemente ou fracamente acoplados. Em um ambiente composto por nós autônomos, as técnicas de escalonamento podem levar a uma situação onde alguns recursos são superutilizados e outros podem estar ociosos. O balanceamento de carga tem como objetivo primário possibilitar a equalização das cargas de trabalho em um ambiente paralelo ou distribuído de maneira que cada recurso tenha aproximadamente a mesma carga [24].

Na literatura, o balanceamento não está ligado apenas em balancear a carga de processamento em um sistema paralelo ou distribuído. Além de prover um balanceamento de carga de processamento, também pode consistir em balancear o fluxo de informações com o objetivo de fornecer uma melhor utilização de ligações de rede e o balanceamento de dados, onde pode ser balanceado também o armazenamento, de tal forma que este também possa ser distribuído. É claro que neste último caso normalmente o balanceamento está associado com técnicas de replicação e de aumento de desempenho na recuperação dos dados. Os agentes podem ser utilizados em qualquer um dos cenários apresentados, embora nosso interesse básico esteja no balanceamento

^{*} O “Anexo D” da especificação que define a arquitetura abstrata da FIPA trata de alguns aspectos de segurança desejáveis em implementações da arquitetura abstrata.

da carga de processamento ocasionada pelas tarefas de análise das informações de gerenciamento e no balanceamento de dados.

Muitos partem do princípio de que quando se tem um bom escalonamento tem-se um bom balanceamento de carga. Isso nem sempre é verdade, pois embora o escalonador possa levar em consideração informações sobre o estado dos recursos durante a etapa de decisão, escolhendo em quais recursos escalonar a execução das tarefas, o cenário pode mudar durante a sua execução. Na taxonomia apresentada por Casavant [24] ele associa o balanceamento de carga com os métodos de escalonamento dinâmicos. Grivas [61] apresenta alguns cenários onde a tecnologia de agentes pode auxiliar no balanceamento de carga. Mecanismos como negociação são úteis na distribuição de tarefas, no que Casavant chama de modelo *bidding*. A mobilidade de agentes é útil no sentido de prover a mobilidade de tarefas e seu estado durante a sua execução, caso o agente constate que o sistema onde ele está em execução não possui mais recursos disponíveis que garantam o término com sucesso e em tempo hábil de sua tarefa.

No modelo de *grid* proposto adotou-se alguns critérios para a distribuição de carga das atividades de análise. Inicialmente pensou-se em ter um escalonador ou distribuidor de tarefas central, estático e que poderia levar em consideração algumas características dos recursos, que era chamado de raiz do *grid* [12]. Porém, tal abordagem não se mostra muito flexível e acaba por eliminar muitas das vantagens de se utilizar um sistema de agentes. Uma segunda abordagem é considerar que cada agente no *grid* de análise de dados pode ser um escalonador em potencial. Ele pode estar incumbido de realizar algumas tarefas de análise que lhe foram delegadas e pode distribuí-las contratando serviços de outros agentes conforme lhe for necessário. Para proceder com esta distribuição alguns fatores podem ser levados em conta, como:

- Distribuir as tarefas entre os agentes com conhecimento e habilidades necessárias para processar o conjunto de dados;
- Utilizar recursos com capacidade computacional para realizar as tarefas de análise necessárias;
- Distribuir tarefas com base na ociosidade de recursos, utilizando recursos que estejam ociosos.

A inclusão do serviço de informação que permita aos agentes descobrirem, além de serviços fornecidos, quais agentes pertencem ao *grid* e as características dos agentes usadas para distribuição, como o conhecimento para realizar a tarefa ou capacidade computacional disponível é extremamente importante. A inclusão de agentes com funcionalidades de *trader* e *broker* no *grid* de agentes é desejável e pode se mostrar interessante no sentido de que um grupo de agentes pode consultar um *trader* ou *broker* que já possui informações sobre os agentes ou algumas avaliações a respeito deles, podendo estar em um *cache*, e assim evitando que cada agente tenha que realizar as operações e avaliações que um *trader* ou *broker* faria. O *trader*, *matchmaker* ou *broker* pode residir em um recurso computacionalmente melhor, evitando que um agente que esteja em um recurso computacional bastante limitado, como um assistente pessoal, tenha que desprender tempo e recurso avaliando as condições e propostas de outros agentes.

5.4 Negociação no *Grid* de Gerência

Os princípios de negociação entre agentes podem ser aplicados primeiramente em dois pontos do *grid*: no *grid* de classificação e dados e no *grid* de análise de informação. No *grid* de análise, cujo interesse é o alvo deste trabalho, os princípios de negociação podem ser usados para proporcionar a distribuição das atividades de análise. Assim o *grid* pode ser visto como um ambiente orientado por comércio, onde agentes gerentes estão incumbidos de realizarem tarefas de análise. Para a realização destas, podem procurar por agentes fornecedores que são capazes de oferecer as “mercadorias”, que neste caso são expressas em serviços ou capacidades, e negociar com estes agentes a realização de tais tarefas. Os agentes responsáveis pelas tarefas podem solicitar que os agentes fornecedores façam as suas propostas para a realização das tarefas. Para formulação das propostas os agentes fornecedores podem levar em consideração a disponibilidade de recursos do sistema onde estão. Com base na quantidade de recursos disponíveis eles podem aumentar ou diminuir o seu interesse em realizá-las. No modelo de comércio pode ser adotada uma “moeda” comum dentro do *grid* e independente das funções usadas pelos agentes na formulação das propostas, estas serão sempre expressas nesta mesma moeda.

Protocolos como **Contract Net**, e os de leilão são úteis neste cenário. Eles são descritos pela **FIPA** como protocolos de interação [49]. Outro modelo de organização aplicável a este *grid* de análise vem da teoria dos jogos. Shen e outros [107] apresentam um modelo de negociação adaptativa para balanceamento de carga em *grids* de agentes que se apóiam em princípios da teoria dos jogos.

No *grid* de dados, a negociação pode ser realizada no sentido de proporcionar uma distribuição adequada de dados para o armazenamento e como meio de facilitar a recuperação. Um exemplo, aplicando os modelos de organização descritos acima pode ser assim descrito:

- Um agente precisa proceder com o armazenamento de um conjunto de dados recebido, classificado e agrupado da melhor maneira possível.
- O agente gerente é o agente que precisa realizar o armazenamento. Ele seleciona os agentes que podem realizá-lo, conhecendo suas capacidades e outros atributos.
- Ele pode negociar com os agentes armazenadores o armazenamento destes dados de maneira que algum agente irá realizá-lo, oferecendo os recursos de seu sistema para que isso seja feito.
- Para a formulação das propostas, os agentes fornecedores podem usar alguns critérios, que além dos recursos disponíveis para estas atividades, podem envolver a quantidade de requisições geralmente recebida a estes dados. Isso significa que um agente pode demonstrar um interesse maior, e conseqüentemente apresentar uma proposta melhor, se as requisições a este tipo de dados feitas pelos agentes de análise são bastante freqüentes. Neste caso, é útil para ele manter estas informações sob sua responsabilidade porque provavelmente alguém irá solicitá-las.

Contudo, é evidente que algum mecanismo de replicação de dados também é desejável, e os dados podem ser replicados nos recursos com maior disponibilidade e interesse em mantê-los.

5.5 Gerência de Redes

O objetivo da gerência de redes é garantir que os usuários recebam os serviços de tecnologia da informação com a qualidade de serviço que eles esperam [112]. Stallings

[111] divide as atividades de gerenciamento de rede em operações de monitoração e de controle. Nas operações de monitoração estão as atividades de obtenção de informações, enquanto que o controle envolve as tarefas de análise de dados e tomada de decisões. Uma abordagem maior é dada por Subramanian [112] que define o gerenciamento de rede como: operações, administração, manutenção e provisão da rede e de serviços.

O grupo de operações se preocupa com as operações gerais e diárias para o provimento dos serviços de rede e são agrupadas em uma organização definida pela OSI. A administração se preocupa com o estabelecimento e administração dos objetivos gerais, políticas e procedimentos do gerenciamento de redes. As áreas de instalação e manutenção se encarregam da instalação e reparo de equipamentos da rede. A provisão consiste de planejamento dos circuitos da rede, atividades que são geralmente exercidas por áreas de engenharia.

5.5.1 Modelo de Gerenciamento de Redes

Um modelo de gerenciamento geral completo é o definido pela ISO em seu modelo OSI. Ele abrange todos os aspectos do gerenciamento de redes e é composto por quatro sub-modelos apresentados na Figura 24: modelo organizacional, modelo de informação, modelo de comunicação e modelo funcional. Alguns padrões de gerenciamento não especificam todos estes modelos e outros não os apresentam de maneira explícita, como é o caso do SNMP, que embora tenha uma organização semelhante ao OSI, não especifica nada a respeito dos modelos utilizados.

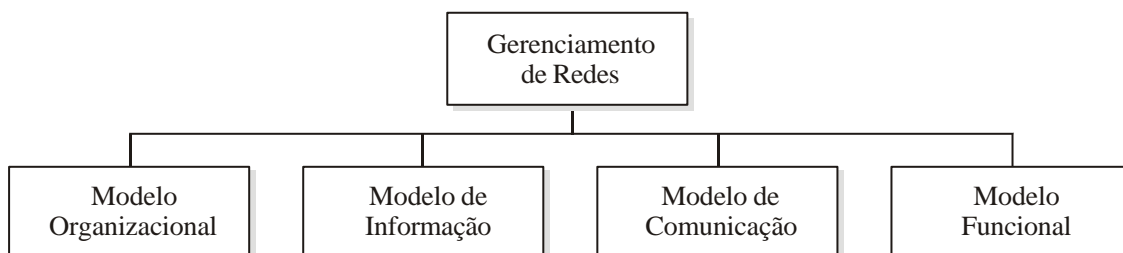


Figura 24 - Modelo de gerenciamento OSI [112].

Um modelo organizacional descreve os componentes do gerenciamento e o seu relacionamento. O modelo organizacional ISO define os termos: *objeto*, *agente* e *gerente*. Os elementos gerenciados possuem um processo em execução dentro deles que é chamado de **agente**. O **gerente** gerencia os elementos gerenciados. Normalmente o

gerente possui uma base de dados para armazenar as informações. O gerente consulta o valor dos **objetos** gerenciados do agente e recebe assim os dados de gerenciamento, processa-os, e armazena em sua base de dados. O agente também pode enviar um conjunto mínimo de alarmes sem a solicitação do gerente.

Um modelo de informação se preocupa com a estrutura e armazenamento da informação. A representação dos objetos gerenciados e a informação relevante ao seu gerenciamento constituem o modelo de informação de gerenciamento. A Estrutura da Informação de Gerenciamento (SMI) define a sintaxe e semântica da informação de gerenciamento armazenada na Base de Informação de Gerenciamento (MIB).

O modelo de comunicação tem três componentes: processos da aplicação de gerenciamento que atuam na camada de aplicação, a camada destes processos com as outras camadas e as operações dentro desta camada.

O quarto modelo é o modelo funcional e trata dos requisitos de gerenciamento do usuário e agrupa estes requisitos e operações dentro das categorias do modelo funcional OSI que engloba: configuração, falhas, desempenho, segurança e contabilização [112].

5.6 Gerência de Sistemas

Quando um usuário não consegue acessar uma aplicação em um servidor, ele pode não saber distinguir se o erro é devido ao transporte de dados da sua estação até o servidor ou da aplicação servidora responsável por prover o serviço solicitado. Este exemplo é usado por Subramanian [112] para criar uma distinção entre gerência de sistemas e de rede. O gerenciamento de sistemas pode ser generalizado como “*o gerenciamento de sistemas e seus recursos na rede*” [112].

A gerência de redes está preocupada com o gerenciamento de dispositivos como *hubs*, *switches*, roteadores e ligações de rede. Ou seja, os elementos necessários ao provimento de serviços de rede. O gerenciamento de sistemas envolve os sistemas existentes na rede, como estações de trabalho com seus sistemas operacionais e servidores de aplicações e outros serviços.

5.6.1 Arquitetura WBEM

São apresentados alguns detalhes da arquitetura **WBEM** (*Web-Based Enterprise Management*) [37] pelo fato de utilizarmos a implementação Microsoft desta tecnologia – *Windows Management Instrumentation (WMI)* [84] – para coletar informações desta plataforma e para formular regras de análise de dados utilizadas pelos agentes do *grid* de processamento.

O **WBEM** é um conjunto de tecnologias de gerenciamento e de padrões Internet desenvolvido para unificar o gerenciamento de ambientes computacionais empresariais. O conjunto de padrões desenvolvido pelo **DMTF** [36] inclui: um modelo de informação chamado *Common Information Model - CIM*; uma especificação para codificação, que é a *xmlCIM Encoding* e; um mecanismo de transporte que constitui as operações **CIM** sobre o protocolo HTTP.

O **CIM** é um modelo de dados orientado por objetos, que contém informações que representam diferentes partes da organização. Ele é uma forma de preservar e estender as fontes tradicionais de gerenciamento - como SNMP e DMI. O **CIM** é independente de linguagem de programação e usa técnicas de orientação a objetos para classificar e descrever as entidades gerenciadas, podendo descrever qualquer elemento gerenciável da organização. O **WBEM** utiliza o conceito de provedores de informação e gerenciador de objeto. Neste caso, a aplicação de gerenciamento solicita as informações através da chamada a métodos e acesso a atributos dos objetos definidos pelas classes do **CIM** através das operações **CIM**, e os provedores são responsáveis por fornecer as informações de maneira transparente.

Os elementos da arquitetura **WBEM** são descritos na Figura 25. Ele consiste basicamente de cinco componentes: Um cliente *Web*, o gerenciador de objetos **CIM** (**CIMON**), o esquema **CIM**, os provedores específicos de cada protocolo de gerenciamento e os objetos gerenciados com os agentes específicos de cada protocolo.

As aplicações podem realizar uma requisição a qualquer objeto gerenciado mantido por qualquer agente específico. Porém, conforme comentado, a requisição é sempre feita como se fosse acessado um objeto do esquema **CIM** e a requisição é enviada ao **CIMON**. O **CIMON** recebe as mensagens das aplicações e usa o esquema **CIM** para decidir qual provedor específico é capaz de prover aquela informação e envia

a mensagem ao provedor. O provedor mapeia a requisição para o protocolo específico usado para obter a informação e depois de obtida devolve ao CIMON. No caso de eventos, as mensagens também são mapeadas do protocolo específico do provedor para os objetos do esquema **CIM** e enviados ao CIMON. O CIMON funciona como um *proxy* para a informação de gerenciamento.

O **CIM** consiste de três módulos: o *core model*, *common models* e os *extension models*. O *core model* é um arcabouço de alto nível e é aplicável a todos os domínios de gerenciamento. Os *common models* são aplicáveis a domínios específicos e incluem informações sobre sistemas, aplicações, dispositivos, usuários e redes. Os *extension models* representam as extensões específicas dos *common models*, feitas por cada fabricante para uma tecnologia específica, como é o caso dos sistemas operacionais Windows e UNIX.

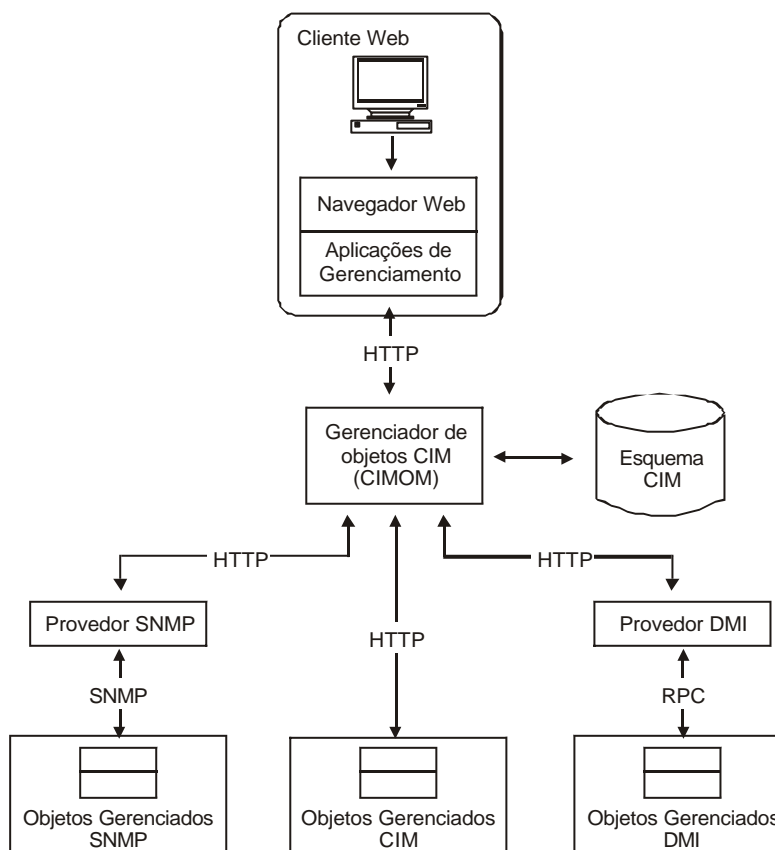


Figura 25 - Visão abstrata do modelo **WBEM** [112].

As operações **CIM** podem ser mapeadas sobre HTTP e permitem que diferentes implementações do **CIM** interoperem de maneira aberta e padronizada.

5.6.2 Utilitários Fornecidos pelos Sistemas

Os sistemas existentes geralmente fornecem ferramentas proprietárias e utilitários que, embora não sigam nenhuma padronização, fornecem funcionalidades importantes aos gerentes. Como exemplo de utilitários usados neste trabalho, podemos citar as ferramentas de linha de comando dos sistemas operacionais UNIX e Linux – como *df*, *ifconfig*, *netstat* e *free* – e do ambiente Windows - como *systeminfo*, *mem*, *chkdsk* e *netstat*.

Embora existam iniciativas de padronização, como **WBEM** e MIBs SNMP específicas [99], o poder destes utilitários não pode ser desprezado. É uma técnica muito comum no “mundo UNIX” usar tais ferramentas, juntamente com *scripts*, para a construção de *probes*, coletores de dados e diversas outras ferramentas de gerenciamento.

Estas ferramentas são também utilizadas na experimentação da proposta de *grid* para a construção de agentes coletores de dados. Estes coletores utilizam estas ferramentas para obter dados, e bibliotecas para realizar a extração de informações relevantes ao gerenciamento e representação destes dados em XML.

5.7 Cenários Possíveis de Utilização

Mediante a caracterização do problema enfrentado pelo gerenciamento de redes e sistemas apresentado na introdução deste trabalho, podemos afirmar que o princípio básico que orienta e justifica a proposta de uma arquitetura de *grid* para a gerência é a existência de um cenário de gerenciamento complexo, onde existe um volume de dados de gerência muito grande e cuja manipulação e processamento requerem um grande poder de processamento. Estas situações podem ser encontradas em ambientes onde não existe o interesse, por questões técnicas ou estratégicas, de manter estações de gerência intermediárias. Por estações de gerenciamento intermediárias nos referimos a, por exemplo, o modelo de gerentes de gerentes apresentados em algumas arquiteturas.

No caso de um suporte a sistemas, provido por empresas especializadas em atividades de gerenciamento, não existe um interesse em manter estações de gerenciamento dentro das dependências das organizações clientes. Tais dados são

coletados através de coletores (*probes*) e são enviadas de alguma forma para as organizações de gerência que proporcionam serviços especializados.

Outro cenário semelhante acontece na gerência de sistemas de telecomunicações. A quantidade de informação da coleta de dados e de alarmes oriundos de tais sistemas alcança facilmente a casa dos gigabytes por dia.

Além destes cenários descritos, que são difíceis de serem trabalhados em uma possível experimentação, podemos citar o gerenciamento baseado em políticas e o gerenciamento baseado em Acordos de Níveis de Serviço – *Service Level Agreement* ou SLA. Procurou-se descrever alguns destes cenários de gerenciamento e como os *grids* de agentes podem ser aplicados em cada um deles.

5.8 O *Grid* na Gerência de Sistemas

Um cenário onde os *grids* de agentes podem ser aplicados envolve uma situação em que uma empresa qualquer desenvolve, instala e gerencia sistemas de telecomunicações para empresas situadas em diferentes países. O gerenciamento e manutenção de tais sistemas não são atividades que podem ser exercidas por outras empresas. Os dados de desempenho coletados dos sistemas operacionais e dos sistemas fornecidos pela empresa são enviados a um centro de gerência geograficamente distante, através de um protocolo como HTTP ou SMTP. O centro de gerenciamento é responsável por analisar estas informações. Este cenário é descrito na Figura 26.

Agentes coletores – ou *probes* escritos em diferentes linguagens – coletam as informações necessárias dos sistemas existentes nos diversos *sites*. Estes dados podem ser armazenados, sumarizados localmente e possivelmente compactados para serem enviados ao centro de gerenciamento. Neste centro de gerenciamento estão concentrados o *grid* de classificação e o *grid* de análise de dados. O *grid* utiliza os recursos de vários micro-computadores - possivelmente padrão IBM-PC – para tais atividades. No modelo de organizações virtuais [9] o *grid* pode contratar recursos de terceiros para prover ciclos de processador ou capacidade de armazenamento para armazenar os dados de gerenciamento.

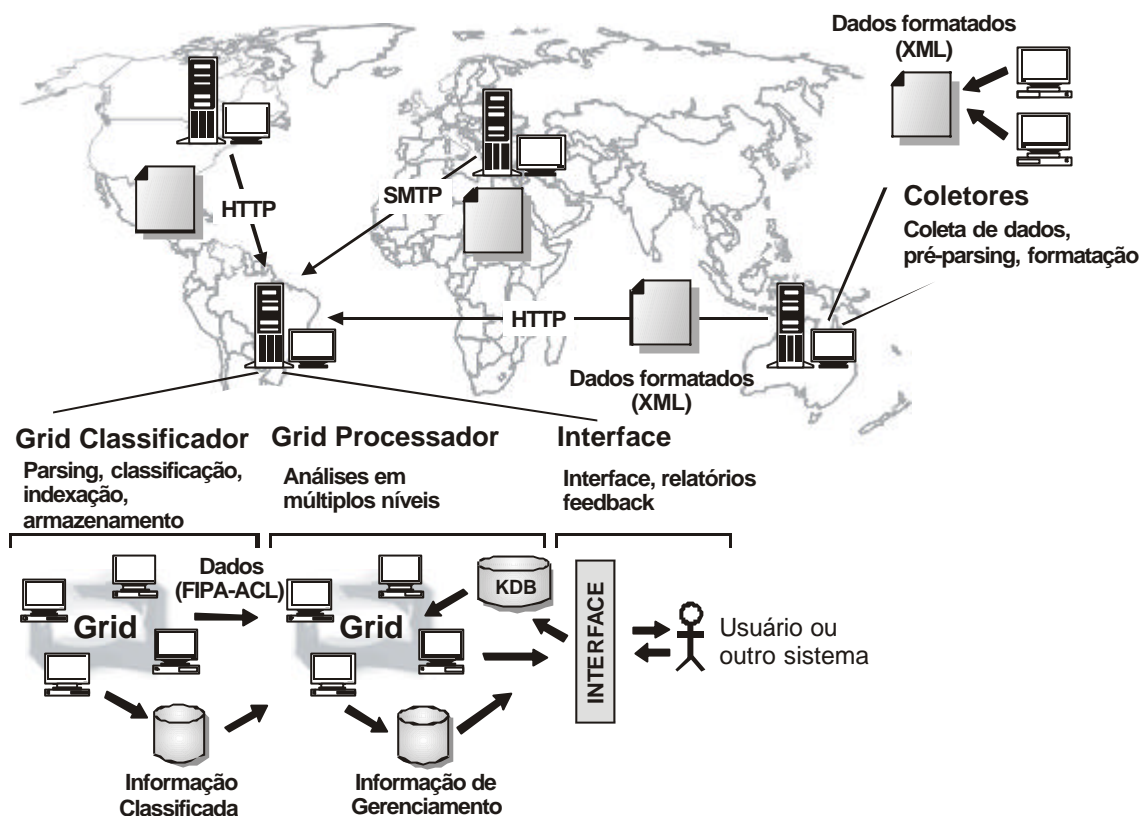


Figura 26 - Cenário exemplo de gerenciamento.

5.8.1 O Grid Formado pela Rede Gerenciada

Em uma situação de gerenciamento onde um enorme volume de dados é coletado da rede gerenciada e onde é necessário um grande poder de processamento para a realização das análises de dados é interessante poder utilizar os próprios recursos dos sites gerenciados para realizar a análise das informações. Nesta situação, os recursos da rede gerenciada funcionam como colaboradores do próprio *grid* nas atividades de gerenciamento. Esta abordagem é interessante porque os próprios recursos da rede são usados para que a mesma seja gerenciada de maneira mais eficiente e onde um grande conjunto de dados pode ser analisado. Esta visão é apresentada de maneira abstrata na Figura 27: os dados são coletados da rede gerenciada pelos agentes coletores de dados (a); os dados a serem analisados são repartidos e dão origem as tarefas de análise (b); as tarefas são atribuídas aos recursos da rede gerenciada para serem processados (c); o resultado deste processamento dá origem aos relatórios de gerenciamento (d) que são apresentados a uma interface para o gerente humano (e).

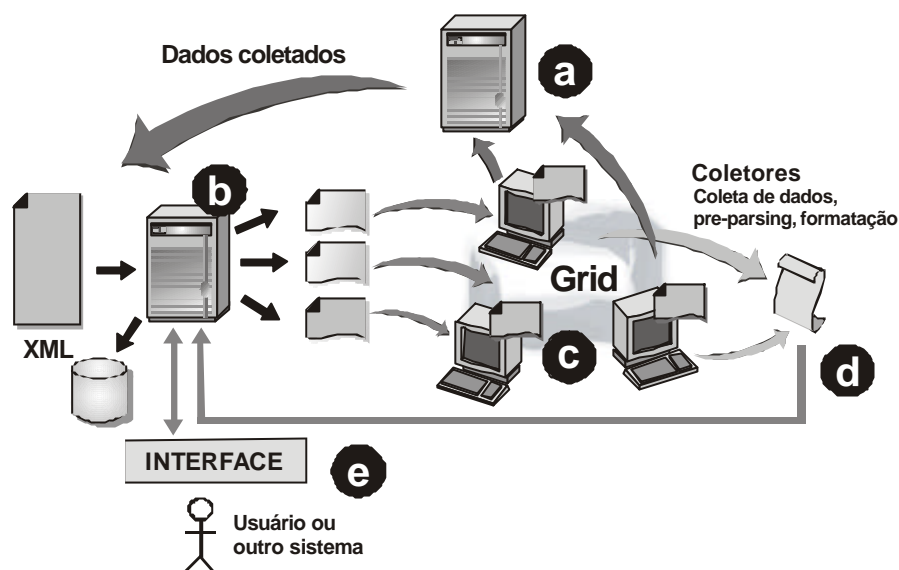


Figura 27 - Grid é formado pela rede gerenciada.

5.9 Conclusões

Foram apresentadas neste capítulo a arquitetura de gerenciamento proposta, baseada em *grid* de agentes e a descrição de seus módulos componentes. Também foram descritas as funcionalidades e objetivos de cada módulo.

As vantagens que a arquitetura se propõe a evidenciar são discutidas neste capítulo. Dentre estas estão a possibilidade de uso de recursos ociosos da rede, a possibilidade de agregação de recursos e conseqüente balanceamento da carga de trabalho das atividades de gerenciamento.

Também, neste capítulo foram mencionados alguns exemplos de utilização dos *grids* de agentes na gerência de redes e de sistemas. A intenção é que em um trabalho futuro seja possível ter uma descrição ampla e a exploração de vários cenários que justifiquem a utilização de tais *grids*.

Ainda, com algumas alterações na arquitetura apresentada, é possível proporcionar o gerenciamento para arquiteturas de *grid* orientadas por serviços, onde uma das formas de gerenciamento mais adequadas é o gerenciamento baseado em acordos de níveis de serviços (SLA). Sob o ponto de vista de uma organização que contrata um serviço e não tem acesso as informações de gerenciamento, o gerenciamento baseado em tais acordos é uma das alternativas mais adequadas.

6 Ambiente e Resultados Experimentais

Com o objetivo de validar os conceitos de *grids* de agentes na gerência de redes e de sistemas, um protótipo e exemplos demonstrando o funcionamento de alguns componentes do sistema de gerenciamento foram desenvolvidos. Nem todos os componentes e funcionalidades exigidas por um *grid* de agentes podem ser implementadas no período de tempo que se tinha disponível. Deste modo, com o foco mantido na aplicação de gerenciamento, optou-se por implementar uma arquitetura com os serviços básicos de um *grid* de agentes e que garantisse a construção de tais aplicações e sua experimentação. Sendo assim, foi escolhida uma plataforma de agentes, a qual foi adaptada às necessidades de desenvolvimento deste trabalho e sobre ela foram implementados os serviços e funcionalidades adicionais. Os detalhes da implementação do protótipo de *grid* de agentes e estes experimentos são apresentados neste capítulo. Também são apresentados detalhes referentes à escolha das ferramentas e da plataforma de agentes utilizada.

O protótipo aqui apresentado procura cobrir os principais pontos da arquitetura de *grid* de agentes proposta, envolvendo: a coleta de informações de gerenciamento, o armazenamento de dados, a análise e a apresentação das informações ao gerente humano. Desta forma, primeiramente são apresentados os recursos utilizados no desenvolvimento e uma visão geral da plataforma construída. Em seguida são detalhados os componentes da arquitetura e após isso, são apresentados cenários de utilização e experimentação da arquitetura proposta.

6.1 A Plataforma *Agentlight*

Para o desenvolvimento da arquitetura de *grid* proposta foi escolhido o *framework* para construção de agentes *Agentlight* [3]. Esta plataforma foi proposta por Koch [3], um dos membros do projeto *AgentGrid* [2] (antiga atividade F do projeto HOPE [63] e atual atividade F do TAGERE [115]) e co-autor em alguns trabalhos resultantes de atividades do projeto [12],[13]. Alguns fatores que influenciaram na escolha desta ferramenta são:

- Disponibilidade do código fonte para alterações.
- Simplicidade e flexibilidade da plataforma.

- O *framework* é todo desenvolvido em Java e, portanto pode ser usado em qualquer plataforma que possua uma máquina virtual Java.
- Um dos objetivos do *framework* é possibilitar a construção de agentes para pequenos dispositivos, o que facilita a sua utilização em diferentes plataformas, desde pequenos assistentes pessoais até servidores com recursos computacionais mais sofisticados.
- Como o *Agentlight* tem como princípio básico proporcionar uma plataforma para desenvolvimento de agentes para pequenos dispositivos, os agentes são criados com o intuito de serem bastante conservativos no que se refere ao tamanho de código compilado e utilização de memória. Estes cuidados são importantes no desenvolvimento da plataforma e são úteis também no *grid*, pois podem garantir a sua *pervasividade** e ubiqüidade.

Além destes fatores, o *Agentlight* está sendo desenvolvido e modificado para facilitar a realização das idéias sobre *grids* de agentes. Alguns módulos desta plataforma estão sendo desenvolvidos para a experimentação das teorias propostas.

O *Agentlight* funciona com o conceito de *container* de agentes. Um container pode ser visto como uma aplicação que possui vários agentes em execução. Os agentes são *Threads* da aplicação *container*.

A escolha pela linguagem de programação Java se deve as várias funcionalidades providas pela linguagem, tais como: como *Threads*, programação concorrente, gerenciamento de memória, primitivas de comunicação, tratamento de exceções, entre outras.

A portabilidade é uma das mais importantes características proporcionadas pelo Java, uma vez que ele funciona com uma máquina virtual que interpreta o código intermediário chamado de *bytecode*. Não é necessário que os programas sejam compilados por um compilador específico para cada plataforma. Em um ambiente de computação onde os recursos são heterogêneos e podem possuir diferentes sistemas operacionais, a adoção de uma linguagem que permita a portabilidade do código compilado é um fator importante. Porém, como destaca Dietz [34] o Java apresenta uma queda de desempenho que pode ser melhorada com o uso de código nativo. Tal

* O termo “pervasividade” foi utilizado para representar *pervasive* do inglês. Tal termo foi usado por desconhecer no português uma única palavra que possa expressar todo o sentido de *pervasive* sem causar dualidades que possam confundir o entendimento.

abordagem não é recomendada em um ambiente heterogêneo porque vai contra a idéia de portabilidade proporcionada pelo Java.

Apesar de ser uma linguagem flexível e de apresentar um grande potencial para a construção de sistemas de computação paralela e distribuída, o Java recebe algumas críticas com relação a dificuldade de explorar toda a diversidade de recursos existentes nos dispositivos de processamento. Coisa que só pode ser realizada através do uso de código nativo. No caso do *Agentlight*, a chamada a sistemas legados e código nativo pode ser realizada através de *Skills* que são carregadas como regras pelos agentes do *grid*.

Outro fator importante da linguagem Java é a segurança proporcionada tanto pelas bibliotecas de criptografia que permitem o uso de certificados digitais que podem ser usados para garantir a confidencialidade e autenticação das mensagens trocadas pelos agentes. Também é possível usar o conceito de *sandbox* (ou caixas de areia) para evitar que um agente execute um código malicioso ou que realize alguma operação indesejada que possa ocasionar problemas para a entidade de processamento.

6.1.1 A Arquitetura do *Agentlight*

Os agentes no *Agentlight* são agentes de inferência que utilizam um mecanismo de raciocínio baseado em um motor de inferência que infere sobre uma base de conhecimento que possui fatos e regras armazenados usando uma estrutura baseada em lógica de primeira ordem. Os agentes no *Agentlight* são compostos basicamente por:

- Pela base de conhecimento onde são armazenados os fatos e regras que constituem o conhecimento do agente. Esta base de conhecimento pode ser carregada na memória do recurso onde o agente está em execução, ou ser armazenada de forma persistente em um banco de dados relacional.
- Uma máquina de inferência que é usada para inferir sobre as regras armazenadas na base de conhecimento. Esta máquina de inferência é uma adaptação do *W-Prolog* [127]. Assim como funciona em um programa PROLOG, quando um agente possui um determinado objetivo, ele infere sobre a base de conhecimento em busca de cláusulas e regras que casem com a cláusula dada como objetivo.

- Os objetivos que o agente deve perseguir durante o seu ciclo de vida.

Conforme mencionado, o *Agentlight* utiliza o conceito de *container* de agentes. Este conceito é utilizado para permitir que se possa ter um bom número de agentes em execução com um uso mínimo de recursos do sistema. Neste caso, os agentes são encadeamentos de uma aplicação que é iniciada quando o *container* é iniciado. No *Agentlight*, um *container* é composto basicamente por:

- Um diretório onde são armazenadas informações inerentes aos agentes do *container*, rotas para outros *containers*, informações sobre as linguagens de conteúdo de mensagens que são aceitas pelos outros *containers*, codificação de mensagem aceita e protocolos utilizados para transporte das mensagens.
- Uma base de conhecimento global que é compartilhada por todos os agentes do *container*. Desta forma, ao inferir, primeiramente a máquina de inferência do agente infere sobre a sua base de conhecimento local e após isso sobre a base de conhecimento global.
- Um agente gerenciador, chamado de *manager*, que é o agente responsável por processar as mensagens destinadas ao *container*. Este agente possui em sua base de conhecimento um conjunto de regras necessárias para realizar as tarefas de gerenciamento do *container*. Sempre que uma mensagem for destinada ao *container* será o agente gerente que irá executá-la.

Outro módulo importante e utilizado no conceito de *grid* de agentes é o de comunicação, ou de rede. Quando um *container* é criado, também pode ser fornecido o seu *Communicator*. O *Communicator* define as primitivas de comunicação utilizadas pelo *container* para que este possa se comunicar com outros *containers* localizados em estações diferentes. O *Communicator* de um *container* é composto por *Listeners* e clientes que permitem que ele possa enviar e receber mensagens de um outro *Communicator* e assim estabelecer a comunicação entre diferentes *containers*.

Os *Listeners* para um protocolo podem ser constituídos de servidores que permitem que o *container* receba uma mensagem através de um determinado protocolo. No caso do protocolo HTTP, por exemplo, o *listener* é um servidor capaz de receber mensagens no padrão **FIPA** através do método POST do protocolo HTTP e de processá-las, extraindo o envelope e a mensagem **FIPA** original enviada, passando-a para o *container* para que seja processada [46].

Um cliente permite o inverso, que o *container* envie uma mensagem através de um protocolo usado para transporte. No mesmo caso do HTTP, o cliente é responsável por proceder com a criação das requisições originadas pela transmissão das mensagens **FIPA** e de postá-las para o servidor HTTP do *container* destino da mensagem.

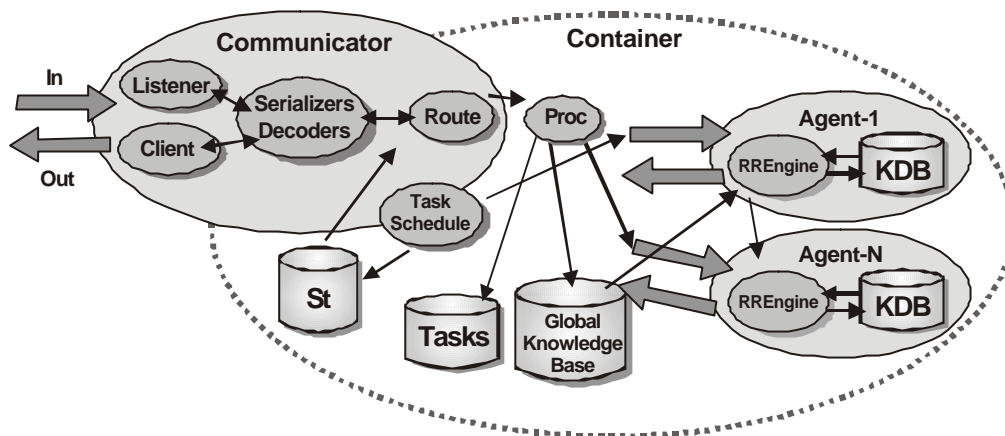


Figura 28 - Visão geral dos componentes do *Agentlight* (Adaptado de [73]).

A relação entre os blocos que compõem o módulo de comunicação do *Agentlight* são descritos na Figura 28.

6.2 O Conceito de *Skills*

O armazenamento dos fatos e regras na base de conhecimento dos agentes no *Agentlight* é feito usando estruturas lógicas, ou seja, lógica de primeira ordem. Quando um agente tem um objetivo e tenta perseguir-lo, a máquina de resolução de regras infere sobre a base de conhecimento e usa estas estruturas para encontrar um fato ou regra que case com o que ele está procurando. Em alguns casos, ao perseguir um objetivo, é necessário que o agente execute código Java através da chamada de funções de uma API ou da execução de algum código que não pode ser representado de forma lógica. Um exemplo disto pode ser o de um agente que precisa realizar uma coleta de dados usando o protocolo de gerenciamento SNMP. As funções de gerenciamento a serem executadas para coletar estas informações devem ser representadas em forma de estruturas lógicas.

A ligação entre o conhecimento dos agentes, que é estruturado em forma de cláusulas de lógica de primeira ordem, e entre as coisas que não são lógicas, como o uso de classes e métodos de bibliotecas deve ser feita através da implementação de uma interface chamada *SkillInterface*, apresentada na Figura 29. Basicamente uma

SkillInterface consiste de apenas um método chamado *skill* que recebe os termos passados pela regra que está chamando a *skill* e a pilha de unificação da máquina de inferência.

```
public interface SkillInterface{
    public boolean skill(Term term, Stack stack);
}
```

Figura 29 – Interface Java *SkillInterface*.

Nos arquivos de definição quando uma regra está associada a uma *skill*, deve ser informada a *skill* correspondente para a qual os termos serão repassados. No momento de carregamento dos arquivos XML de definição, o nó realizará a instanciação da *skill* correspondente. Na Figura 30 apresenta um exemplo de mapeamento de regras SNMP com uma *skill* responsável por realizar as operações SNMP. Neste caso, quando alguma regra procurada na base de conhecimento fechar com a regra *snmp*, tiver aridade 4, e os termos passados unificarem também com a regra, os termos serão passados para a *SNMPSkill* que executará a ação correspondente.

```
snmp(Cmd,Arg1,Arg2,Arg3) :-
    !br.ufsc.lrg.agentgrid.skill.SNMPSkill(Cmd,Arg1,Arg2,Arg3).

[...]

snmpcollect(Address,Community,OIDList,Result) :-
    snmp(connection,Connection,Address,Community),
    snmp(get,Connection,OIDList,Result).
```

Figura 30 – Mapeamento de regras de coleta SNMP para a *skill*.

Durante o desenvolvimento deste trabalho foram desenvolvidas *skills* para diversos fins. Para a coleta de informações de gerenciamento foram desenvolvidas *skills* SNMP, para a execução de utilitários de linha de comando e para extração de informações do resultado de execução de comandos. Para o armazenamento e recuperação de informações foram desenvolvidas *skills* para o armazenamento em bases de dados relacionais, em arquivos de textos, dentre outras. Também foram desenvolvidas *skills* para outros fins diversos, como por exemplo, a geração de gráficos de gerenciamento.

6.3 O Núcleo do *Grid* de Agentes

O núcleo básico do *grid* de agentes é construído sobre a infra-estrutura de agentes proporcionada pelo *Agentlight*. Aqui se chama núcleo, uma infra-estrutura básica necessária para que se possa iniciar um nó do *grid* de agentes, o que se pode chamar também de *kernel* do *grid* de agentes. A idéia é a de que um usuário que queira construir um *grid* de agentes ou doar os seus recursos para serem utilizados por um *grid* de agentes para que este possa usá-los para realizar atividades de gerência, tenha que baixar apenas alguns arquivos necessários à iniciação do aplicativo que ele irá executar, e a sua configuração. Este conjunto de arquivos poderá ser constituído basicamente de um ou mais pacotes que correspondem ao *kernel* do *grid* de agentes – arquivos com a extensão *jar* no caso do Java - e documentos XML que conterão todas as definições que se façam necessárias para a criação e configuração. Estes arquivos de definição contêm todos os detalhes específicos da aplicação para a qual o nó será criado.

Então, a aplicação que podemos chamar de núcleo do *grid* é composta pelas classes necessárias para que seja iniciado um nó do *grid*. Um nó é uma especialização do *Container* do *Agentlight* contendo métodos e conhecimento necessários para carregar documentos XML contendo definições do nó.

Um dispositivo pode conter um ou vários nós *grid* uma vez que vários *Containers* podem estar em execução em uma mesma estação de trabalho, mas com os *Listeners* de seus *Communicators* “ouvindo”^{*} em portas diferentes. Uma estação de trabalho, por exemplo, não representa um único nó no *grid*, não havendo restrições a este respeito e podendo existir vários nós *grid* em um mesmo recurso.

Na Figura 31 é apresentado um diagrama de seqüência que demonstra como acontece o carregamento dos arquivos e a iniciação de um nó do *grid*. Quando um usuário inicia a aplicação, através da invocação a classe *GridNodeJ2SE*, que é uma extensão de *Container*, é carregada com um conhecimento básico que garante o funcionamento do nó. Os comandos passados através da linha de comando, são carregados pelo *CMDLine*. Os parâmetros são usados para configurar os *Listeners* e para passar ao leitor de arquivos XML de configuração quais arquivos devem ser carregados. O *parser* realiza o *parser* destes dados e carrega o conhecimento e

* O *listener* no caso do HTTP, por exemplo, é composto de um servidor associado a determinada porta TCP.

elementos que devem ser criados no *Container*. Também é iniciado o medidor de desempenho que é responsável por coletar informações a respeito do dispositivo, através de um protocolo de gerenciamento ou alguma ferramenta de instrumentação e de construir as informações de utilização de recursos do dispositivo.

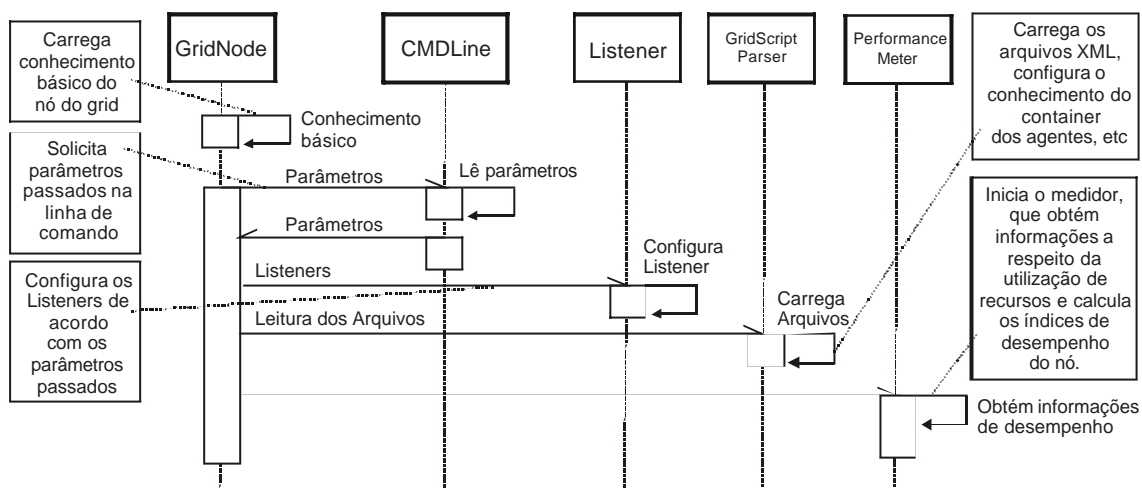


Figura 31 - Seqüência de iniciação de um nó do grid.

Alguns dos componentes descritos, como o *GridScriptParser* são implementações da interface *SkillInterface* do *Agentlight*. Neste caso eles são carregados como parte do conhecimento do nó do *grid* de agentes.

6.4 Início de um Nó Grid

Para que seja iniciado um nó *grid* em uma estação de trabalho com uma máquina virtual Java compatível com o J2SE, o usuário deve executar a classe *GridNodeJ2SE* que contém o método *main* que desencadeará o início do processo de carregamento e iniciação do nó. Conforme apresenta a Figura 31, o primeiro passo a ser executado é a leitura dos parâmetros passados pelo usuário na linha de comando fornecida para iniciação do nó. Na Tabela 2 temos uma lista de parâmetros que são aceitos quando é realizada a invocação de um nó do *grid*.

Tabela 2 - Parâmetros fornecidos para a iniciação de um nó *grid*.

Parâmetro	Tipo	Descrição
-name	Texto	Nome de identificação do <i>grid</i> . Os agentes do nó criado serão identificados por <i>agente@nome</i>
-port	Numérico	Número da porta HTTP que será utilizada pelo

		<i>Listener</i> deste nó.
-bind	Texto	Se este nó deve se ligar a outro nó já existente, deve ser informado o endereço ao qual ele deve se ligar. Ex: <i>http://150.162.63.2:5000</i>
-meter	Texto	Este parâmetro é usado para informar a classe do medidor de desempenho que será utilizado para coletar as informações estatísticas do nó. Ex: <i>br.ufsc.lrg.agentgrid.meter.SNMPMeter</i>
-load	Texto	Lista de arquivos XML que o nó deve carregar. Estes arquivos possuem informações de configuração, agentes que devem ser criados, fatos e regras que estes agentes devem possuir em suas bases de conhecimento e objetivos que eles devem perseguir durante o seu ciclo de vida. Vale lembrar que esta é a configuração inicial do nó. Ex: <i>http://localhost/analize-example-001.xml</i>
-trace	Texto	Neste parâmetro pode ser fornecida uma série de itens dos quais deseja-se realizar o <i>trace</i> . As informações geradas pelo <i>trace</i> são usadas para acompanhamento, identificação e correção de erros. O nível do <i>trace</i> pode variar de 0 a 3. Ex: <i>rrengine=3,script=2,agent=3,node=0</i>

Um exemplo de iniciação de um agente coletor que extrai os dados de gerenciamento através do pacote de instrumentação WMI proporcionado pelas plataformas Windows é apresentado na Figura 32. Neste exemplo o nó é criado com o nome de *collector1* e este deverá se ligar a um nó já existente e em execução no mesmo *host*. A porta em que o servidor HTTP (*Listener*) deste nó deverá usar para receber solicitações será a porta 50002. É fornecida uma lista de arquivos XML de definição que devem ser carregados e, por fim, são configurados alguns parâmetros de *trace* para os itens *script* e *skill*. Alguns destes parâmetros não precisam ser fornecidos na linha de comando durante o início do nó, podendo ser inseridos como elementos dos documentos XML de definição.

```
java -cp node.jar br.ufsc.lrg.agentgrid.GridNodeJ2SE -name collector1 -
bind http://localhost:50000 -port 50002 -load
http://localhost/collector/collector.bootstrap.xml,
http://localhost/collector/cmdline-wmi-common.xml,
http://localhost/collector/wmi-win-xp.xml,
http://localhost/collector/cmdline-wmi-test-2.xml -meter
br.ufsc.lrg.agentgrid.meter.SNMPMeter -trace script=1,skill=3
```

Figura 32 – Exemplo de iniciação de um nó grid.

Após terem sido lidos os parâmetros fornecidos, o segundo item a ser executado é o carregamento do módulo de comunicação que será utilizado pelo nó. Este módulo de

comunicação será iniciado com os *Listeners* dos protocolos correspondentes, sendo que o protocolo HTTP é utilizado por padrão. O servidor correspondente para este protocolo será carregado e ficará em execução recebendo requisições na porta informada pelo parâmetro *port*. No exemplo apresentado na Figura 32 o servidor está em execução ligado à porta 50002.

Conforme apresentado na arquitetura original [12], o *grid* de agentes é formado por vários *containers* de agentes interligados por meio de ligações feitas usando protocolos HTTP ou SMTP. Quando um nó *grid* é iniciado ele pode se ligar a um nó já existente ou ser a raiz para um determinado *grid*. Quando é fornecida alguma informação no parâmetro *bind*, significa que o nó deve se ligar a um nó existente. Para realizar esta ligação o nó que está sendo iniciado envia uma mensagem FIPA para o nó ao qual ele está se ligando e este atualiza a sua tabela de rotas. O nó iniciado também atualiza a sua tabela de rotas, adicionando o nó ao qual ele se ligou como uma rota padrão. Esta idéia é ilustrada na Figura 33 onde temos um exemplo de um nó que foi iniciado e se ligou a um nó já existente e que já possuía um nó ligado a ele: o nó C solicita que A registre sua presença (1). A confirma a mensagem de C (2) e ambos atualizam suas tabelas de rotas (3).

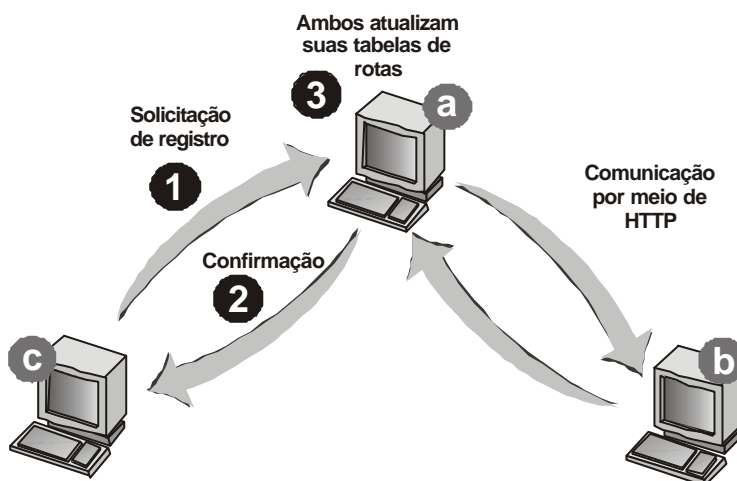


Figura 33 - Ligação de um nó a outro.

No parâmetro *meter* do nó iniciado na Figura 32 é fornecida a classe do medidor de desempenho utilizado. Esta classe estende uma classe abstrata Java que possui métodos que permitem a obtenção de informações de desempenho do recurso onde o nó está em execução. Estas informações de desempenho são utilizadas para montar as estratégias de distribuição das atividades de análise de dados. Neste exemplo do coletor

não faz muito sentido o nó possuir um medidor de desempenho, mas este parâmetro foi colocado para ilustrar a idéia de como ele funciona. Durante a etapa de iniciação, o nó faz o carregamento desta classe com os seus devidos parâmetros e inicia a execução do medidor de desempenho através da chamada ao método *init()* implementado pela classe do medidor. A partir deste momento o medidor inicia a sua tarefa de coleta de informações que possam demonstrar o grau de utilização do recurso. A classe abstrata que define o medidor de desempenho define um método *get(String param)* que é usado para recuperar as informações de desempenho obtidas pelo medidor. No exemplo apresentado na Figura 32, a classe do medidor de desempenho que deve ser carregada é *br.ufsc.lrg.agentgrid.meter.SNMPMeter*. A Tabela 3 apresenta uma lista das informações que podem ser obtidas através deste medidor.

Tabela 3 - Informações obtidas através do medidor de desempenho.

Informação	Descrição
overall-rate	Taxa geral demonstrando o grau de utilização do dispositivo.
cpu-number	Número de processadores que o recurso possui.
cpu-description	Descrição da arquitetura da CPU do recurso computacional.
cpu-total	Total de MFlops/s proporcionada pelo dispositivo.
cpu-usage	Uso dos processadores expresso em valor percentual.
memory-total	Total de memória do dispositivo em Kbytes.
memory-usage	Uso da memória expresso em valor percentual.
network-total	Total da capacidade de transmissão das interfaces do recurso, medido em Mbits.
network-usage	Uso das interfaces de rede expresso em valor percentual.
storage-total	Capacidade de armazenamento dos dispositivos de disco medido em Kbytes.
storage-usage	Uso dos discos expresso em valor percentual.
Location	Descrição do local onde o recurso se encontra.
Description	Descrição do recurso.

No passo seguinte são carregados os documentos de definição contendo configurações, regras e fatos a serem aplicadas ao nó que está sendo iniciado.

6.5 Sistema de Comunicação

A FIPA define uma linguagem de comunicação entre agentes que é chamada de FIPA ACL. A especificação que define a estrutura da mensagem é a 61 [44]. A linguagem é composta por um número de vinte atos comunicativos para definir a

intenção da mensagem trocada e a sua correta interpretação [48]. Não existe uma linguagem obrigatória para ser usada no conteúdo da mensagem, sendo que podem ser usadas linguagens como KIF, FIPA SL ou mesmo XML.

A fim de garantir a interoperabilidade entre diferentes plataformas de agentes, além de definir o formato e sintaxe de uma mensagem, a FIPA define diferentes formas de codificação de mensagens. Algumas especificações existentes apresentam a codificação baseada em *strings*, em XML e representação binária destinada a pequenos dispositivos com recursos de comunicação escassos.

Também são apresentadas especificações que definem como estas mensagens devem ser transportadas usando diferentes protocolos. As especificações apresentam como uma mensagem FIPA-ACL deve ser transportada usando protocolos como WAP, IIOP e HTTP [46]. Plataformas diferentes que desejam que seus agentes troquem mensagens, devem adotar um mecanismo comum de comunicação. Além destas especificações, a FIPA sugere meios para que o módulo de transporte de mensagens de uma plataforma possa escolher entre um ou outro mecanismo de comunicação a ser usado de acordo com a plataforma com quem se deseja trocar mensagens, sendo possível que uma plataforma escolha dentre vários mecanismos implementados e disponíveis.

Neste trabalho, para a comunicação inter-*container* o formato de mensagens adotado é o definido pela FIPA-ACL, bem como alguns dos atos comunicativos definidos por ela. O protocolo e padrão usado para transporte das mensagens FIPA é o HTTP. A escolha por tal mecanismo se deve ao fato deste protocolo ser ubíquo e de não encontrar problemas para ser utilizado em ambientes abertos e heterogêneos. Contudo, embora este protocolo seja utilizado por padrão, módulos de comunicação para utilização de protocolos como SMTP e POP3 estão sendo construídos.

Dos atos comunicativos definidos pela FIPA, alguns estão implementados na plataforma: como *request*, *inform*, *query-ref* e *query-if*. Embora apenas estes atos estejam implementados na plataforma, os demais podem ser inseridos de uma maneira flexível, sem alterações no código, apenas com a adição das regras que definem o seu comportamento.

Internamente a troca de mensagens no *container* ocorre através da passagem de mensagens entre objetos. Quando se faz necessário a comunicação com um agente

externo ao *container*, a mensagem é passada para o módulo de comunicação que identifica a rota para a mensagem e realiza a representação da mesma usando o módulo cliente do protocolo que será usado para realizar a transferência. Na Figura 34 é apresentado um exemplo de uma mensagem enviada por um agente de análise solicitando a um agente armazenador que ele retorne o conteúdo dos 10 últimos valores da porcentagem de uso do processador do sistema cujo endereço IP é “10.0.9.53”. Neste caso o conteúdo enviado na mensagem deve ser unificado com uma cláusula na base de conhecimento do armazenador e a cláusula resultante da unificação será então devolvida.

```

POST /
content-type: multipart/mixed; boundary="part171329824138561070811400866"
connection: keep-alive
accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
cache-control: no-cache
host: 10.0.9.53:50001
content-length: 725
user-agent: Java/1.4.1_02
mime-version: 1.0

[...]

--part171329824138561070811400866
Content-Type: application/fipa.acl.rep.string.std

(query-ref
  :receiver
  (agent-identifier
    :name store_agent@storage
    :addresses (sequence http://localhost:50000)
  )
  :reply-with msg17291351000
  :sender
  (agent-identifier
    :name analysis_agent@analysis-1
    :addresses (sequence http://10.0.9.53:50004)
  )
  :language agentlight-term
  :content (
    listLastCollections("10.0.9.53", "processor", "Load", ListColl)
  )
)

--part171329824138561070811400866

```

Figura 34 - Mensagem FIPA transportada por HTTP.

6.6 Arquivos XML de Definição

Os arquivos de definição carregados por um nó *grid* no momento em que ele é iniciado são documentos XML que possuem uma estrutura simples e flexível. Estes

documentos contêm a lógica da aplicação para a qual o nó será utilizado. Por exemplo, se um nó *grid* desempenhará o papel de um coletor de dados, sendo que ele estará em execução em um elemento da rede gerenciada, ele deve ser provido com conhecimento e habilidades necessárias para que ele possa realizar estas atividades. Se o nó for usado para realizar atividades de análise de informações, ele deverá ter o conhecimento inicial necessário para realizar as atividades para as quais ele se destina. Estas informações iniciais ditam o comportamento que o nó terá no *grid*.

O elemento raiz de um arquivo de definição é o elemento `<grid>`. O elemento *grid* de um documento XML de definição pode ser composto por uma série de outros elementos que serão descritos a seguir. O DTD destes arquivos pode ser visto no Anexo I deste trabalho.

- O documento pode conter um elemento *DEBUG*. Este elemento é composto por um atributo chamado *action* que especifica a ação de *debug* que deve ser realizada, e por um segundo atributo chamado *name* que contém o valor para a ação a ser executada. Esta opção é usada para testes.
- Os elementos *classpath* informam caminhos de pacotes que a aplicação deve usar como caminho para encontrar as classes requeridas pela aplicação.
- Podem ser inseridos vários elementos *trace* nos arquivos de definição que têm a mesma função dos vários itens que podem ser passados com o parâmetro *trace* no momento de iniciação do *grid*. Quando o *trace* para um elemento é ativado, são exibidas mensagens para depuração deste item em específico.
- O elemento *communicator* é usado para inserir informações a serem usadas pelo módulo de comunicação do nó que está carregando o arquivo de definição.
- Um elemento *grid* pode ser composto de um ou vários elementos *node*. Neste caso, um documento pode conter a definição necessária para mais de um nó *grid*. Os elementos *kdb* (base de conhecimento), *execute* (objetivos) e *schedule* (agendamento de tarefas) podem existir nos elementos *nodes* representando o conhecimento e objetivos globais do sistema, ou dentro do elemento agente, onde representam o conhecimento e objetivos específicos do agente. Estes elementos podem aparecer também no contexto do elemento *grid*, onde

representam dados que devem ser carregados por qualquer nó que carregue o arquivo de definição.

- O elemento *agent* representa um agente a ser criado pelo sistema. Os elementos *kdb*, *execute* e *schedule* representam a base de conhecimento, objetivos e tarefas agendadas respectivamente, conforme destacado anteriormente.
- Também são fornecidos os parâmetros de configuração do medidor de desempenho utilizado pelo nó *grid*. Tais parâmetros são fornecidos no elemento *meter*.

Um exemplo de um arquivo de definição usado por um agente de análise simples usado para testes durante o desenvolvimento pode ser visto no Anexo II deste trabalho.

6.7 Medidores de Desempenho

Os medidores de desempenho são componentes que foram desenvolvidos com o objetivo de coletar informações do recurso úteis na realização da distribuição de carga de trabalho das atividades de análise, quando estas se fizerem necessárias. Um dos objetivos deste trabalho é proceder com a implementação de um protótipo da arquitetura proposta e com isso, para que a proposta de distribuição da carga de trabalho das atividades de análise de informações usando recursos ociosos ou menos utilizados na rede possa ser feita é necessário existir uma forma de medir quando e quanto os recursos disponíveis estão sendo usados.

Conforme descrito, quando um nó *grid* é iniciado pode ser informado o medidor de desempenho que será utilizado para extrair as informações. Esta possibilidade de escolher qual o medidor a ser usado é provida com a intenção de proporcionar uma flexibilidade em poder optar por um medidor de desempenho ou outro. Isto significa que podemos ter um medidor que extrai as informações do recurso usando o protocolo SNMP, por exemplo. Em uma estação que não tem suporte ao protocolo SNMP e que deseja ser inserida no *grid*, podemos optar por um medidor de desempenho que extrai as informações usando outro pacote de instrumentação ou utilitários.

Durante o desenvolvimento deste trabalho, para os testes, foi desenvolvido um medidor que obtém as informações estatísticas usando o protocolo SNMP e a biblioteca

Linpack* [79]. As informações sobre capacidade e utilização da memória, dos dispositivos de armazenamento e do processador são obtidas através da HOST-RESOURCES-MIB [64] dos equipamentos envolvidos.

No caso das análises de dados de gerenciamento, quando um agente possui um conjunto de atividades de análise que ele não é capaz de realizar sozinho, ele pode requerer que agentes que estão em execução em recursos com melhores condições de recursos realizem algumas destas atividades. No caso do *grid* de agentes, um agente residente em um *container* pode perguntar aos agentes de outros *containers* aos quais ele esteja ligado quais agentes estão aptos a realizarem tal atividade, ou seja, quais possuem os recursos para tal.

Vale lembrar que a ligação entre as estruturas lógicas da base de conhecimento dos agentes e o que é código que deve ser executado deve ser feita através do uso de *skills* e, é por isso que os medidores de desempenho também devem ser implementações da interface Java que define uma *skill*. Um nó *grid* ao ser iniciado já possui em sua base de conhecimento regras que fazem o mapeamento das solicitações para o medidor de desempenho. Para exemplificar como isto é feito é apresentado um cenário onde dois nós *grid* foram iniciados e um deles iniciou um medidor de desempenho SNMP durante a etapa de iniciação do nó. O segundo nó precisa obter as informações de capacidade de armazenamento e utilização deste nó com medidor. Um exemplo de regras usadas pelo agente para obter as informações de utilização dos recursos de armazenamento é apresentado na Figura 35.

```
<node name="node-1">
  <kdb>
    [...]
    printDiskUsage :-
      receiver(Receiver),
      agent(query-ref,Receiver,meter("storage-total",Total)),
      agent(query-ref,Receiver,meter("storage-usage",Usage)),
      print(["\nTotal=",Total,"\nUsage",Usage]).
  </kdb>
  <execute>
    printDiskUsage().
  </execute>
</node>
```

Figura 35 – Regras para um agente obter e exibir informações de armazenamento.

* O Linpack Benchmark é um teste que tem sido usado durante anos para medir o desempenho de computadores na realização de operações de ponto flutuante.

6.8 Os Agentes Coletores

Os agentes coletores constituem as entidades responsáveis por interagir com os dispositivos gerenciados, coletando informações ou configurando parâmetros nos recursos. São a primeira etapa da arquitetura e conforme apresentado no Capítulo 5 que descreve a arquitetura, estes agentes são providos de ‘interfaces’ que lhes permitem obter as informações desejadas usando os protocolos de gerenciamento ou outras ferramentas. Durante o desenvolvimento do trabalho foram desenvolvidos agentes coletores com habilidades SNMP e linha de comando. Como um aprimoramento dos agentes coletores de linha de comando foram desenvolvidos agentes com habilidades para extrair informações através da instrumentação **WMI** das plataformas Microsoft Windows [84].

Os agentes coletores são agentes simples e reativos. Eles podem apenas reagir a situações do ambiente e não precisam guardar nenhum estado com relação ao seu passado ou entrar em um estado de deliberação que lhe permita traçar planos para atingir um estado futuro. Eles são compostos de objetivos que podem consistir, por exemplo, em coletar as informações de um conjunto de objetos gerenciados de um dispositivo em particular. Desta forma, os agentes simplesmente tentam perseguir os seus objetivos de coleta sem levar em consideração o estado do ambiente ou os possíveis estados futuros.

Eles também são responsáveis por criar uma representação dos dados em um formato comum. Este formato comum adotado consiste de documentos XML que irão conter os dados coletados pelos agentes. Desta forma, tanto os agentes coletores SNMP como os agentes de linha de comando apresentam ao *grid* de armazenamento as informações no mesmo formato.

O diagrama de seqüência apresentado na Figura 36 mostra de maneira simples como ocorrem as etapas de coleta, de *parser* dos dados e de construção do documento resultante enviado ao *grid* de armazenamento de dados.

As mensagens enviadas pelos agentes coletores aos agentes que irão proceder com o armazenamento dos dados seguem o padrão de mensagens estabelecido pela **FIPA**. Os dados coletados são representados usando documentos XML. A conformidade com tais padrões garante que os dados possam ser interpretados por agentes construídos usando outra plataforma de agentes. Os agentes coletores não precisam ser construídos

necessariamente em Java e usando o *Agentlight*. Seguindo o padrão de mensagens **FIPA** e o modelo de representação de dados, agentes construídos em uma outra linguagem como *perl* poderiam ser utilizados.

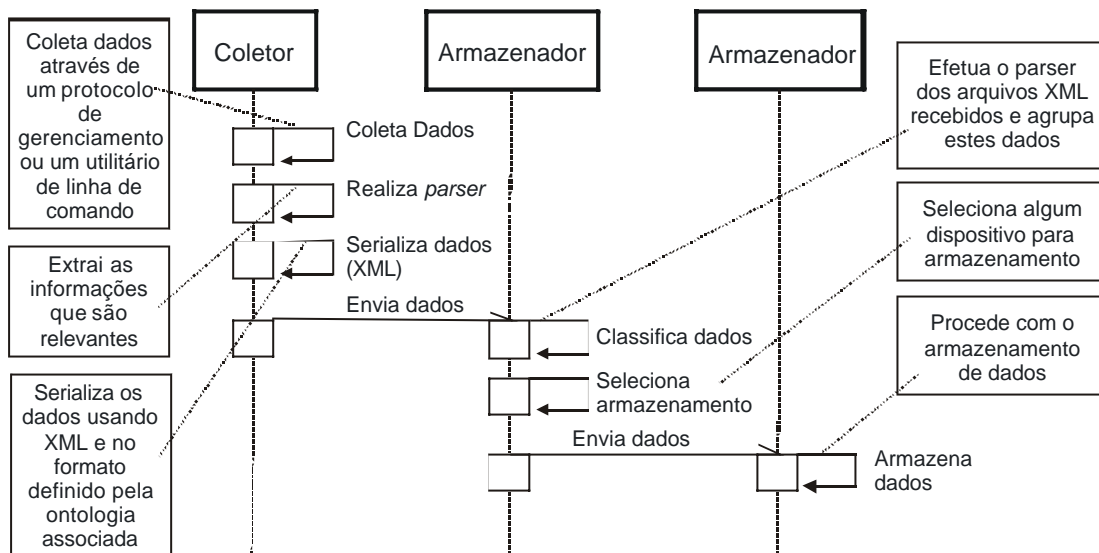


Figura 36 - Coleta e armazenamento de dados.

6.8.1 Agentes Coletores SNMP

Conforme comentado anteriormente para a ligação das estruturas lógicas da base de conhecimento e do código Java são utilizadas *skills*. Durante o desenvolvimento deste trabalho, utilizou-se uma biblioteca SNMP para realizar a coleta de informações de gerenciamento. Esta biblioteca suporta a versão 1 e 2 do protocolo SNMP. Ela foi remodelada e adaptada às condições de uso do projeto, que era possuir um código reduzido, que fosse de fácil utilização e fácil de ser compreendida. Embora suporte a versão 2 do protocolo SNMP, esta biblioteca implementa apenas as primitivas GET, GET-NEXT e SET do SNMP, que são suficientes para atender as necessidades do projeto.

Tabela 4 - Ações executadas pela SNMPSkill.

Comando	Descrição	Sintaxe	Exemplo
connection	Inicia uma conexão* com a entidade par de quem os dados serão coletados.	skill(connection, Conn, host, community)	snmp(connection, Conn, "127.0.0.1", "public")

* Na realidade não acontece uma conexão, pois o protocolo SNMP utiliza o protocolo de transporte UDP, o qual não é orientado por conexão. Porém, a visão de conexão é usada para apresentar uma forma de relacionamento com a entidade par de onde serão coletados os dados.

Get	Extraí o valor de um objeto gerenciado, ou de uma lista de objetos.	skill(get, Conn, OID, Var) ou skill(get, Conn, List, Vars)	snmp(get, Conn, "1.3.6...", Var) snmp(get, Conn, List, Vars)
get_next	Obtém o valor de um ou de uma lista de objetos lexicograficamente após os identificadores fornecidos	skill(get-next, Conn, OID, Var) ou skill(get-next, Conn, List, Vars)	snmp(get_next, Conn, "1.3.6...", Var) snmp(get_next, Conn, List, Vars)
Set	Configura o valor de um objeto.	skill(set, Conn, OID, Value)	snmp(set, Conn, "1.3.6...", "2")
oid	Retorna a oid de uma variável ligada do SNMP.	skill(oid, Varbind, String)	snmp(oid, Varbind, "1.3.6...")
value	Retorna o valor de uma variável ligada do SNMP	skill(value, Varbind, String)	snmp(value, Varbind, "Laborat...")

Contudo, para que as funcionalidades desta biblioteca possam ser utilizadas pelos agentes do *grid* foi construída a *skill* que fará a ligação com as estruturas lógicas da base de conhecimento dos agentes. Esta *skill* foi nomeada de *SNMPSkill*. Não faz muito sentido apresentar neste documento o seu código fonte, porém é apresentada uma visão básica de como ela funciona e em seguida são apresentados alguns exemplos a respeito de sua utilização.

Quando invocada pela máquina de inferência, uma *skill* recebe uma lista de termos que são os termos passados para a regra que corresponde ao objetivo a ser alcançado. Como padrão, optou-se por usar o primeiro termo desta lista para denotar a ação que deve ser realizada pela *skill*, chamado de comando, sendo que os demais termos correspondem aos parâmetros ou valores de retorno resultantes da unificação. Sendo assim, na Tabela 4 é apresentada uma lista de funcionalidades providas pela *SNMPSkill*.

Possuindo a infra-estrutura proporcionada pelo *grid* de agentes e com esta *skill* SNMP implementada, para a construção dos agentes coletores SNMP, é necessária a construção de regras de coleta. Estas regras são construídas e inseridas nos documentos XML de definição que são carregados pelo nó *grid* no momento em que ele é iniciado. São apresentados alguns exemplos de agentes coletores e regras de coleta de informações SNMP. Na Figura 37 são apresentados trechos dos fatos e regras de um agente coletor de dados SNMP. Neste trecho primeiramente são adicionadas regras para fazer um mapeamento para a *skill* SNMP correspondente. Existem alguns fatos usados para associar os nomes dos objetos gerenciados com os seus identificadores correspondentes, com o objetivo de aumentar a clareza das regras. Em seguida existem regras comuns usadas pelos agentes coletores.

```

<kdb>
  snmp(Cmd,Arg1,Arg2) :-
    !br.ufsc.lrg.agentgrid.skill.SNMPSkill(Cmd,Arg1,Arg2).
  snmp(Cmd,Arg1,Arg2,Arg3) :-
    !br.ufsc.lrg.agentgrid.skill.SNMPSkill(Cmd,Arg1,Arg2,Arg3).
  [...]
  mib("ifIndex","1.3.6.1.2.1.2.2.1.1").
  mib("ifDescr","1.3.6.1.2.1.2.2.1.2").
  mib("ifType","1.3.6.1.2.1.2.2.1.3").
  mib("ifMtu","1.3.6.1.2.1.2.2.1.4").
  mib("ifSpeed","1.3.6.1.2.1.2.2.1.5").
  [...]
  snmpConnection(Address,Community,Connection) :-
    snmp(connection,Connection,Address,Community),!.
  snmpGet(Connection,OIDList,Result) :-
    snmp(get,Connection,OIDList,Result),!.
  snmpGetNext(Connection,OIDList,Result) :-
    snmp(get_next,Connection,OIDList,Result),!.
  snmpSet(Connection,OID,Value) :-
    snmp(set,Connection,OID,Value),!.
  snmpOid(Varbind,OID) :-
    snmp(oid,Varbind,OID),!.
  snmpValue(Varbind,Value) :-
    snmp(value,Varbind,Value),!.
</kdb>

```

Figura 37 – Exemplo de regras de um agente coletor SNMP.

Estas regras são regras gerais usadas pelos agentes coletores para a construção de regras mais elaboradas e a conseqüente realização das tarefas de coleta. Um nó pode carregar diversos documentos XML de definição contendo as regras necessárias ao seu funcionamento. Neste caso o nó pode carregar um documento contendo as regras apresentadas na Figura 37 e outros documentos contendo outras regras, criadas usando as regras contidas nos documentos carregados anteriormente. No fim todas estas regras serão inseridas na base de conhecimento dos agentes coletores. Na Figura 38 temos um segundo exemplo de regras. Estas regras possibilitam que um agente coletor possa obter as instâncias de interfaces de rede existentes no equipamento gerenciado e obter características das interfaces de rede.

```

getInstances(Connection,OID,ResultList) :-
  list(create,ResultList,_),
  snmpGetNext(Connection,OID,Varbind),
  snmpOid(Varbind,LastOID),
  getUntilDiffer(Connection,Varbind,OID,LastOID,ResultList),!.
getUntilDiffer(Connection,Varbind,OID,LastOID,ResultList) :-
  string(startswith,LastOID,OID),
  snmpValue(Varbind,Value),
  list(add,ResultList,Value),
  snmpGetNext(Connection,LastOID,Varbind2),
  snmpOid(Varbind2,OIDCollected),
  getUntilDiffer(Connection,Varbind2,OID,OIDCollected,ResultList).
getUntilDiffer(Connection,Varbind,OID,LastOID,ResultList) :- !.
snmpExtract('interfaces','config',Connection,TreeResult) :-
  mib("ifIndex",OIDifIndex),!,
  getInstances(Connection,OIDifIndex,Instances),
  getIfConfig(Connection,Instances,TreeResult),!.

```

```

getIfConfig(Connection,[],TreeResult) :- !.
getIfConfig(Connection,Instances,TreeResult) :-
    list(head,Instances,Instance),
    list(tail,Instances,RestInstances),
    [...],!.
completeOID(OID,Instance,Joined) :-
    string(join,[OID,".",Instance],Joined),!.
snmpGetValue(Connection,OIDList,Result) :-
    snmpGet(Connection,OIDList,Varbind),
    snmpValue(Varbind,Result),!.
monitor(What,KindInfo,Connection,TreeResult) :-
    getId(OS,IP,TIME,HostName,Locale),
    buildId(OS,IP,TIME,HostName,TreeResult,KindInfo),
    snmpExtract(What,KindInfo,Connection,TreeResult),
    sendToStore(Tree),!.

```

Figura 38 – Regras para um agente coletor obter informações de interfaces.

Deve ser destacado que as regras apresentadas têm o objetivo de apresentar como funciona um agente SNMP. Algumas regras não foram inseridas devido à limitação de espaço e também com o objetivo de aumentar a clareza do código. Embora tenham sido definidas as regras para o agente coletor poder coletar as informações SNMP, não foram definidos os objetivos do agente. Ou seja, o que ele deve coletar e de quais elementos gerenciados da rede. Na Figura 39 é apresentado um exemplo de arquivo de definição apresentando como tais objetivos de coleta podem ser especificados. Neste exemplo, as informações das interfaces do equipamento onde o agente está sendo executado são recuperadas a cada 60 segundos. O elemento *<schedule>* é usado para agendar a execução de um objetivo a cada 60 segundos. Vale ressaltar que os objetivos, assim como em qualquer mecanismo de inferência Prolog, são especificados através de cláusulas que devem ser buscadas na base de conhecimento.

```

<?xml version="1.0"?>
<grid>
  <DEBUG action="echo" name="=== snmp-test v0.1 ==="/>
  <agent name="collector-1">
    <kdb>
      equipment('127.0.0.1','public').
      netConfig :-
        equipment(Address,Community),
        snmpConnection(Address,Community,Connection),
        monitor('interfaces','config',Connection,TreeResult),!.
    </kdb>
    <schedule name="collectNet" time="60">
      netConfig().
    </schedule>
  </agent>
</grid>

```

Figura 39 – Regras e objetivo a ser atingido por um agente coletor SNMP.

O agente coletor SNMP envia estas informações para o armazenador de dados depois de realizar a coleta e montar a árvore com os dados coletados. Neste caso os

agentes coletores enviam as informações para realizar o armazenamento tão logo elas tenham sido coletadas e os documentos resultantes tenham sido montados. Contudo, no gerenciamento de um *site* geograficamente distante estas informações podem ser armazenadas em um repositório temporário, e o seu transporte é realizado em grandes lotes com um volume maior de dados de gerência. Pode existir um nó com agentes cujos objetivos consistem em enviar estas informações para o armazenamento. A árvore de dados resultante da coleta exemplificada na Figura 39 é apresentada na Figura 40. O formato apresentado pelo documento de resultado consiste basicamente de: um elemento de identificação, que contém o número IP do dispositivo de onde os dados foram coletados, a hora em que a coleta foi realizada e o nome da estação onde o agente estava sendo executado no momento em que fez a coleta de dados; e pelos blocos com os dados correspondentes resultantes da coleta de dados. No exemplo apresentado, o dispositivo possui duas interfaces de rede na sua tabela de interfaces, sendo que uma é *loop* local e a segunda é a interface propriamente dita cujo índice corresponde ao número 65539.

```
<config>
  <id>
    <ip>10.0.9.53</ip>
    <time>1070402198557</time>
    <from>C104</from>
  </id>
  <interface index="1">
    <ifDescr>MS TCP Loopback interface </ifDescr>
    <ifType>24</ifType>
    <ifMtu>1520</ifMtu>
    <ifSpeed>10000000</ifSpeed>
    <ifPhysAddress></ifPhysAddress>
    <ifLastChange>0</ifLastChange>
  </interface>
  <interface index="65539">
    <ifDescr>Realtek RTL8139 Family PCI Fast Ethernet NIC #2 </ifDescr>
    <ifType>6</ifType>
    <ifMtu>1500</ifMtu>
    <ifSpeed>10000000</ifSpeed>
    <ifPhysAddress>00-E0-7D-9B-07-97</ifPhysAddress>
    <ifLastChange>0</ifLastChange>
  </interface>
</config>
```

Figura 40 – Árvore resultante de coleta de dados por um agente SNMP.

O formato de documento XML adotado como resultado das atividades de coleta de dados pode não ser o mais adequado. Porém, ele é simples e para os princípios deste trabalho, que é apresentar resultados de implementação e como a arquitetura pode ser utilizada em uma situação prática de gerência, ele atende a necessidade. Conforme

discutida anteriormente a transferência dos dados coletados é feita por meio de mensagens FIPA-ACL transportadas por meio do protocolo HTTP.

6.8.2 Agentes Coletores de Linha de Comando

Os agentes coletores de linha de comando extraem as informações de gerenciamento através da execução de utilitários de linha de comando disponíveis nos diversos sistemas operacionais onde os agentes podem ser executados. No ambiente UNIX os utilitários de linha de comando podem consistir de comandos como *netstat*, *top*, *df*, *free*, *ifconfig*, dentre outros. A execução destes comandos retorna um resultado do qual as informações relevantes devem ser extraídas. Para esta categoria de agentes foram desenvolvidas *skills* que permitem a execução de um comando na console do sistema operacional e a obtenção do resultado desta execução. Também foi desenvolvida uma *skill* que, através do uso de expressões regulares, permite que os dados relevantes sejam extraídos facilmente do resultado da execução destes comandos. Esta biblioteca utilizada pelos agentes de linha de comando é provida pela equipe de desenvolvimento Apache [7] sob o nome de Apache Regexp [6].

A *skill* responsável pela execução dos comandos no sistema operacional tem o nome de *CmdLineExecuterSkill* e não utiliza o conceito de comandos apresentado na *skill* SNMP em virtude de não existir esta necessidade, pois a única função exercida por ela é a execução de comandos no sistema operacional e a obtenção de seu resultado. A funcionalidade desta *skill* é apresentada na Tabela 5.

Tabela 5 - Ações executadas pela *CmdLineExecuterSkill*.

Descrição	Sintaxe	Exemplo
Executa um comando no sistema operacional e usa a lista de parâmetros fornecida	<code>skill(Command, ListParameters, Result)</code>	<code>skill(netstat, [-s], Result)</code>
Executa um comando, mas a lista de parâmetros é fornecida na mesma linha do comando a ser executado.	<code>skill(Command, Result)</code>	<code>skill('netstat -s', Result)</code>

A execução de comandos através do uso desta *skill* retorna um fluxo de bytes do qual são extraídos os dados relevantes. A *skill* que encapsula as funcionalidades providas pela biblioteca usada para a extração de dados usando expressões regulares é a *RegExpSkill*. Um resumo de suas funções é apresentado na Tabela 6.

Tabela 6 - Ações executadas pela RegExpSkill.

Comando	Descrição	Sintaxe	Exemplo
create	Cria um objeto expressão regular a ser utilizado	skill(create, RegExp, String)	regexp(create, RegExp, "(+)\s*=\s*(d+)")
match	Encontra um padrão dentro de uma determinada <i>string</i> ou fluxo de bytes e retorna uma lista de elementos.	skill(match, [RE REString], String, Result,[Index])	regexp(match, RE,"key=value",Result)
match-add	Encontra um padrão dentro de uma determinada <i>string</i> ou fluxo de bytes e adiciona em uma lista os elementos.	skill(match-add, [RE REString], String, Result,[Index])	regexp(match-add, RE,"key=value",Result)

Assim como no caso dos agentes SNMP estas *skills*, juntamente com a estrutura proporcionada pelo *grid* de agentes, fornecem as funcionalidades necessárias para a construção de agentes de coleta com a flexibilidade e as características requeridas por uma aplicação de gerenciamento de sistemas. Para a compreensão a respeito de como estas funcionalidades podem ser usadas para a construção dos agentes de coleta de linha de comando, serão apresentados alguns exemplos de regras utilizadas por agentes que usam deste princípio para coletar as informações de gerenciamento. Assim como no caso do exemplo de agente coletor SNMP apresentado, serão apresentadas as regras gerais e em seguida regras mais específicas e os objetivos dos agentes. Dentre outras, as regras gerais para que um agente coletor de linha possa ser usado são apresentadas na Figura 41. No início deste exemplo existem as regras responsáveis por fazer um mapeamento para as *skills* correspondentes. As demais regras são usadas para executar um comando e obter o seu resultado. Podemos perceber que neste caso, a última regra ao ser invocada tem em seu corpo um sub-objetivo chamado *extract* que será uma regra que possui em seu corpo o comando a ser executado e o que deve ser extraído desta execução. Esta idéia é mostrada na Figura 42 que apresenta um conjunto de regras necessárias para que o agente execute o comando *'netstat -s'* e extraia estatísticas do protocolo IP.

```
<kdb>
[... ]
regexp(Cmd,Arg1,Arg2,Arg3) :-
    !br.ufsc.lrg.agentgrid.skill.RegExpSkill(Cmd,Arg1,Arg2,Arg3).
cmdline(Cmd,Arg1,Arg2) :-
    !br.ufsc.lrg.agentgrid.skill.CmdLineExecuterSkill(Cmd,Arg1,Arg2).
[... ]
appendValues(Tree,[],[]) :- !.

appendValues(Tree,Template,Values) :-
```

```

list(head,Template,TBranch),
list(head,Values,Value),
tree(add,Tree,TBranch,Value),
list(tail,Template,NextTemplates),
list(tail,Values,NextValues),
appendValues(Tree,NextTemplates,NextValues).

retrieve(What,Type) :-
  getId(OS,IP,TIME,HostName,Locale),
  extract(OS,Locale,What,SubTree),
  buildId(OS,IP,TIME,HostName,Tree,Type),
  tree(add,Tree,SubTree),
  sendToStore(Tree).
[...]
```

Figura 41 – Regras gerais de um agente de linha de comando.

```

<kdb>
extract("Windows XP","en_US",'ipv4Stat',Tree) :-
  cmdline(netstat,['-s'],CmdResult),
  regexp(match-add,'Packets Received\\s*=\\s*(\\d+)\\r',
          CmdResult,Result),
  regexp(match-add,'Received Header Errors\\s*=\\s*(\\d+)\\r',
          CmdResult,Result),
  regexp(match-add,'Received Address Errors\\s*=\\s*(\\d+)\\r',
          CmdResult,Result),
  [...],
  tree(create,Tree,ipv4,['index','1']),
  appendValues(Tree,['PacketsReceived','ReceivedHeaderErrors',
                    'ReceivedAddressErrors','DatagramsForwarded',
                    'UnknownProtocolsReceived','ReceivedPacketsDiscarded',
                    'ReceivedPacketsDelivered','OutputRequests',
                    'RoutingDiscards','DiscardedOutputPackets',
                    'OutputPacketNoRoute','ReassemblyRequired',
                    'ReassemblySuccessful','ReassemblyFailures',
                    'DatagramsSuccessfullyFragmented',
                    'DatagramsFailingFragmentation','FragmentsCreated'],
              Result).

ipv4Stat :- retrieve('ipv4Stat','data').

</kdb>
```

Figura 42 – Regras para coletor extrair informações usando ‘netstat’.

Não existe a necessidade de apresentar os objetivos de um agente deste tipo em virtude deles se assemelharem aos objetivos de um agente SNMP. No exemplo apresentado imaginemos que um agente possui como objetivos chamar a regra de coleta passando os devidos parâmetros em intervalos de 360 segundos. Assim como no caso do agente SNMP, o arquivo resultante da coleta pode ser armazenado em um repositório temporário, ou ser enviado para os agentes que irão armazená-lo. Um exemplo de arquivo resultante da execução do comando pelo agente descrito anteriormente pode ser visto na Figura 43.

```

<data>
<id>
<ip>150.162.63.4</ip>
```



```

<time>1070449019156</time>
<from>blue</from>
</id>
<ipv4 index="1">
  <PacketsReceived>36620</PacketsReceived>
  <ReceivedHeaderErrors>0</ReceivedHeaderErrors>
  <ReceivedAddressErrors>6</ReceivedAddressErrors>
  <DatagramsForwarded>0</DatagramsForwarded>
  <UnknownProtocolsReceived>0</UnknownProtocolsReceived>
  <ReceivedPacketsDiscarded>0</ReceivedPacketsDiscarded>
  <ReceivedPacketsDelivered>36620</ReceivedPacketsDelivered>
  <OutputRequests>23818</OutputRequests>
  <RoutingDiscards>0</RoutingDiscards>
  <DiscardedOutputPackets>0</DiscardedOutputPackets>
  <OutputPacketNoRoute>0</OutputPacketNoRoute>
  <ReassemblyRequired>0</ReassemblyRequired>
  <ReassemblySuccessful>0</ReassemblySuccessful>
  <ReassemblyFailures>0</ReassemblyFailures>
  <DatagramsSuccessfullyFragmented>0</DatagramsSuccessfullyFragmented>
  <DatagramsFailingFragmentation>0</DatagramsFailingFragmentation>
  <FragmentsCreated>0</FragmentsCreated>
</ipv4>
</data>

```

Figura 43 – Resultado da coleta de um agente de linha de comando.

6.8.3 Agentes Coletores WMI

Os agentes coletores WMI são extensões dos agentes coletores de linha de comando. Estes agentes extraem dados necessários às atividades de gerenciamento usando o pacote de instrumentação WMI proporcionado pela plataforma Microsoft Windows [84]. Para isso estes agentes executam *scripts*, desenvolvidos em Visual Basic Script, que são responsáveis por interagir com o gerenciador de objetos do WMI e por recuperar estes dados. Quando passados como parâmetros de utilitários existentes para a execução destes *scripts*, eles apresentam o resultado na console do sistema operacional, como se fosse executado um comando do sistema. Estes resultados são usados pelos agentes que realizam o mesmo processo que um agente de linha de comando para extrair estas informações. Em seguida o agente efetua a extração das informações relevantes do resultado da mesma forma que é feito com os agentes de linha de comando. A Figura 44 ilustra de maneira abstrata como este processo acontece, onde primeiramente o *script* é executado (1), posteriormente o resultado desta execução é tratado (2) e os dados são formatados em XML (3).

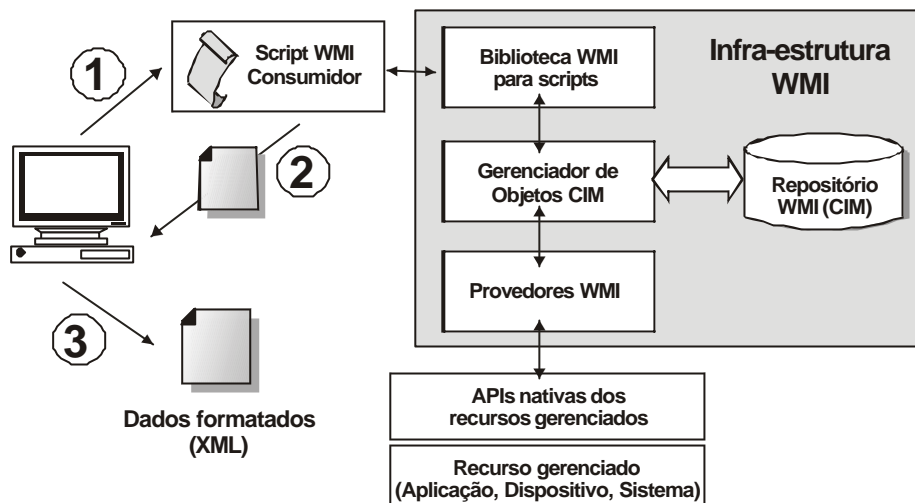


Figura 44- Funcionalidade de um agente coletor WMI.

As regras para funcionamento dos agentes coletores WMI são semelhantes as usados pelos agentes de linha de comando. Porém, no caso destes primeiros agentes, deve ser informado o *script* que ele deve executar durante a etapa de coleta de dados, sendo que este *script* deve ser previamente construído. Durante o desenvolvimento deste trabalho foram desenvolvidos *scripts* para a coleta de informações de configuração e desempenho a respeito de processadores, memória, periféricos e outros. Um exemplo de um agente que coleta informações de desempenho do processador é apresentado na Figura 45.

```
<kdb>
```

```
[...]
```

```
cmdScript(Directory,WMIScript,CmdResult) :-
  string(join,[Directory,WMIScript],Joined),
  cmdline(cscript,[Joined],CmdResult).
```

```
wmiRetrieve(What,Type) :-
  getId(OS,IP,TIME,HostName,Locale),
  wmiExtract(OS,Locale,What,SubTree),
  buildId(OS,IP,TIME,HostName,Tree,Type),
  tree(add,Tree,SubTree),
  sendToStore(Tree).
```

```
wmiExtract("Windows XP","pt_BR",processorStat,Tree) :-
  scriptDirectory(Directory),
  cmdScript(Directory,'\\Win32_Processor.vbs',CmdResult),
  regexp(match-add,'Availability:\\s*(\\d+)\\r',CmdResult,Result),
  regexp(match-add,'CpuStatus:\\s*(\\d+)\\r',CmdResult,Result),
  regexp(match-add,'CurrentClockSpeed:\\s*(\\d+)\\r',CmdResult,Result),
  regexp(match-add,'LoadPercentage:\\s*(\\d+)\\r',CmdResult,Result),
  regexp(match-add,'Status:\\s*(.+)\\r',CmdResult,Result),
  tree(create,Tree,processor,['index','1']),
  appendValues(Tree,['Availability','CPUStatus','CurrentClockSpeed',
    'LoadPercentage','Status'],Result).
```

```
processorStat :- wmiRetrieve(processorStat,'data').
</kdb>
```

Figura 45 – Agente WMI para a coleta de informações do processador.

A definição não difere muito dos agentes de linha de comando, com a diferença de que deve ser informado onde o agente encontrará o *script* correspondente a ser executado.

6.9 Os Agentes do *Grid* de Armazenamento

Os agentes armazenadores possuem o conhecimento necessário para proceder com o armazenamento dos dados coletados pelos agentes coletores. Nesta parte do *grid* de gerência podem existir também agentes responsáveis por classificar as informações coletadas. Ou seja, estes agentes agrupam as informações por meio de suas afinidades e tentam armazenar no lugar mais adequado e da melhor maneira possível, facilitando a recuperação dos dados e eventualmente procedendo com uma possível replicação dos mesmos.

Da mesma forma que os agentes coletores, para que tais agentes possam proceder com o armazenamento dos dados, eles precisam de “interfaces” que lhes permitam interagir com os dispositivos de armazenamento necessários. No caso de um agente que interage com um banco de dados relacional, ele pode ter uma interface JDBC que lhe permita armazenar e recuperar os dados de um banco de dados relacional. A exemplo do que acontece com os agentes coletores quando interagem com os dispositivos usando um protocolo de gerenciamento, os agentes coletores precisam ter implementações de *SkillInterface* que lhes dêem todas as funcionalidades de que precisam para utilizar os meios de armazenamento disponíveis. Os agentes armazenadores que armazenam dados em bancos de dados relacionais usam uma *skill* chamada de *DatabaseSkill* responsável por encapsular uma classe que é uma implementação de uma interface que contém os métodos necessários para inserir, apagar e consultar fatos na base de dados. Vale lembrar que o armazenamento de dados é feito como se fatos e regras fossem inseridos na base de conhecimento do agente armazenador, embora estes fatos e regras estejam sendo armazenados de forma persistente no banco de dados. Esta abordagem permite uma abstração no que se refere ao mecanismo de armazenamento utilizado, uma vez que o armazenamento e recuperação acontecem de maneira transparente através da

manipulação de fatos e regras expressos em lógica de primeira ordem. O conhecimento necessário para que este tipo de agente possa realizar as suas tarefas também é carregado de documentos XML que contém os fatos e regras iniciais necessárias para o seu funcionamento.

Os itens a serem explorados nesta modalidade do *grid* estão relacionados ao balanceamento de carga no que se refere ao armazenamento dos dados, aproveitando a disponibilidade dos recursos de armazenamento das estações que compõem este *grid*, a replicação dos dados com o objetivo de prover uma maior agilidade na recuperação dos mesmos e evitar que eles estejam indisponíveis em virtude do *grid*, ser um ambiente dinâmico, onde novos participantes podem entrar no *grid* e existentes podem deixá-lo. Neste caso é importante a escolha de uma forma de armazenamento padronizada, baseada em XML, e de mecanismos que permitam a catalogação destes dados.

Outra atividade de agentes do *grid* de armazenamento é proporcionar o que pode ser chamado de ‘envelhecimento dos dados’. Ou seja, existem informações que não precisam ficar eternamente disponíveis e que podem ser retiradas das tabelas de dados principais e armazenadas em locais de onde não seja necessária uma recuperação freqüente. Ainda, tais informações podem também ser removidas quando a sua utilização não seja mais necessária.

O *grid* de análise dos dados de gerência não trata diretamente da recuperação dos dados de gerenciamento armazenados. Eles solicitam as informações ao *grid* de armazenamento que, por sua vez, retorna as informações necessárias. Uma estrutura onde exista replicação com o objetivo de agilizar a recuperação dos dados faz sentido onde o volume de informação é muito grande e requer cuidados especiais no seu armazenamento.

No início do desenvolvimento deste trabalho foram desenvolvidas *skills* para a manipulação de bancos de dados relacionais. Estas *skills* eram usadas como uma forma de mapear as requisições de armazenamento expressas através de regras para as instruções SQL de inserção e consulta dos dados, além de outras operações como geração de médias de valores. Um exemplo de como estas regras eram expressas pode ser visto na Figura 46. Porém, o tempo e o uso destas funcionalidades se mostrou não ser eficiente, pois além de quebrar um pouco da estrutura lógica com as regras que eram construídas, teriam que ser construídas regras diferentes para cada tipo de

armazenamento. Ou seja, se um agente mudasse a forma de armazenamento de um banco de dados para arquivos de texto, todas as regras devem ser alteradas. Por fim, optou-se por deixar estas diferenças fora das regras utilizadas pelos agentes armazenadores, deixando-as similares para os diferentes tipos de armazenamento e adotando um esquema onde o armazenamento ocorre como se o agente estivesse manipulando fatos em sua base de conhecimento. A ligação com os recursos de armazenamento é feita através de uma camada de *software* que dá a percepção de que o recurso de armazenamento é uma extensão da base de conhecimento do agente.

```
[...]
sqlPrepare(Connection,Statement,Result) :-
    storedb(preparation,Connection,Statement,Result).
sqlExecute(Values,Prepared,Result) :-
    storedb(execute,Values,Prepared,Result).
[...]
dbConnect(Connection) :- dbDriver(Driver),
    dbUrlconn(URL),dbRegister(Driver),
    dbOpen(Connection,URL).
insertTree(Connection,Tree) :- storedb(insert,Connection,Tree).
storeTree(Tree) :- dbConnect(Connection),
    tree(string,Tree,String,_),
    insertTree(Connection,Tree),
    dbClose(Connection).
```

Figura 46- Trecho usadas anteriormente pelos agentes armazenadores.

Atualmente os dados são recebidos em um formato XML como o descrito na Figura 47. Este formato é simples e para os propósitos iniciais do *grid* de agentes, que é prover um protótipo funcional o modelo ele é satisfatório. Procurou-se manter este formato o mais simples possível em um primeiro estágio para não dificultar as etapas de desenvolvimento. O elemento raiz deste formato de documento define o tipo de dados que o agente coletor está enviando ao *grid* de armazenamento. Os dados podem ser classificados em dados dinâmicos, que são dados de gerência que foram coletados e que tendem a se alterar no futuro em que forem coletados novamente, e em dados não dinâmicos que são normalmente informações de configurações de equipamentos que não possuem uma alteração freqüente de valores. Deste primeiro tipo de dados é necessário manter um histórico, pois as análises poderão requerer informações armazenadas previamente para as suas atividades. No segundo caso os dados não precisam de um histórico e isso significa que sempre que estas informações são coletadas novamente, elas substituem a cópia mantida pelos agentes de armazenamento. A Figura 47 exhibe um exemplo contendo parte de um arquivo XML que contém informações referentes a um disco de um dispositivo. Estas informações, ao serem

tratadas pelo armazenador substituirão a cópia existente. Na Figura 48 é apresentado um exemplo de informações de desempenho do processador de um determinado recurso. Estas informações serão mantidas em um histórico, pois o elemento raiz deste documento possui um nome que no sistema está configurado para manter histórico.

```
<config>
  <id><ip>150.162.63.3</ip>
  <time>1070540017745</time>
  <from>sirius</from>
</id>
<disk index="1">
  <BytesPerSector>512</BytesPerSector>
  <Caption>Maxtor 32049H2</Caption>
  <InterfaceType>IDE</InterfaceType>
  <Model>Maxtor 32049H2</Model>
  <Partitions>5</Partitions>
  <Size>20415144960</Size>
  <TotalCylinders>2482</TotalCylinders>
  <TotalHeads>255</TotalHeads>
  <TotalSectors>39873330</TotalSectors>
  <TotalTracks>632910</TotalTracks>
  <TracksPerCylinder>255</TracksPerCylinder>
</disk>
</config>
```

Figura 47- Arquivo de configuração gerado por um agente coletor.

Apesar das definições, o que deve ser mantido como histórico e o que não deve, pode ser configurável, permitindo ao usuário do sistema de gerenciamento definir o que pode ser configurável e o que não deve ser.

```
<data>
  <id>
    <ip>150.162.63.3</ip>
    <time>1070540525154</time>
    <from>sirius</from>
  </id>
  <processor index="1"><Availability>3</Availability>
  <CPUStatus>1</CPUStatus>
  <CurrentClockSpeed>801</CurrentClockSpeed>
  <LoadPercentage>0</LoadPercentage>
  <Status>OK</Status>
</processor>
</data>
```

Figura 48- Exemplo de arquivo de dados gerado por um agente coletor.

6.9.1 O Armazenamento de Dados

Para proceder com o armazenamento de dados, foram desenvolvidos agentes com habilidades para manipular bases de dados relacionais e para nossos testes o banco de

dados utilizado foi o MySQL [90], por ser simples, de fácil instalação e utilização. Estes agentes fazem uso da *skill* chamada *DatabaseSkill* e durante a etapa de abertura de conexão, deve ser fornecida a classe que fará a ligação com o recurso de armazenamento. Ou seja, a *skill DatabaseSkill* é genérica para os diversos recursos de armazenamento e implementa primitivas para que fatos e regras sejam inseridas em uma base de dados. Um exemplo prático de como isto acontece pode ser visto na Figura 49. Quando iniciado, o agente estabelece uma conexão com a base de dados informando a classe que será usada para interagir com o recurso de armazenamento, que neste caso é a *org.agentlight.db.JDBCDatabaseInterface*. Depois de estabelecida a conexão, o objeto resultante é armazenado como um fato na base de conhecimento do agente, e recuperado quando alguma operação de armazenamento se faça necessária.

```

<kdb>

db(Cmd,Arg1) :-
    !org.agentlight.skill.DatabaseSkill(Cmd,Arg1).
db(Cmd,Arg1,Arg2) :-
    !org.agentlight.skill.DatabaseSkill(Cmd,Arg1,Arg2).
db(Cmd,Arg1,Arg2,Arg3) :-
    !org.agentlight.skill.DatabaseSkill(Cmd,Arg1,Arg2,Arg3).
assert(Term,DB) :-
    db(assert,DB,Term),!.
delete(Term,DB) :-
    db(delete,DB,Term),!.
query(Term,DB,Set) :-
    db(query,DB,Term,Set),!.
assertInfo(Term) :-
    db(DB),
    assert(Term,DB), !.
connectDB(Driver,String,Prefix,Class) :-
    hash(create,Param,_,_),
    hash(put,Param,driver,Driver),
    hash(put,Param,string,String),
    hash(put,Param,prefix,Prefix),
    db(open,DB,Class,Param),
    assert(db(DB)),!.
</kdb>
<execute>
connectDB("com.mysql.jdbc.Driver",
    "jdbc:mysql://127.0.0.1/DB_TEST?user=usr&password=pass",
    "STORE_AGENT", "org.agentlight.db.JDBCDatabaseInterface").
</execute>

```

Figura 49- Regras comuns usadas por um armazenador.

Quando um agente precisa armazenar, apagar ou recuperar informações, ele utiliza as primitivas *assert*, *delete* e *query* providas por esta *skill* de armazenamento. Os

agentes coletores enviam as mensagens FIPA com ato comunicativo *request** forçando o agente a realizar o armazenamento de um conjunto de dados. Isto significa que o agente coletor pode enviar uma mensagem cujo conteúdo consiste de uma cláusula PROLOG como: *storeTreeCollected(Tree)*, por exemplo, onde *Tree* possui o resultado da atividade de coleta de dados. O agente armazenador, tendo recebido uma mensagem do tipo *request* tentará encontrar em sua base de conhecimento uma cláusula que case com a recebida na mensagem e efetuará a chamada a esta regra que procederá com o armazenamento.

Para a recuperação de dados, as mensagens recebidas pelos agentes de armazenamento devem possuir como ato comunicativo o *query-ref*** . Neste caso, um agente receptor receberá uma cláusula com algumas variáveis não unificadas, ou seja, sem valor, e tentará fazer uma unificação desta cláusula com alguma existente em sua base de conhecimento. Caso este processo de unificação ocorra com sucesso, o agente que recebeu a mensagem irá retornar ao agente emissor a cláusula unificada com o conteúdo de sua base de conhecimento. Este é o processo usado nos agentes armazenadores para recuperação de informação de gerenciamento. Na Figura 50 é apresentado um exemplo de regras de um agente de interface que envia uma mensagem a um agente armazenador solicitando que ele recupere a lista de sistemas gerenciados que ele possui em sua base de conhecimento. O agente armazenador irá procurar por uma regra que unifique com a regra enviada como conteúdo da mensagem e responderá com o conteúdo desta unificação. O envio da mensagem do agente de interface ao agente de armazenamento ocorre de maneira transparente ao agente de interface, uma vez que o envio e retorno da mensagem são tratados como parte dos sub-objetivos do corpo da regra que inicia a recuperação.

```
<kdb>
[... ]
getReport(Name,Param,Out) :-
    headerReport(Name,Out),!,
    storeURL(StoreURL),
    agent(query-ref, StoreURL, buildReport(Name,Param,TreeReport)),
    print(TreeReport),
    print(Out,TreeReport),!.
```

* A FIPA define um conjunto de atos comunicativos. O ato comunicativo “request” diz que um agente que recebe uma mensagem deste tipo deve executar a operação especificada no conteúdo da mensagem.

** Este ato comunicativo é usado quando uma cláusula é enviada para saber o conteúdo de determinada variável na base de conhecimento do agente receptor. O agente receptor faz a unificação da variável com o conteúdo da sua base de conhecimento e retorna a mensagem ao emissor.


```

gui(http-request,Request,Param,ContentType,Out) :-
  string(startswith,Request,"/grid_data/"),
  string(substr,Request,'11',Name),
  getContentType(Name,ContentType),!,
  getReport(Name,Param,Out),!.
[... ]
</kdb>

```

Figura 50 - Parte das regras para recuperação de informações.

Em um trabalho futuro, o *grid* de armazenamento pode distribuir o armazenamento baseando-se no “interesse” dos agentes analisadores pelos dados. Isso significa que os dados podem ser destinados para um agente armazenador em particular, se outros agentes solicitarem um conjunto de dados com mais frequência a eles. Eles podem então maximizar o seu interesse pelos dados e forçar que o armazenamento seja realizado por eles, uma vez que este tipo de dados lhe é sempre solicitado.

6.10 A Etapa de Análise dos Dados

Uma das vantagens a serem evidenciadas da utilização do *grid* de agentes no gerenciamento é a possibilidade de distribuir a carga de processamento das atividades de gerenciamento diminuindo o tempo gasto nas atividades de análise e utilizando recursos ociosos. Porém, antes que possa existir uma distribuição de atividades de análise, é necessário que as regras para análise de dados sejam construídas. Como os agentes da plataforma utilizada para a implementação da arquitetura têm seu conhecimento representado em estruturas de lógica de primeira ordem, tais regras de análise devem ser construídas e modeladas de acordo com esta estrutura.

Os critérios para distribuição das atividades de análise descritos anteriormente consistem na disponibilidade de conhecimento e de recursos computacionais para realizar as atividades. No primeiro caso, é necessário que o agente que irá realizar as tarefas de análise tenha o conhecimento necessário para tal, ou seja, as regras para efetuá-las. No segundo caso, além de possuir as regras necessárias para realizar tais atividades, os agentes devem estar em execução em recursos com capacidade computacional para processar os dados.

A construção de regras para analisar informações e identificar problemas de *hardware* que possam estar ocorrendo em estações da rede, como escassez de memória,

de processador, ou problemas com sistemas instalados são exemplos de dados analisados no *grid* de análise.

Uma abordagem para a construção destas regras e identificar quais itens devem ser analisados foi a análise de informações de atributos contidos nas especificações do modelo **CIM** [37] e do pacote de instrumentação **WMI** [84] da plataforma Windows da Microsoft.

O plano de ação adotado para elaboração das regras foi:

1. Identificar os itens que se deseja verificar.
2. Baseando-se nas classes definidas no modelo **CIM** e as extensões do sistema operacional Windows, analisar quais atributos de quais classes fornecem a informação necessária para a verificação dos itens cuja verificação é desejada.
3. De posse destas informações, definir de forma descritiva quais são as condições que devem ser verificadas dos sistemas. Apresentar nesta descrição as classes definidas no item 2 que fornecem as informações necessárias.
4. Depois de definidas de forma descritiva, as regras devem ser representadas em **PROLOG** para serem utilizadas pelos agentes no *grid* de análise.

Depois de criado um cenário de gerência, com um conjunto de regras reais que possam ser utilizadas e uma vez que os coletores e armazenadores do *grid* estejam funcionando, testes para a distribuição de atividades podem ser realizados. Um cenário simples de distribuição pode ser imaginado. Em um ambiente centralizado, uma única estação de gerenciamento seria encarregada de analisar todas as informações de utilização de memória e de processador das estações de uma rede – são usados valores de uso de memória e processador apenas como um exemplo. Se a rede fosse composta por um número de 50 estações gerenciadas, a estação de gerenciamento teria que analisar dados de todas estas estações. Podemos realizar uma distribuição dos dados de análise por dispositivo gerenciado. Neste caso se o *grid* de agentes, em um determinado momento, fosse composto de 10 estações, ou seja, cada estação será responsável por analisar os dados de 10 estações. É claro que no *grid*, recursos podem ter mais capacidade computacional do que outros e são capazes de processar mais informação. Então, simplesmente repartir estes dados, sem levar este fator em consideração vai contra a idéia original que é a de aproveitar os recursos com base em sua capacidade.

A forma como a distribuição é realizada, ou quem realiza o escalonamento das atividades de análise, ou atribui as atividades aos agentes, é um outro fator a ser considerado. Na proposta original [12] o *grid* possuía uma raiz que era responsável por realizar toda a distribuição, ou seja, o escalonamento. Tal arquitetura é limitada por possuir um ponto único de falha e sobrecarregar a raiz com as atividades de escalonamento. Nos modelos implementados, os nós que são inseridos no *grid* possuem tarefas de análise pré-definidas. Quando eles não são capazes de analisar estas informações usando os recursos que eles têm disponíveis, eles procuram no *grid* por outros agentes e delegam as atividades a eles. Eles então iniciam as atividades de análise, requerendo os dados de que precisam. Neste caso, cada agente responsável por um conjunto de análises é um escalonador em potencial, pois ele pode escalonar tarefas ou negociar com outros agentes a execução delas. O *grid* de análise é então um ambiente orientado por tarefas [125] onde as tarefas de análise são distribuídas e alocadas entre os agentes do *grid* [41].

6.11 As Interfaces do Sistema

Os agentes de interface têm como objetivo apresentar as informações de gerenciamento, como gráficos, estatísticas e relatórios, ao gerente humano, ou formatar os dados para servirem como entrada para outro sistema de gerência. Estes agentes também têm como princípio receber o *feedback* do usuário e fornecê-lo ao sistema. Para a construção e experimentação destes agentes no protótipo, foram desenvolvidas *skills* específicas para o carregamento dos relatórios, geração de gráficos, além das regras para manipulação destas funcionalidades.

Um recurso implementado e utilizado para apresentação dos relatórios e gráficos de gerência é o servidor HTTP proporcionado pelos nós do *grid*. Ou seja, um nó *grid* que foi adicionado possui um servidor HTTP que é utilizado para a transferência de mensagens para os agentes do nó. Este servidor também é utilizado para receber solicitações de relatórios, e para provê-los ao usuário. O nó *grid* de interface é capaz de diferenciar uma mensagem FIPA postada no servidor de uma requisição para uma página mantida pelo nó, ou para um relatório que deve ser recuperado do *grid* de armazenamento. Este cenário é descrito na Figura 51, onde existe um nó *grid* que é usado como interface para o sistema de gerência. Neste caso, quando o usuário faz uma

solicitação de um relatório (1), o servidor HTTP deste nó é capaz de tratar estas solicitações (2), solicitando do sistema os dados necessários para que eles sejam apresentados (3), usando o sistema de comunicação adotado. Quando uma mensagem é postada no servidor HTTP do nó e esta requisição não corresponde a uma mensagem de comunicação entre agentes, esta solicitação é enviada ao agente gerente do nó, que irá tratá-la da forma adequada.

Durante o desenvolvimento deste trabalho, adotou-se uma abordagem baseada em XML para a apresentação dos relatórios de gerenciamento. Os relatórios gerados pelo *grid* são passados aos agentes de interface em um formato XML. Estes agentes são responsáveis pela transformação dos relatórios de acordo com a interface usada pelo usuário do sistema para visualização. Esta transformação é realizada usando ferramentas XSL e XSLT [122] que permitem a transformação e formatação de documentos XML em diferentes formatos como XHTML, PDF e outros. Embora esta transformação possa ser realizada pelos agentes e estes devolverem o documento resultante, para os nossos testes os agentes de interface devolvem o documento XML resultante da solicitação ao relatório e o documento XSL usado para formatação e apresentação pelo dispositivo que está requerendo o relatório. A formatação para apresentação do relatório é feita pelo cliente.

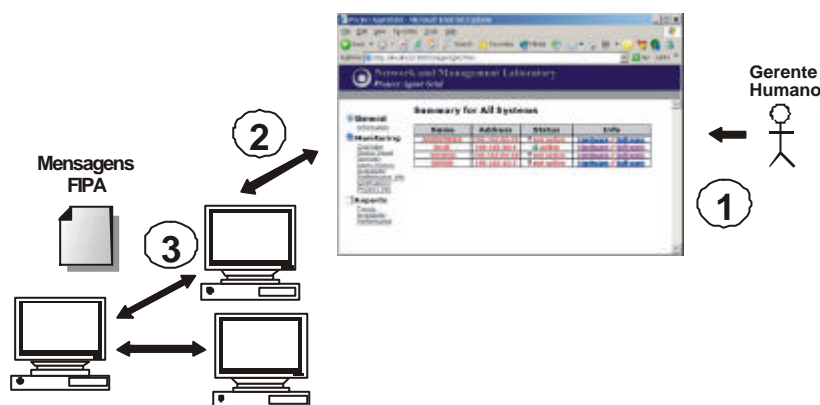


Figura 51 - Servidor HTTP usado pelo sistema de comunicação e interface.

Um exemplo do resultado desta formatação é apresentado na Figura 52, onde existem: um trecho de um documento XML que constitui um relatório (1), e o resultado da formatação apresentado ao usuário (2).

```

① <?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="hard_info.xsl"?>
<hardware_info>
  <system>
    <address>150.162.63.4</address>
    [...]
    <bios>
      <Description>Default System BIOS</Description>
      <Manufacturer>American Megatrends Inc.</Manufacturer>
      <ReleaseDate>20010719*****.******+***</ReleaseDate>
      <SerialNumber>BR21110214</SerialNumber>
      <SMBIOSBIOSVersion>JN.01.00US</SMBIOSBIOSVersion>
    </bios>
    [...]
  </system>
</hardware_info>

```

② Basic Input/Output System	
Caption:	Default System BIOS
Manufacturer:	American Megatrends Inc.
Release Date:	20010719*****.******+***
Serial Number:	BR21110214
SMBIOS BIOS Version:	JN.01.00US

Figura 52 - Exemplo de formatação usando XSL.

A *skill* construída para a geração de gráficos de gerenciamento é chamada de *ChartSkill*. Ela fornece aos agentes a habilidade de criar gráficos do tipo pizza, barras verticais ou horizontais, linhas, pontos e gráficos de área. Esta *skill* faz a ligação das estruturas lógicas da base de conhecimento com as funcionalidades providas por uma biblioteca para a geração de gráficos [67], de código fonte aberto e de distribuição livre. Na Figura 53 é apresentado um exemplo simples de gráfico gerado por esta *skill* e apresentado por um agente de interface. Este gráfico apresenta porcentagem de utilização dos recursos de um determinado nó do *grid*.

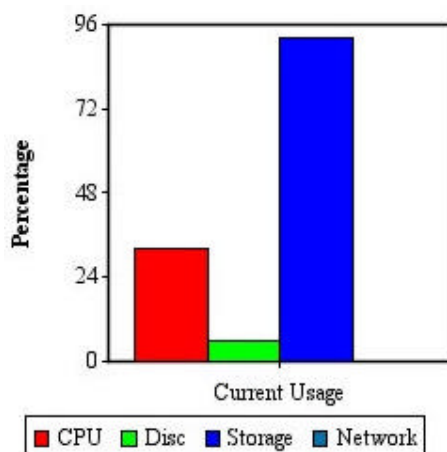


Figura 53 - Exemplo de gráfico gerado pela *ChartSkill*.

Apesar das funcionalidades providas pela interface HTTP possibilitarem a realização de alguns testes e a visualização de relatórios gerados pelo *grid*, elas não são muito úteis quando a necessidade é o envio de alertas ao gerente humano sobre a ocorrência de problemas. Conforme foi apresentado na arquitetura original de *grid* [12], as interfaces deveriam ser diversas e possibilitar o envio de alertas ao gerente. Uma solução encontrada é dotar os agentes das habilidades e conhecimento necessários para a manipulação e utilização dos protocolos usados para comunicação. Para isso uma abordagem foi a criação de uma *skill* que possibilita que os agentes façam uso das funcionalidades da API Java para a utilização de *sockets*, e da construção de regras correspondentes para o envio destes alertas. Neste caso os agentes podem enviar as notificações através de mensagens de e-mail, usando o protocolo SMTP para transferência das informações. Um exemplo de um trecho de regras usadas para o envio de uma notificação usando o protocolo SMTP é apresentado na Figura 54.

```
<kdb>
[... ]
sendMail(Host,Port,From,To,Subject,Message) :-
    socket(open,Host,Port,Socket),
    socket(output,Socket,Output),
    string(join,["MAIL FROM: <",From,">\r\n"],Whole),
    string(join,[Whole,"RCPT TO: <",To,">\r\n"],Whole),
    string(join,[Whole,"DATA\r\n"],Whole),
    string(join,[Whole,"From:",From,"\r\n"],Whole),
    string(join,[Whole,"To:",From,"\r\n"],Whole),
    string(join,[Whole,"Subject:",Subject,"\r\n\r\n"],Whole),
    string(join,[Whole,Message],Whole),
    string(join,[Whole,"\r\n.\r\n"],Whole),
    print(Output,Whole),
    socket(close,Socket),!.
[... ]
</kdb>
```

Figura 54 - Exemplo de regras para envio de notificação.

6.12 Aplicação para Inventário e Configuração – Estudo de Caso

Duas das categorias mais comuns de gerenciamento de sistemas que as empresas utilizam são o gerenciamento de mudanças e configuração e o gerenciamento de operações. O gerenciamento de mudanças e configuração engloba a instalação e configuração de *software* e *hardware* durante todo o ciclo de vida dos sistemas computacionais envolvidos [114]. O gerenciamento de operações proporciona eventos em tempo real e monitoramento de desempenho de servidores e suas aplicações, possibilitando que os administradores maximizem o tempo de execução dos serviços e

minimizem o tempo de parada para os usuários finais e as operações comerciais da organização. Estas duas disciplinas provêm serviços complementares de gerenciamento: as operações de gerenciamento alertam os gerentes da necessidade de gerenciamento de mudança e de configuração. Prover um gerenciamento que permita uma visão centralizada da base instalada é uma tarefa cada vez mais difícil quando a base de sistemas é muito grande. Os administradores devem apresentar freqüentes relatórios de desempenho e outros, como inventários dos bens e recursos da plataforma computacional disponível. Tais atividades requerem tempo e recursos para serem realizadas.

À medida que o número de nós do ambiente aumenta, um fator crítico aparece: a carga de processamento do sistema de gerência aumenta, criando um grande retardo nas notificações e relatórios enviados ao gerente. Neste cenário o *grid* de agentes pode ser aplicado: para distribuir a carga de processamento destas atividades. Contudo, pela ausência de um cenário de gerenciamento que pudesse apresentar tais condições de carga e requisitos de processamento, optou-se por desenvolver protótipos simples onde todos os módulos do sistema possam ser testados de maneira satisfatória, apresentando o seu funcionamento e onde algumas interfaces que apresentem o resultado do funcionamento do sistema de modo geral. Além disso, tal validação, embora simples, é importante e tem o objetivo de facilitar a compreensão do funcionamento da arquitetura como um todo e de possibilitar a extensão e melhoramento em seus vários módulos.

Para realizar um gerenciamento de mudanças e configuração, alguns requisitos se fazem necessários, dentre eles a montagem de inventários que apresentem uma relação do *hardware* e *software* instalados nos sistemas existentes no ambiente gerenciado. A seguir descrevemos alguns cenários e como as interfaces são montadas nestas modalidades de gerenciamento a seguir.

6.12.1 Criando Inventários de *Hardware* e *Software*

Baseado nas funcionalidades providas pelo pacote de instrumentação das plataformas Windows e em utilitários de gerenciamento usados no ambiente UNIX, um primeiro cenário de gerenciamento de mudanças, configuração e inventário foi criado. Tais inventários são úteis no controle dos recursos, no controle de licença de *softwares*

instalados nos sistemas disponíveis e na manutenção de equipamentos. Os agentes coletores **WMI** executam *scripts* escritos na linguagem *Visual Basic Script* para solicitar as informações de gerenciamento ao gerenciador de objetos dos sistemas Windows disponíveis na rede. Os agentes coletores se comportam como clientes da plataforma de gerenciamento ou de agentes armazenadores para os quais eles enviam as informações de gerenciamento coletadas. O sistema implementado apresenta interfaces simples construídas para demonstrar o funcionamento do sistema. Porém estas interfaces podem ser expandidas no caso de se desejar construir um sistema de gerenciamento mais sofisticado, o que está fora do escopo deste trabalho.

Estas informações são extraídas dos objetos existentes nos sistemas e seguem o padrão estabelecido pelo modelo de informação **CIM** [37] e das extensões ao modelo feitas pela Microsoft para a plataforma Windows, que permite um bom nível de informação a respeito do *hardware* e *software* instalados. No ambiente **Linux** estas informações podem ser obtidas através da execução de utilitários de linha de comando ou através da leitura de arquivos de configuração. Além disso, o protocolo de gerenciamento SNMP é usado para extrair informações de ambas as plataformas.

Alguns exemplos de informações de *hardware* obtidas para montar os relatórios de inventário destes sistemas são:

- Número e características dos discos instalados.
- Número e características dos processadores.
- Quantidade de memória física.
- Versão e detalhes do sistema operacional.
- Configurações de monitor e de placa de vídeo.
- Informações sobre periféricos conectados ao sistema.
- Características e informações das interfaces de rede.

Para exemplificar, as informações a respeito do processador, em sistemas Windows, são obtidas solicitando ao gerenciador de objetos **CIM** do sistema onde o agente coletor está sendo executado (veja Figura 44 que contém esboço do modelo WMI).

A apresentação das informações em interfaces HTML não constitui uma tarefa complexa que requeira um tratamento muito apurado dos dados no caso dos inventários. Caso haja a necessidade de alguma atividade por parte dos agentes de análise, estas são

mais no sentido de construir os arquivos XML resultantes que seriam já apresentados nas interfaces do sistema ao invés de fazer com que eles sejam construídos no momento em que são solicitados. Assim, os agentes de análise solicitam os dados aos agentes armazenadores e constroem os arquivos XML usados para apresentação pelos agentes de interface, e enviam os relatórios construídos aos agentes armazenadores, de onde são recuperados pelos agentes de interface.

6.12.2 Ambiente de Experimentação

Para realização de testes dos componentes da arquitetura e para a construção de inventários úteis à gerência de configuração e mudanças, os componentes foram aplicados na construção de um ambiente composto por: alguns agentes coletores, armazenadores de dados, analisadores simples e agentes de interface. O ambiente de experimentação é descrito na Figura 55 e os recursos utilizados para esta validação são os oferecidos pelo Laboratório de Redes e Gerência da UFSC. Conforme descrito anteriormente as informações coletadas pelos agentes coletores são extraídas das estações Windows usando o pacote de instrumentação WMI e no ambiente Linux são obtidas usando utilitários de linha de comando. Em ambas plataformas, o SNMP é utilizado para extrair informações, tais como relação de softwares instalados nos equipamentos. Embora os papéis de coletores e analisadores sejam apresentados como estando em estações diferentes, nos testes realizados os equipamentos geralmente desempenham mais de um papel, como o de analisador e coletor de dados.

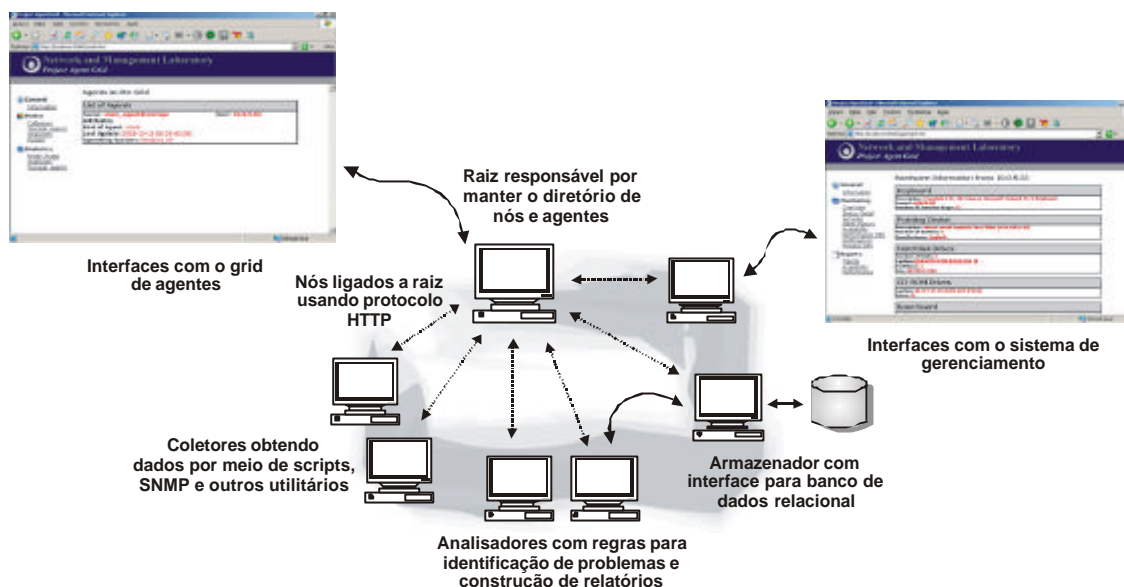


Figura 55 - Visão geral do sistema.

Quando iniciados os nós se ligam ao elemento raiz, registram as suas habilidades com este nó e informam se eles possuem agentes de coleta, armazenamento ou de análise. A raiz é responsável por manter um sistema de diretórios onde os nós do ambiente são registrados. As interfaces com informações sobre o estado do *grid* e informações do sistema de gerência são apresentadas ao usuário utilizando o servidor HTTP do *listener* provido pelo nó que é usado como raiz para o *grid* de agentes. Neste caso é importante fazer uma distinção entre as interfaces que são utilizadas para exibição de informações do *grid* e para exibição de informações do sistema de gerência formado pelo *grid*. Um exemplo de interface provida pelo *grid* de agentes é apresentada na Figura 56, onde é exibida uma lista dos sistemas cadastrados e monitorados pelos agentes.

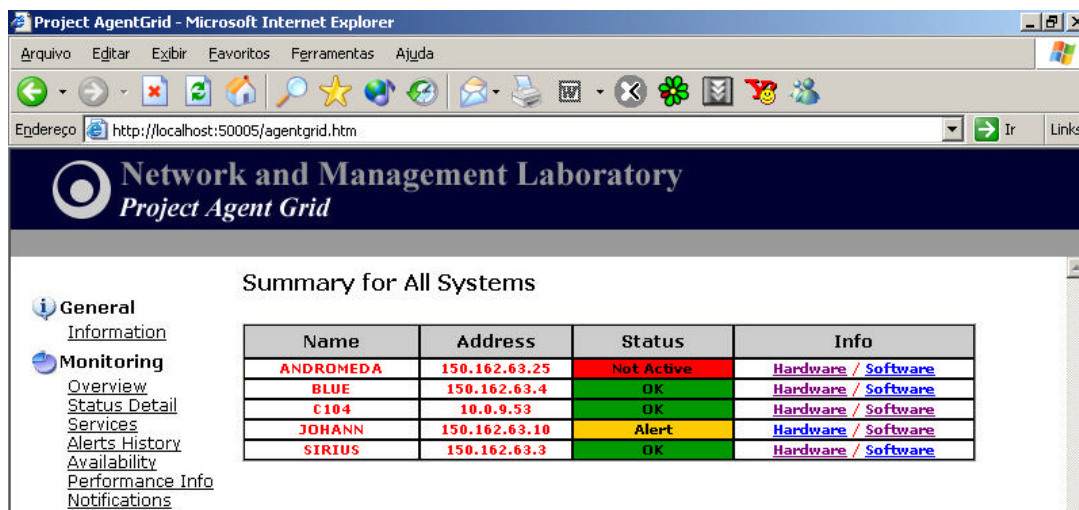


Figura 56 - Exemplo de interface do sistema.

6.12.3 Inventários de Hardware e Software

Conforme destacado anteriormente as informações de *hardware* e *software* oriundas dos sistemas monitorados em um ambiente Windows são extraídas usando WMI e SNMP. Não convém entrar em detalhes aqui a respeito das regras usadas para coleta, pois a idéia geral e sua construção já foram apresentadas na sub-seção que descreve os agentes coletores (sub-seção 6.8). Alguns exemplos de arquivos XML resultantes da coleta das informações de gerenciamento podem ser apresentados. A Figura 57 por exemplo, apresenta parte do resultado da coleta usando SNMP da lista de softwares instalados em um dos equipamentos do sistema.

```
<config>
  <id>
    <ip>150.162.63.3</ip>
    <time>1071485631043</time>
    <from>collector_agent@collector_snmp_1_sirius</from>
  </id>
  <software index="1">
    <hrSWInstalledName>Ad-aware 6 Personal</hrSWInstalledName>
    <hrSWInstalledID>0.0</hrSWInstalledID>
    <hrSWInstalledType>4</hrSWInstalledType>
  </software>
  <software index="2">
    <hrSWInstalledName>Adobe Acrobat 5.0</hrSWInstalledName>
    <hrSWInstalledID>0.0</hrSWInstalledID>
    <hrSWInstalledType>4</hrSWInstalledType>
  </software>
  [...]
  <software index="115">
    <hrSWInstalledName>Microsoft .NET Framework 1.1</hrSWInstalledName>
    <hrSWInstalledID>0.0</hrSWInstalledID>
    <hrSWInstalledType>4</hrSWInstalledType>
  </software>
</config>
```

```
</software>
</config>
```

Figura 57 - Lista de softwares instalados em um sistema.

Estas informações são apresentadas ao usuário através de uma interface *web* baseada em XML. Os dados são formatados para exibição no navegador do computador do usuário usando as funcionalidades proporcionadas pela tecnologia XSL e XSLT. O arquivo de formatação é escolhido de acordo com as características do dispositivo que o usuário está utilizando. Um exemplo da lista de *software* visualizada pelo usuário pode ser vista na Figura 58. Esta interface apresenta uma lista dos programas instalados em um determinado equipamento. Na Figura 59 é apresentado o documento XML que irá compor este relatório, sendo que o documento é gerado levando em consideração o documento de formatação que será utilizado pelo usuário para exibição das informações.

Software Installed on 150.162.63.10	
Installed Software	
Adobe Acrobat 5.0	
FlashGet ads support	
FlashGet(JetCar)	
ICQ Lite	
Java 2 Runtime Environment, SE v1.4.1_06	
Java 2 SDK, SE v1.4.1_06	
Java Web Start	
Microsoft Internet Explorer 6 SP1	
Microsoft Office 2000 Premium	
PowerArchiver	
SSH Secure Shell	
WebFldrs	
Yahoo! Messenger	
Yahoo! Messenger Explorer Bar	

Figura 58 - Relação de *softwares* instalados em 150.162.63.10.

Os relatórios apresentados ao usuário são gerados pelos agentes de armazenamento. Por serem apenas relatórios de inventário e que não precisam de grande manipulação de dados, eles são criados no momento em que são solicitados. O mesmo não acontece com relatórios estatísticos e de desempenho, que são previamente trabalhados pelos agentes de análise e armazenados pelos agentes armazenadores. No momento em que são solicitados, eles são apenas recuperados por estes agentes. A Figura 60 apresenta uma interface com as características de *hardware* de um determinado equipamento gerenciado. Este relatório é montado como um documento XML e formatado usando o interpretador XSLT do navegador do usuário.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="../xsl/software.xsl"?>
<software_list>
  <system>
    <address>150.162.63.10</address>
  </system>
  <software index="1">
    <hrSWInstalledName>Adobe Acrobat 5.0</hrSWInstalledName>
  </software>
  <software index="2">
    <hrSWInstalledName>FlashGet ads support</hrSWInstalledName>
  </software>
  [...]
</software_list>

```

Figura 59 - Documento XML do relatório de softwares instalados.

Basic Input/Output System
Caption: Default System BIOS Manufacturer: American Megatrends Inc. Release Date: 20010719*****.*****+*** Serial Number: BR21110214 SMBIOS BIOS Version: JN.01.00US
Processors
Number of Processor: 1 Description: x86 Family 15 Model 1 Stepping 2 Description: x86 Family 15 Model 1 Stepping 2 Manufacturer: GenuineIntel Socket Designation: Socket 478 Max Clock Speed: 1594 Ext Clock: 100 L2 Cache Size: 256 Architecture: 0
Video Controller
Description: ATI Technologies Inc. RAGE 128 PRO Ultra GL AGP Current Bits Per Pixel: 32 Current Horizontal Resolution: 1152 Current Vertical Resolution: 864 Video Architecture: 5 Video Processor: RAGE128 PRO II, (AGP 4X/PCI)

Figura 60 - Parte da interface para exibição do *hardware* instalado.

6.12.4 Relatórios de Gerência

Conforme descrito anteriormente, os relatórios referentes ao inventário dos sistemas são construídos no momento em que são solicitados pelo usuário, pois apenas necessitam da recuperação de informação armazenada para que sejam construídos. Um outro caso é o dos relatórios de gerenciamento como de desempenho, alertas de problemas e outros. Neste caso os relatórios são construídos pelos agentes de análise de informações que após analisarem um conjunto de dados que lhes é fornecido devolvem o relatório resultante ou as notificações de problemas ao armazenador de dados, para

que estes sejam armazenados e depois recuperados para posteriormente serem apresentados ao usuário do sistema.

Um exemplo de relatório gerado desta forma é o relatório diário de disponibilidade dos sistemas. Os agentes de análise recebem um conjunto de dados que representam os tempos em que o sistema ficou parado durante o período de análise. Este agente então calcula a disponibilidade, constrói o relatório correspondente e devolve-o ao *grid* de armazenamento onde é armazenado para ser recuperado quando for solicitado pelo usuário. A Figura 61 apresenta parte do conjunto de regras necessárias ao agente de análise para gerar o relatório de disponibilidade e na Figura 62 é apresentado o XML resultante.

```
[...]
analysis('system_availability',System,InitialDate,FinalDate) :-
    storeURL(Store),
    date(diff,FinalDate,InitialDate,TotalTime),
    tree(create,TreeReport,"availability"),
    tree(add,TreeReport,"system",System),
    tree(add,TreeReport,"initialdate",InitialDate),
    tree(add,TreeReport,"finaldate",FinalDate),
    agent(query-ref,Store,
           getOutages(System,InitialDate,FinalDate,ListOutages)),
    eq('0',TimeOutage),
    list(string,ListOutages,String),
    sumOutages(ListOutages,TimeOutage),
    calculateAvailability(TimeOutage,TotalTime,TreeReport),
    tree(string,TreeReport,SerialReport),
    storeReport(System,InitialDate,"accumulative",
               'system_avail_daily',SerialReport),!.
[...]
```

Figura 61 - Parte das regras para análise da disponibilidade.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<availability>
  <system>150.162.63.13</system>
  <initialdate>2004-01-06 00:00:00.0</initialdate>
  <finaldate>2004-01-07 00:00:00.0</finaldate>
  <availability>100.0</availability>
</availability>
```

Figura 62 - XML resultante da análise de disponibilidade.

Outro exemplo de análise apresentado é o de um agente que é responsável por verificar a utilização do processador de um conjunto de sistemas em um determinado período. Neste caso, é evidente que devem ser especificados valores limites para estes itens. Ou seja, no caso do processador, o sistema deve conhecer qual é o valor que pode ser considerado uma superutilização do mesmo. Para efeito dos testes realizados assumimos que:

- O agente analisa as últimas coletas de uso do processador e calcula quantas das vezes que o uso do processador foi coletado, este estava com um grau de utilização acima de 80%.
- Calcula se a soma das vezes em que o mesmo se encontra acima do uso considerado ideal excede o valor de 50% do total de coletas realizadas. Se este valor for atingido, cria uma notificação que será apresentada ao usuário.

Estes eventos alertando sobre a ocorrência de uma situação anormal podem ser visualizados na interface fornecida pelo sistema. As notificações são apresentadas como um documento XML, sendo que são formatadas de acordo com o arquivo de formatação escolhido.

				above the accepted threshold
7	2004-01-06 10:06:58.0	150.162.63.11	HIGH	The processor utilization is above the accepted threshold
6	2004-01-06 10:06:58.0	150.162.63.10	HIGH	The processor utilization is above the accepted threshold
5	2004-01-06 10:05:23.0	150.162.63.13	LOW	System seems to be working correctly again.
4	2004-01-06 10:05:19.0	150.162.63.12	LOW	System seems to be working correctly again.
3	2004-01-06 10:05:16.0	150.162.63.10	LOW	System seems to be working correctly again.
2	2004-01-06 10:04:58.0	150.162.63.10	HIGH	System is not responding.
				The processor utilization is

Figura 63 - Tela com alertas gerados pelo sistema.

6.12.5 Informações Sobre o *Grid*.

O nó usado como raiz do *grid* de agentes provê uma interface que apresenta informações a respeito dos nós pertencentes ao *grid*, as rotas para estes nós e os agentes existentes no *grid*. Também é possível visualizar as informações de desempenho dos nós que são usados para a análise dos dados de gerenciamento. Na Figura 64 é exibida uma interface onde é apresentada uma lista dos agentes coletores registrados com a raiz do *grid* de agentes.

The screenshot shows a web interface titled "Agents on the Grid". On the left is a navigation menu with sections: "General" (Information), "Status" (Collectors, Storage Agents, Analyzers, Routes), and "Statistics" (Analyzers, Memory Usage, CPU Usage, Storage Usage, Storage agents). The main content area is titled "List of Agents" and displays three entries for collector agents:

List of Agents	
Name: collector_agent@collector_001_c104	Host: 10.0.9.53
Attributes	
Kind of Agent: collector	
Last Update: 2004-01-03 12:41:09.77	
Operating System: Windows XP	
Name: collector_agent@collector_002_c104	Host: 10.0.9.53
Attributes	
Kind of Agent: collector	
Last Update: 2004-01-03 12:41:11.332	
Operating System: Windows XP	
Name: collector_agent@collector_003_c104	Host: 10.0.9.53
Attributes	

Figura 64 - Lista de agentes coletores registrados.

Esta interface também apresenta informações dos agentes analisadores registrados no *grid* de agentes, assim como informações de utilização dos recursos onde estes agentes estão em execução. Na Figura 65 é possível ver um exemplo de parte da tela que apresenta as informações de um agente analisador registrado no *grid*.

The screenshot shows a detailed view of an analyzer agent in the "List of Agents" section:

List of Agents	
Name: analysis_agent@analysis_1_c104	Host: 150.162.63.19
Attributes	
Kind of Agent: analyzer	
Last Update: 2004-01-06 11:29:40.078	
Operating System: Linux	
CPU Capacity: 52	
Memory Capacity: 247684	
Storage Capacity: 37986533	
Kind of analysis able to carry out	
processor_usage	
system_availability	
Information from the performance meter	

Figura 65 - Informações sobre um agente analisador.

6.12.6 Análise de Informações

Para realização dos testes da arquitetura proposta foram criadas algumas regras para análise e identificação de problemas de algumas das informações coletadas pelos agentes coletores. Estas regras abrangem principalmente os seguintes itens:

- Utilização do processador dos sistemas gerenciados.
- Utilização da memória destes equipamentos.
- Uso dos dispositivos de armazenamento.
- Processos em execução.

Conforme apresentado anteriormente como resultado destas análises temos os relatórios que são apresentados ao cliente e as notificações a respeito dos problemas que podem estar ocorrendo nos equipamentos. Embora sejam regras simples, elas são usadas para demonstrar o funcionamento do *grid* de análise, uma vez que o desenvolvimento de uma aplicação que abranja os pontos da arquitetura é um dos objetivos deste trabalho.

Parte de um relatório apresentado ao usuário, contendo alguns alertas gerados pelo sistema, é apresentado na Figura 66. Nesta figura existem três notificações, uma avisando que um sistema deixou de funcionar, outro que o sistema retornou, e uma alertando que a utilização do processador em um determinado sistema ultrapassou os limiares considerados ideais.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" ref="../xsl/notifications.xsl"?>
<notifications>
[... ]
<notification>
  <system>150.162.63.11</system>
  <id>7</id>
  <text>The processor utilization is above the accepted threshold</text>
  <date>2004-01-06 10:06:58.0</date>
  <subject>Processor Overuse</subject>
  <severity>high</severity>
</notification>
<notification>
  <system>150.162.63.13</system>
  <id>17</id>
  <text>System is not responding.</text>
  <date>2004-01-06 12:20:57.0</date>
  <subject>System Outage</subject>
  <severity>high</severity>
</notification>
<notification>
  <system>150.162.63.13</system>
  <id>20</id>
  <text>System seems to be working correctly again.</text>
  <date>2004-01-06 14:25:54.0</date>
  <subject>System Responding</subject>
  <severity>low</severity>
</notification>
[... ]
</notifications>
```

Figura 66 - Relatório em XML contendo notificações.

Além das notificações, estes agentes de análise são responsáveis por construir os relatórios de gerenciamento. Na seção 6.12.4 é apresentado um exemplo de um relatório de disponibilidade diário gerado por um agente de análise.

6.12.7 Distribuição das Atividades de Análise

Esta seção apresenta resultados de testes de distribuição das atividades de análise de dados de gerenciamento realizadas no Laboratório de Redes e Gerência da UFSC. Estes testes foram realizados em uma rede local usando um número pequeno de equipamentos que estava à disposição para a realização dos testes. A descrição da configuração e capacidade dos equipamentos utilizados é descrita na Tabela 7. Embora simples estes testes demonstram os princípios de distribuição de tarefas de análise apresentados pela arquitetura, usando técnicas de escalonamento simples, mas que apresentam a viabilidade do sistema.

Tabela 7 – Configuração dos equipamentos usados para os testes.

Equipamento	Descrição
Máquina A	Processador Pentium III 800 MHz 192MB de Memória RAM 20GB de Disco Rígido Sistema Operacional Windows XP
Máquina B	Processador Duron 1.2 GHz 256MB de Memória RAM 40GB de Disco Rígido Sistema Operacional Linux
Máquina C	Processador Pentium II 233 MHz 64MB de Memória RAM 4GB de Disco Rígido Sistema Operacional Windows 2000
Máquina D	Processador Pentium MMX 233 MHz 64MB de Memória RAM 1.6GB de Disco Rígido Sistema Operacional Windows 2000

Para a realização dos testes optou-se por usar um exemplo de tarefa de análise. Esta tarefa é designada aos agentes de análise no *grid*, de acordo com o critério usado para distribuição, para que estes realizem as análises correspondentes. Nos testes realizados estas tarefas de análise consistem em:

- Analisar a porcentagem de uso do processador dos sistemas que são gerenciados pelo *grid*. Neste caso para efeito de testes são variados o intervalo de chegada das tarefas, e o período de dados que deve ser analisado. Ou seja, a hora inicial e final do lote de dados a ser analisado, o que influencia diretamente no tamanho do conjunto de dados.
- Analisar a utilização de memória dos sistemas. Também, da mesma forma que o uso dos processadores, o tempo entre chegada das tarefas e o intervalo de dados pode ser variado.

No primeiro cenário de testes montado, um agente de análise tem como função realizar um conjunto de tarefas de análise que chegam em intervalos de tempo. Este agente foi configurado para que nunca tenha capacidade de realizar esta tarefa, e por sua vez, deve enviá-la para ser executada por um agente registrado no *grid* que tenha capacidade de realizá-la. Para isso, ele consulta a raiz do *grid* em busca dos agentes aptos a realizarem aquela tarefa – vale lembrar que o agente deve ter o conhecimento para realizá-la – e envia esta tarefa para que ele a realize.

Pelo fato das máquinas disponíveis serem heterogêneas em termos de recursos que oferecem, primeiramente optou-se por realizar um teste sem nenhuma carga de trabalho sendo submetida às estações usadas para a análise. Os resultados destas medições podem ser visualizados no gráfico apresentado na Figura 67. Como podemos perceber, esta é praticamente a utilização dos sistemas pelos agentes que estão em execução, sem realizar tarefas de análise, e pelas atividades desempenhadas pelo sistema operacional. Percebe-se que as estações C e D apresentam um grau de utilização maior, por serem equipamentos mais limitados no que se refere aos recursos oferecidos.

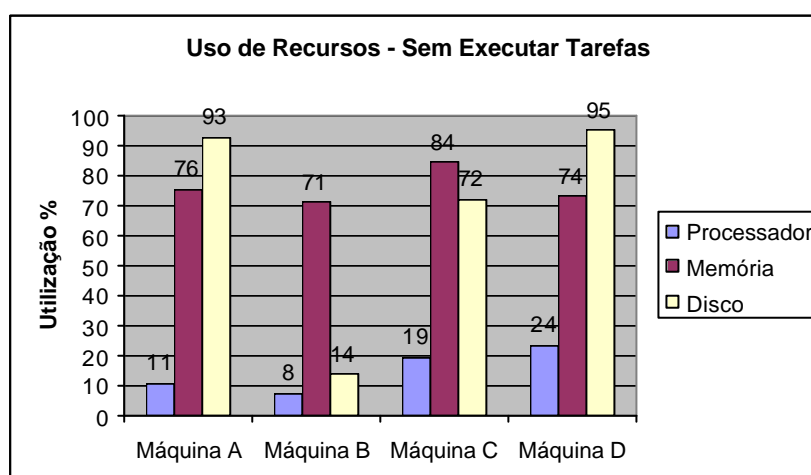


Figura 67 - Utilização de recursos sem execução de tarefas.

O segundo teste realizado foi efetuado usando apenas um agente de análise – além do agente que está delegando a tarefa e que não irá executá-la - registrado no *grid* de agentes (**Máquina A**) para realizar as tarefas de análise. Para a realização deste experimento optou-se por usar os parâmetros descritos na Tabela 8. O tempo entre chegada de tarefas foi configurado para 10 segundos. Enquanto que o intervalo de dados foi configurado como 2 horas. Neste caso, o agente irá verificar todas as coletas do uso de processador dos sistemas durante este período. Vale lembrar que em uma situação

real de gerenciamento, os agentes realizariam tarefas de análise mais apuradas. Utilizou-se a avaliação do uso de recursos para facilitar a elaboração dos experimentos. Durante a realização dos experimentos, foram medidos alguns valores referente à utilização dos recursos onde os analisadores registrados no *grid* estavam em execução durante a elaboração dos testes. O resultado da média da utilização de recursos no **Equipamento A**, realizando a tarefa de análise descrita anteriormente, e não realizando a tarefa, são apresentados na Figura 68.

Tabela 8 – Parâmetros da simulação com apenas uma máquina.

Parâmetro	Valor
Tempo de chegada de uma tarefa de análise	10 segundos
Tempo de duração das medições	Aproximadamente 10 minutos
Intervalo de dados	2 hora

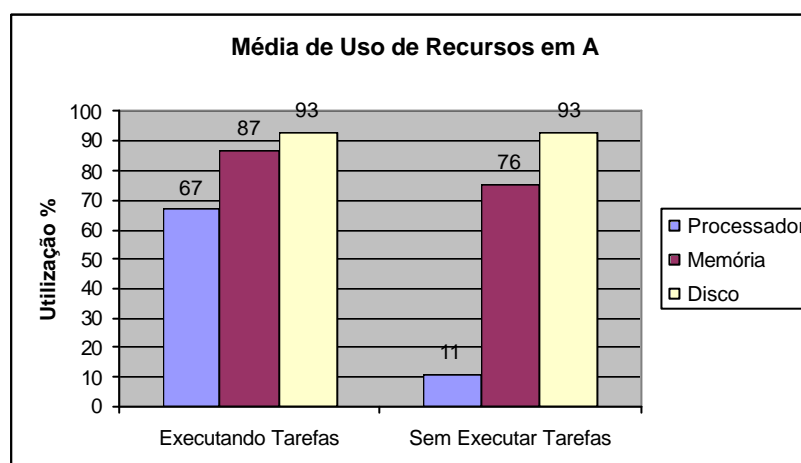


Figura 68 - Uso de recursos na Máquina A.

Neste gráfico apresentado na Figura 68 pode-se perceber algumas características da tarefa usada para estes testes, como o fato dela acarretar em um considerável uso do processador, elevando o seu uso médio para 67%, além de utilizar um pouco da memória RAM do sistema. A utilização do disco é apresentada neste gráfico, embora as tarefas escalonadas não façam uso do disco e não acarretam no armazenamento de dados por parte da estação que está realizando as atividades, o que não causa uma variação na sua utilização. A utilização da rede não foi considerada nestes experimentos por ser considerada proporcionalmente baixa, uma vez que os testes foram realizados em uma rede local.

Em um segundo experimento, todos os agentes foram registrados no *grid*, e configurados como aptos a realizarem as tarefas de análise, ou seja, possuíam o conhecimento ou regras para tal. O agente responsável por delegar as tarefas efetuou uma distribuição cíclica das tarefas usando como parâmetros os mesmos utilizados na Tabela 8. O gráfico com a utilização média de recursos pode ser visto na Figura 69.

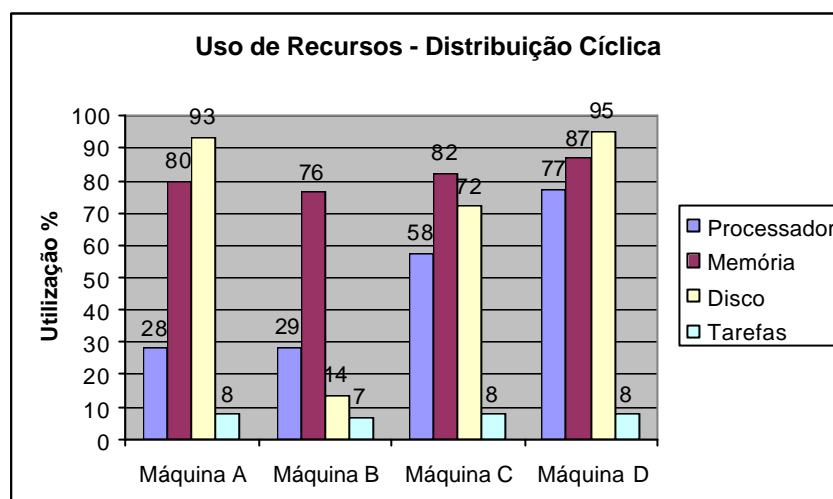


Figura 69 - Distribuição cíclica das tarefas.

Neste gráfico apresentado na Figura 69, pode ser percebido uma equalização no uso médio dos recursos. O uso do processador e de memória da **Máquina A** diminuiu, passando a 28% e 80% respectivamente. Porém, como é apresentado no gráfico da Figura 75, o tempo médio de resposta para realização das tarefas tende a ser um pouco maior. Isto ocorre porque tarefas são escalonadas para máquinas com poucos recursos computacionais sem que estes fatores de escassez de recursos sejam levados em consideração. Esta característica é bastante evidente quando o lote de dados analisado em cada tarefa é maior. No gráfico da Figura 70 é possível visualizar o resultado de outros experimentos, nos quais o tempo entre chegada de tarefas foi alterado para 15 segundos e o intervalo de dados para 1 hora. Embora os parâmetros tenham acarretado um volume de dados menor, é possível ver a mesma equalização no uso de recursos. A utilização de recursos quando as tarefas são executadas apenas pela **Máquina B** são também apresentadas neste gráfico.

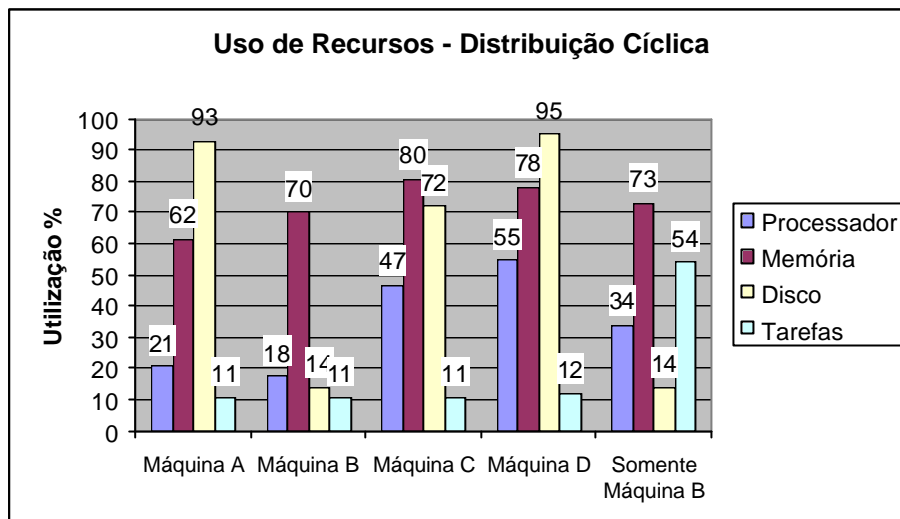


Figura 70 - Distribuição cíclica - segunda tomada.

Também foi testada uma distribuição aleatória entre os agentes aptos a realizarem a tarefa. Tais resultados podem ser vistos no gráfico apresentado na Figura 71. Esta abordagem apresentou um tempo de resposta um pouco menor. Porém, a mesma situação, a respeito do escalonamento de tarefas a recursos não aptos a realizá-las também ocorre nesta abordagem. Por exemplo, a Figura 73 apresenta os resultados de uma outra tomada de dados de uma distribuição aleatória. Neste exemplo pode ser visualizada uma situação onde várias tarefas foram atribuídas a **Máquina C**, a qual é computacionalmente pior do que os equipamentos **A** e **B**.

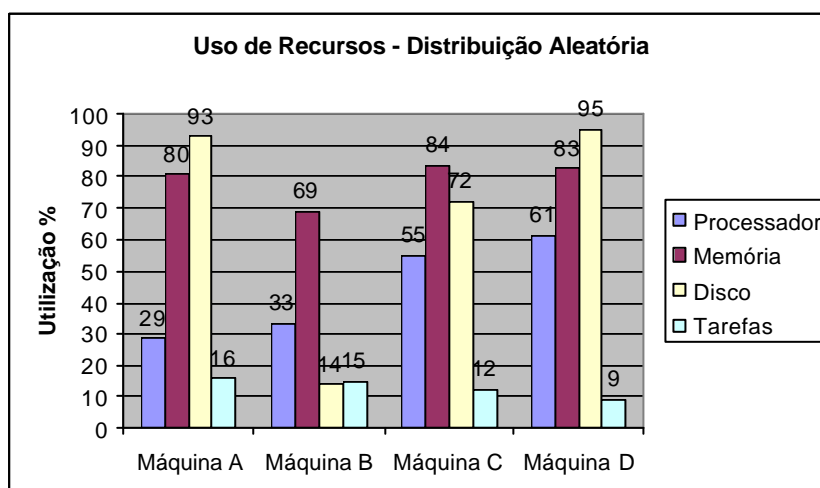


Figura 71 - Distribuição aleatória de tarefas.

Tanto a abordagem de distribuição aleatória, quanto à distribuição cíclica apresentam uma equalização da utilização dos recursos nas atividades de análise de

dados de gerência. Porém, ambas recaem no problema de usar estações não capazes ou com recursos escassos nas atividades de análise. Uma outra abordagem é a utilização dos medidores, descritos anteriormente, para o fornecimento de informações de desempenho e capacidade dos recursos computacionais de forma a auxiliar a atribuição das tarefas. Estas informações são usadas para guiar o agente responsável pelo escalonamento, na decisão, a respeito de qual agente está mais apto a realizar uma determinada atividade de análise. A Figura 72 apresenta de forma abstrata como funciona esta abordagem de distribuição. O agente responsável pela tarefa solicita que a raiz informe quais agentes poderiam realizar tal tarefa (1). A raiz monta uma lista dos agentes e retorna ao analisador (2). Em seguida o agente solicita aos agentes analisadores que estes lhe informem como está a utilização atual do recurso processador nos equipamentos onde eles estão sendo executados (3). Neste cenário cada nó responde a solicitação do agente escalonador com um valor que é o produto da porcentagem não usada do processador pelo número de Mflops, obtidos pelo medidor, que a máquina consegue realizar. Baseado nestas respostas, o agente escolhe o melhor dos analisadores e envia a tarefa a ser realizada por ele (4).

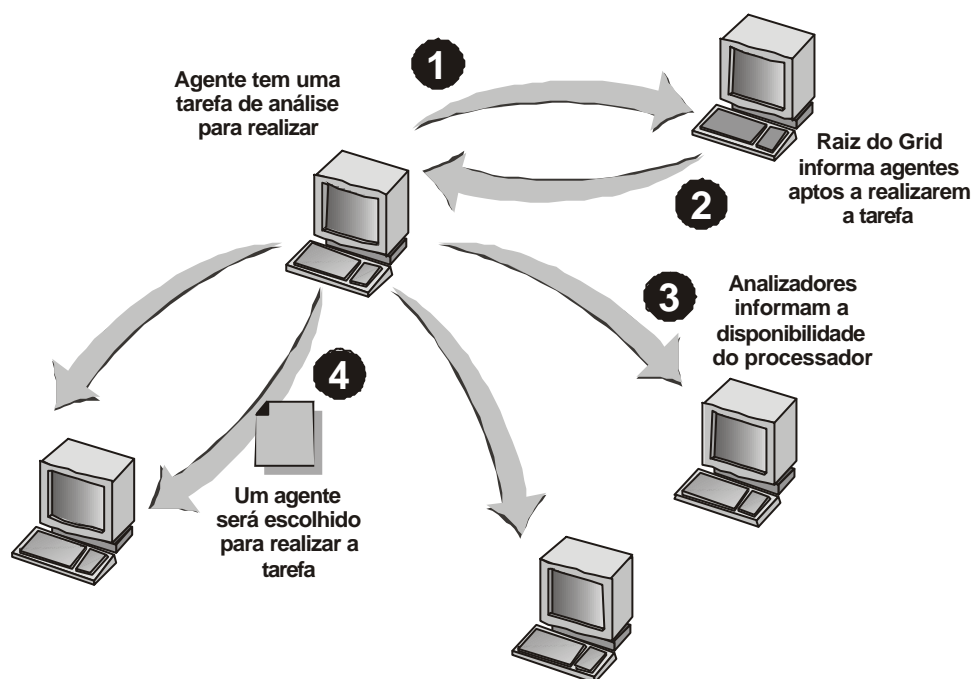


Figura 72 - Distribuição baseada no processador.

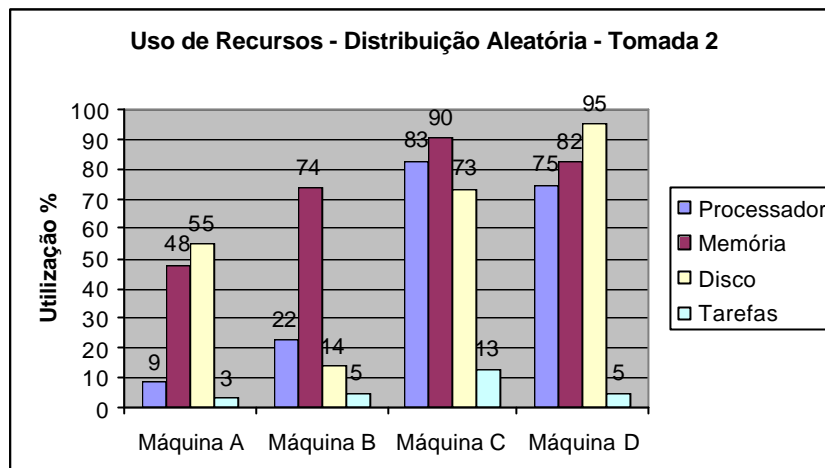


Figura 73 - Distribuição aleatória, segunda tomada de dados.

As características e uso do processador foram usados nos testes, porém poderiam ser usados outros fatores coletados pelo medidor de desempenho do nó *grid*, como memória ou uma taxa baseada no uso de todos os recursos. Os resultados desta abordagem são apresentados no gráfico da Figura 74. A utilização do recurso processador é maior, evidentemente, nos equipamentos que participam da realização das atividades. Embora o tempo de resposta desta para realização das tarefas nesta abordagem seja menor, como pode ser visto na Figura 75, o tempo usado pelo mecanismo de decisão, e pergunta aos agentes do atual uso de recursos não foi contabilizado nestes experimentos.

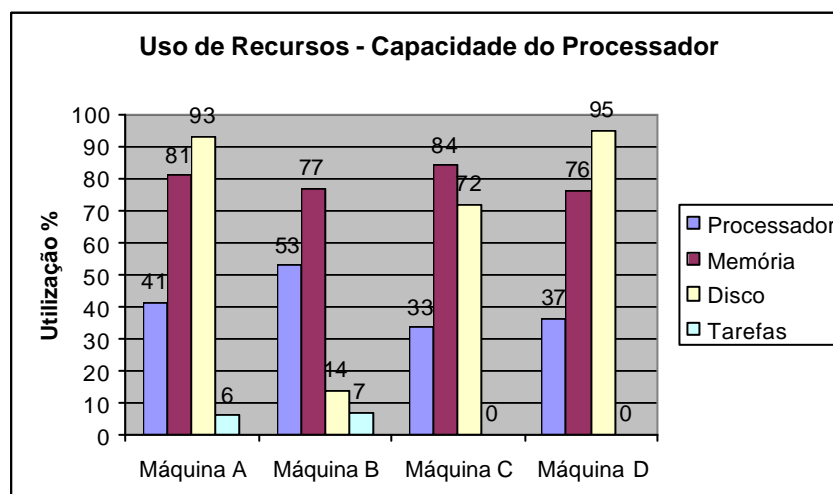


Figura 74 - Distribuição com base no uso/capacidade do processador.

Estas perguntas feitas a cada vez que uma tarefa precisa ser atribuída a um agente, podem causar uma série de problemas, como o aumento do tempo de resposta, o aumento dos custos de comunicação, devido a grande quantidade de mensagens trocadas entre os agentes, além de poder tornar inviável o seu uso em ambientes abertos. Uma proposta apresentada na arquitetura de *grid* [12] é que o serviço de diretórios mantido pela raiz do *grid* armazene um conjunto de informações a respeito da capacidade computacional dos recursos existentes no *grid*, evitando que estas informações sejam buscadas a cada vez que uma tarefa precisa ser executada.

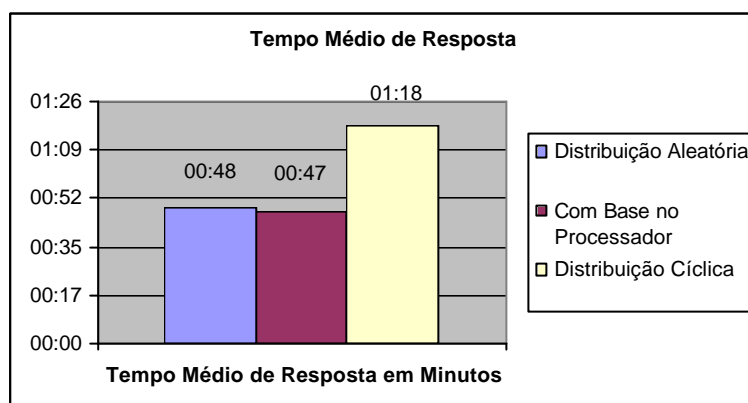


Figura 75 - Tempo médio de resposta na execução das tarefas.

6.13 Conclusões

Este capítulo apresentou detalhes da implementação da arquitetura de *grid* proposta neste trabalho. Embora não tenham sido descritos todos os aspectos, a descrição fornece uma idéia de como a implementação foi conduzida e apresenta exemplos e cenários simples mostrando como a arquitetura pode ser aplicada em ambientes maiores.

Os resultados apresentados mostram como foram implementados os agentes coletores de dados, os armazenadores de informação e os agentes de análise de dados. Também foram construídas regras de análise de dados para serem usadas pelos agentes analisadores. O desenvolvimento destes componentes constitui uma plataforma que pode ser usada na realização de experimentos melhores envolvendo cenários de gerência mais sofisticados, que não estavam ao alcance durante o desenvolvimento deste trabalho.

Pode-se dizer que a implementação cobre todos os aspectos inicialmente descritos, ou seja, os itens mencionados durante a proposta de arquitetura para a gerência de redes e de sistemas. É evidente que alguns detalhes, como segurança na arquitetura, não são tratados neste trabalho pelo fato deste não ser o seu foco primário, embora sejam de extrema importância na aplicação da mesma em condições reais de gerenciamento.

Os agentes de interface implementados são responsáveis por apresentar as informações de gerenciamento ao usuário do sistema e por obter o retorno dele. Isto é feito usando as funcionalidades de comunicação proporcionadas pela plataforma de agentes usada para a validação e tecnologias baseadas em XML com XSL e XSLT.

Uma vantagem da arquitetura de agentes proposta neste trabalho é que ela proporciona uma maneira flexível para a construção de sistemas de gerenciamento. Através da confecção das regras de inferência necessárias, que constituem o conhecimento dos agentes, é possível construir rapidamente qualquer módulo adicional da arquitetura. Ou seja, se for necessário um novo agente coletor para coletar um conjunto de dados específico basta construir as regras necessárias para isso, não necessitando qualquer alteração no código ou estrutura do sistema. Da mesma forma podem ser adicionados novos agentes de análise com regras específicas para tratar um conjunto de dados que foi coletado. A análise dos dados será distribuída entre os agentes com as regras necessárias para processar estas informações, sendo equalizada entre os recursos disponíveis para processá-las.

Além destes fatores, uma abordagem de representação baseada em lógica de primeira ordem tanto do conhecimento de coleta de dados quanto do conhecimento necessário para a análise de dados facilita a migração deste conhecimento entre os nós participantes do *grid*. Os agentes de coleta e de análise também podem importar as regras necessárias para a análise de um conjunto de dados de um ponto estratégico como, por exemplo, um servidor de conhecimento.

Pode-se dizer que a validação da arquitetura obteve sucesso no sentido de proporcionar um conjunto de funcionalidades e componentes que podem ser usados para a construção de sistemas de gerenciamento baseados em *grids* de agentes.

7 Conclusões

Como objetivo geral este trabalho apresentou uma arquitetura baseada em *grids* de agentes para o gerenciamento de redes e sistemas. Analisando os objetivos específicos pode-se dizer que estes foram cumpridos:

- O trabalho procurou apresentar o que são os *grids* de agentes e como estes podem ser aplicados na gerência de redes de computadores e sistemas. Quais os benefícios de tal abordagem e as vantagens no sentido de proporcionar uma melhor utilização de recursos da rede na realização de atividades de gerenciamento.
- A implementação do protótipo de sistema e sua experimentação foram apresentadas em um estágio de desenvolvimento capaz de abordar todos os pontos básicos da arquitetura sugerida no trabalho. Exemplos de uso, resultados do sistema de gerenciamento e da distribuição das atividades de análise de dados são abordadas no âmbito deste trabalho.
- As ferramentas e a infra-estrutura de testes para a arquitetura de *grid* proposta para o gerenciamento foram desenvolvidas. Estas funcionalidades permitem a realização de testes de distribuição e balanceamento da carga das tarefas de análise de dados, as quais são intensivas em termos de recursos que requerem.

A implementação e a experimentação apresentadas neste trabalho abrangem o desenvolvimento dos pontos essenciais da arquitetura sugerida. Porém, é evidente que em um cenário de *grid* muitos outros aspectos devem ser levados em conta, dentre eles os requisitos de segurança que não foram tratados neste trabalho. Outro fato que deve ser lembrado, é que se acredita que a utilização de uma arquitetura complexa pode ser justificável e aplicável em cenários onde o volume de informação e o ambiente gerenciado são muito grandes.

Além disso, a experimentação da arquitetura de *grid* foi realizada no âmbito de uma rede local, que era o ambiente que se tinha à disposição para o desenvolvimento do trabalho e realização dos testes. Embora tenha procurado utilizar protocolos ubíquos e ferramentas multi-plataformas para o desenvolvimento do sistema, este não foi testado em um cenário envolvendo múltiplos domínios administrativos.

Um estudo de caso envolvendo um cenário de composição de inventários e gerência de configuração foi apresentado. Este estudo de caso teve como objetivo apresentar como os módulos desenvolvidos podem ser utilizados na composição de sistemas de gerenciamento, bem como a integração de tecnologias de gerenciamento existentes.

Alguns testes, envolvendo as etapas de análise de informações de gerenciamento e distribuição destas atividades de análise, foram realizados. Tais testes foram realizados utilizando os recursos dos computadores de uma rede local. Estes experimentos apresentam alguns resultados da distribuição proporcionada pela arquitetura. Contudo, como um trabalho futuro, testes mais apurados devem ser conduzidos.

7.1 Principais Contribuições

Este trabalho tem como principal contribuição a apresentação de uma abordagem utilizando *grids* de agentes no gerenciamento de redes e sistemas para proporcionar uma distribuição das tarefas de gerenciamento e prover uma melhor utilização dos recursos da rede usados nestas atividades, tentando resolver problemas do gerenciamento centralizado como a grande quantidade de dados que deve ser manipulada por estes sistemas. Este trabalho teve o seu valor reconhecido durante o seu desenvolvimento através de várias publicações como [12][10][14][13].

Outra contribuição inerente a este trabalho é o de proporcionar uma infra-estrutura baseada em agentes para a construção de sistemas de gerenciamento de redes e sistemas. Esta infra-estrutura pode ser estendida para comportar novas funções de gerenciamento sem alterar o seu princípio básico de funcionamento, e ser usada na construção de *grids* de agentes.

7.2 Trabalhos Futuros

Como trabalhos futuros podem ser destacadas as seguintes atividades:

- A aplicação de agentes móveis no *grid* de processamento, para prover mecanismos que possibilitem o balanceamento de carga, constitui um estudo importante. Os agentes móveis podem migrar para nós com recursos computacionalmente melhores para realizar as atividades de gerenciamento.

- O melhoramento da forma como o armazenamento de dados é realizado pelos agentes armazenadores, visando facilitar a adoção de técnicas de replicação e que diminuam a velocidade de acesso a estas informações.
- O estudo e aplicação de mecanismos de negociação entre agentes no *grid* de análise de informações e no *grid* de armazenamento de dados. Aplicando um modelo de organização baseado em comércio, tais mecanismos podem ser úteis para atingir um equilíbrio e balanceamento de carga nestes *grids*.
- Fazer os ajustes na infra-estrutura de agentes para esta siga os padrões estabelecidos pela **FIPA**, a fim de garantir a interoperabilidade com outras plataformas de agentes.
- A aplicação da arquitetura em um cenário de gerência de rede com um grande número de equipamentos, onde esta possa ser justificada pela presença de um grande volume de dados para processamento.
- A aplicação de tecnologias baseadas em XML no armazenamento e representação dos dados de gerenciamento no intuito de facilitar a sua distribuição e manipulação por diferentes plataformas e por sistemas construídos em diferentes linguagens.
- O estudo, desenvolvimento e aplicação de agentes no *grid* com finalidades especiais como *brokers*, *traders* e *matchmakers*, para melhorar as capacidades de pesquisa, e diminuir o tráfego de mensagens que pode existir no ambiente de *grid* de agentes proposto.
- Inclusão de mecanismos de segurança, como autenticação e confidencialidade das mensagens trocadas. Como o *grid* de agentes pode envolver múltiplos domínios administrativos, a existência destas funcionalidades é essencial.
- O estudo e desenvolvimento de ontologias a serem associadas aos dados armazenados pelo *grid* de classificação e armazenamento.
- Um trabalho futuro importante a ser realizado é a integração da arquitetura de *grid* de agentes com *middlewares* e outras ferramentas existentes para a computação em *grid*. Como exemplo pode ser citado a integração da arquitetura de agentes com os componentes da ferramenta *Globus*, usando componentes como o diretório de informações de recursos, o *broker* de recursos oferecido por esta plataforma, assim como protocolos como GridFTP

e outras. Outra tecnologia de rede que pode ser explorada neste contexto é o Java JINI, a qual é também usada pelo projeto CoABS [28].

- No escopo do gerenciamento, o desenvolvimento de protótipos melhores de gerência, assim como a criação de regras de inferência para compor as bases de conhecimento dos agentes do sistema também caracteriza um trabalho de extrema importância.
- A adoção de tecnologias baseadas em XML para a representação do conhecimento e dos objetivos dos agentes do *grid* também é um trabalho futuro a ser realizado. Uma abordagem baseada em XML pode facilitar e proporcionar um ganho de tempo no desenvolvimento dos agentes que integram o sistema.
- Outro trabalho futuro a ser realizado consiste em efetuar uma análise do tráfego de mensagens ocasionado pela distribuição das tarefas de gerenciamento. À medida que aumenta a distribuição, aumenta também o número de nós no *grid* de agentes e também o número de mensagens trocadas entre estes elementos. Neste caso, a partir de uma determinada situação, o *grid* de agentes não será uma estrutura viável devido à alta utilização da capacidade de transmissão da rede. Uma análise do tráfego gerado e em que ponto a arquitetura deixa de ser uma vantagem, caracteriza um importante trabalho futuro a ser realizado.
- Realizar estudos no sentido de garantir tolerância à falhas e a alta disponibilidade da arquitetura. Em um ambiente de *grid* de agentes, devem existir garantias de que a plataforma funcionará em casos de falhas, e dos mecanismos que possam garantir esta disponibilidade e funcionamento, mesmo quando estas condições acontecerem. Embora algumas funcionalidades tenham sido adicionadas durante o desenvolvimento deste trabalho, um estudo mais detalhado, com uma profunda investigação destes mecanismos se faz necessária.

Referências

- [1] AAMAS, 1st International Workshop on Challenges in Open Agent Systems. Bologna, Itália, jul. 2002. Sítio do evento contendo informações e artigos publicados. Disponível em: <<http://www.agentcities.org/Challenge02/>>. Acesso em: 03 mar. 2003.
- [2] AGENT GRID, Laboratório de Redes e Gerência. Sítio com informações sobre o projeto. Disponível em: <<http://agentgrid.lrg.ufsc.br/>>.
- [3] AGENTLIGHT – PLATFORM FOR LIGHTWEIGHT AGENTS. Sítio com informações sobre o projeto e o software para baixar. Disponível em: <<http://www.agentlight.org>>.
- [4] ALLSOPP, D. N.; BEAUTEUMENT, P.; BRADSHAW, J. M.; DURFEE, E. H.; KIRTON, M.; KNOBLOCK, C. A.; SURI, N.; TATE, A.; THOMPSON, C. W. Coalition Agents Experiment: Multi-Agent Co-operation in an International Coalition Setting. Special Issue on Knowledge Systems for Coalition Operations (KSCO), IEEE Intelligent Systems, vol.17 num.3 p.26-35. may/jun. 2002.
- [5] ALVARENGA, A. T. H. Um Ambiente Para Processamento Paralelo Oportunístico na Internet. 2003, 186f. Dissertação de Mestrado, Universidade de Brasília.
- [6] APACHE REGEXP. Sítio do projeto. Disponível em: <<http://jakarta.apache.org/regexp/>>. Acesso em: 02 fev. 2003.
- [7] APACHE. The Apache Software Foundation. Sítio do projeto. Disponível em: <<http://www.apache.org>>. Acesso em: 05 mar. 2002.
- [8] ARANTES, J. A.; WESTPHALL, C. B.; CUSTÓDIO, R. F. Modelo Analítico para Avaliar Plataformas Cliente/Servidor e Agentes Móveis Aplicado à Gerência de Redes. Simpósio Brasileiro de Redes de Computadores, 20., 2002, Búzios-RJ. **Anais...** Simpósio Brasileiro de Redes de Computadores, Vol. II. p.424-439. maio 2002.
- [9] ASSUNÇÃO, M. D.; KOCH, F. L. & WESTPHALL, C. B. Grids of Autonomous Agents based in Virtual Organizations for Network Management Activities. **Artigo Submetido**, ago. 2003.
- [10] ASSUNÇÃO, M. D.; KOCH, F. L.; WESTPHALL, C. B. Vantagens do Uso de Grids de Agentes no Gerenciamento de Redes de Computadores. Congresso

- Brasileiro de Computação, 3., 2003, Itajaí-SC. **Anais...** III - Congresso Brasileiro de Computação - Agent's Day. Agosto de 2003, ISSN: 1677-2822.
- [11] ASSUNÇÃO, M. D.; SOBRAL, J. B. M.; WESTPHALL, C. B. Agentes Móveis na Gerência de Redes. I2TS–International Information Technology Symposium, 2002, Florianópolis – SC, Brasil. **Resumos...** I2TS, Florianópolis – SC, Brasil, Out. 2002, p.15.
- [12] ASSUNÇÃO, M. D.; WESTPHALL, C. B.; KOCH, F. L. Arquitetura de *Grids* de Agentes Aplicada à Gerência de Redes de Computadores e Telecomunicações. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 21., 2003, Natal-RN. **Anais...** Natal: 21º Simpósio Brasileiro de Redes de Computadores, 2003, p. 789-804.
- [13] ASSUNÇÃO, M. D.; WESTPHALL, C. B.; KOCH, F. L. Grids of Agents for Computer and Telecommunication Network Management. In: *Acm/Ifip/Usenix International Middleware Conference. International Workshop On Middleware For Grid Computing*, 1., Rio de Janeiro, **Proceedings...** Rio de Janeiro: International Middleware Conference - 1st Workshop on Middleware for Grid Computing, jun, 2003. p.186-193.
- [14] ASSUNÇÃO, M. D.; KOCH, F. L., WESTPHALL, C. B. Grids of Agents for Computer and Telecommunication Network Management. **Journal of Concurrency and Computation: Practice and Experience, John Wiley & Sons, Inc. (A ser publicado na edição de: março/abril de 2004).**
- [15] BEER M.; D'INVERNO M.; LUCK M.; JENNINGS N. R.; PREIST C; SCHROEDER M. Negotiation in Multi-Agent Systems. *Knowledge Engineering Review*. vol.14, num.3, 1999, p.285-289.
- [16] BERMAN, F.; FOX, G.; HEY, T. The Grid: Past, Present, Future. *Grid Computing: Making the Global Infrastructure a Reality*. Editores: BERMAN, F.; FOX, G.; HEY, T. [S.l.]: Wiley and Sons, mar. 2003. A special issue of *Concurrency and Computation: Practice and Experience*.
- [17] BOINC. Berkeley Open Infrastructure for Network Computing, 2002. Sítio do projeto. Disponível em: <<http://boinc.berkeley.edu>>. Acesso em: 10 dez. 2002.
- [18] BRADSHAW, J. M. *Software Agents*. [Menlo Park, Calif]: The AAI Press, 490p. ISBN 0-262-52234-9.

- [19] BRENNER, W.; ZARNEKOW R.; WITTIG H. Intelligent Software Agents: Foundations and Applications. Tradução para inglês de RUDD, A. S. Berlin, Heidelberg: Springer-Verlag, 1998. ISBN 3-540-63411-8.
- [20] BUYYA, R. Economic-based Distributed Resource Management and Scheduling for Grid Computing. 2002, 166f. Tese de PhD. School of Computer Science and Software Engineering Monash University, Melbourne, Australia, 2002.
- [21] BUYYA, R. High Performance Cluster Computing: Programming and Applications. NJ, USA: Prentice Hall PTR, 1999. ISBN 0-13-013785-5.
- [22] CANNATARO, M.; TALIA, D. The Knowledge Grid. Communications of ACM. vol.46 num.1, p.89-93, jan. 2003.
- [23] CASANOVA, H. Distributed computing research issues in grid computing. ACM SIGACT News, New York, NY, USA: ACM Press, v.33, n.3, p.50-70, set. 2002. ISSN:0163-5700.
- [24] CASAVANT T. L.; KUHL J. G. A Taxonomy of Scheduling in General-Purpose Distributing Computing System. IEEE Transactions on Software Engineering, v.14, n.2, p.141-154, fev. 1988.
- [25] CATLETT, C. Standards for Grid Computing: Global Grid Forum. Journal of Grid Computing, v.1, n.1, p.3-7, 2003.
- [26] CHAPPEL, D. JEWELL, T. Java Web Services. 1st ed. [S.l.]: O'Reilly, mar. 2002. 276 p. ISBN: 0-596-00269-6.
- [27] CHETTY, M.; BUYYA, R. Weaving Computational Grids: How Analogous Are They with Electrical Grids? IEEE Computing in Science and Engineering Magazine, v.4, n.4, p.61-71, jul./ago. 2002.
- [28] COABS – CONTROL OF AGENT BASED SYSTEMS. Sítio com informações sobre o projeto. Disponível em: < <http://coabs.globalinfotek.com>>. Acesso em: 28 set. 2002.
- [29] COABS GRID - USER MANUAL. Manual do usuário contendo informações sobre operação do *grid* CoABS e instrumentação fornecida pelo software do *grid*. Disponível em: <<http://coabs.globalinfotek.com/public/downloads/Grid/documents/GridUsersManual.v5.0.doc>>. Acesso em: 03 mar. 2003.

- [30] COSTA, T. F. S. Avaliação Analítica Do Uso De Agentes Móveis Na Gerência De Redes, 1999, 115f. Dissertação de Mestrado (Curso de Pós Graduação em Ciência da Computação). Universidade Federal de Santa Catarina.
- [31] DAVIDSONN, P. Concept Acquisition by Autonomous Agents: Cognitive Modeling versus the Engineering Approach. Lund University Cognitive Studies 12. [S.l.]: Lund University, Suécia, 1992. ISSN 1101-8453.
- [32] DE ROURE, D.; JENNINGS, N. R.; SHADBOLT, N. R. The Semantic Grid: A Future e-Science Infrastructure, p.437-470. In: Grid Computing: Making The Global Infrastructure a Reality. [S.l.]: John Wiley & Sons, apr. 2003, ISBN: 0470853190.
- [33] DESIC, SASA; HULJENIC, DARKO. Agents Based Load Balancing with Component Distribution Capability. IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02), 2., may 2002, Berlin, Germany. **Proceedings...** CCGRID'02, p. 354-358.
- [34] DIETZ, H. G. Heterogeneous Parallel Computing with Java: Jabber or Justified? Heterogeneous Computing Workshop, 1998, Orlando, FL, USA. **Proceedings...** Heterogeneous Computing Workshop, mar. 1998, p.159-162.
- [35] DISTRIBUTED.NET PROJECT, 2002. Sítio do projeto contendo informações sobre ele, andamento das atividades relacionadas e o software para ser baixado. Disponível em: <<http://www.distributed.net>>. Acesso em: 01 ago. 2002.
- [36] DMTF. Distributed Management Task Force, Inc. Sítio com informações sobre a organização e especificações produzidas. Disponível em: <<http://www.dmtf.org>>. Acesso em: 15 set. 2002.
- [37] DMTF. Web-Based Enterprise Management (WBEM) Initiative. Disponível em: <<http://www.dmtf.org/standards/wbem>>. Acesso em: 02 mar. 2003.
- [38] ECHELON - AGENT BASED GRID COMPUTING ARCHITECTURE. Sítio com informações sobre o projeto. Disponível em: <<http://www.geocities.com/echelongrid>>. Acesso em: 6 jun. 2003.
- [39] EDWARDS, W. K. Core Jini. [S.l.]: Addison Wesley, 1999. 832 p.
- [40] FAFNER-FACTORING VIA NETWORK-ENABLED RECURSION. Disponível em: <<http://www.lehigh.edu/~bad0/fafner.html>>. Acesso em: 15 jan. 2003.

- [41] FERBER, J. Multi-Agent System: An Introduction to Distributed Artificial Intelligence. [Reading, Massachusetts]: Addison Wesley Longman, 1999, 496 p. ISBN 0-201-36048-9.
- [42] FIPA - The Foundation For Intelligent For Physical Agent. Sítio da organização com informações sobre ela e as especificações produzidas. Disponível em: <<http://www.fipa.org>>. Acesso em: 04 mar. 2002.
- [43] FIPA. Abstract Architecture Specification. Disponível em: <<http://www.fipa.org/specs/fipa00001/>>. Acesso em: 15 ago. 2002.
- [44] FIPA. ACL Message Structure Specification. Disponível em: <<http://www.fipa.org/specs/fipa00061/>>. Acesso em: 26 set. 2002.
- [45] FIPA. Agent Management Specification. Disponível em: <<http://www.fipa.org/specs/fipa00023/>>. Acesso em: 15 ago. 2002.
- [46] FIPA. Agent Message Transport Protocol for HTTP Specification. Disponível em: <<http://www.fipa.org/specs/fipa00084/>>. Acesso em: 10 nov. 2002.
- [47] FIPA. Agent Software Integration Specification. Disponível em: <<http://www.fipa.org/specs/fipa00079/>>. Acesso em: 05 jan. 2003.
- [48] FIPA. Communicative Act Library Specification. Disponível em: <<http://www.fipa.org/specs/fipa00037/>>. Acesso em: 06 set. 2002.
- [49] FIPA. Interaction Protocols Specifications. Informações e especificações dos protocolos de interação da FIPA. Disponível em: <<http://www.fipa.org/repository/ips.php3>>. Acesso em: 01 jun. 2002.
- [50] FIPA-OS V2.2.0 Distribution Notes. Disponível em: <<http://www.emorphia.com/research/about.htm>>. Acesso em: 04 ago. 2003.
- [51] FOSTER I.; IAMNITCHI, A. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. International Workshop on Peer-to-Peer Systems (IPTPS'03), 2., feb. 2003, Berkeley, CA. **Proceedings...** 2nd International Workshop on Peer-to-Peer Systems.
- [52] FOSTER, I.; GEISLER, J.; NICKLESS, W.; SMITH, W.; TUECKE S. Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment. IEEE Symposium on High Performance Distributed Computing, 5., 1997, Syracuse-NY. **Proceedings...** 5th IEEE Symposium on High Performance Distributed Computing. 1997, p.562-571.

- [53] FOSTER, I.; KESSELMAN, C. The Grid: Blueprint for a New Computing Infrastructure. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1999. 677p.
- [54] FOSTER, I.; KESSELMAN, C. Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, v.11, n.2, p.115-128, 1997.
- [55] FOSTER, I.; KESSELMAN, C.; NICK, J.; TUECKE, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, jun. 2002.
- [56] FOSTER, I.; KESSELMAN, C.; TUECKE S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications, v.15 n.3, 2001.
- [57] FRANKLIN, S., GRAESSER, A. Is It an Agent or Just a Program? A Taxonomy for Autonomous Agents. International Workshop on Agent Theories, Architectures, and Languages, 3., New York, 1996. **Proceedings...** Third International Workshop on Agent Theories, Architectures, and Languages. New York: Springer-Verlag, 1996.
- [58] FULLER, W. Network management using expert diagnostics. International Journal of Network Management, vol.9, num.4, 1999, p.199-208.
- [59] GLOBAL INFOTEK, INC. AND ISX CORPORATION. The CoABS Grid: Technical Vision. Disponível em:
<[http://coabs.globalinfotek.com/public/downloads/Grid/documents/Grid Vision Doc Draft 2-3 2001-09-30.doc](http://coabs.globalinfotek.com/public/downloads/Grid/documents/Grid_Vision_Doc_Draft_2-3_2001-09-30.doc)>. Acesso em: 10 jan. 2003.
- [60] GRAND CHALLENGING APPLICATIONS. Disponível em
<<http://www.mcs.anl.gov/Projects/grand-challenges/>>. Acesso em: 28 jun. 2003.
- [61] GRIVAS, M.; TURNER, S. J. Agent Technology in Load Balancing for Network Applications. International Workshop on Intelligent Agents on the Internet and Web, 1998, Mexico. **Proceedings...** 4 World Congress on Expert Systems, México, mar. 1998, 13 p.
- [62] GRUBER T. R. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal of Human-Computer Studies, v.43, n.5-6, p.907-928, 1995.

- [63] HOPE - "Hot Topics" em Gerência de Redes. Laboratório de Redes e Gerência - LRG. Documento contendo informações e áreas abrangidas pelo projeto. Disponível em: <<http://www.lrg.ufsc.br/page/hope.pdf>>.
- [64] IETF RFC 2790 - Host Resources MIB. Disponível em: <<http://www.ietf.org/rfc/rfc2790.txt?number=2790>>. Acesso em: 30 mar. 2003.
- [65] INTERNET PIONEER. J.C.R. Licklider, Disponível em: <<http://www.ibiblio.org/pioneers/licklider>>. Acesso em: 24 fev. 2003.
- [66] ISC - INTERNET SOFTWARE CONSORTIUM. Internet Domain Survey. Disponível em: <<http://www.isc.org/ds>>. Acesso em: 20 set. 2002.
- [67] JCHARTS - 100% Java based charting utility. Disponível em: <<http://jcharts.sourceforge.net/>>. Acesso em: 04 set. 2003.
- [68] JEFFERY, K. G. Knowledge, Information and Data, A briefing to the Office of Science and Technology. Disponível em: <www.itd.clrc.ac.uk/Publications/1433/KnowledgeInformationData20000124.htm>. Acesso em: 02 set. 2002.
- [69] JENNINGS, N. R. An agent-based approach for building complex software systems. *Communications of the ACM*, v.44, n.4, p.35-41, 2001.
- [70] JUHASZ, Z.; ANDICS, A.; POTA, S. JM: A Jini Framework for Global Computing. International Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems, 2., may 2002, Berlin, Germany. **Proceedings...** IEEE International Symposium on Cluster Computing and the Grid (CCGrid'2002), may 2002, Berlin, Germany.
- [71] KAHN M. L.; CICALESE C. D. T. CoABS Grid Scalability Experiments. *Autonomous Agents and Multi-Agent Systems*. Vol.7, num.1-2, p.171-178, 2003.
- [72] KOCH, F. L. Agentes Autônomos para Gerenciamento de Redes de Computadores. 1997. Dissertação de mestrado (Curso de Pós Graduação em Ciência da Computação), Universidade Federal de Santa Catarina.
- [73] KOCH, F. L.; MEYER, J.J. C. Knowledge Based Autonomous Agents for Pervasive Computing using AgentLight. ACM/IFIP/USENIX International Middleware Conference, MIDDLEWARE 2003 WORK-IN-PROGRESS, 2003,

- Rio de Janeiro. **Proceedings...** IEEE Distributed Systems. Disponível em: <<http://dsonline.computer.org/0306/f/koc.htm>>, Rio de Janeiro, Brazil, 2003.
- [74] KOCH, F. L.; WESTPHALL, C. B. Decentralized Network Management using Distributed Artificial Intelligence. *Journal of Network and Systems Management*. Plenum Publishing Corporation. Meddletown, USA, v.9, n.4, p.291-313, dez. 2001.
- [75] KOCH, F. L.; WESTPHALL, C. B.; ASSUNÇÃO, M. D.; XAVIER, E.; SYPERREK B. Advances in Distributed Artificial Intelligence for Network Management Systems. *Artigo Submetido*, 2003.
- [76] LANGE, B. D.; OSHIMA, M. Seven Good Reasons for Mobile Agents. *Communications of ACM*, vol.42, num.3. p.88-89. mar. 1999.
- [77] LEGION. A Worldwide Virtual Computer, 1997. *Sítio do projeto Legion*. Disponível em: <<http://legion.virginia.edu>>. Acesso em: 30 set. 2002.
- [78] LIM CHOI KEUNG, H. N.; CAO, J.; SPOONER, D. P.; JARVIS, S. A.; NUDD, G. R. Grid Information Services using Software Agents. *Annual UK Performance Engineering Workshop (UKPEW' 2002)*, 18., 2002, University of Glasgow, UK. **Proceedings...** UKPEW' 2002, jul. 2002, p. 187-198.
- [79] LINPACK BENCHMARK -- Java Version. Disponível em: <<http://www.netlib.org/benchmark/linpackjava/>>. Acesso em: 02 jun 2003.
- [80] LUCK, M., MCBURNEY, P., PREIST, C. *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*, AgentLink, 2003, ISBN 0854 327886.
- [81] MANOLA, F. *Characterizing Computer-Related Grid Concepts*, 1999. Disponível em: <<http://www.objs.com/agility/tech-reports/9903-grid-report-fm.html>>. Acesso em: 25 out. 2002.
- [82] MANOLA, F., THOMPSON, C. *Characterizing the Agent Grid*. For: *Bradshaw's Handbook of Agent Technology*. Disponível em: <<http://www.objs.com/agility/tech-reports/000304-characterizing-the-agent-grid.doc>>. Acesso em: 10 set. 2002.
- [83] METCALFE R.; BOGGS, D. Ethernet: Distributed Packet Switching for Local Computer Networks, *Proceedings of ACM National Computer Conference* vol.19, num.5, July 1976.

- [84] MICROSOFT. WMI: Introduction to Windows Management Instrumentation. Visão geral sobre a instrumentação e gerenciamento da plataforma Window. Disponível em: <<http://www.microsoft.com/whdc/hwdev/driver/WMI/WMI-intro.msp>>. Acesso em: 24 fev. 2003.
- [85] MOORE, G. E. Cramming More Components Onto Integrated Circuits. Electronics. abril 19, 1965. Disponível em: <<ftp://download.intel.com/research/silicon/moorespaper.pdf>>. Acesso em: 19 abr. 2003.
- [86] MOREAU, L. Agents for the Grid: A Comparison for Web Services (Part 1: the transport layer). Editores: BAL, H. E.; LOHR, K-P; REINEFELD, A. IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2002), 2., 2002, Berlin, Germany. Proceedings... CCGRID Germany, p.220-228, May 2002.
- [87] MOREAU, L.; GIBBINS, N.; DE ROURE, D.; EL-BELTAGY, S.; HALL, W.; HUGHES, G.; JOYCE, D.; KIM, S.; MICHAELIDES, D.; MILLARD, D.; REICH, S.; TANSLEY, R.; WEAL, M. SoFAR with DIM Agent: An Agent Framework for Distributed Information Management. International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Systems, 5., 2000, Manchester, UK. **Proceedings...** International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Systems, apr. 2000, p.369-388.
- [88] MULLER, J. P. The Design of Autonomous Agents: A Layered Approach. Lecture Notes in Artificial Intelligence, vol.1177, Springer-Verlag, 1996.
- [89] MYGRID, Sítio com informações sobre o projeto. Disponível em: <<http://www.mygrid.org.uk>>. Acesso em: 12 dez. 2002.
- [90] MYSQL. Sítio do projeto. Disponível em: <<http://www.mysql.com/>>. Acesso em: 01 dez. 2001.
- [91] NATIONAL COORDINATION OFFICE FOR INFORMATION TECHNOLOGY RESEARCH AND DEVELOPMENT. High Performance Computing And Communications: Foundation For America's Information Future. 1996. The Annual Supplement to the President's Budget. Disponível em: <<http://www.ccic.gov/pubs/blue96>>. Acesso em: 05 abr. 2003.

- [92] NETRAVALI, A. N. The Impact of Solid State Electronics on Computing and Communications. [1997]. [S.l]: Bell Labs Journals. Disponível em: <http://www.bellsystemmemorial.com/pdf/bell_labs_journals/paper07.pdf>. Acesso em: 25 mar. 2003.
- [93] NWANA, H. S. Software Agents: An Overview. Knowledge Engineering Review, v.11, n.3, p.1-40, set. 1996.
- [94] NWANA, H. S.; NDUMU, D. An Introduction to Agent Technology. Intelligent Systems Research, Applied Research and Technology, BT Labs. Disponível em: <http://more.btexact.com/projects/agents/publish/papers/intro_agents.htm>. Acesso em: 25 out. 2002.
- [95] PFISTER, G. F. In Search of Clusters. 2.ed., Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN 0-13-899709-8.
- [96] PORTER, J. N. Disk Drives' Evolution. DISK/TREND Given at the 100th Anniversary Conference on Magnetic Recording and Information Storage. Santa Clara University, dec. 1998. Disponível em: <<http://www.disktrend.com/pdf/portrpkg.pdf>>. Acesso em: 17 mai. 2003.
- [97] RANA, O. F.; WALKER D. W. The Agent Grid: Agent based resource Integration in Problem Solving Environments, IMACS World Congress On Scientific Computation, 16., 2000, Lausanne, Switzerland. **Proceedings...** Lausanne, Switzerland: Applied Mathematics and Simulation, 2000.
- [98] RANA, O.; MOREAU, F. Issues in Building Agent-Based Computational Grids. Workshop of the UK Special Interest Group on Multi-Agent Systems (UKMAS' 2000), 3., 2000, Oxford, UK. **Proceedings...** UKMAS' 2000.
- [99] RFC2790. Host Resources MIB. The Internet Engineering Task Force. Disponível em: <<http://www.ietf.org/rfc/rfc2790.txt?number=2790>>. Acesso em: 07 abr. 2002.
- [100] RIPEANU, M.; FOSTER, I; IAMNITCHI, A. Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. IEEE Internet Computing, Special issue on Peer-to-Peer Networking, v.6 n.1, p.50-57, fev. 2002. Disponível em: <<http://people.cs.uchicago.edu/~anda/papers/ic.pdf>>

- [101] ROCHA, M. A, WESTPHALL, C. B. Pro-active Management of Computer Networks Using Artificial Intelligence Agents and Techniques. IFIP/IEEE International Symposium on Integrated Network Management, 5., 1997, San Diego, USA. Proceedings... Fifth IFIP/IEEE International Symposium on Integrated Network Management, p.610-621, 1997.
- [102] ROURE, D.; BAKER, M.; JENNINGS, N. R.; SHADBOLT, N. The evolution of the Grid. International Journal of Concurrency and Computation: Practice and Experience, v.15, n.11, 2003.
- [103] SEARLE, J. R. Speech Acts: an Essay in the Philosophy of Language. [S.l]: Cambridge University Press, Cambridge [1969].
- [104] SELMAS 2002. 1st International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2002). May 19, 2002, Orlando, Florida, USA. Disponível em: <<http://www.teccomm.les.inf.puc-rio.br/selmas2002/>>. Acesso em: 05 jan. 2003.
- [105] SELMAS 2003. 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2003). May 3-4, 2003. Portland, Oregon, USA. Disponível em: <<http://www.teccomm.les.inf.puc-rio.br/selmas2003/>>. Acesso em: 01 jun. 2003.
- [106] SETI@HOME. Search for Extraterrestrial Intelligence, 1996. Sítio do projeto. Disponível em: <<http://setiathome.ssl.berkeley.edu>>. Acesso em: 01 ago. 2002.
- [107] SHEN, W.; LI, Y.; GHENNIWA, H. H.; WANG, C. Adaptive Negotiation for Agent-Based Grid Computing. AAMAS 1st International Workshop on Challenges in Open Agent Systems, 2002, Bologna, Italy. **Proceedings...** AAMAS 1st International Workshop on Challenges in Open Agent Systems, Bologna, Italy, jul. 2002.
- [108] SHI, Z.; DONG, M.; ZHANG, H.; SHENG, Q. Agent-based Grid Computing. International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES 2002), 2002, Wuxi. **Proceedings...** Wuxi: DCABES 2002.
- [109] SHIRKY, C. What Is P2P... and What Isn't? 2000. Disponível em: <www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html>. Acesso em: 20 jun. 2003.

- [110] SHOHAM, Y. An Overview of Agent-oriented Programming. Software Agents, Menlo Park, Calif.: AAAI Press, Edited by J. M. Bradshaw, 1997.
- [111] STALLINGS, W. SNMP, SNMPv2, SNMPv3, and RMON 1 and 2. Reading, Massachusetts: Addison Wesley Longman, Inc, 1999. ISBN: 0-201-48534-6.
- [112] SUBRAMANIAN, M. Network management: An introduction to principals and practice. [Reading, Massachusetts]: Addison-Wesley, 2000. 644p, ISBN: 0-201-35742-9.
- [113] SYCARA, K. Multiagent Systems. AI Magazine, Intelligent Agents, v.19, n.2, p.79-92, 1998.
- [114] SYSTEMS MANAGEMENT SERVER 2003 REVIEWER'S GUIDE. Microsoft Corp. Sep. 2002. Disponível em: <<http://www.microsoft.com>>. Acesso em: Ago. 2003.
- [115] TAGERE - Tópicos Avançados em Gerência de Redes de Computadores e Telecomunicações. Laboratório de Redes e Gerência - LRG. Documento contendo informações e áreas abrangidas pelo projeto. Disponível em: <<http://www.lrg.ufsc.br/page/tagere.pdf>>.
- [116] THE GRID – JINIGRID. Sítio com informações sobre o projeto. Disponível em: <http://www.epcc.ed.ac.uk/computing/research_activities/grid/jinigrd/>. Acesso em: 01 mar. 2003.
- [117] THE SEMANTIC GRID. Sítio contendo informações sobre a iniciativa. Disponível em: <<http://www.semanticgrid.org/>>. Acesso em: 03 mar. 2003.
- [118] UNITED DEVICES INC, 2002. Sítio da organização com informações sobre a organização, seus produtos e sobre soluções baseadas em *grid*. Disponível em: <<http://www.ud.com/home.htm>>. Acesso em: 05 ago. 2002.
- [119] VASUDEVAN, V.; FORD, S.; THOMPSON, C. eGents: Agents communicating via Email. Sítio com informações sobre o projeto. Disponível em: <<http://www.objs.com/agility/final/eGents/EGENTS-PROJECT-SUMMARY.html>>. Acesso em: 23 mar. 2003.
- [120] VON LASZEWSKI, G.; PIEPER, G.; WAGSTROM, P. Gestalt of the Grid. Disponível em: <<http://phase.hpcc.jp/mirrors/globus/cog/documentation/papers/gestalt.pdf>>. Acesso em: 02 mai. 2003.

- [121] W3C, Extensible Markup Language (XML). Disponível em: <<http://www.w3.org/XML>>. Acesso em: 21 jul. 2002.
- [122] W3C. The Extensible Stylesheet Language Family (XSL). Disponível em: <<http://www.w3.org/Style/XSL/>> Acesso em: 01 set. 2003.
- [123] WIJNGAARDS, N. J. E.; OVEREINDER, B. J.; STEEN, M. V.; BRAZIER, F. M. T. Supporting Internet-Scale Multi-Agent Systems. *Data and Knowledge Engineering*. v.41, n.2-3, p.229-245, Jun. 2002.
- [124] WILLMOTT, S. N., DALE, J., BURG, B., CHARLTON, C., O'BRIEN, P. Agentcities: A Worldwide Open Agent Network. *Agentlink News*. nov. 2001, p.13-15. Disponível em: <<http://www.AgentLink.org/newsletter/8/AL-8.pdf>>. Acesso em: 30 mar. 2003.
- [125] WOOLDRIDGE, M. J. *An Introduction to Multiagent Systems*. Baffins Lane, Chichester, West Sussex, England: John Wiley & Sons Ltd., 2001. 348p. ISBN 0-471-49691-X.
- [126] WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, v.10, n.2, p.115-152, 1995.
- [127] W-PROLOG. Disponível em: <<http://goanna.cs.rmit.edu.au/~winikoff/wp/>> Acesso em: 1 abr. 2002.
- [128] XAVIER, E.; WESTPHALL, C. B.; KOCH, F. L. Uso de Inteligência Computacional na Escolha Automatizada da Melhor Configuração de Agentes Móveis para a Gerência de Redes de Computadores. ENIA – Encontro Nacional de Inteligência Artificial, 4., 2003, Campinas, SP. **Anais...** XXIII Congresso da SBC - ENIA – IV Encontro Nacional de Inteligência Artificial, ago. 2003.

Anexos

Anexo I – DTD dos Arquivos de Definição de Nós do *Grid*

```

<!ELEMENT grid (
  DEBUG|classpath|communicator|kdb|agent|trace|node|load|execute|schedule)*>
<!ELEMENT DEBUG EMPTY>
<!ATTLIST DEBUG
  action CDATA #REQUIRED
  name CDATA #REQUIRED>
<!ELEMENT trace EMPTY>
<!ATTLIST trace
  name CDATA #REQUIRED
  level CDATA #REQUIRED>
<!ELEMENT agent (kdb?, execute?, schedule?)>
<!ATTLIST agent
  name CDATA #REQUIRED>
<!ELEMENT classpath EMPTY>
<!ATTLIST classpath
  name CDATA #REQUIRED>
<!ELEMENT communicator (listener?, route?)>
<!ELEMENT listener (#PCDATA)>
<!ATTLIST listener
  port CDATA #REQUIRED>
<!ELEMENT route (#PCDATA)>
<!ATTLIST route
  name CDATA #REQUIRED>
<!ELEMENT execute (#PCDATA)>
<!ELEMENT node (communicator?,kdb?,meter?,agent?,kdb?,execute?,schedule?)>
<!ATTLIST node
  name CDATA #REQUIRED>
<!ELEMENT kdb (#PCDATA)>
<!ELEMENT meter (url?,sample?,interval,community?,problemsize?)>
<!ATTLIST meter
  name CDATA #REQUIRED>
<!ELEMENT interval (#PCDATA)>
<!ELEMENT community (#PCDATA)>
<!ELEMENT load (#PCDATA)>
<!ELEMENT problemsize (#PCDATA)>
<!ELEMENT sample (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT schedule (#PCDATA)>
<!ATTLIST schedule
  name CDATA #REQUIRED
  time CDATA #REQUIRED>

```

Anexo II – Exemplo de Arquivo de Definição de Um Nó Grid

```

<?xml version="1.0"?>

<!--
  Configuration for an example of simple analysis agent.
  @author Marcos Assuncao
          Project Tagere-F
          Laboratorio de Redes e Gerencia - CTC
          Universidade Federal de Santa Catarina - Brasil
  @version 1.0 -->

<DEBUG action="echo" name==" analyze-example-001.xml v0.1 =="/>
<meter name="br.ufsc.lrg.agentgrid.meter.SNMPMeter">
  <sample>10</sample>
  <interval>30</interval>
  <problemsize>500</problemsize>
  <community>public</community>
</meter>

<kdb>
  directory('df_root@root').
  storeURL("store_agent@storage").
</kdb>

<agent name="analysis_agent">
<kdb>
  date(Cmd,Arg1) :-
    !br.ufsc.lrg.agentgrid.skill.DateTimeSkill(Cmd,Arg1).
  date(Cmd,Arg1,Arg2) :-
    !br.ufsc.lrg.agentgrid.skill.DateTimeSkill(Cmd,Arg1,Arg2).
  date(Cmd,Arg1,Arg2,Arg3) :-
    !br.ufsc.lrg.agentgrid.skill.DateTimeSkill(Cmd,Arg1,Arg2,Arg3).
  math(Cmd,Arg1,Arg2) :-
    !br.ufsc.lrg.agentgrid.skill.MathSkill(Cmd,Arg1,Arg2).
  math(Cmd,Arg1,Arg2,Arg3) :-
    !br.ufsc.lrg.agentgrid.skill.MathSkill(Cmd,Arg1,Arg2,Arg3).
  math(Cmd,Arg1,Arg2,Arg3,Arg4) :-
    !br.ufsc.lrg.agentgrid.skill.MathSkill(Cmd,Arg1,Arg2,Arg3,Arg4).

  analyzeProcessorUsage(System) :-
    storeURL(Store),
    date(now,Date),
    date(string,Date,DateString),
    tree(create,TreeReport,"disk_usage"),
    tree(add,TreeReport,"system",System),
    agent(query-ref,Store,

# Rules for building a availability report of a given system.
# It is provided an initial and a final dates that the
# agent will use for ask the store for the necessary data
# for building this report.
verifyAvailability(system,System,InitialDate,FinalDate) :-
  storeURL(Store),
  date(diff,FinalDate,InitialDate,TotalTime),
  tree(create,TreeReport,"availability"),
  tree(add,TreeReport,"system",System),
  tree(add,TreeReport,"initialdate",InitialDate),
  tree(add,TreeReport,"finaldate",FinalDate),
  agent(query-ref,Store,
        getOutages(System,InitialDate,FinalDate,ListOutages)),
  eq('0',TimeOutage),
  sumOutages(ListOutages,TimeOutage),
  calculateAvailability(TimeOutage,TotalTime,TreeReport),

```

```

tree(string,TreeReport,SerialReport),
storeReport(System,InitialDate,"accumulative",
            'system_avail_daily',SerialReport),!.

# for summing the outage time.
sumOutages([],TimeOutage) :- !.

sumOutages(ListOutages,TimeOutage) :-
    list(head,ListOutages,[Begin,End]),
    date(diff,End,Begin,Time),
    __plus(TimeOutage,TimeOutage,Time),
    list(tail,ListOutages,RestOfList),
    sumOutages(RestOfList,TimeOutage),!.

# rule just for calculating the percentage
# of availability of the requested system
calculateAvailability(TimeOut,TotalTime,TreeReport) :-
    __minus(TimeUp,TotalTime,TimeOut),
    math(percentage,TimeUp,TotalTime,Percentage),
    tree(add,TreeReport,"availability",Percentage),!.

# this rule is used for sending the generated report to
# the storage agent, which will perform the storing of this information
storeReport(System,Time,KindReport,Descr,SerialReport) :-
    storeURL(Store),
    agent(request,
        Store,storeReport(System,Time,KindReport,Descr,SerialReport)),!.
</kdb>

<schedule name="analysis" time="5">
    verifyAvailability(system,'150.162.63.19',
        '2003-12-17 0:00:01.000000000','2003-12-17 23:59:59.000000000').
</schedule>
</agent>

```