Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Serviço Baseado em Semântica para Descoberta de Recursos em Grade Computacional

José Nelson Costa Allemand

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientadora
Profª Drª Célia Ghedini Ralha

Brasília
2006

Universidade de Brasília – UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenadora: Profª. Drª. Alba Cristina M. A. de Melo

Banca examinadora composta por:

Profª. Drª. Célia Ghedini Ralha (Orientadora) – CIC/UnB
Profª. Drª. Alba Cristina M. A. de Melo – CIC/UnB
Prof. Dr. Celso Massaki Hirata – Comp/ITA

CIP – Catalogação Internacional na Publicação

José Nelson Costa Allemand.

Serviço Baseado em Semântica para Descoberta de Recursos em Grade Computacional/ José Nelson Costa Allemand. Brasília : UnB, 2006.
120 p. : il. ; 29,5 cm.

Tese (Mestre) – Universidade de Brasília, Brasília, 2006.

1. Computação em Grade, 2. Conhecimento, 3. Ontologia,
4. Grade Semântica

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro – Asa Norte
CEP 70910–900
Brasília – DF – Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Serviço Baseado em Semântica para Descoberta de Recursos em Grade Computacional

José Nelson Costa Allemand

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Profª. Drª. Célia Ghedini Ralha (Orientadora)
CIC/UnB

Profª. Drª. Alba Cristina M. A. de Melo
CIC/UnB

Prof. Dr. Celso Massaki Hirata
Comp/ITA

Profª. Drª. Alba Cristina M. A. de Melo
Coordenadora do Mestrado em Informática

Brasília, 15 de dezembro de 2006

Dedicatória

Dedico este trabalho à minha querida esposa Lorena, uma companhia sempre agradável, em todos os momentos de minha vida.

Dedico também a meu pai João Henrique, um grande incentivador nesta jornada.

Agradecimentos

Agradeço a minha família pelo carinho, incentivo e suporte necessário, sem o qual este trabalho não seria possível.

Agradeço a Célia, professora de Universidade de Brasília, pela orientação e confiança depositada em mim.

Agradeço aos colegas de Mestrado Alê! Gomes, Daniela, Edward, Lorena, Marcelo Nardeli, Marcelo Pacote Sousa, Roberto Cantanhede, Taninha e Zé Geraldo por terem ajudado com sugestões e por terem compartilhado comigo esta experiência. Um agradecimento especial à Rosa Amariles, secretária da Pós-Graduação, por ser sempre tão prestativa.

E por fim, agradeço meus amigos, que tiveram importância relevante, pela colaboração e compreensão no decorrer deste período tão importante de minha vida.

Resumo

Nesta dissertação de mestrado são apresentados o projeto e o protótipo de uma arquitetura de Grade Semântica visando o casamento entre os recursos computacionais disponíveis na grade e os requisitos da aplicação a ser executada, uma vez que em um ambiente que possui diversas máquinas este casamento pode ser muito trabalhoso sem o auxílio de uma ferramenta. A arquitetura proposta é composta por uma camada de Conhecimento com a finalidade de prover um serviço de descoberta de recursos computacionais de forma semântica, que está localizada acima das camadas Física e a de *Middleware* da grade. A camada de Conhecimento é composta por dois elementos básicos: o Repositório Semântico e o Raciocinador. O Repositório Semântico permite a descoberta semântica dos recursos da grade por meio do uso de um *template* ontológico. Para a edição deste *template* e tratamento dos diversos tipos de recursos computacionais no ambiente de grade foi utilizado o Protégé-OWL. O Raciocinador, segundo componente da camada de Conhecimento, interage com o Repositório Semântico para selecionar os recursos de forma apropriada. Neste trabalho foi utilizado o *middleware* de grade Globus Toolkit 4 (GT4) e seu serviço *Monitoring and Discovery System* (MDS4), juntamente com o Ganglia, para coletar automaticamente as informações sobre os recursos da grade computacional. Como motor de inferência empregou-se a ferramenta Pellet-OWL. Por meio da realização de um estudo de caso com uso do protótipo, verificou-se que a busca baseada em semântica resulta na obtenção de recursos na grade computacional de forma mais adequada, quando comparada às abordagens tradicionais.

Palavras-chave: Computação em Grade, Conhecimento, Ontologia, Grade Semântica

Abstract

In this work we present the design and the prototype of a Semantic Grid architecture that deals with resource matching in a grid environment where an application has to be executed with certain requirements. This tool can be very useful in an environment with lots of machines, where the resource matching is a hard work. The proposed architecture is composed by a Knowledge layer with the purpose of providing a computational resource discovery service in a semantic way. This Knowledge layer is located above the Fabric and the grid Middleware layer. It is composed by two basic elements: the Semantic Repository and the Reasoner. The Semantic Repository allows the semantic discovery of grid resources using an ontological template. To edit this template and to handle a lot of computational resource types in the grid environment we use Protégé-OWL tool. The Reasoner, the second component of the Knowledge layer, interacts with the Semantic Repository to select resource in an appropriate way. In this work, we used the Globus Toolkit 4 (GT4) grid middleware and its Monitoring and Discovery System (MDS4) with Ganglia to automatically collect information about the computational grid resources. We use Pellet-OWL as the inference engine. We developed a study case using the prototype, which showed that more accurate results are possible than conventional approaches.

Keywords: Grid Computing, Knowledge, Ontology, Semantic Grid

Sumário

Lista de Figuras

Lista de Tabelas

Acrônimos

API	*Application Programming Interface*
CGO	*Core Grid Ontology*
CPU	*Central Processing Unit*
DAML	*DARPA Agent Markup Language*
DARPA	*Defense Advanced Research Projects Agency*
DAS	*Data Access Services*
DIG	*Description Logics Implementors Group Interface*
DL	*Description Logics*
EMBOSS	*European Molecular Biology Open Software Suite*
EPMS	*Execution Plan Management Service*
GGF	*Global Grid Forum*
GLUE	*Grid Laboratory Uniform Environment*
GRIP	*Grid Interoperability Project*
GT4	*Globus Toolkit 4*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IA	Inteligência Artificial
Java WSDP	*Java Web Services Developer Pack*
JAXB	*Java Architecture for XML Binding*
JDBC	*Java Database Connectivity*
JDK	*Java Development Kit*
KBR	*Knowledge Base Repository*
KDD	*Knowledge Discovery in Database*
KDS	*Knowledge Directory Service*
KERP	*Knowledge Execution Plan Repository*
KMR	*Knowledge Metadata Repository*

LABPOS	Laboratório da Pós-Graduação
LAICO	LAboratório de sistemas Integrados e COncorrentes
LSF	*Load Share Facility*
MB	*Megabyte*
MDS	*Monitoring and Discovery System*
MonALISA	*Monitoring Agents using a Large Integrated Services Architecture*
OGF	*Open Grid Forum*
OGSA	*Open Grid Services Architecture*
OIL	*Ontology Interchange Language ou Ontology Inference Layer*
OMMS	*Ontology-based Resource Matchmaker Service*
OMS	*Ontology Management Service*
OV	Organização Virtual
OWL	*Ontology Web Language*
PDKD	*Parallel and Distributed Knowledge Discovery*
QoS	*Quality of Service*
RAEMS	*Resource Allocation and Execution Management Service*
RAM	*Random Access Memory*
RDF	*Resource Description Framework*
RDFS	*RDF Schema*
RDQL	*RDF Data Query Language*
RPS	*Results Presentation Service*
S-OGSA	*Semantic-OGSA*
SGBD	Sistema de Gerenciamento de Banco de Dados
SGE	*Sun Grid Engine*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
TAAS	*Tools and Algorithms Access Service*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UDDI	*Universal Description, Discovery and Integration*
UML	*Unified Modeling Language*

VEGA	*Visual Environment for Developing Complex Grid Applications*
W3C	*World Wide Web Consortium*
WS-Inspection	*Web Services Inspection Language*
WS-Notification	*Web Services Notification*
WSDL	*Web Services Definition Language*
WSRF	*Web Services Resource Framework*
XML	*Extensible Markup Language*

Capítulo 1

Introdução

Existem atualmente vários tipos de problemas que demandam um alto poder computacional. Tais problemas levariam anos, décadas ou talvez centenas de anos para serem resolvidos em uma máquina apenas. Algumas soluções computacionais foram propostas para contornar este problema, como é o caso de supercomputadores, *clusters* e computação em grade.

A computação em grade é uma tecnologia recente que permite o compartilhamento de recursos e a resolução coordenada de problemas em organizações virtuais e dinâmicas de múltiplas instituições [Foster et al., 2001]. As grades são utilizadas para compartilhar recursos computacionais heterogêneos e distribuídos geograficamente e entregar estes recursos para comunidades de usuários distintos. Os recursos podem estar em diferentes instituições, ter diferentes políticas de uso e possuir diferentes requisitos de aceitação de requisições.

Não só os recursos da grade precisam ser tratados semanticamente, mas também as aplicações da grade necessitam ter diferentes restrições que podem apenas ser satisfeitas por certos tipos de recursos com capacidades específicas. Esta é a idéia da Grade Semântica [Roure, 2006]. Antes dos recursos serem alocados para executar uma aplicação, o usuário ou agente deve selecionar os recursos apropriadamente para os requisitos da aplicação [Czajkowski et al., 2002]. Este processo de seleção de recursos baseado nos requisitos da aplicação é conhecido como casamento de recursos ou *resource matching*. Em um ambiente de grade dinâmico, onde os recursos são adicionados e removidos dinamicamente e assim podem ou não estar disponíveis, é desejável e às vezes necessário automatizar o casamento de recursos conhecendo-se quais são os requisitos da aplicação.

O ambiente de grade tem uma tarefa fundamental, que é decidir quais tarefas executar e em quais recursos de computação, baseado na tarefa ou nos requisitos da aplicação. O problema do casamento de recursos neste ambiente envolve associar recursos à tarefas de modo a satisfazer os requisitos da tarefa e as políticas

do recurso. Estes requisitos e políticas são freqüentemente expressos em aplicações disjuntas e modelos de recursos heterogêneos, forçando um selecionador de recursos realizar o casamento semântico entre os dois.

Em [Foster and Kesselman, 1999] é descrita a infra-estrutura de computação em grade, que inclui manipulação de informação e suporte ao processamento de conhecimento em um processo científico distribuído. Mais recentemente, com a definição do conceito de Grade Semântica, informação e serviços possuem um significado bem definido, permitindo que pessoas e computadores trabalharem em cooperação [Roure et al., 2003; Li et al., 2003]. Neste ambiente, a habilidade de descrever os recursos da grade necessários por agentes ou aplicações é essencial para desenvolver o acesso aos recursos na grade [Brooke et al., 2004].

Existem muitas iniciativas da pesquisa em grade para ajudar neste desafio. Dentre estas iniciativas podemos destacar o Sistema de Monitoração e Descoberta do conjunto de ferramentas Globus ou *Globus Toolkit Monitoring and Discovery System* (MDS). O MDS suporta apenas o casamento tradicional de recursos baseado em atributo, que é baseado em simetria, e não suporta a descrição semântica dos recursos ou serviços da grade [Akkiraju et al., 2003; Zhang and Song, 2004]. No entanto, em ambientes de grade onde recursos são controlados por diferentes comunidades, com várias políticas e capacidades, a obtenção e o gerenciamento destes recursos, mesmo sem utilização de semântica, é algo não trivial.

Nesta dissertação de mestrado são apresentados o projeto de uma arquitetura de Grade Semântica e o protótipo implementado em Java, visando o casamento entre os recursos computacionais disponíveis na grade e os requisitos da aplicação a ser executada, uma vez que em um ambiente que possui diversas máquinas este casamento pode ser muito trabalhoso sem o auxílio de uma ferramenta. O foco deste trabalho está na definição da arquitetura de grade semântica que viabiliza ao casamento semântico entre os recursos e não aborda a execução, a submissão e o escalonamento das aplicações.

A arquitetura proposta compõe-se de quatro camadas, sendo o foco deste trabalho a Camada de Conhecimento, que tem como objetivo melhorar de forma semântica a descoberta de recursos na grade computacional. Nesta camada existem dois elementos principais. O primeiro é o Repositório Semântico, o qual possibilita a descoberta semântica dos recursos da grade, com a ajuda de um *template* ontológico, que nada mais é que um repositório que trata os diferentes tipos de recursos computacionais do ambiente que podem ser monitorados. Para a edição e tratamento do *template* ontológico foi utilizado o Protégé-OWL. O segundo elemento existente na Camada de Conhecimento é o Raciocinador que

interage com o Repositório Semântico para a descoberta de recursos de forma adequada. Como motor de inferência deste Componente de Raciocínio foi utilizado o Pellet-OWL. Para a coleta e tratamento das informações sobre os recursos da grade foram utilizados o Sistema de Monitoramento Ganglia juntamente com o Sistema de Descoberta e Monitoramento do Globus Toolkit 4 (GT4), o *Monitoring and Discovery System* (MDS4), o *middleware* de grade utilizado neste trabalho.

Quanto à estrutura desta dissertação, o documento está organizado como se segue, primeiramente serão abordados temas relacionados aos fundamentos deste trabalho, incluindo: o capítulo 2 sobre computação em grade, o capítulo 3 sobre tratamento de conhecimento e o capítulo 3.1 sobre tratamento semântico. O capítulo 4 descreve a proposta do trabalho incluindo a arquitetura, o fluxo de execução lógico da proposta implementada e alguns trabalhos correlatos. O capítulo 5 apresenta um estudo de caso para ilustrar o uso do protótipo, enquanto o capítulo 6 conclui com idéias de trabalhos futuros. Há também o apêndice A, em que são mostradas as ontologias utilizadas e o apêndice B em que são descritos passos de instalação das ferramentas utilizadas neste trabalho.

Capítulo 2

Computação em Grade

Nos últimos anos, tem sido observada a evolução progressiva do poder de processamento dos computadores. Observou-se também que estes computadores muitas vezes ficam ociosos por um certo período de tempo. Com o objetivo de se aproveitar esse tempo ocioso dos processadores conectados em rede para executar aplicações paralelas e distribuídas, nasceu a área de computação em grade ou *grid computing*.

O termo grade surgiu em meados de 1990 para denotar uma infra-estrutura de computação distribuída proposta para ciência avançada e engenharia [Foster et al., 2001]. A expressão *Grid Computing* vem da analogia de *Power Grid* (rede elétrica em inglês), pois apresentam uma característica em comum: a transparência. Não importa ao usuário, consumidor do serviço, onde exatamente está sendo gerada ou produzida a computação ou a energia elétrica. Não importa também qual tecnologia está sendo utilizada, mas sim se o serviço prestado satisfaz às suas necessidades. Um projeto recente baseado nesta idéia é o *World Community Grid* [World Community Grid, 2005].

Conforme citado no capítulo 1, a grade surgiu como um ambiente computacional de alto desempenho e uma de suas principais características é prover o compartilhamento geograficamente distribuído de serviços oferecidos por organizações distribuídas.

De uma forma geral uma grade é conceituada por alguns autores [Foster and Kesselman, 1999; The Globus Project, 2000; Buyya, 2006; Berman et al., 2003] como:

> *Um ambiente computacional distribuído que permite o compartilhamento, a seleção e a agregação de recursos autônomos e geograficamente distribuídos. Estas operações e recursos podem ser utilizados durante a execução de uma aplicação, dependendo de sua disponibilidade, capacidade, desempenho e custo. O objetivo é prover aos*

usuários serviços com os requisitos de qualidade corretos para o perfeito funcionamento de suas aplicações.

Desta forma, pode-se considerar que uma grade computacional tem como seu principal objetivo alcançar a interoperabilidade entre as Organizações Virtuais (seção 2.1), através da habilidade de cooperação de compartilhamento e agregação de recursos computacionais distribuídos e disponibilizá-los como serviços.

Padronização

Os esforços de padronização da tecnologia de grade já ocorrem há alguns anos, todavia nos últimos anos estes esforços começaram a tomar um vulto maior devido ao uso da tecnologia por inúmeras organizações. A organização que se destaca no sentido de padronização do ambiente de grade é a *Global Grid Forum* (GGF) [Global Grid Forum, 2003], que agora forma o *Open Grid Forum* (OGF) [Open Grid Forum, 2006]. É uma organização composta por uma comunidade de milhares de pesquisadores e por diversos profissionais com grande experiência em sistemas distribuídos. O OGF tem como objetivo promover e prover suporte ao desenvolvimento, utilização e implementação de tecnologias e aplicações de grade. As recomendações são criadas e documentadas a partir das melhores práticas técnicas, experiência de usuários e guias de implementação. Dentre os membros do OGF podemos citar Hewlett-Packard, IBM, Intel, Microsoft, Silicon Graphics e Sun dentre diversos outros.

Um outro aspecto interessante quanto ao trabalho do OGF diz respeito ao entendimento da organização quanto a uma grade computacional. A organização faz a leitura de uma grade computacional como sendo aquele ambiente de computação geograficamente distribuída que provê o alicerce para inúmeros esforços de utilização global da Internet para a construção de infra-estruturas de computação distribuída e comunicação. Desta forma, a interoperabilidade de seus componentes (software e hardware) deve ser um serviço comum na configuração de uma grade.

A camada de serviços da grade, também chamada de *middleware* da grade possibilita que aplicações possam ser executadas na grade. Existem algumas ferramentas que provem este *middleware*, por exemplo: o Globus [The Globus Alliance, 2005], o Legion [University of Virginia, 2006], o *Sun Grid Engine* (SGE) [Sun Microsystems, 2005], o Unicore [UNICORE Forum, 2005] e o Alchemi [Alchemi, 2006], dentre outras.

As grades computacionais podem também ser consideradas, sob o paradigma da melhoria de alguns aspectos físicos nas redes de comunicação de computado-

res [Baird, 2002], tais como:

- Utilização agregada de recursos gerando um grande poder computacional;
- Melhor utilização de largura de banda;
- Acesso rápido a dados, pacotes de software e dispositivos remotos com qualidade de serviço ou *Quality of Service* (QoS);
- Melhor utilização de *Central Processing Unit* (CPU), memória e espaço em disco remoto; e
- Aproveitamento de recursos ociosos disponíveis no ambiente de grade.

2.1 Organização Virtual

No contexto de computação em grade, uma organização é conhecida como Organização Virtual (OV) [Foster et al., 2001]. Traçando uma comparação com a Internet, a entidade OV seria semelhante a um domínio, todavia com a possibilidade de prover serviços solicitados pelo usuário. Uma OV é uma entidade que compartilha recursos sob uma determinada política em uma configuração de grade. Exemplos de OVs são universidades, centros de pesquisas e empresas que proporcionam facilidades de armazenamento de dados, poder de processamento e o uso de equipamentos (por exemplo, telescópios) e aplicações (por exemplo, pacotes de software de simulação que podem ser executados com dados fornecidos pelo próprio usuário).

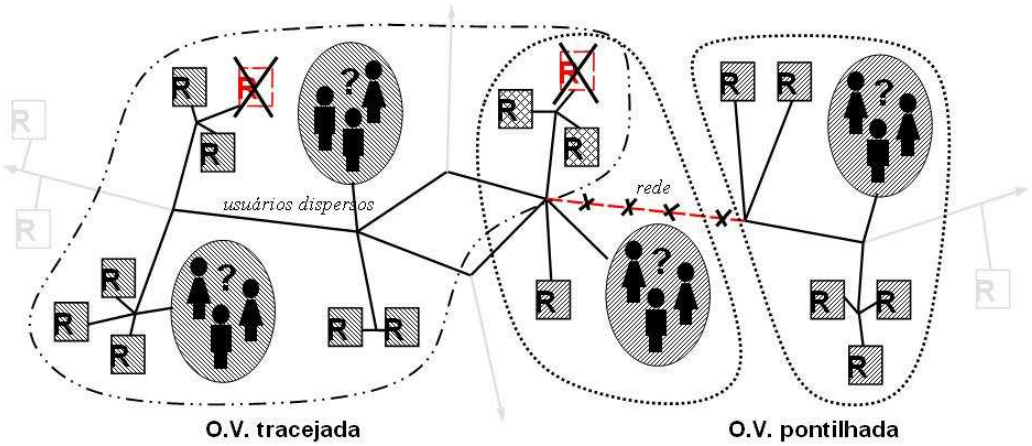

Figura 2.1: Exemplo de Organização Virtual [Clifford, 2005].

Observe a figura 2.1, que representa duas OVs. É interessante notar que tanto os usuários quanto os recursos estão distribuídos geograficamente e que os recursos

computacionais podem estar ativos, ou não, em determinado momento. Podem ser criadas diferentes OVs para determinados recursos computacionais, ou seja, podem ser feitos diferentes agrupamentos representando as possíveis OVs. Note que na figura 2.1 que a OV pontilhada tornou-se particionada, uma vez que houve uma falha na rede [Clifford, 2005].

2.2 Arquitetura da Grade

De maneira análoga à Internet, e principalmente devido à experiência com a mesma, pesquisadores vêm propondo modelos de arquitetura com o objetivo de uma padronização que permita a interoperabilidade entre diferentes OVs. Aspectos tais como autenticação, autorização, mecanismos de troca de mensagem, compartilhamento de recursos, escalonamento e balanceamento de tarefas são alguns dos pontos que devem existir e ser estabelecidos de uma maneira uniforme para uma arquitetura de grade computacional. Com este objetivo a seguir, na figura 2.2, apresenta-se a arquitetura de protocolos [Foster and Kesselman, 1999], [Berman et al., 2003] que ganhou aceitação na comunidade internacional como padronização da arquitetura de grade.

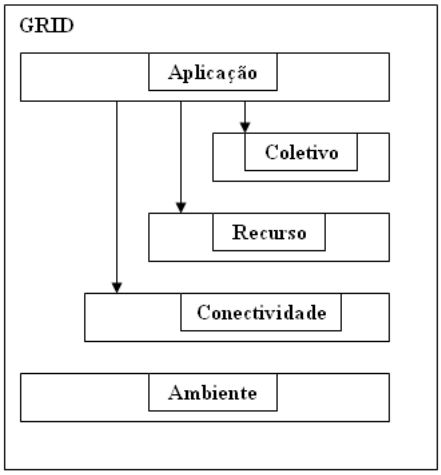

Figura 2.2: Modelos de Grades Computacionais [Foster and Kesselman, 1999].

Os cinco níveis da arquitetura de protocolos [Foster and Kesselman, 1999], ilustrados na figura 2.2 podem ser entendidos como:

- Ambiente: nesta camada existe a interface para controle local dos recursos disponibilizados. Devem ser implementados mecanismos que disponibilizem ao usuário informações sobre a estrutura, e as possibilidades de utilização do recurso, bem como mecanismo que possibilitem monitorar a qualidade dos serviços.

- Conectividade: esta camada define os protocolos básicos de comunicação e autenticação necessários para as transações de rede específicas da grade. Os protocolos de comunicação permitem a troca de dados entre os níveis de ambiente e recursos. Entre os requisitos de comunicação estão o transporte e o roteamento. Os protocolos de autenticação constroem os serviços de comunicação de modo a prover mecanismos seguros e criptográficos para a verificação da identidade de usuários e recursos.

- Recursos: é nesta camada onde encontra-se a definição dos protocolos utilizados, e são eles: os Protocolos de Informação, que são utilizados para obtenção de informações sobre a estrutura e o estado dos recursos compartilhados e os Protocolos de Gerenciamento, que negociam acesso a recursos compartilhados. As implementações dos protocolos dessa camada chamam as funções da camada de ambiente para acessar e controlar recursos locais.

- Coletivo: enquanto no nível de recursos são tratadas as operações no âmbito de cada recurso individualmente, neste nível os componentes atuam sobre coleções de recursos. Os componentes dessa camada baseiam-se nos níveis recursos e aplicação e implementam serviços tais como:

  - Serviços de diretório: que permitem aos membros de uma organização virtual descobrir quais são os recursos desta organização;
  - Serviços de alocação conjunta e agendamento;
  - Serviços de monitoramento e diagnostico;
  - Serviços de pesquisa de software;
  - Serviços colaborativos; e
  - Sistema de programação.

- Aplicação: esta camada compreende as aplicações dos usuários que operam no ambiente da organização virtual. Os níveis anteriores provém serviços úteis às aplicações desenvolvidas que as solicitam.

Na figura 2.3 apresenta-se uma comparação funcional entre o padrão *Transmission Control Protocol / Internet Protocol* (TCP/IP), utilizado na rede Internet, e o primeiro modelo de grade. Esta equivalência é interessante de ser ressaltada, pois permite uma melhor visualização dos ambientes de grades computacionais e também demonstra a preocupação de uma padronização de todos aqueles envolvidos com esta nova abordagem tecnológica.

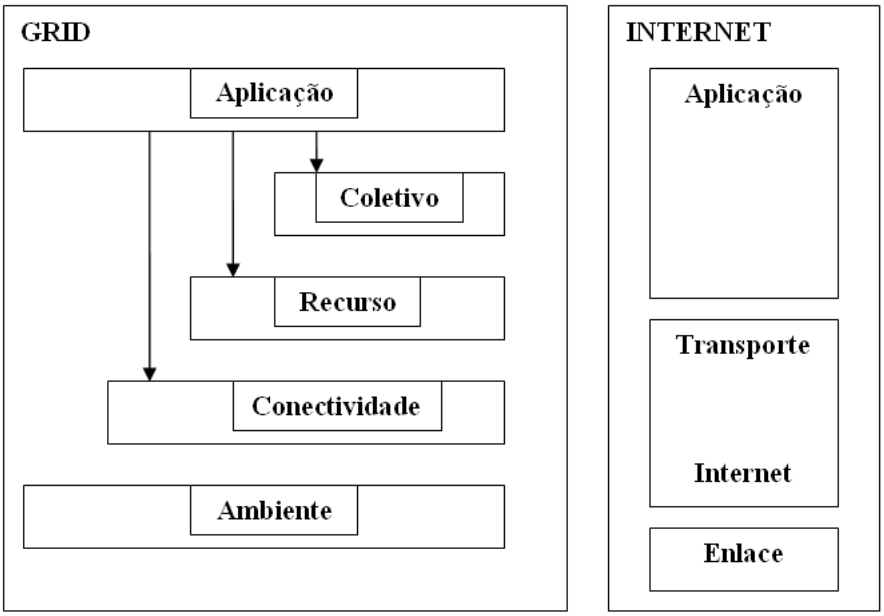

Figura 2.3: Equivalência Funcional entre os Modelos da Grade e da Internet [Foster et al., 2001].

2.3 Classificação das Aplicações

É comum a classificação das aplicações utilizadas em um ambiente de grade segundo um paradigma de uso de computadores, como apresentado a seguir [Foster and Kesselman, 1999]:

- Supercomputação Distribuída: aplicações com grande necessidade computacional, cuja execução só é possível se for em vários supercomputadores. Este ambiente de computadores aqui referido pode ser representado por todos os supercomputadores de um país, por exemplo, ou por todos os computadores de uma empresa. Como exemplo tem-se a simulação de processos físicos complexos, como a previsão do tempo e a cosmologia.

- Computação de Alto Desempenho: cujo objetivo é distribuir uma grande quantidade de tarefas independentes e com baixo acoplamento entre os diversos recursos que o formam (muitas vezes máquinas ociosas). Diferentemente da Supercomputação Distribuída, esse tipo de aplicação envolve poucas dependências (ou nenhuma) entre as tarefas. Como exemplos desta abordagem pode-se citar os problemas de criptografia e o projeto *Load Share Facility*((LSF)), da empresa Platform, no qual é relatado o caso da fabricante de microprocessadores AMD, que utilizou todas as máquinas de seus engenheiros que não estavam sendo utilizadas em um determinado período,

durante o desenvolvimento dos seus processadores K6 e K7.

- Computação Sob Demanda: aplicações que se enquadram nesta classe podem ser compreendidas como aquelas que requerem o uso intensivo de recursos computacionais que não estão disponíveis localmente. Os recursos computacionais devem ser entendidos como pacotes de software, dados, arquivos e poder computacional para obtenção de seus resultados experimentais. É diferente das aplicações classificadas como supercomputação distribuída, que dispõem de todos os recursos para seu melhor desempenho.

- Computação para Grande Quantidade de Dados: as aplicações e o paradigma computacional nesta classe são caracterizados pelo processamento de enorme quantidade de dados distribuídos geograficamente. Conhecidos exemplos são bibliotecas digitais e os bancos de dados distribuídos. Pode-se ainda citar o caso do projeto de pesquisa digital do espaço, o *Digital Sky Survey [2006]*.

- Computação Colaborativa: tem como objetivo possibilitar e melhorar a interação entre pessoas via simuladores ou mundos virtuais. Tais aplicações são freqüentemente estruturadas como um espaço virtual compartilhado, contendo diversos recursos computacionais que poderão ser utilizados de forma colaborativa entre os usuários da aplicação.

2.4 Globus *Toolkit*

O Globus *Toolkit* [Foster and Kesselman, 1997] surgiu em 1997 como um projeto de código aberto e rapidamente se tornou um padrão *de facto* para a infraestrutura de computação em grade. O Globus define e implementa um conjunto de protocolos, APIs e serviços utilizados por centenas de aplicação de grade ao redor do mundo. Além disto, ele foi pioneiro no desenvolvimento de sistemas de grade interoperáveis.

O projeto de pesquisa Globus [The Globus Project, 2000] tem como foco o desenvolvimento de tecnologias fundamentais que são necessárias para a construção das grades computacionais. O projeto é dividido em quatro atividades principais: aplicações, pesquisas, ferramentas de software e testes. O ambiente Globus auxilia o desenvolvimento da tecnologia de grade através de um conjunto de ferramentas abertas denominadas de Globus Toolkit. O desenvolvimento dessas ferramentas iniciou-se pelo esforço dos pesquisadores Ian Foster (*University of Chicago*) e Carl Kesselman (*University of Southern California*). Estes pesquisadores propuseram

que o conjunto de serviços do ambiente fosse desenvolvido com o objetivo de que as aplicações pudessem se utilizar dos serviços de grade sem a necessidade de uma adaptação a um modelo particular de programação. Alguns exemplos de serviços do conjunto de ferramenta são a alocação de recursos e gerenciamento de processos, os serviços de comunicação, autenticação e segurança, o acesso distribuído à informação de estrutura da grade. Em adição, o estado dos processos, o acesso remoto a dados e o monitoramento do sistema são ainda aspectos importantes no ambiente.

Segundo este grupo, as grades podem ser entendidas como ambientes onde aplicativos de software permitem a integração de instrumentos, dispositivos de visualização, recursos computacionais e de informação que são gerenciadas por diversas organizações dispersas geograficamente.

Em 2002, a Arquitetura Aberta de Serviços de Grade ou *Open Grid Services Architecture* (OGSA) [The Globus Alliance, 2006c] foi introduzida pelo *Global Grid Forum* (GGF) para expandir a padronização. A OGSA provê uma nova arquitetura para aplicações de grade baseada em Serviços Web de modo a realizar a interoperabilidade utilizando padrões industriais. Muitas aplicações da arquitetura OGSA foram desenvolvidas, incluindo uma para o Globus. Além da definição de um conjunto de interfaces padronizadas, a arquitetura OGSA provê um *framework* para definição de serviços interoperáveis e portáveis e assim provê uma base para o desenvolvimento da grade. O *WS-Resource Framework* (WSRF) [The Globus Alliance, 2006d] é um conjunto de especificações de Serviços Web que implementam as habilidades OGSA utilizando Serviços Web. No protótipo desenvolvido nesta dissertação foi utilizado o Globus *Toolkit* versão 4 (GT4).

Dentro os principais desafios técnicos do projeto Globus [The Globus Project, 2000] destacam-se:

- o gerenciamento de recursos;
- os serviços de comunicação;
- os serviços de escalonamento;
- os serviços de informação;
- os protocolos de segurança;
- o acesso aos dados;
- os serviços de tolerância a falhas; e

- facilidades de acesso a dados remotos.

Quanto à evolução do Globus *Toolkit*, segue abaixo as últimas versões estáveis lançadas e os respectivos anos de lançamento, até a presente data:

- GT1 v.1.1.3 (2002)
- GT2 v.2.4.3 (2003)
- GT3 v.3.2.1 (2004)
- GT4 v.4.0.3 (2006)

O GT1 e GT2 são conhecidos como Globus *Toolkits pre-Web Services*, pois a partir do GT3 começam a são utilizados Serviços Web, ao invés de protocolos de rede *ad-hoc*. Já o GT4 é totalmente compatível com Serviços Web. A figura 2.4 mostra uma visão geral dos componentes do GT4.

Serviços Web

Os padrões para construção de aplicação baseada em Serviços Web ou *Web Services* são responsáveis por especificar os mecanismos para troca de mensagens entre as aplicações e os Serviços Web, pesquisa de serviços em registros e configuração dinâmica de serviços.

Desta forma, Basiura et al. [2003] publicou uma pilha básica de tecnologias formada por cinco camadas lógicas para construção de qualquer Serviço Web:

- Camada de rede de transporte: *HyperText Transfer Protocol*(HTTP);
- Camada decodificação de dados: *eXtensible Markup Language*(XML);
- Camada de troca de mensagens entre Serviço Web: *Simple Object Access Protocol*(SOAP);
- Camada de descrição de serviços: *Web Service Definition Language*(WSDL);
- Camada de publicação de serviços: *Universal Description, Discovery and Integration*(UDDI).

Para a construção de um Serviço Web, é definido um modelo genérico, que é baseado nas interações entre papéis e operações, também conhecido como arquitetura e integração de Serviço Web, este modelo está apresentado na figura 2.5.

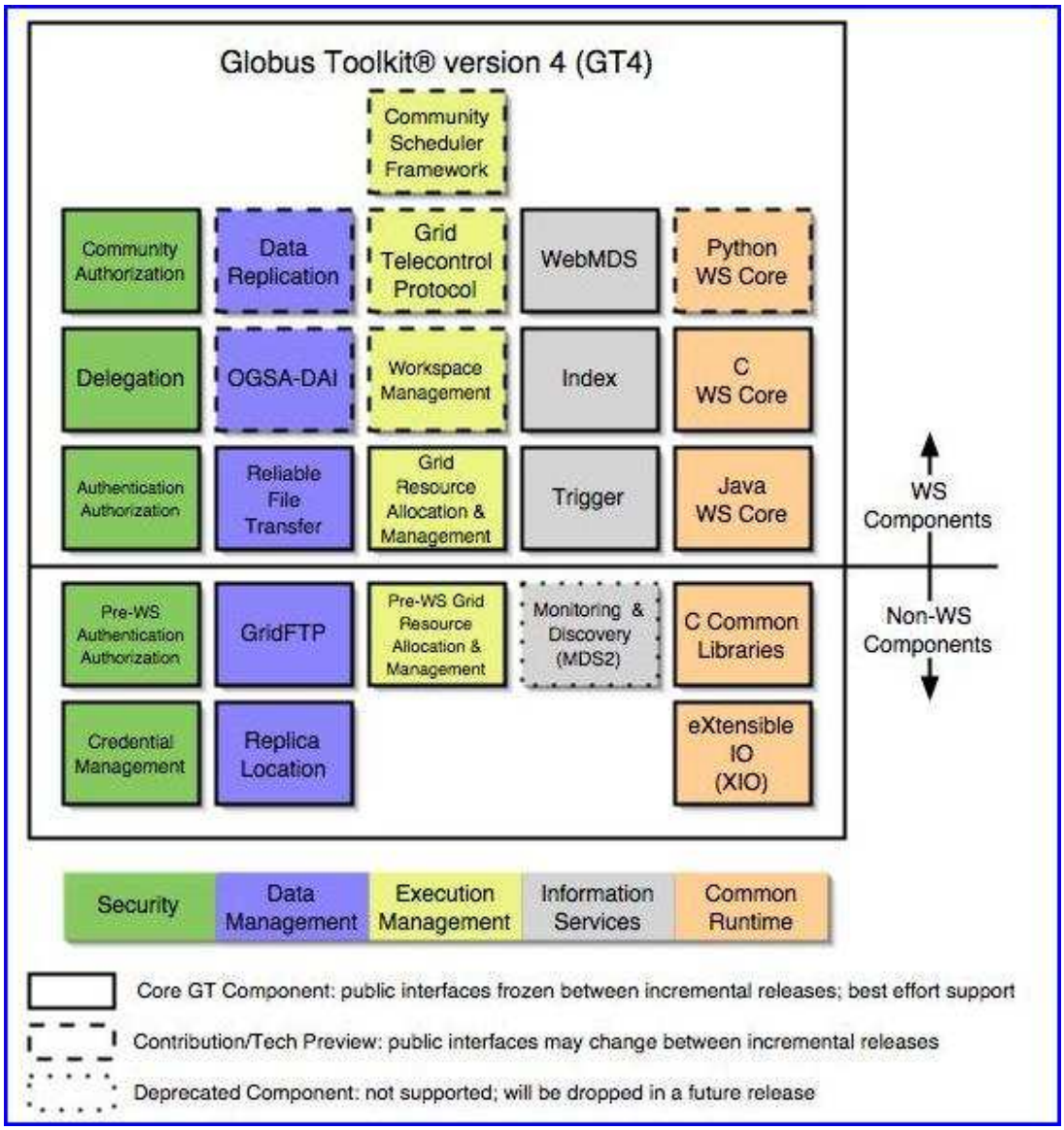

Figura 2.4: Componentes do GT4 [The Globus Alliance, 2006a].

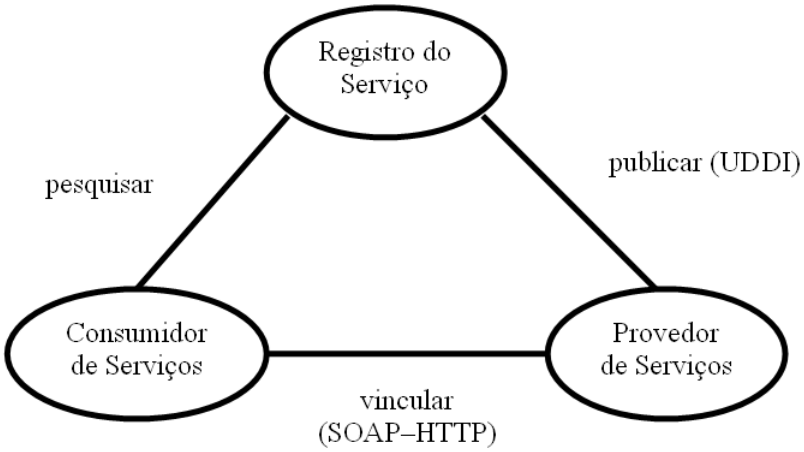

Figura 2.5: Modelo Genérico de um Serviço Web [Hendricks et al., 2002].

Neste modelo são considerados como papéis: registro do serviço, provedor do serviço e consumidor do serviço. E como operações definidas tem-se: pesquisar, publicar e vincular.

A partir da figura 2.5 pode-se inferir que um Serviço Web é baseado no modelo cliente-servidor, onde a aplicação servidora registra o serviço em um catálogo de endereços que contém as informações do Serviço Web (UDDI). Este registro é feito pelo Provedor de Serviços que publica o Serviço Web [Hendricks et al., 2002]. Ao implementar a aplicação cliente, o consumidor de serviços pesquisa a existência de um determinado serviço. Caso seja encontrado, é realizado o vínculo com o provedor de serviços através de mensagens SOAP sobre o protocolo de transporte, que pode ser o HTTP.

Grid Services

A tecnologia de grade está evoluindo para uma arquitetura aberta de serviços de grade, conhecida como OGSA. Esta arquitetura provê um extenso conjunto de serviços que podem ser agregados de vários modos pelas OVs [Congiusta et al., 2002, 2004].

A OGSA define uma semântica uniforme de exposição dos serviços, os então chamados *Grid Services*, baseados em conceitos e tecnologias tanto da comunidade da computação em grade quanto da comunidade de Serviços Web. Os Serviços Web definem técnicas para descrever os componentes de software a serem acessados, os métodos de acesso a esses componentes e os métodos de descoberta que permitem a identificação de provedores de serviços relevantes. Os Serviços Web

são, em princípio, independentes de linguagem de programação e de sistemas de software. Os padrões estão sendo definidos pelo *World Wide Web Consortium* (W3C).

O modelo da OGSA adota três padrões de Serviços Web:

- *Simple Object Access Protocol* (SOAP);
- *Web Services Description Language* (WSDL); e
- *Web Services Inspection Language* (WS-Inspection).

O objetivo do esforço da OGSA está em definir um modelo comum dos recursos, que é a representação tanto dos recursos reais, por exemplo: processadores, processos, discos, sistemas de arquivos, quanto dos recursos lógicos, independente de implementação, localização e plataforma. A OGSA provê algumas operações comuns e suporta múltiplos modelos de recursos representando os recursos como instâncias dos serviços. A OGSA define mecanismos padronizados para a criação, nomeação e descoberta de instâncias de *Grid Services* transientes e persistentes, provê localização de forma transparente e um protocolo múltiplo de ligação para as instâncias dos serviços e ainda suporta e integração com as facilidades providas nativamente pelas plataformas.

Na OGSA todos os serviços mantém-se fiéis às interfaces e comportamentos especificadas pelo *Grid Service* definidos em termos das interfaces e convenções WSDL e os mecanismos requeridos para a criação e a composição de sistemas distribuídos sofisticados. As ligações de serviço podem suportar invocação confiável, autenticação, autorização e delegação. Por isso a OGSA define o *Grid Service* como um Serviço Web que provê um conjunto de interfaces WSDL bem definidas e que seguem convenções específicas no uso para a computação em grade.

O grupo de pesquisa da Itália [Congiusta et al., 2002] liderado pelo professor Talia descreve a implementação de *Knowledge Grid* seguindo o modelo OGSA. Nesta proposta, cada um dos serviços é exposto como um serviço persistente, utilizando as convenções e os mecanismos da OGSA. Por exemplo, o serviço *Execution Plan Management Service* (EPMS) implementa muitas interfaces, dentre estas, a interface de notificação que permite a entrega assíncrona para o EPMS de mensagens de notificação advindas dos serviços invocados, como os indicados nos planos de execução. Ao mesmo tempo, os serviços básicos de descoberta de conhecimento podem ser projetados e implantados utilizando os serviços *Knowledge Directory Services* (KDS) para descoberta dos recursos da grade computacional

que podem ser utilizados para a composição das aplicações de descoberta de conhecimento.

Provedores de Informação sobre Recursos da Grade

O trabalho de Schopf et al. [2006] aborda o Sistema de Descoberta e Monitoramento do Globus ou *Monitoring and Discovery System* (MDS), que é um conjunto de componentes para o monitoramento e descoberta de recursos e serviços da grade. MDS4, a versão existente no Globus Toolkit 4 [Foster, 2006] utiliza as interfaces padrão definidas pelo *Web Services Resource Framework* (WSRF) e especificações *Web Services Notification* (WS-Notification) [Foster et al., 2005] para prover a consulta e a subscrição de interfaces para detalhar arbitrariamente os recursos de dados (modelados em XML). Uma interface *trigger* pode ser configurada para agir quando condições pré-configuradas são conhecidas. Os serviços MDS4 adquirem suas informações através de uma interface extensível que pode ser utilizada para consultar serviços WSRF para:

- informações sobre as propriedade de um recurso;
- executar um programa para aquisição de dados; ou
- fazer a interface com sistemas de monitoração de terceiros.

Ainda no trabalho de Schopf et al. [2006], os recursos e serviços da grade de computação podem fornecer uma grande quantidade de dados. O MDS4 foi projetado para permitir o acesso a tais dados por múltiplas pessoas de múltiplos domínios administrativos. Intrinsicamente, ele não é um sistema de manipulação de eventos, como o NetLogger [Gunter and Tierney, 2003], ou um monitor de *cluster*, como é o Ganglia [Massie et al., 2004], mas possui uma interface para estes sistemas de monitoração mais detalhados.

Os principais provedores de informação sobre recursos da grade existentes atualmente são:

- O Ganglia [Ganglia Development Team, 2006] é um sistema de monitoramento distribuído e escalável para sistemas de computação de alto desempenho, tais como *clusters* e grades. Ele permite ao usuário uma visão remota de estatísticas, tanto atual quanto históricas (tais como média da carga da CPU ou a utilização da rede) para todas as máquinas que estão sendo monitoradas.

- O Hawkeye [Hawkeye, 2006] utiliza as tecnologias já presentes no Condor e ClassAds para prover ricos mecanismos para coletar, armazenar e utilizar as informações sobre os computadores. Um sistema Hawkeye pode ser utilizado para monitorar vários atributos de uma coleção de sistemas. O mecanismo de monitoramento pode também ser utilizado para fazer o gerenciamento dos sistemas.

- O MonALISA (*Monitoring Agents using a Large Integrated Services Architecture*) [Caltech, 2006; Legrand, 2003] é um projeto desenvolvido pela Caltech e seus parceiros, com suporte do laboratório de pesquisa *U.S. Compact Muon Solenoid* (US CMS). O framework é baseado na *Dynamic Distributed Service Architecture* e está apto a prover o completo monitoramento, controle e serviços de otimização globais para sistemas complexos.

Capítulo 3

Tratamento de Conhecimento Semântico

Segundo definição encontrada em [de Holanda Ferreira, 1986]:

> *O conceito de conhecimento, no sentido mais amplo, é o atributo geral que tem os seres vivos de reagir ativamente ao mundo circundante, na medida de sua organização biológica e no sentido de sua sobrevivência.*

Neste intuito, segundo Rich and Knight [1993], o uso do conhecimento proporciona um meio de solucionar problemas complexos explorando as estruturas dos objetos envolvidos.

No começo dos estudos em Inteligência Artificial (IA), a resolução de problemas era simplesmente um mecanismo de busca de uso geral que possibilitava reunir passos elementares de raciocínio para encontrar soluções completas [Russell and Norvig, 2003]. Tal abordagem ficou conhecida como método fraco, pois embora geral, não se podia apenas aumentar a escala para instâncias de problemas grandes e de maior complexidade.

Segundo Luger [2002], existe uma outra abordagem em IA que trata do uso do conhecimento de forma mais ampla, porém, de domínio específico, que permite passos de raciocínio maiores e que pode tratar com mais facilidade casos que ocorram tipicamente em especialidades estritas, tais como a medicina interna ou cálculos integrais, ao invés de se projetar métodos heurísticos que generalize soluções em diversas áreas do domínio.

Estes métodos ficaram conhecidos como métodos fortes para resolução de problemas e incluem os sistemas baseados em conhecimento, os quais podem ser representados por regras de produção e baseados em modelos de raciocínio com recursos de aprendizagem simbólica. Assim, pode-se dizer que os métodos fortes enfatizam cada questão como a quantidade de conhecimento necessário para a resolução de problemas, aprendizado e aquisição do conhecimento, a representação

sintática do conhecimento, o gerenciamento de incertezas e questões relacionadas com a qualidade do conhecimento.

Para se tratar o conhecimento é importante se estabelecer uma linguagem que expresse através de uma sintaxe e semântica o que se deseja modelar. Essa modelagem pode ser realizada por sistemas lógicos, quaisquer que sejam eles (por exemplo, lógica proposicional de primeira ordem, modal, descritiva, temporal), uma vez que estes são sistemas expressos por linguagens cuja sintaxe e semântica estão muito bem definidas possibilitando uma forma de representação do conhecimento. Nesta dissertação, quando se fala em conhecimento, estamos nos referindo ao conteúdo que deve ser colocado na base de conhecimento e que representa fatos sobre o mundo real, representados através de sistemas de produção.

Dentre os possíveis modos de se representar o conhecimento declarativo, ideal para modelar o raciocínio humano [Schor, 1986], encontramos os sistemas de produção como modelos bem utilizados [Riley, 2004].

Em [Howe, 2003] encontramos como definição de sistemas de produção os sistemas que utilizam regras de produção para representação de conhecimento. Segundo o autor, um sistema de produção pode ser definido como:

- um conjunto de regras de produção;
- uma base de conhecimento onde são armazenados os fatos; e
- um algoritmo conhecido por encadeamento para frente ou *forward-chaining*, que produz novos fatos a partir de fatos já conhecidos.

Uma regra se torna pronta para ser disparada quando suas condições são satisfeitas por fatos da base de conhecimento. Uma estratégia de resolução de conflitos determina qual, dentre as regras que estão prontas, a próxima que será disparada.

Segundo Russell and Norvig [2003], as regras de produção são utilizadas para se representar um conhecimento heurístico sobre o mundo, especificando um conjunto de ações que devem ser realizadas para uma dada situação. Uma regra é composta por uma parte de condições, também conhecida por parte *se* da regra, ou lado esquerdo da regra; e uma parte de ações, que é a parte *então* da regra, ou o lado direito da regra. Uma condição apresenta uma lista de símbolos ou variáveis, que devem ser unificadas (ou casadas) com os fatos da memória de trabalho. Este processo de unificação ou casamento de padrão (*pattern matching*) é feito por um motor de inferência.

A regra a seguir ilustra um exemplo de regra de produção utilizada no sistema que foi implementado:

```
SE usuário solicita ao aplicativo as máquinas M da grade G com
  memória RAM disponível: RAM = 128 MB
  e
  nome do sistema operacional: SO = Linux
ENTÃO o sistema devolve
  máquinas MAQ que têm 128 MB de memória RAM disponível
  e
  nome do sistema operacional = Linux
```

A regra acima poderia ser lida como: para grade G, se RAM=128 e SO=Linux indica máquinas M com estes requisitos. Esta expressividade não existe, por exemplo, em sistemas onde não se pode definir quantificações para as variáveis que são declaradas.

Segundo da Figueira Filho [2000], o ciclo de execução de um sistema de produção com encadeamento para frente tem três passos principais:

- Unificação: no qual o sistema de produção procura unificar os fatos da memória de trabalho com as declarações das regras de modo a tornar verdade os predicados que compõem as condições das mesmas; a cada unificação bem sucedida, o par <regra, fatos> é inserido no conjunto de conflitos.

- Resolução de Conflitos: onde é escolhido um par <regra, fatos> do conjunto de conflitos, de acordo com a política de resolução de conflitos utilizada.

- Disparo da Regra: no qual as ações da regra escolhida na etapa anterior são executadas com suas variáveis substituídas pelos fatos que tornaram a regra disparável.

Este ciclo se repete até que não haja mais nenhuma regra disparável. A ação da regra pode alterar a lista de regras disparáveis, através do acréscimo ou da remoção de fatos da memória de trabalho. Estas operações, que convencionou-se chamar de *assert* e *retract* respectivamente, são a base da manipulação de fatos da memória de trabalho.

A definição de sistemas de produção a qual cita o encadeamento para frente como algoritmo não é a única. Outros autores, como [Jackson, 1998], definem sistemas de produção como podendo apresentar tanto encadeamento para frente ou *forward-chaining* quanto encadeamento para trás ou *backward-chaining*. No encadeamento para frente, a inferência é feita de fatos que acredita-se que sejam verdadeiros para novos estados que as condições nos permitem estabelecer. No

encadeamento para trás, o encadeamento se dá de um objetivo para as condições necessárias para que esse possa ser estabelecido.

Em sistemas do tipo encadeamento para frente, o motor de inferência tenta unificar o lado esquerdo da regra com as variáveis da base de conhecimento, executando em seguida as ações definidas no lado direito. Em sistemas com encadeamento para trás, o motor de inferência tenta unificar o lado direito das regras com os objetivos os quais se tenta provar, gerando como novos sub-objetivos as condições definidas no lado esquerdo, até que se atinjam condições atômicas (fatos já presentes na base de conhecimento).

Neste trabalho foi utilizado o sistema de produção com encadeamento para frente para tratamento das regras de inferência (seção 3.4) através da utilização do raciocinador Pellet. O objetivo do uso deste sistema de produção foi o de fazer a descoberta dos recursos da grade computacional. Além da descoberta de recursos feita de forma literal, este trabalho também trata da descoberta semântica dos recursos, a qual passamos a expor.

3.1 Tratamento Semântico

O principal aspecto da semântica é o de eliminar ambigüidades no tratamento de significados específicos a cada conceito de um domínio. Um exemplo prático desta abordagem é a Web Semântica, proposta por Tim Berners-Lee [Jacobs, 2005], e que tem como finalidade atribuir um significado aos conteúdos publicados na Internet de modo que seja perceptível tanto pelo humano quanto para o computador. Desta forma, podemos dizer que o objetivo principal da Web Semântica é o de desenvolver tecnologias e linguagens que tornem a informação legível para as máquinas. A idéia é desenvolver um modelo que permita o compartilhamento do conhecimento entre as máquinas. A integração de linguagens, como *eXtensible Markup Language* (XML) e *Resource Description Framework* (RDF), meta-dados, ontologias, agentes computacionais, dentre outras características, irá facilitar a criação de serviços Web que garantam a interoperabilidade e a cooperação.

3.2 Web Semântica

A Web atual é interoperável apenas sintaticamente, ou seja, ela é projetada para processamento humano dos significados dos dados. Dessa forma, a Web Semântica é considerada uma ampliação da Web, no sentido de prover uma infra-estrutura aberta, com tecnologia de Serviços Web, para que haja compartilhamento e tratamento da informação e do conhecimento, que será entendido também por má-

quina. Esta infra-estrutura é baseada em um modelo de domínio formal denominado ontologia.

O W3C é a organização responsável pela padronização da Web Semântica. Como órgão internacional formado por várias organizações e dirigido por Berners-Lee tem como objetivo o desenvolvimento de padrões da Web. De acordo com o W3C, para a Web alcançar seu potencial completo é fundamental que as tecnologias sejam compatíveis umas com as outras de forma a permitir que qualquer hardware ou software possa acessar a Web [Jacobs, 2005].

De acordo com o W3C, a Web só pode atingir seu pleno potencial se ela se tornar um local onde os dados possam ser compartilhados, processados e entendidos por ferramentas automatizadas da mesma forma que o são pelas pessoas. Os programas futuros devem ser capazes de compartilhar, processar e entender dados mesmo quando os mesmos são projetados independentemente uns dos outros [Fromm et al., 2005].

A Web Semântica se estende sobre as capacidades da Web atual e as tecnologias de informação existentes, permitindo uma maior colaboração e decisões mais inteligentes aos sistemas. Segundo Fromm et al. [2005], é uma agregação de *sites* Web inteligentes e armazenadores de dados acessíveis por um conjunto de tecnologias semânticas, *frameworks* e contratos de interação para permitir às maquinas fazer mais que o trabalho de responder a requisições de serviços, mas propiciar maior relevância de informações e inferência inteligente.

O W3C tem trabalhado no sentido de formalizar novas linguagens e estabelecer padrões relacionados, baseados em ontologias. Em 2000, o W3C apresentou uma proposta [Koivunen and Miller, 2001] que definia diversas novas camadas para a Web e sugeria linguagens e padrões para as tecnologias dessas camadas, como mostrado na figura 3.1.

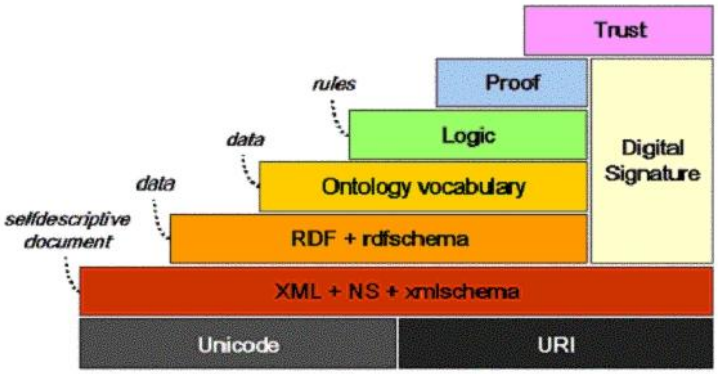

Figura 3.1: Camadas da Web Semântica [Koivunen and Miller, 2001].

As aplicações da Web Semântica consistem de duas camadas separadas e ao mesmo tempo, interligadas: a Camada da Web Semântica torna as ontologias e as interfaces disponíveis para o público, enquanto a Camada Interna consiste dos mecanismos de controle e raciocínio. Enquanto os últimos componentes podem residir dentro de uma caixa preta, os artefatos na camada da Web Semântica são compartilhados com outras aplicações, e, portanto devem encontrar padrões da mais alta qualidade do que os componentes internos. Os modelos na Camada da Web Semântica são utilizados para controlar o comportamento interno, em particular o resultado dos algoritmos de raciocínio. Como resultado, o código dentro dos componentes internos pode ser de tamanho relativamente pequeno e encontrar padrões de mais baixa qualidade do que os módulos visíveis externamente [Knublauch, 2004].

Atualmente, boa parte dos esforços de pesquisa tem sido dirigida para ontologias de sistemas. Ferramentas de desenvolvimento de ontologias, como o Protégé-OWL, permitem aos usuários encontrar erros e detectar inconsistências, semelhante a um depurador em um ambiente de programação. Além disto, o Protégé gera código a partir de uma ontologia em OWL, criando classes correspondentes em Java.

Em [Djuric et al., 2005] são definidos três níveis distintos na arquitetura da Web Semântica que introduzem primitivas expressivas: *metadata layer*, *schema layer* e *logical layer*. A figura 3.2, de [Djuric et al., 2005], ilustra esta arquitetura. A linguagem XML/XML *Schema* proporciona uma sintaxe comum, bem definida e de fácil processamento, porém, não diz nada sobre os dados que ela descreve. A semântica dos dados é abordada a partir da camada de metadados (*metadata layer*), através do *Resource Description Framework* (RDF), e da camada de esquema (*schema layer*), com a linguagem RDF *Schema* (RDFS). O modelo RDF define os relacionamentos entre recursos, mas para permitir raciocínio na Web Semântica seria necessário outra camada. A camada lógica (*logical layer*) introduz linguagens ontológicas, tais como OIL, DAML e OWL, que são baseadas na arquitetura de meta-modelo da camada mais baixa.

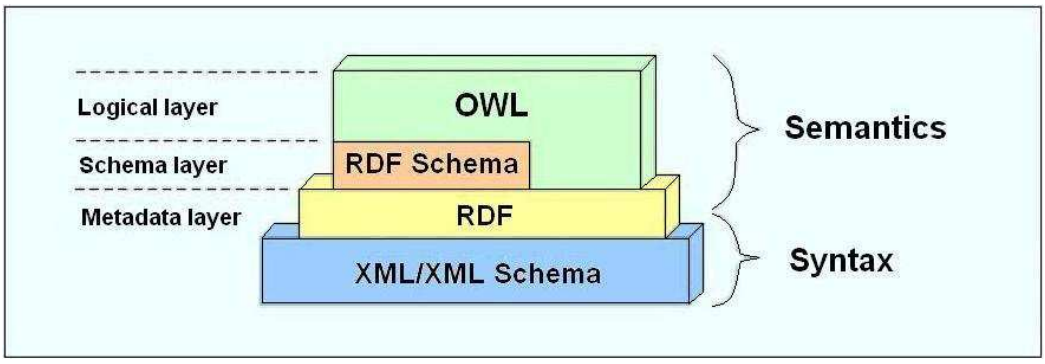

Figura 3.2: Linguagens na Arquitetura da Web Semântica [Djuric et al., 2005].

3.3 Ontologia

O termo ontologia tem origem no grego *ontos*, ser, e *logos*, palavra. Originalmente vem da palavra categoria, definida por Aristóteles, que pode ser utilizada para classificar coisas. O Dicionário Oxford [Blackburn, 1997] de filosofia define ontologia como:

> *[...] o termo derivado da palavra grega que significa 'ser', mas utilizado desde o século XVII para denominar o ramo da metafísica que diz respeito àquilo que existe.*

De acordo com Sowa [1999], o termo ontologia tem um sentido específico quando se trata de organização da informação, diferente do adotado na filosofia e em outras áreas. Ontologia é um catálogo de tipos de coisas em que se supõe existir um domínio, na perspectiva de uma pessoa que utiliza uma determinada linguagem.

Uma das definições mais conhecidas para ontologias é apresentada por [Gruber, 1993b, 2005]:

> *Uma ontologia é uma especificação explícita de uma conceitualização. [...] Em tal ontologia, definições associam nomes de entidades no universo do discurso (por exemplo, classes, relações, funções, etc. com textos que descrevem o que os nomes significam e os axiomas formais que restringem a interpretação e o uso desses termos) [...].*

Ou seja, ontologia é uma descrição (como uma especificação formal de um programa) dos conceitos e relacionamentos que podem existir para um agente ou comunidade de agentes.

O termo conceitualização corresponde a uma coleção de objetos, conceitos e outras entidades que se assume existirem em um domínio e os relacionamentos entre eles [Genesereth and Nilsson, 1987].

A ontologia provê uma linguagem compartilhada para uma comunidade de provedores e consumidores de serviços, sendo eles máquinas (por exemplo, agentes de software) ou pessoas [Gruber, 1993a]. As ontologias podem ser utilizadas como a espinha dorsal para cada uma das tarefas no ciclo de vida do gerenciamento do conhecimento. Elas servem para estruturar e recuperar as informações de um modo compreensivo e são utilizadas essencialmente para busca, troca e descoberta.

Segundo Guarino [1998], ontologia se refere a um sistema particular de categorias, de acordo com uma certa visão do mundo. Ela refere-se a um artefato de engenharia, constituído por um vocabulário específico utilizado para descrever uma certa realidade. O autor classifica as ontologias conforme ilustrado na figura 3.3 como:

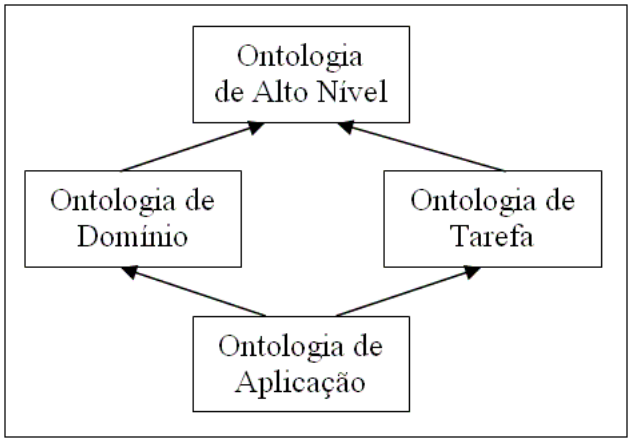

Figura 3.3: Classificação das Ontologias [Guarino, 1998].

- De Alto Nível: descreve os conceitos gerais como espaço, tempo, assunto, objeto, evento, ação, etc... os quais são independentes de um problema ou domínio específico.

- De Domínio / Tarefa: descreve o vocabulário relacionado ao domínio genérico (por exemplo, medicina, automóvel), ou uma tarefa/atividade genérica (exemplo: diagnóstico, venda), especializando os termos introduzidos na ontologia de alto nível.

- De Aplicação: descreve conceitos dependendo de um domínio e tarefa específicos, os quais são freqüentemente especializações das ontologias relacionadas. Esses conceitos correspondem aos papéis das entidades do domínio enquanto desempenham uma certa atividade como unidade substituível ou componente dispensáveis.

Segundo Studer et al. [1998], na área de Ciência da Computação, uma ontologia é uma especificação explícita e formal de uma conceitualização compartilhada. Esclarecendo os requisitos desta definição:

- Por especificação explícita, podemos entender as definições de conceitos, instâncias, relações, restrições e axiomas.
- Por formal, que é declarativamente definida, portanto, compreensível para agentes e sistemas, isto é, interpretável por máquina.
- Por conceitualização, que se trata de um modelo abstrato de uma área de conhecimento ou de um universo limitado de discurso.
- Por compartilhada, por tratar-se de um conhecimento consensual apresentado por um grupo, e não por apenas um indivíduo, seja uma terminologia comum da área modelada, ou acordada entre os desenvolvedores dos agentes que se comunicam.

Em [Russell and Norvig, 2003] encontramos uma ontologia para qualquer coisa relacionada ao mundo real, tal como ilustrada na figura 3.4. Segundo o autor, existe uma grande complexidade em se fazer uma descrição completa de alguma entidade do mundo real. Representar tudo o que existe no mundo então, é algo inimaginável. Por esta razão é que se opta por representar domínios restritos de conhecimento cujos conceitos devem ser elaborados preferencialmente por especialistas da área.

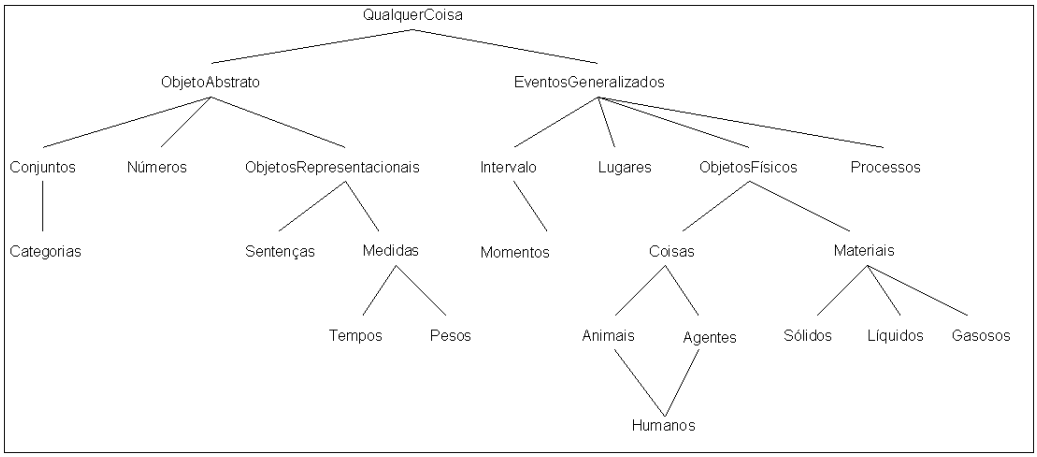

Figura 3.4: Ontologia de Qualquer Coisa [Russell and Norvig, 2003].

Conforme Knublauch [2004] e Gruber [2005] os componentes básicos de uma ontologia são:

- Classes: expressam qualquer coisa sobre a qual alguma coisa é dita (figura 3.4), são organizadas em uma taxonomia.
- Relações: que representam o tipo de interação entre os conceitos de um domínio.
- Axiomas: utilizados para modelar sentenças sempre verdadeiras.
- Instâncias: utilizadas para representar elementos específicos, ou seja, os próprios dados.

3.3.1 Linguagens

As considerações abaixo sobre as linguagens de ontologia foram retiradas de [Óscar Corcho et al., 2003].

No começo dos anos 1990 foram criadas linguagens de implementação de ontologia baseadas em IA. Basicamente o paradigma de Representação do Conhecimento lidava com linguagens de ontologia baseadas em lógica de primeira ordem (por exemplo, *Knowledge Interchange Format* ou KIF [Genesereth, 2006]), com *frames* combinados com lógica de primeira ordem (por exemplo, Ontolingua [Knowledge Systems AI Laboratory Stanford University, 2006b], *Operational Conceptual Modeling Language* ou OCML [Knowledge Media Institute, 2006] e *Frame Logic* ou FLogic [Kifer et al., 1990]) ou com lógica descritiva (por exemplo, Loom [University of Southern California's Information Sciences Institute, 2006]).

O crescimento do Internet conduziu à criação das linguagens de ontologia que exploram características da Web. Tais linguagens são chamadas geralmente linguagens de ontologia baseadas em Web ou linguagens de marcação de ontologia (*ontology markup languages*). Estas linguagens estão em fase de desenvolvimento e os relacionamentos entre elas foram mostrados na figura 3.2.

No ano de 2001, o W3C criou um grupo de trabalho chamado *Web-Ontology* (WebOnt), com o objetivo de desenvolver uma nova linguagem de marcação de ontologia para a Web Semântica, chamada *Ontology Web Language* ou OWL.

Atualmente, as linguagens para a representação de ontologias são divididas em dois grupos: as baseadas na Lógica de Primeira Ordem; e as baseadas em XML. Estes grupos de linguagens possuam diferentes expressividades e propriedades computacionais e são descritas a seguir.

- Linguagens Baseadas em Lógica de Primeira Ordem: são apenas citadas aqui, pois não fazem parte do foco do trabalho.
  - *Knowledge Interchange Format* (KIF) [Genesereth, 2006]

- Ontolingua [Knowledge Systems AI Laboratory Stanford University, 2006b]
- Loom [University of Southern California's Information Sciences Institute, 2006]
- *Operational Conceptual Modeling Language* (OCML) [Knowledge Media Institute, 2006]
- *Frame Logic* (FLogic) [Kifer et al., 1990]

- Linguagens Baseadas em *EXtensible Markup Language* (XML): A XML provê uma sintaxe para documentos estruturados e permite a definição de *tags* específicas de uma área, mas não impõe restrições semânticas sobre o significado. Apesar destas linguagens serem baseadas em XML elas obedecem fundamentos de IA, por exemplo, lógica descritiva, redes semânticas, *frames*, dentre outros, permitindo que muitas ontologias estejam sendo desenvolvidas em laboratórios de IA [Gómez-Pérez and Corcho, 2002].

  - RDF: segundo Lassila and Swick [1999], foi desenvolvido pela W3C (the World Wide Web Consortium) como uma linguagem baseada em redes semânticas para descrever recursos da Web e, também, o RDF *Schema* [Brickley and Guha, 2000; Connolly et al., 2006] que também foi desenvolvido pela W3C como uma extensão do RDF com primitivas baseadas em *frame*. A combinação de RDF com o RDF-*Schema* é conhecido como RDF(S). É mais expressiva, permite a representação de conceitos, taxonomias de conceitos e relações binárias. Algumas máquinas de inferência têm sido criadas para esta linguagem, principalmente para checar as restrições.

    Segundo Su and Ilebrekke [2002], uma das áreas em que o RDF é bastante aplicado é na descoberta de recursos, onde ela é capaz de prover melhores capacidades aos mecanismos de busca. O maior objetivo da linguagem RDF é definir um mecanismo para descrição de recursos que não faça suposições a respeito de um domínio de aplicação em particular e que não defina as semânticas de nenhum domínio de aplicação, nesta, a definição do domínio deve ocorrer de forma imparcial, mesmo que o mecanismo deva ser apropriado para a descrição de informações a respeito de qualquer domínio [Lassila and Swick, 1999].

    A linguagem RDF, como muitos outros sistemas de modelagem orientados a objetos, é baseada em um sistema de classes. Uma coleção

de classes é chamada de *schema*, e estas classes são organizadas em uma hierarquia, provendo extensibilidade através das subclasses definidas [Lassila and Swick, 1999]. O modelo de dados do RDF consiste de três tipos de objetos: recursos, propriedades (predicados descrevendo os recursos) e valor (objetos definindo um valor a uma propriedade a em um recurso). Uma definição de um recurso em RDF normalmente é vista com o um grafo RDF, onde sempre há triplas RDF. O RDF não possui mecanismos para a definição de relacionamento entre estes objetos, mas o RDF-Schema possui. RDF-Schema pode ser utilizado diretamente para descrever ontologias, apesar de seu principal propósito não ser a especificação de ontologias [Su and Ilebrekke, 2002]. RDF-Schema provê um conjunto fixo de primitivas de modelagem para a definição de ontologias (classe, recurso, propriedade, *is-a* e relacionamento *element-of*) e uma maneira padrão de codificá-las em XML. Mas o RDF-Schema possui um poder limitado de expressividade, uma vez que axiomas não podem ser definidos diretamente.

- *DARPA Agent Markup Language* (DAML) é uma linguagem que foi desenvolvida como uma extensão para XML e RDF. Provê uma infraestrutura básica que permite às máquinas fazerem a mesma classificação de inferências simples que os seres humanos fazem. Um sistema em DAML pode inferir conclusões que não estão explicitamente declaradas em DAML [Berners-Lee, 2006].
- *Ontology Inference Layer* (OIL) é uma camada de inferência e representação baseada na Web, que combina a utilização de modelagem de primitivas provenientes das linguagens baseadas em *frames* com a semântica formal. A sintaxe de OIL é baseada em RDF-Schema e provê definição de classes e *slots* (relações), e um conjunto limitado de axiomas [Su and Ilebrekke, 2002].
- A linguagem *DARPA Markup Language + Ontology Inference Layer* (DAML+OIL) foi construída com base na linguagem DAML em um esforço para combinar vários componentes da linguagem OIL. É construída sobre padrões W3C tais como RDF e RDF-Schema, e estende estas linguagens com primitivas de modelagem mais ricas, além de possuir semântica clara e bem definida [Connolly et al., 2001].
- A *Ontology Web Language* (OWL) é uma linguagem de marcação semântica para publicação e compartilhamento de ontologias na Web. A OWL foi desenvolvida como uma extensão do vocabulário RDF e de-

rivada da linguagem ontológica para Web DAML+OIL [Djuric et al., 2005]. A OWL representa o significado de termos em vocabulários e relacionamentos entre eles, facilitando a interpretação do conteúdo Web por máquinas, já que oferece um vocabulário adicional com uma semântica formal, por exemplo, relações entre classes, cardinalidade, características de propriedades.

Enquanto o RDF promove a associação e integração de dados distribuídos, a OWL possibilita o raciocínio sobre esses dados [Niemann et al., 2005]. A OWL também é mais completa do que a RDFS, pois permite definir tudo que a RDFS é capaz e algo mais. É possível, por exemplo, estabelecer que duas classes são disjuntas, o que não ocorre com a RDFS. Na OWL é possível também estabelecer nas subclasses restrições de propriedades herdadas das superclasses, podendo ter tais restrições relações com outras classes.

A OWL é dividida em três sub-linguagens, de acordo com o nível de formalidade exigido e a liberdade dada ao usuário para a definição de ontologias: OWL *Lite*, OWL DL e OWL *Full*. Em ordem crescente de expressão, estas três sub-linguagens servem para atender tanto os mais exigentes quanto aqueles que precisam apenas de uma maneira rápida para desenvolver uma ontologia. Vale lembrar para aqueles que pretendem estudar a OWL que os três níveis da linguagem se baseiam num incremento dos recursos disponíveis, assim uma boa forma de começar é aprender a OWL *Lite*. Posteriormente poderão ser assimilados os conceitos da OWL DL e finalmente da OWL *Full*.

* A OWL *Lite* é a sub-linguagem mais simples sintaticamente. É indicada para usuários que precisam de uma hierarquia de classificação e restrições simples. Por exemplo, dá suporte a cardinalidade, no entanto esta só pode assumir os valores 0 ou 1. É mais simples fornecer suporte a OWL *Lite* do que a suas irmãs, e também é um caminho rápido para fornecer a migração de taxonomias existentes para OWL. OWL DL é para aqueles usuários que querem o máximo de expressividade mas que querem reter toda a computabilidade (todas as conclusões podem ser computadas) e decidabilidade (o processamento não é infinito). E por fim, a OWL *Lite* possui complexidade formal mais baixa que a OWL DL.
  O protótipo implementado neste trabalho fez uso da ontologia representada em OWL *Lite*, por isso citamos algumas características

desta linguagem relacionadas a RDFS:

  - **Class**: Uma classe define um grupo de indivíduos porque eles partilham algumas propriedades.
  - **rdfs:subClassOf**: A hierarquia entre classes pode ser criada através de uma ou mais declarações de que uma classe é subclasse de outra classe.
  - **rdf:Property**: Propriedades podem ser utilizadas para declarar relacionamentos entre indivíduos ou de indivíduos para valores de dados.
  - **rdfs:subPropertyOf**: Hierarquia de propriedades podem ser criadas a partir de uma ou mais declarações de que uma propriedade é subpropriedade de uma ou mais subpropriedades.
  - **rdfs:domain**: O domínio de uma propriedade limita os indivíduos para as quais a propriedade pode ser aplicada. Se uma propriedade relaciona um indivíduo com outro, e a propriedade tem uma classe como um de seus domínios, então o indivíduo deve pertencer a essa classe.
  - **rdfs:range**: A imagem (*range*) de uma propriedade limita os indivíduos que a propriedade pode ter como valor. Se uma propriedade relaciona um indivíduo com outro indivíduo, e a propriedade tem uma classe como sua imagem, então o outro indivíduo deve pertencer a classe da imagem.
  - **Individual**: Indivíduos são instâncias de classes, e propriedades podem ser utilizadas para relacionar um indivíduo com outro.

* OWL DL proporciona expressividade máxima e garantia de completude computacional e decidibilidade. Esta linguagem inclui todos os construtores OWL mas adiciona algumas restrições. As restrições mais significativas são: uma classe não pode ser um indivíduo ou propriedade e uma propriedade não pode ser um indivíduo ou classe. OWL DL possui esse nome devido à sua correspondência com a lógica descritiva (*Description Logics*), um campo de pesquisa da lógica formal sobre o qual se baseia a linguagem OWL [McGuinness and van Harmelen, 2004].
* A OWL *Full*, segundo Djuric et al. [2005], é uma extensão da OWL DL, que é uma extensão da OWL *Lite*; desta forma, toda ontologia OWL *Lite* é uma ontologia OWL DL e OWL *Full*, e toda ontologia

OWL DL é uma ontologia OWL *Full*. A OWL *Full* provê expressividade máxima, além da sintaxe livre do RDF, mas não provê qualquer garantia computacional. Por exemplo, uma classe pode ser tratada, como uma coleção de indivíduos e como um indivíduo simultaneamente. OWL *Full* pode aumentar o significado do vocabulário pré-definido (em RDF ou OWL). A principal característica da OWL *Full* em comparação com a OWL DL e OWL *Lite* é que uma classe, que é por definição uma coleção de indivíduos, pode ser o próprio indivíduo, como em RDF-Schema. OWL *Full* pode ser visto como uma extensão ao RDF, ao passo que OWL *Lite* e OWL DL seriam extensões a visões restritas do RDF. Todo documento OWL é um documento RDF e todo documento RDF é um documento OWL *Full*. No entanto, nem todo documento RDF é um documento OWL *Lite* ou OWL DL (pois estas linguagens são mais restritas). Por essa razão, deve haver um certo cuidado na hora de migrar documentos RDF para OWL *Lite* ou OWL DL.

3.3.2 Ferramentas

Existem diversas ferramentas para o tratamento de ontologias. Passaremos a expor sobre o Protégé, que foi a ferramenta utilizada neste trabalho, além de citarmos algumas outras ferramentas existentes, tais como o OilEd, OntoEdit, DOE e Chimaera.

O Protégé [Stanford Medical Informatics at the Stanford University School of Medicine, 2006] é um ambiente integrado utilizado por desenvolvedores de sistemas e especialistas em um domínio específico para desenvolver e manter uma ontologia. Através dele é possível descrever os conceitos pertencentes à ontologia, juntamente com seus atributos e relacionamentos. O editor Protégé-2000 possui uma arquitetura de metaclasses, documentos de formato padrão utilizados para definir novas classes em uma ontologia, que o tornam facilmente extensível e permite o seu uso juntamente com outros modelos de conhecimento. O Protégé apresenta interoperabilidade com outros sistemas de representação do conhecimento e além disto possui uma interface intuitiva que torna simples a sua utilização e configuração.

Esta ferramenta foi desenvolvida na Universidade de Stanford e está disponível para utilização gratuitamente. Fornece uma interface gráfica amigável para o desenvolvimento de ontologias, proporcionando o acesso rápido às informações importantes, podendo-se manipular diretamente a ferramenta para navegação e

gerenciamento da ontologia. Ao contrário de outras ferramentas, o Protégé utiliza a instalação local ao invés de utilizar uma arquitetura cliente-servidor através da Internet. Além disso, gera saídas em diversas linguagens ontológicas e possibilita aos usuários modificá-lo para ser editor de uma linguagem específica [Mizoguchi, 2003]. Utilizaremos esta ferramenta para implementação da proposta deste trabalho, tendo em vista as vantagens apresentadas, objetivos e a disponibilidade da ferramenta sem custo.

Segundo Stanford Medical Informatics at the Stanford University School of Medicine [2006], o Protégé é uma ferramenta que permite aos usuários construir domínios ontológicos e customizar formulários de entrada de dados. É também uma plataforma que pode facilmente ser estendida para incluir componentes gráficos como tabelas, arquivos de som, imagens e vídeos, e vários formatos de armazenamento como OWL, RDF, XML, além de tratar HTML.

A *Protégé* API possibilita a outras aplicações utilizar, acessar e disponibilizar bases de conhecimento criados com o Protégé. Protégé é um editor de ontologias livre, com código aberto, que inclui um *framework* de base de conhecimento. O Protégé é baseado em Java e suportado por uma grande comunidade de desenvolvedores e acadêmicos, usuários do governo e corporativos, que o estão utilizando como solução em diversas áreas como biomedicina e modelagem de entidades e relacionamentos em corporações.

O Protégé pode ser utilizado como um *plugin* do Eclipse e pode ser utilizado em qualquer máquina que tenha uma JDK instalada. Suporta qualquer banco de dados que tenha driver JDBC como: *Oracle*, *MySQL*, *Microsoft SQL Server* e *Microsoft Access*. Permite múltiplos usuários simultaneamente e tem uma API documentada para auxílio aos usuários.

Os desenvolvedores do Protégé, percebendo o potencial de desenvolvimento de seus usuários, tornou o código da ferramenta aberto, permitindo a colaboração de toda a comunidade de desenvolvedores. A partir daí, surgiu uma arquitetura integrável a diversas aplicações, através de componentes que podem ser conectados ao sistema. Essa arquitetura propiciou o desenvolvimento de *plugins* que acrescentam novas funcionalidades acopladas às já existentes. O Protégé permite que aplicações externas utilizem sua API. Portanto, seria possível utilizar partes da ferramenta, mas em uma interface desenvolvida especificamente para uma determinada aplicação.

Dentre os diversos *plugins* existentes e em desenvolvimento pela comunidade de usuários do Protégé, em nosso trabalho utilizamos o OntoViz Tab, que permite lidar com ontologias no Protégé através da ajuda de um sofisticado software para

visualização em grafo (Graphviz da ATeT). A ferramenta dá a opção de selecionar um conjunto de classes e instâncias para visualizar partes de uma ontologia e especificar cores para nós do grafo e outros elementos.

Além de *plugins* para modificações na interface com o usuário, a arquitetura do Protégé-2000 permite *backend plugins* que lêem e escrevem em diferentes formatos de armazenamento. Como padrão, a ferramenta armazena suas bases de conhecimento em um formato de arquivo de propósito especial. Atualmente podem ser criadas classes e instâncias em CLIPS, OIL, Jess, XML, RDF, F-Logic, Prolog, Topic Maps (linguagem padrão ISO para descrever estruturas de conhecimento em páginas da Web) , além do armazenamento em bancos de dados relacionais. Essa última capacidade é fundamental para grandes bases de conhecimento, que excedem o tamanho físico da memória principal e precisam de uma maneira eficiente de recuperar a informação de uma memória secundária.

- OilEd [Bechhofer and Ng, 2006]: é um editor de ontologia cujo objetivo inicial era demonstrar o uso e estimular o interesse na linguagem Oil. Ele utiliza o raciocinador FaCT para verificar a consistência das ontologias junto com a interface DIG, que permite o uso de outros raciocinadores para classificação das ontologias. O OilEd permite exportação para OWL, além de DAML+OIL e RDFS.

- OntoEdit [Mizoguchi, 2003]: é um ambiente gráfico de desenvolvimento para projeto e manutenção de ontologias. Possui embutida uma máquina de inferência chamada FaCT. O processo de desenvolvimento de ontologia no OntoEdit é baseado em uma metodologia própria denominada *On-To-Knowledge*.

  O sucessor do OntoEdit é o OntoStudio [OntoPrise, 2006], que é baseado no ambiente de desenvolvimento Eclipse, da IBM, e está disponível nas versões: profissional e livre (para uso não comercial). As linguagens suportadas são: OWL e RDF, além de F-logic e OXML. Estas últimas são linguagens otimizadas para processamento de ontologias baseado em lógica. O OntoStudio inclui também um avaliador, chamado *OntoStudio Evaluator*, utilizado na implementação de regras durante a modelagem.

- DOE (*The Differential Ontology Editor*) [Isaac and Troncy, 2006]: é um editor simples para criação de ontologias. No DOE são construídas as taxonomias dos conceitos e relações, justificando explicitamente a posição de cada item na hierarquia. Feito isto, o usuário pode então adicionar sinônimos e definições. Considera-se duas taxonomias do ponto de vista semân-

tico extensional. O usuário poderá aumentá-las ou adicionar restrições às relações. A ontologia pode ser traduzida em uma linguagem de representação do conhecimento. As linguagens disponibilizadas são: RDFS, OWL, DAML+OIL, OIL, CGXML (linguagem para especificação de gráficos conceituais). O DOE não é um ambiente de desenvolvimento completo de ontologia. Ele oferece técnicas inspiradas em lingüística, que atribuem uma definição léxica aos conceitos e relações utilizados e justifica a hierarquia do ponto de vista teórico, que pode ser entendido por pessoas.

- Chimaera [Knowledge Systems AI Laboratory Stanford University, 2006a]: é um sistema de software que permite aos usuários criar e manter ontologias distribuídas na Web. As duas maiores funções que ele suporta são mesclagem de múltiplas ontologias e diagnóstico de várias ontologias ou individualmente. Também permite o carregamento de bases de conhecimento em diferentes formatos, reorganização de taxonomias, resolução de conflitos de nome, edição de termos, entre outras opções. O Chimaera está disponível para uso na internet, solicitando registro para acesso completo à sua versão funcional. Ele pode carregar e exportar arquivos no formato OWL e DAML.

3.3.3 Ontologia de Grade Computacional

O GLUE *Schema* [Andreozzi, 2006] é uma modelagem abstrata para recursos de grade e mapeia para esquemas concretos que podem ser utilizados em Serviços de Informação da Grade. A definição deste esquema iniciou em 2002 como um esforço em colaboração entre os projetos EU-DataTAG e US-iVDGL. Os projetos atuais que estão participando desta atividade são EGEE, LCG, Grid3/OSG, Globus e NorduGrid. Quanto a estrutura do *Schema*, a estrutura básica do GLUE é mostrada na figura 3.5. Um ambiente de computação em grade é geralmente formado por vários *hosts* independentes, porém pode também incluir *hosts* que estejam agrupados em um *cluster*. Os componentes principais do modelo inicial da informação do GLUE são:

- Elemento de Computação ou *Computing Element*. Um elemento de computação representa um ponto de entrada em um sistema de filas. Há um elemento de computação por a fila. Os sistemas com filas múltiplas são representados criando um elemento de computação por fila. A informação associada com um elemento da fila é limitada somente à informação relevante à fila. Toda a informação sobre os recursos físicos acessados por uma

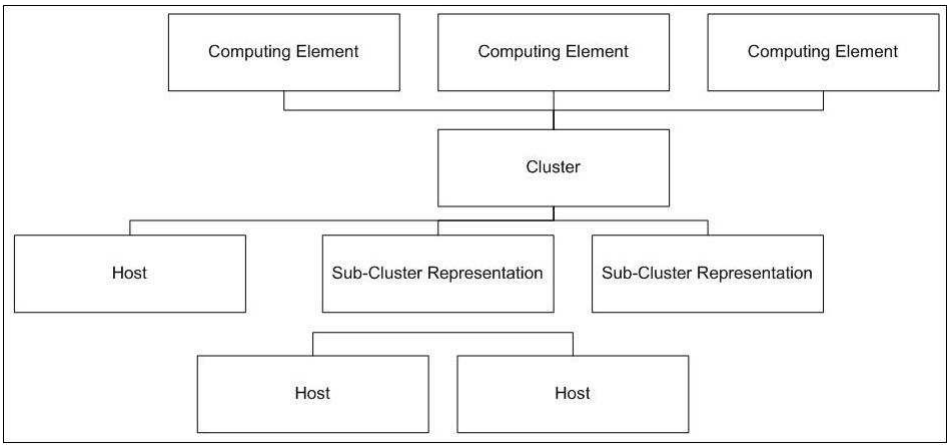

Figura 3.5: Estrutura do GLUE *Schema* [Andreozzi, 2006].

fila é representada pelo elemento de informação do *Cluster*.

- *Cluster*. Um *cluster* é um *container* que agrupa *subclusters* ou nós. Os elementos do *subcluster* representam coleções "homogêneas" de nós computacionais, enquanto os nós representarem nós únicos, tais como os nós *head*, ou nós individuais de computação. Um *cluster* pode ser referenciado por mais de um elemento de computação.

- *SubCluster*. Um *subcluster* representa uma coleção "homogênea" de nós onde a homogeneidade é definida por uma coleção cujos atributos dos nós requeridos têm todos o mesmo valor. Por exemplo, um *subcluster* representa um conjunto de nós com o mesmo processador, memória, sistema operacional, interfaces de rede, etc. Estritamente falando, os *subclusters* não são necessários, mas fornecem uma maneira conveniente de representar coleções úteis dos nós. Um *subcluster* captura uma contagem do nó e um conjunto de atributos para os quais valores homogêneos estão sendo declarados.

- *Host*. Representa um elemento físico de computação. Este elemento caracteriza a configuração física de um nó de computação, incluindo processadores, software, elementos do armazenamento, etc.

Serão descritas agora cada uma das classes ontológicas que inserimos na ontologia utilizada em nosso trabalho. Observe na figura 3.6 que o GLUE *Schema* provê alguns outros elementos de computação [Andreozzi, 2006], porém vamos descrever apenas aqueles que estão sendo utilizados na ontologia de nosso protótipo, que são justamente aqueles que conseguimos obter com o Ganglia, que foi o serviço de monitoramento de recursos utilizado [Efstathiadis, 2006].

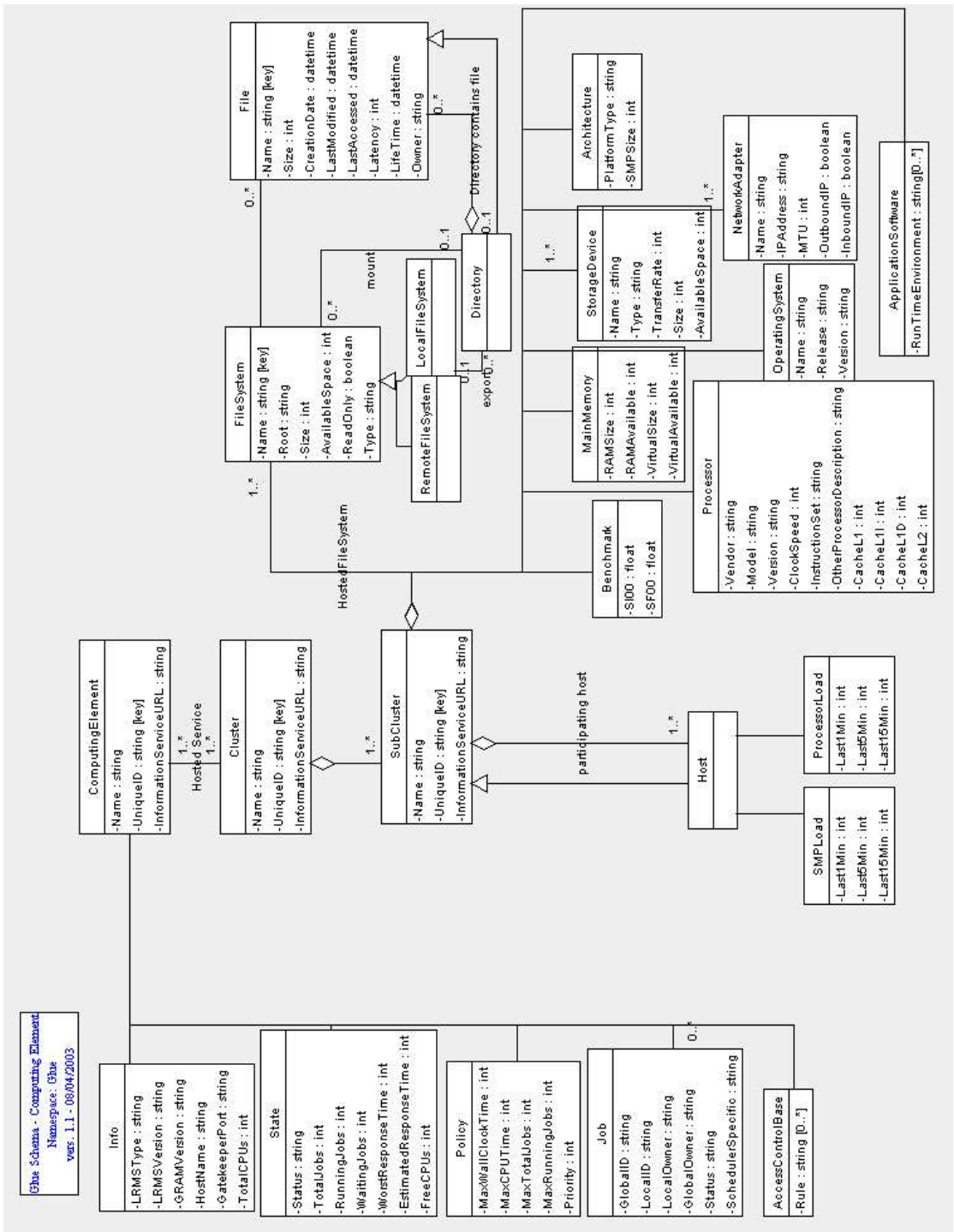

Figura 3.6: Diagrama de Classe UML do Elemento de Computação [Andreozzi, 2006].

- *Host*: o elemento *Host* é utilizado para representar detalhes de um elemento de computação específico.
  - *Name*: um nome para o *host*.
  - *UniqueID*: um identificador único para este *host*.
- *Architecture*: Arquitetura
  - *SMPSize*: o número de processadores no modo de multiprocessamento simétrico ou *Symmetric Multi-Processing* (SMP).
- *Operating System*: Sistema Operacional
  - *Name*: indica informalmente o nome sistema operacional utilizando a convenção específica do vendedor.
  - *Release*: indica informalmente o nome da *release* do sistema operacional utilizando a convenção específica do vendedor.
- *Processor*: Processador
  - *CacheL1*: tamanho da *cache* nível 1 (em KB) de uma CPU.
  - *CacheL1I*: tamanho da *cache* nível 1 de instruções (em KB) de uma CPU.
  - *CacheL1D*: tamanho da *cache* nível 1 de dados (em KB) de uma CPU.
  - *Cachel2*: tamanho da *cache* nível 2 (em KB) de uma CPU.
  - *ClockSpeed*: a freqüência em MHz associada com a CPU.
  - *InstructionSet*: nomeia informalmente a arquitetura do conjunto de instruções ou *Instruction Set Architecture* (ISA) do Host.
- *MainMemory*: Memória Principal
  - *RAMSize*: a memória física configurada em um processador em MB.
  - *RAMAvailable*: o tamanho da RAM que não está alocado em MB.
  - *VirtualSize*: memória virtual baseada no disco configurada no *host* em MB.
  - *VirtualAvailable*: memória virtual disponível.
- *FileSystem*: Sistema de Arquivo

  - *Root*: o nome do caminho ou alguma outra informação que define a raiz do sistema de arquivo.
  - *Name*: o nome de um sistema de arquivos.
  - *ReadOnly*: o sistema de arquivo é somente leitura?
  - *Size*: espaço total relacionado a este tipo de arquivo (em MB).
  - *AvailableSpace*: espaço total disponível para este tipo de arquivo (em MB).

- *ProcessorLoad*: Carga do Processador
  - *Last1min*: a média de utilização do processador durante o último minuto.
  - *Last5min*: a média de utilização do processador durante os últimos 5 minutos.
  - *Last15min*: a média de utilização do processador durante os últimos 15 minutos.

- *NetworkAdapter*: Adaptador de Rede
  - *Name*: o nome de uma interface de rede.
  - *IPAddress*: o endereço IP de uma interface de rede.
  - *OutboundIP*: define se é permitida a conectividade de dentro para fora da grade pelos *worker nodes*, ou seja, pode um *worker node* iniciar uma conectividade de dentro para fora da grade?
  - *InboundIP*: define se é permitida a conectividade de fora para dentro da grade.

3.4 Mecanismos de Inferência

Os raciocinadores ou *reasoners* são capazes de fazer inferências sobre objetos de dados utilizando a informação de seus relacionamentos definidos através de uma ontologia, ou seja, verifica se novas informações podem ser derivadas de uma ontologia. Os raciocinadores permitem a verificação ou a validação de uma ontologia, verificando se todas as regras definidas pela ontologia são obedecidas. Nesta dissertação foi utilizado o Pellet, que será citado a seguir, juntamente com outros raciocinadores:

- Pellet: o mecanismo de inferência Pellet [University of Maryland's Mindswap Lab, 2006; Sirin and Parsia, 2004] é um software de código aberto feito em Java usando o OWL e baseado no tipo de lógica *Description Logics* ou DL. O Pellet possibilita tratamento de dados no formato XML e pode ser utilizado em conjunto tanto com o Jena quanto a bibliotecas da API OWL e também provê uma inteface DIG. O Pellet provê funcionalidades para verificar validações de espécies, faz checagem de consistência das ontologias, classificação das taxonomias e responder a consultas RDQL.

- Racer: o Racer [Racer Systems, 2006] é um sistema de representação de conhecimento baseado em DL. O sistema provê suporte para especificações de terminologias em geral, através do uso de axiomas, inclusive axiomas que descrevam a relação hierárquica entre conceitos, definições múltiplas ou cíclicas. O sistema também provê a verificação de consistência, além de várias funções e serviços de recuperação de informação pré-programados.

- FaCT: o sistema FaCT [Horrocks, 2006] é um classificador de DL que também pode ser usado para teste de satisfabilidade em lógica modal. Além de implementar uma DL com alto grau de expressividade, o sistema provê suporte para raciocínio envolvendo bases de conhecimento que contenham axiomas gerais de inclusão de conceitos.

A tabela 3.1, obtida de [Zou et al., 2004], representa uma comparação entre estes mecanismos de inferência.

Tabela 3.1: Comparação entre Mecanismos de Inferência [Zou et al., 2004].

	Pellet	Racer	FaCT
Lógica	DL	DL	DL
Suporta	OWL-DL	OWL-DL	OWL-DL
Baseada em	Java	Lisp	Lisp
Tipo de dado em XML	Sim	Sim	Não
Decidível	Sim	Sim	Sim
Checagem de consistência	Sim (OWL-Lite)	Sim (OWL-Lite)	Sim
Interface	DIG, Java e Jena	DIG, Java, GUI	DIG, Linha de comando
Query	RDQL	Racer query language	-

3.4.1 Linguagens de Consulta de Inferência

Como dito anteriormente, neste trabalho a linguagem de consulta de inferência utilizada foi a RDQL. A seguir será feita uma breve descrição desta e de outras linguagens de consulta, conforme o estudo comparativo apresentado no artigo de Haase et al. [2004].

- RDQL: é atualmente a linguagem padrão, segundo o W3C [Seaborne, 2004]. A RDQL tem como propósito prover um meio de acesso aos documentos RDF e é uma linguagem orientada a dados, pois não possibilita aplicar o mecanismo de inferência, permite apenas extrair dados de documentos RDF disponível na web ou armazenados em um meio físico qualquer. A sintaxe do RDQL segue o padrão de seleção semelhante ao SQL, onde a cláusula *where* é omitida.

```
SELECT ?n WHERE
<s:Paper>, <s:author>, ?c),
(?c, ?collection, ?a), (?a, <s:name>,
?n) USING s FOR ...
```

- RQL [Karvounarakis et al., 2002]: é uma linguagem tipada que segue uma abordagem funciona e que suporta variáveis que caracterizam de expressões generalizadas, tanto para os nós quanto para aos arcos de um grafo RDF.

```
select PersonName from {X} s:author {y}. rdfs:member
{z}. s:name {PersonName}
using namespace
s = ... rdfs = ...
```

- SeRQL [Broekstra and Kampman, 2004]: significa Linguagem de Consulta RDF Sesame ou *Sesame RDF Query Language* e é uma linguagem de consulta e transformação fracamente baseada em várias linguagens existentes, mais notavelmente RQL, RDQL e N3.

```
select PersonName
from {X} <s:author> {} <rdfs:member>
{} <s:name> {PersonName}
```

```
using namespace
s = <!...>
```

- TRIPLE: o termo *Triple* denota uma linguagem de regras e de consulta [Sintek and Decker, 2001].

```
FORALL C,X,A,N <-
( s:Paper[s:author->C] AND
C[X->A] AND
A[s:name->N] )@rdfschema(s:ln).
```

- N3: a Notation3 (N3) provê uma sintaxe baseada em texto para o RDF. Assim, o modelo de dados da N3 está de acordo com o modelo de dados da RDF. Adicionalmente a N3 permite a definição de regras, que é denotada utilizando uma sintaxe especial.

```
{ ?y s:author ?z.
?z ?p ?x.
?x a s:Person; s:name ?result.
} => { ?result a :QueryResult.}.
```

- Versa: tem uma abordagem interessante em que o bloco principal de construção da linguagem é uma lista de recursos RDF.

```
QUERY=((s:Paper
- s:author -> *) - properties(.) -> *)
- s:name -> *
```

Capítulo 4

Proposta de Trabalho

Neste capítulo apresentamos a proposta de trabalho, a qual inclui a apresentação da arquitetura proposta, aspectos de implementação do protótipo e alguns trabalhos correlatos.

4.1 Arquitetura

Neste trabalho é estudado e proposto um projeto de arquitetura de Grade Semântica com o desenvolvimento de um protótipo implementado em Java. É importante salientar que a arquitetura proposta tem como foco a descoberta de recursos via múltiplos atributos, ela não aborda aspectos de execução, submissão e escalonamento das tarefas.

A arquitetura de Grade Semântica proposta tem quatro camadas, que serão descritas neste capítulo. A figura 4.1 ilustra a arquitetura que apresenta a Camada de Conhecimento implementada no topo da Camada de *Middleware*, que por sua vez se encontra em cima da Camada Física ou *Fabric*.

A Camada *Fabric*

Esta camada está diretamente ligada aos recursos computacionais e de rede, os *clusters* de computadores, super-computadores, os sistemas de armazenamento e entidades lógicas, tais como dispositivos de armazenamento distribuído em que o acesso distribuído é mediado por protocolos de grade.

A Camada de *Middleware*

Nesta camada é provido o acesso unificado e seguro aos recursos remotos, podendo-se utilizar diferentes *middlewares* de grade, tais como o Globus, Legion, Unicore, Alchemi, dentre outros. No presente trabalho esta camada foi implementada utilizando o Globus *Toolkit* versão 4 [The Globus Alliance, 2006b; Foster,

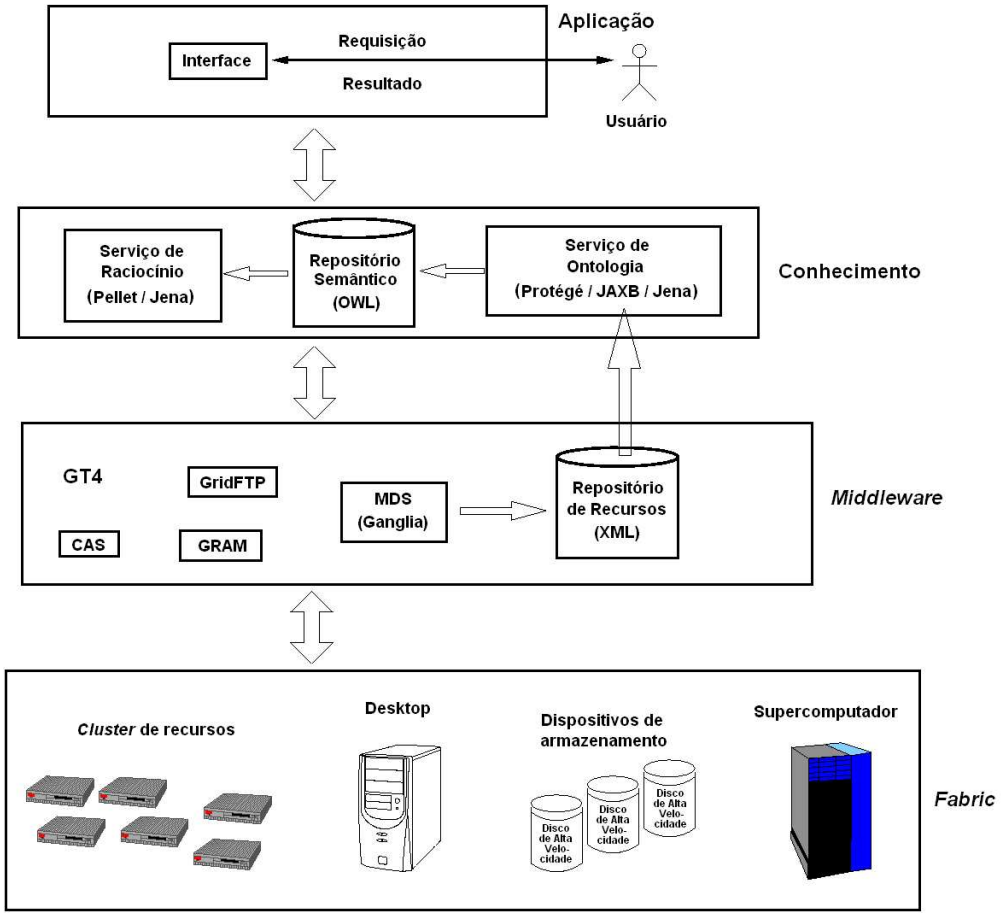

Figura 4.1: Arquitetura de Quatro Camadas de Grade Semântica.

2006]. Foi utilizado também o provedor de informação Ganglia, que é escrito em Perl [Efstathiadis, 2006]. A informação de saída do Ganglia, que é publicada no MDS, está determinado pelo GLUE-CE Schema (um modelo abstrato para recursos e mapeamentos da grade para esquemas concretos que podem ser utilizado nos Serviços de Informação da Grade [Andreozzi, 2006]). Na Camada de *Middleware*, os dados sobre os recursos da grade capturados pelo provedor de informações Ganglia são armazenados em um Repositório de Recursos em formato XML.

A Camada de Conhecimento

Esta camada é o foco principal deste trabalho e provê o serviço de descoberta semântica de recursos de uma grande quantidade de dados agregados da camada de serviços de informação que estão armazenados no Repositório de Recursos. Para capturar as informações dos recursos da grade em XML e transferi-las para o Repositório Semântico, na linguagem de ontologia Web OWL [Dean and Schreiber, 2004; Patel-Schneider et al., 2003], foi utilizado o Jena, que é um *framework*

para aplicações de Web Semântica. O Jena é um software de código aberto e provê um ambiente programático para OWL e uma API OWL. O projeto Jena surgiu como um programa de Web Semântica dos laboratórios da HP [HP Labs Semantic Web Research, 2006]. A arquitetura do Jena provê um mecanismo para anexar raciocinadores externos aos modelos Jena, através da reimplementação da interface gráfica. A versão 2.1 do Jena provê uma ilustração desta técnica provendo um *gateway* transparente entre os modelos de ontologia do Jena e raciocinadores externos implementando a interface de descrição lógica do raciocínio ou *Description Logics Implementors Group Interface* (DIG). Para o ambiente da Grade Semântica proposta, os usuários e os agentes de software devem ser capazes de descobrir os nós da grade oferecendo serviços requeridos e tendo propriedades particulares. Para o que está sendo proposto, a Camada de Conhecimento é formada pela ontologia e os componentes de raciocínio, como já apresentado na figura 4.1. Resumindo, a Camada de Conhecimento é orientada a domínio e utiliza o conhecimento da grade, que foi construído através da ontologia da grade. Serão descritos abaixo os componentes implementados na Camada de Conhecimento.

O Componente Ontológico da Camada de Conhecimento

Para o componente ontológico nós utilizamos o Protégé (seção 3.3.2) para lidar com os recursos da grade semanticamente. Para cada tipo de informação recuperada do nó da grade, criamos instâncias dos conceitos apropriados no *template* ontológico conforme os respectivos tipos de conceitos. Desta forma, um modo em que uma instância será exatamente relacionada com apenas um tipo de conceito e os valores de várias propriedades recuperadas são associadas às respectivas propriedade dos conceitos apropriados no *template* ontológico. Neste trabalho o *template* ontológico é definido como a ontologia dos recursos da grade que prevêem a hierarquia de conceitos nos quais algumas propriedades definem suas características.

A figura 4.2 mostra o *template* da ontologia com a hierarquia de conceitos considerada. Os valores das propriedades consideradas para definir os conceitos são recuperados através do sistema MDS do *middleware* Globus (seção 2.4). Este *template*, que não é definitivo e por isto pode sofrer evoluções, define um mínimo de conceitos e propriedades providos pelo serviço de informação de recursos MDS/Ganglia para cada um dos *host* da grade.

Pode-se notar que este *template* da ontologia é formado por conceitos e propriedades que correspondem aos valores das instâncias e propriedades que constituem

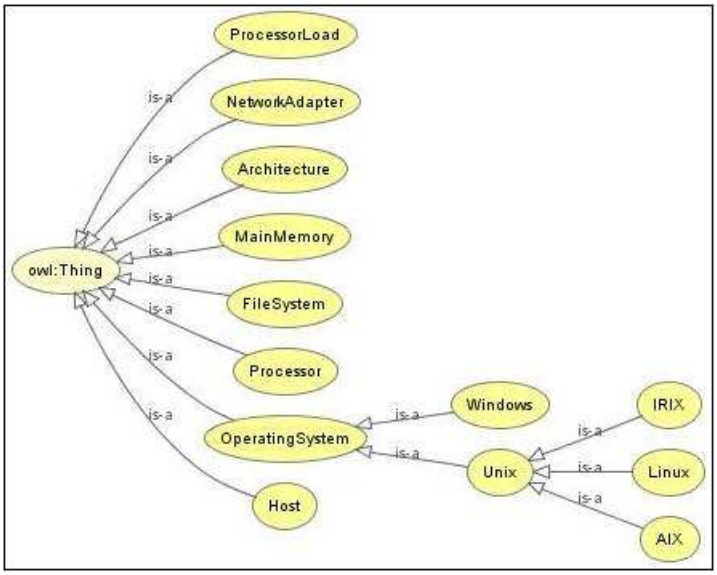

Figura 4.2: O *Template* da Ontologia.

o Repositório Semântico dos recursos da grade. A descrição semântica dos recursos no repositório facilita o uso de mecanismos de inferência para interagir com a base e recuperar informações semanticamente. O repositório de recursos é atualizado periodicamente, assim, a adição ou remoção de recursos é contabilizada no repositório semântico dinamicamente.

É importante observar que este trabalho não trata de questões relacionadas à política de segurança, controle de acesso aos recursos e nem das restrições de tempo de uso dos recursos pelos usuários, mas que tais aspectos podem ser tratados pela ontologia da grade computacional.

O Componente Raciocinador da Camada de Conhecimento

O Componente de Raciocínio utiliza o motor de inferência Pellet, que pode ser utilizado juntamente com o Jena ou com bibliotecas da API OWL. O Pellet é baseado em algoritmos *tableaux* desenvolvido para a expressiva *Description Logics* (DL). O algoritmo *tableaux* é utilizado nos sistemas de raciocínio DL mais modernos, sua grande vantagem é iniciar com um algoritmo base relativamente simples que pode ser sucessivamente estendido de modo a suportar construções de linguagens adicionais [Horrocks, 1997; Baader et al., 2003].

O Pellet suporta completamente a expressividade do OWL DL incluindo raciocínio sobre *nominals* (classes enumeradas). Com isto, construções OWL do tipo owl:onOg e owl:hasValue podem ser utilizadas livremente. A API do Pellet provê funcionalidades para verificação da validade de espécies, checagem de consistência de ontologias, classificação de taxonomia, checar inferências e responder um con-

junto e consultas RDQL (conhecidas como consultas ABox na terminologia DL). Por questões de conveniência o Pellet tem um servidor DIG para ser utilizado com aplicações DIG e que é importante para nós, uma vez ele foi utilizado como um raciocinador externo ao Protégé.

A Camada de Aplicação

E finalmente a Camada de Aplicação é a que fica no topo da arquitetura, a qual provê o uso dos recursos do ambiente de grade. A interface gráfica do protótipo permite que o requisitante de recursos submeta uma consulta e obtenha o resultado semântico de sua requisição através do serviço de ontologia que utiliza o motor de inferência.

4.2 Aspectos da Implementação

Para validar a arquitetura da Grade Semântica para o serviço de descoberta de recursos da grade, foi implementado um protótipo utilizando a linguagem de programação Java. Um motivo importante por termos escolhido Java é que o GT4 também foi implementado em Java, com isto, facilita a integração de nossa aplicação com os serviços do GT4.

Em nossa implementação utilizamos a versão 4.0.3 do Globus Toolkit [Foster, 2006] em nosso ambiente de teste na Universidade de Brasília, que será posteriormente descrito, na tabela 5.1. A figura 4.3 representa os módulos principais implementados na Camada de Conhecimento, a qual está dividida em duas partes principais: o Serviço de Ontologia e o Serviço de Raciocínio.

Será descrito o *workflow* de execução do protótipo desenvolvido. Na parte do serviço de ontologia, foi primeiramente definido o *template* ontológico utilizando o Protégé, versão 3.1.1 (*build* 284), para definir algumas das classes do GLUE *Schema* versão 1.1 relacionados com o Ganglia versão 3.0.3. Neste ponto, o *template* da ontologia com os conceitos e propriedades relacionadas formam o Repositório Semântico na linguagem OWL (vide figura 4.2).

Após o *template* da ontologia ser definido, o serviço de descoberta monitora os recursos utilizando o Ganglia e salva dinamicamente os recursos reais do ambiente de grade relacionados a um determinado instante. Estes dados foram salvos no Repositório de Recursos utilizando-se a linguagem XML com a informação provida pelo serviço MDS4 do *middleware* GT4. Os testes realizados pelo protótipo implementado foram feitos em um ambiente de grade com vinte e duas máquinas localizadas em dois diferentes laboratórios de pesquisa, que estão ilustrados na

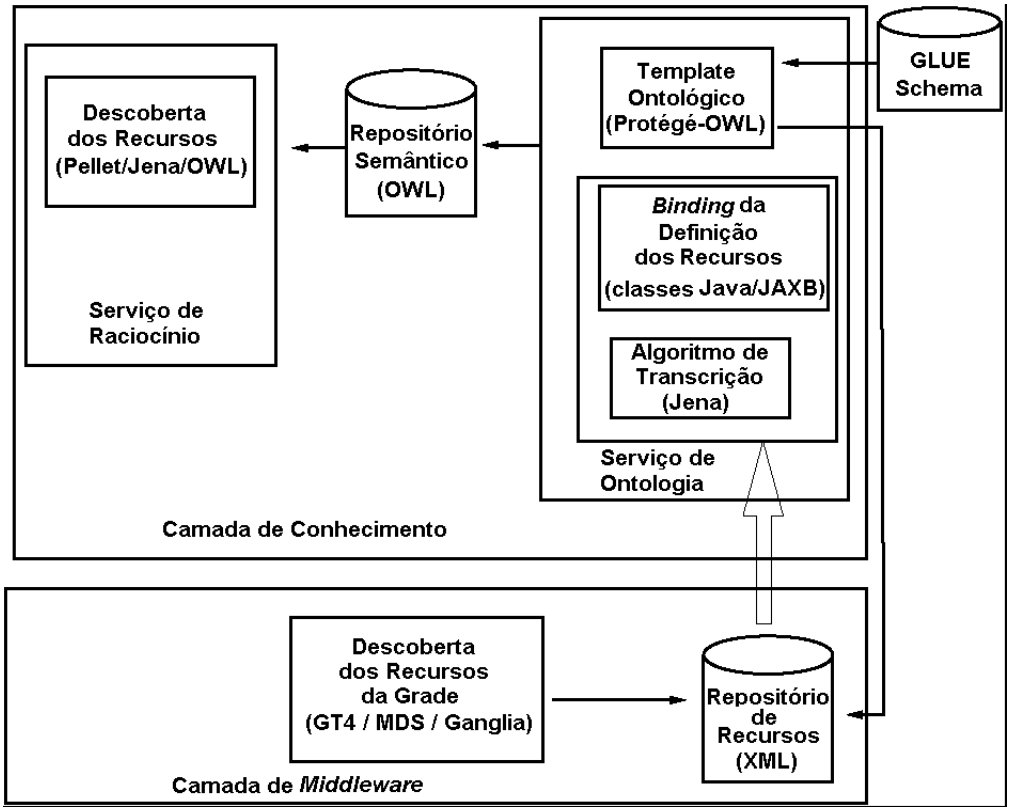

Figura 4.3: Ações Envolvidas nos dois Serviços Providos pela Arquitetura de Grade Semântica.

figura 5.1 e respectiva tabela 5.1. Observe o seguinte trecho de código, que mostra o nome do sistema operacional e a quantidade de memória RAM disponível do *host* atm-01.cic.unb.br que está armazenado no Repositório de Recursos, que foi instanciado pelo MDS4 utilizando o GLUE/Ganglia.

```
<ns1:Host ns1:Name="atm-01.cic.unb.br">
  <ns1:OperatingSystem
       ns1:Name="AIX"
       ns1:Release="5.1.0.65"/>
  <ns1:MainMemory
       ns1:RAMAvailable="888"
       ns1:RAMSize="1024"
       ns1:VirtualAvailable="1258"
       ns1:VirtualSize="2012"/>
  <!-- other machine resources -->
</ns1:Host>
```

De modo a instaciar o *template* de ontologia, nós primeiramente fazemos um *bind* dos recursos da grade definidos em XML para classes Java utilizando a

Java Architecture for XML Binding (JAXB), que provê um modo conveniente de se fazer o *bind* do *schema* XML para sua representação em código Java e está disponível no *Java Web Services Developer Pack* (Java WSDP) 1.5 [Sun Microsystems, 2006]. Em seguida, foi desenvolvido um algoritmo para leitura dos recursos do Repositório de Recursos em XML para daí então instanciar as classes Java, que permitem salvar no Repositório Semântico o *template* da ontologia instanciada no formato OWL utilizando o framework Jena versão 2.4. Veja abaixo um trecho do código OWL do Repositório Semântico para sistema operacional e memória principal do *host* atm-01.cic.unb.br:

```
<?xml version="1.0"?>
<Host rdf:ID="atm-01">
<Name
  rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    atm-01.cic.unb.br
</Name>
<hasOperatingSystem>
  <OperatingSystem rdf:ID="OS_aix">
    <OSRelease
      rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        5.1.0.65
    </OSRelease>
    <OSName
      rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        AIX
    </OSName>
  </OperatingSystem>
</hasOperatingSystem>
<hasMainMemory>
  <MainMemory rdf:ID="Mem_a01">
    <VirtualSize
      rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
        2012
    </VirtualSize>
    <RAMAvailable
      rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
        888
    </RAMAvailable>
```

```
    <VirtualAvailable
      rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
        1258
    </VirtualAvailable>
    <RAMSize
      rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
        1024
    </RAMSize>
  </MainMemory>
</hasMainMemory>
<!-- other machine resources -->
</Host>
```

No Serviço de Raciocínio foi utilizado o Repositório Semântico instanciado com o *framework* Jena, versão 2.4 e o raciocinador Pellet, versão 1.3, para a recuperação semântica da informação dos recursos da grade. De acordo com as consultas dos usuário o raciocinador provê o melhor resultado tanto direto quando semântico para o casamento de recursos. É mostrado a seguir a ilustração da regra do Pellet no formato RDQL que está incluída no Componente Raciocinador.

Segue uma ilustração de regra do Pellet para memória RAM disponível e sistema operacional:

```
SELECT ?host, ?name,
       ?memory,?ramAvailable,
       ?operatingSystem, ?operatingSystemName
WHERE (?host, gt4:Name, ?name)
      (?host, gt4:hasMainMemory, ?memory)
      (?memory, gt4:RAMAvailable, ?ramAvailable)
      (?host, gt4:hasOperatingSystem, ?operatingSystem)
      (?operatingSystem, gt4:OSName, ?operatingSystemName)
AND ?ramAvailable opRAM paramAvailableRAM
AND ?operatingSystemName opOS paramNameOS
USING gt4 for <http://www.cic.unb.br/josenelson.owl#>
}
```

Note que o *paramAvailableRAM* e *paramNameOS* são variáveis passadas ao Pellet pelo Java, de acordo com a solicitação do usuário. Note também que *opRAM* e *opOS* indicam os operadores a serem utilizados para realização de consultas flexíveis. Por exemplo, se o usuário deseja buscar por máquinas com memó-

ria RAM disponível maior que 128 MB, a consulta deve ser ?ramAvailable>=128. O protótipo implementado suporta os operadores =, >, <, >= e <= para consultas sobre disponibilidade de memória RAM, enquanto o recurso de nome do sistema operacional é suportado pelos operadores = e ≠. O mecanismo de consulta foi projetado para suportar tanto restrições simples quanto restrições múltiplas, por exemplo, se o usuário deseja busca por máquinas com memória RAM disponível de 256 MB e além disto por sistema operacional AIX então a consulta deverá ser ?ramAvailable=256 e ?operatingSystemName=AIX.

Para a Camada de Aplicação foi desenvolvida uma interface para validar a proposta. A tela de entrada/saída do serviço de descoberta de recursos é mostrado na figura 4.4. Note que foram desenvolvidas duas opções de busca: direta e semântica; utilizando os recursos de máquina: Sistema Operacional e Memória RAM Disponível (MB). A tela de entrada/saída apresentada mostra uma consulta da busca semântica com múltiplos recursos incluindo *?operatingSystemName=AIX* e *?ramAvailable=128*. A figura 4.5 exibe informações sobre o sistema e é acessada pelo menu da aplicação.

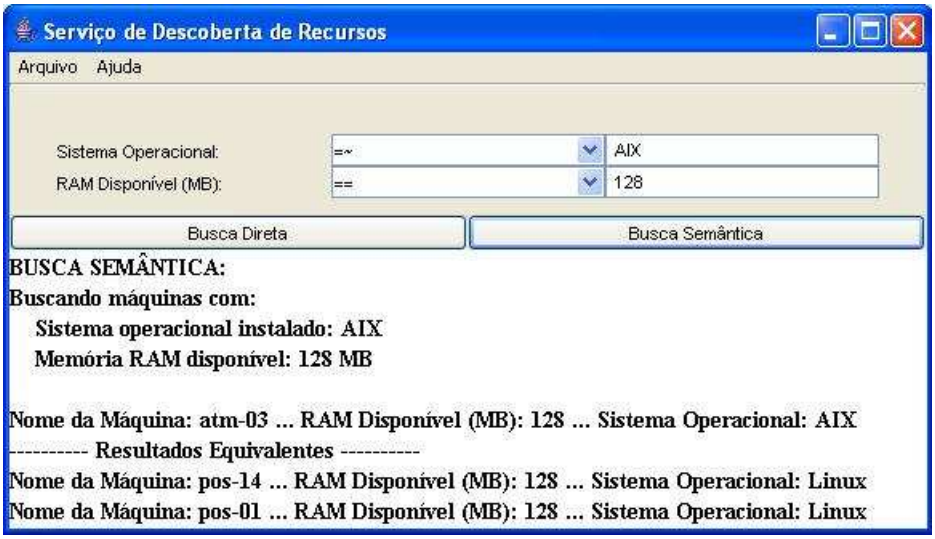

Figura 4.4: Tela de Entrada/Saída do Serviço de Descoberta de Recursos.

O resultado apresentado na figura 4.4 demonstra que o *host* atm-03 é o melhor *match* direto, o que semanticamente equivale aos hosts pos-14 e pos-01, que também têm 128 MB e são compatíveis com Unix. Note que o protótipo foi desenvolvido para ilustrar apenas dois diferentes recursos de máquinas, porém é possível estendê-lo para incluir outros conceitos do *template* ontológico para buscas de consultas simples e múltiplas. O *template* da ontologia pode também ser estendido para incluir uma variedade de conceitos de acordo com os propósitos de gerenciamento desejados.

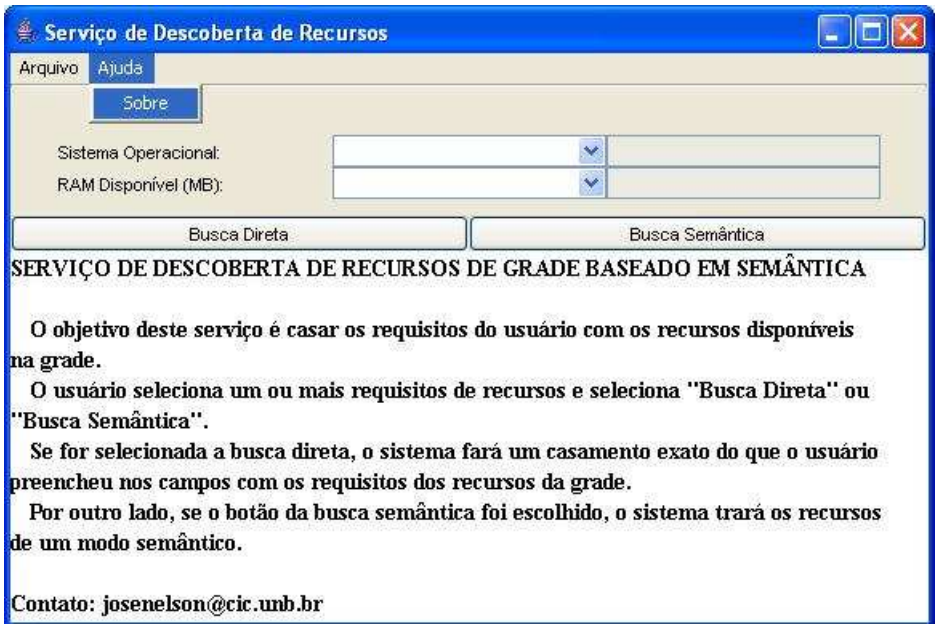

Figura 4.5: Tela com Informações sobre o Sistema.

4.3 Trabalhos Correlatos

Segundo Berman [2001], o Conhecimento em Grade, ou *Knowledge Grid*, trata de um mecanismo capaz de sintetizar conhecimento a partir de dados através da mineração e de métodos de referência, o qual permite que mecanismos de busca façam inferências, respondam questões e cheguem a conclusões partindo de uma massa de dados.

Dentre várias pesquisas recentes sobre *Knowledge Grid* destacam-se: o grupo de pesquisa da China [Zhuge, 2004], liderado pelo professor Hai Zhuge, da Academia Chinesa de Ciência, e o grupo de pesquisa da Itália [Talia, 2003], liderado pelos professores Domenico Talia, da Universidade de Calábria, e Mario Cannataro, da Universidade de Catanzaro.

Grupo de Pesquisa da China

Segundo Zhuge [2004], a Internet e a Web têm tido um importante papel no compartilhamento de informações. Os cientistas têm utilizado-as cada vez mais como ferramentas de apoio às suas pesquisas. Eles podem comunicar-se via e-mail ou *NetMeeting* e postar os dados experimentais de suas pesquisas em páginas pessoais na Web ou nos *sites* de suas instituições. Eles podem obter fontes para suas pesquisas de bibliotecas digitais on-line já conhecidas ou descobrir novas simplesmente buscando-as em ferramentas de busca tais como Google [Google Inc., 2006]. O conhecimento é a base a realização de serviços inteligentes. A

infra-estrutura do *knowledge grid* suporta a *e-science* através de um conjunto de serviços relevantes da aplicação e de recursos semânticos.

Ainda segundo Zhuge [2004], o número de páginas na Web aumenta consideravelmente a cada instante, o que dificulta que o compartilhamento de informações seja feito de forma eficiente e eficaz. Muito se tem feito para resolver este problema, mas pouco sucesso se tem conseguido. A dificuldade é que devido à forma como a Web foi definida, torna-se muito difícil usá-la de forma inteligente. Sabe-se que o conhecimento é a base da realização de serviços inteligentes. Para superar este defeito da Web, os cientistas estão trabalhando na Web da próxima geração, cujo foco da pesquisa está em duas abordagens: aprimorar a Web existente e criar uma nova plataforma de aplicação independente da Web. O *Knowledge Grid* visa pesquisar estas áreas, utilizando-as na construção de uma plataforma de aplicação inteligente eficiente e eficaz. Assim, o *Knowledge Grid* pode ser definido como um ambiente inteligente e sustentável de aplicação da Internet que possibilita que pessoas ou agentes inteligentes de software façam de forma eficaz a captura, publicação, compartilhamento e o controle dos recursos de conhecimento. Ele visa também prover serviços sob demanda de suporte à inovação tecnológica, ao trabalho cooperativo em equipe, à resolução de problemas e à tomada de decisão. Ele incorpora a epistemologia e a ontologia para refletir características humanas da cognição; explora princípios sociais, ecologicos e econômicos; e adota técnicas e padrões desenvolvidos durante o trabalho para a Web do futuro.

Grupo de Pesquisa da Itália

Segundo Talia [2003], o *Knowledge Discovery in Databases* (KDD) é um conjunto de técnicas e ferramentas de software que utiliza a mineração de dados para analisar uma grande quantidade de informação armazenada em repositórios de dados com o intuito de encontrar um padrão útil entre eles, de modo a se entender o que é importante e útil neste emaranhado de informações. Já o *Parallel and Distributed Knowledge Discovery* (PDKD) ou Descoberta de Conhecimento Paralelo e Distribuído é uma plataforma onde sistemas KDD manipulam e analisam múltiplos domínios e repositórios de dados de proprietários diferentes [Talia, 2003].

O *Knowledge Grid* é uma arquitetura de software proposta para sistemas PDKD distribuídos geograficamente. Esta arquitetura é montada no topo da grade computacional, que provê confiabilidade, consistência e acesso seguro aos recursos computacionais. O *Knowledge Grid* utiliza os recursos básicos da grade e define um conjunto de camadas adicionais para implementar serviços de desco-

berta de conhecimento distribuídos computadores conectados, onde cada nó pode ser uma máquina seqüencial ou paralela.

Sobre a arquitetura do projeto do grupo de pesquisa da Itália, tem-se na base os serviços básicos da grade. Foi utilizado o Globus Toolkit versão 2 (GT2), que é projetado para prover um conjunto de serviços básicos para construir grades computacionais. Observe na figura 4.6 esses serviços. Ainda sobre a arquitetura, têm-se acima dos serviços básicos da grade, duas principais camadas de software, representados na figura 4.7.

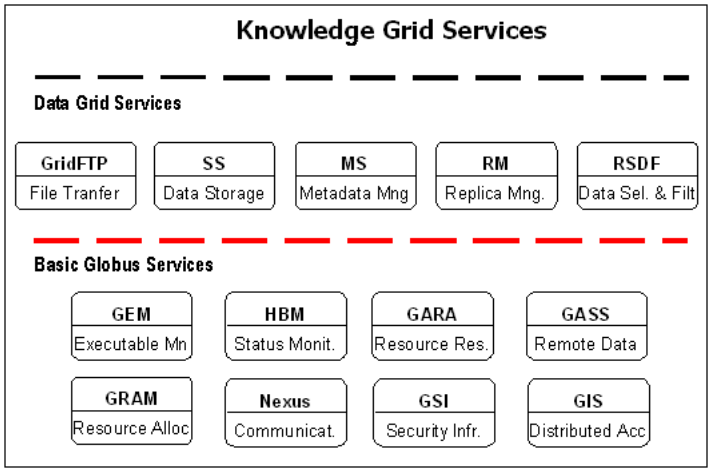

Figura 4.6: Os Serviços Genéricos do Globus e os Serviços de Dados da Grade [Talia, 2003].

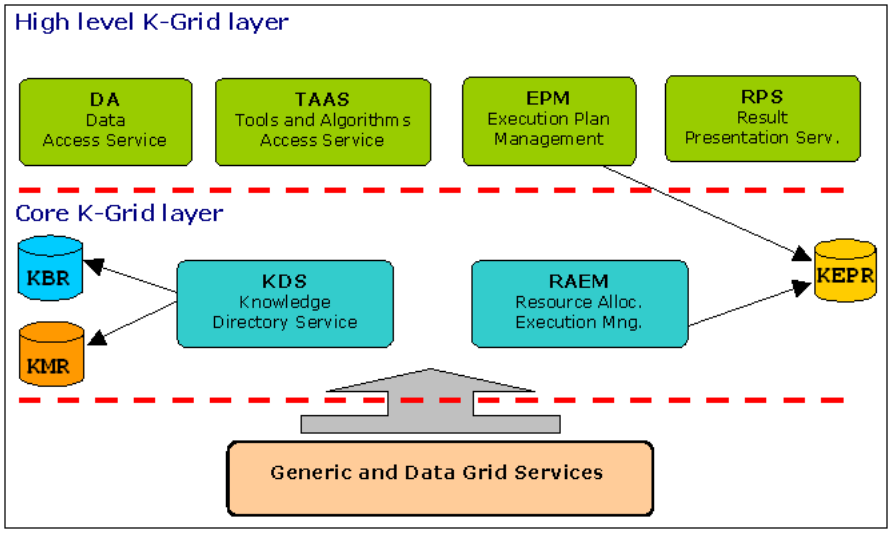

Figura 4.7: As Camadas da Arquitetura do *Knowledge Grid* [Talia, 2003].

- Camada *Core K-Grid*: refere-se aos serviços diretamente implementados no topo dos serviços da grade genérica. Responsável pela definição, composição e execução de computações PDKD sob a grade. Seus dois serviços são:
  - *Knowledge Directory Service* (KDS): estende do MDS do Globus (seção 2.4). Gerencia os metadados, descrevendo as características dos objetos relevantes das aplicações PDKD, tais como fontes de dados, softwares de mineração de dados, resultados de computações, ferramentas de manipulação de resultados e os planos de execução. A informação gerenciada pelo KDS é armazenada em três repositórios *ad hoc*:
    - *Knowledge Metadata Repository* (KMR): são armazenados os metadados descrevendo as características do dado, software e ferramentas.
    - *Knowledge Base Repository* (KBR): são armazenadas as informação sobre o conhecimento descoberto após a computação PDKD.
    - *Knowledge Execution Plan Repository* (KERP): armazena os planos de execução descrevendo as aplicações sobre a grade.
  - *Resource Allocation and Execution Management Service* (RAEMS): o objetivo dos serviços RAEM é encontrar um mapeamento entre o plano de execução e os recursos disponíveis na grade, tais que satisfaçam tanto ao usuário, quanto aos dados e aos algoritmos requeridos e suas respectivas restrições. É baseado no GRAM do Globus (seção 2.4).
- Camada *High-level K-Grid*: utilizada para descrever, desenvolver e executar computações PDKD sobre a *Knowledge Grid*.
  - *Data Access Services* (DAS): são utilizados para a busca, seleção, extração, transformação e entrega de dados a serem minerados. São baseados nos serviços GASS do Globus (seção 2.4) e utilizam o KDS.
  - *Tools and Algorithms Access Service* (TAAS): responsáveis pela busca, seleção e *download* das ferramentas e algoritmos de mineração de dados.
  - *Execution Plan Management Service* (EPMS): é uma ferramenta semi-automática que obtém os dados e os programas selecionados pelo usuário e gera um conjunto de diferentes possíveis planos de execução. Os planos de execução são armazenados no KEPR para permitir a implementação de processos iterativos de descoberta de conhecimento, por

exemplo, análise periódica das mesmas fontes de dados que variam ao longo do tempo.

– *Results Presentation Service* (RPS): especifica como gerar, apresentar e visualizar os resultados PDKD (regras, associações, modelos, classificações, etc.), e oferece métodos para armazenar estes resultados em diferentes formatos no KBR.

VEGA

O *Visual Environment for Grid Applications* (VEGA) é um protótipo da arquitetura *Knowledge Grid* que foi implementado utilizando a linguagem de programação Java e o Globus *Toolkit* 2 com o objetivo de permitir que um usuário construa uma aplicação de mineração de dados baseada em grade. Em [Talia, 2003] foi citada a implementação dos serviços e estruturas principais do KDS e do DAS.

O VEGA oferece ao usuário suporte a:

- Composição de tarefas: definição das entidades envolvidas e as relações entre elas.
- Checagem da consistência das tarefas planejadas.
- Geração de planos de execução para uma tarefa de mineração de dados.
- Execução de plano de execução através de um gerente de alocação de recursos.

Observando a figura 4.8 podemos notar que no VEGA, a representação dos planos de execução é modelada como grafos onde os nós representam os recursos computacionais (fontes de dados, softwares, resultados, etc) e os arcos representam as relações entre os recursos (movimentação dos dados, filtragem dos dados, execução dos programas, transferência de arquivos, etc). Uma aplicação de *Knowledge Grid* pode ser composta por vários *Workspaces*. Planeja-se futuramente considerar diferentes parâmetros da rede, tais como topologia, largura de banda e latência para a otimização da execução de programas de descoberta de conhecimento paralelo e distribuído.

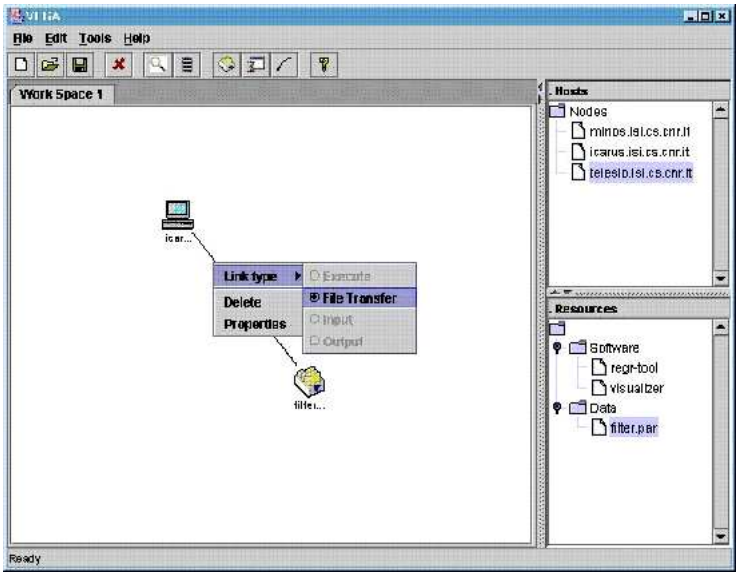

Figura 4.8: Tela do VEGA [Talia, 2003].

Proteus

O Proteus é um ambiente de resolução de problemas baseado em grade para compor, compilar e executar aplicações em bioinformática desenvolvido pelo grupo de pesquisa da Itália [Cannataro et al., 2004]. As aplicações em bioinformática, em particular a *Proteomics* executa diferentes algoritmos de diferentes bases de dados, que podem ter sido gerados em um experimento ou estão disponíveis em bases de dados públicas. Estas aplicações necessitam diversas modelagens semânticas de seus componentes básicos e requer um grande poder computacional.

Segundo os autores, utilizando o Proteus o pesquisador da área de bioinformática pode facilmente: formular um problema para resolver um dado problema, por exemplo, o seqüenciamento de DNA; executar a aplicação no ambiente de grade utilizando os recursos disponíveis; e finalmente visualizar e analisar os resultados obtidos. A arquitetura do Proteus é composta por quatro componentes principais:

- Repositórios de Metadados: contém informações sobre os recursos instalados em cada máquina.
- Ontologias: subdivididas em dois tipos.
  - Ontologias do Domínio: descreve e classifica os conceitos e usos do domínio, no caso o da bioinformática, bem como as ferramentas de software (*European Molecular Biology Open Software Suite* [Rice et al., 2000] ou EMBOSS) e as fontes de dados (Swiss-Prot [Swiss Institute of Bioinformatics, 2006]).

  - Ontologias da Aplicação: descreve e classifica as aplicações de bioinformática e é representado como fluxo de dados.

- Projetista de aplicação baseado em ontologia: que irá auxiliar o pesquisador sugerindo ao pesquisador quais máquinas têm as ferramentas instaladas e as que contém os bancos de dados.

- Gerente de execução em grade baseado em fluxo de dados: são representações gráficas das aplicações que representam os *scripts* que foram submetidos à grade.

O Proteus utiliza uma metodologia baseada em ontologia para modelar a semântica das aplicações de bioinformática. Na figura 4.9 observa-se o *Ontology Repository*, que é a grande diferença entre o Proteus em relação ao *Knowledge Grid*. Neste repositório descreve os recursos de bioinformática. Existe também o *Ontology Management Service* (OMS), que busca os componentes de software mais apropriados para solucionar um determinado problema de bioinformática.

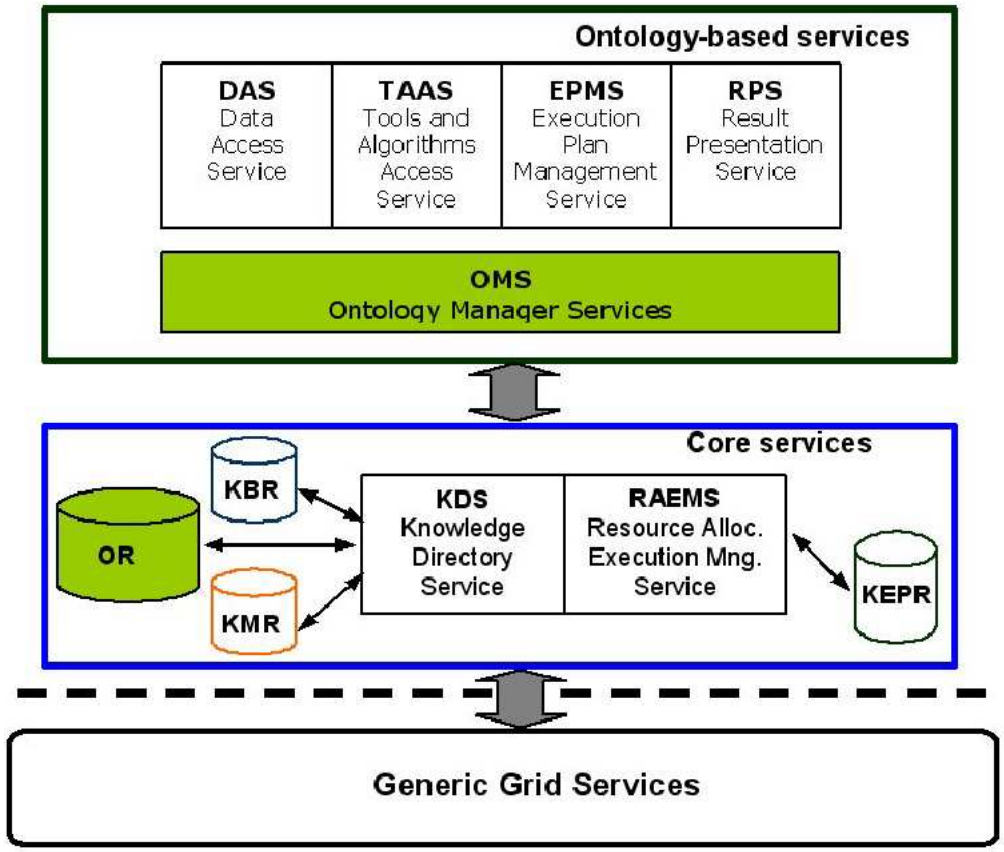

Figura 4.9: Módulos de Software do Proteus [Cannataro et al., 2004].

Na figura 4.10 observa-se uma ferramenta gráfica que foi implementada para navegar nas ontologias. Esta ferramenta lê um arquivo .daml, que é gerado pela linguagem geradora de ontologia DAML+OIL [DAML+OIL, 2001].

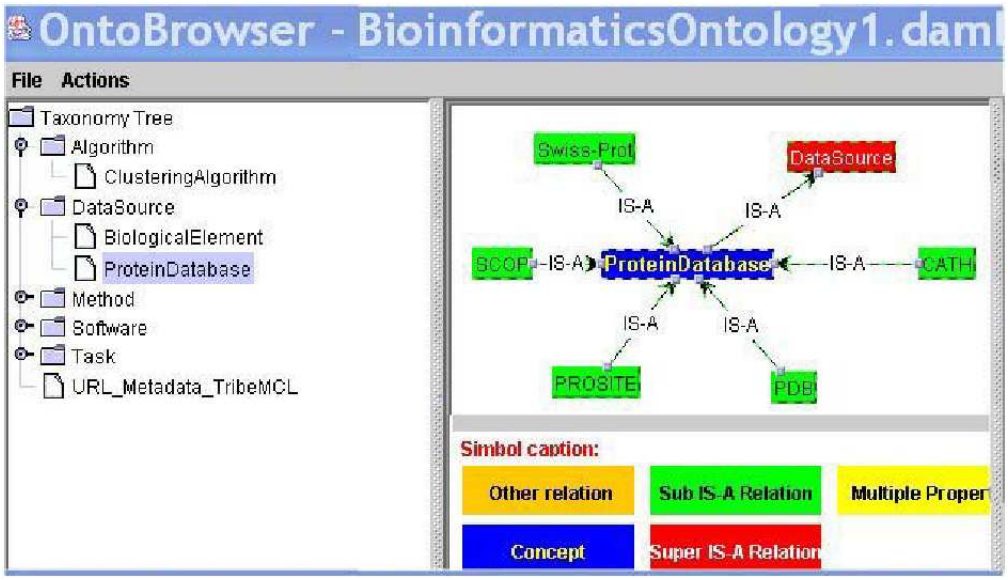

Figura 4.10: O *Browser* de Ontologia do Proteus [Cannataro et al., 2004].

Outros Trabalhos Correlatos

Em [Geldof, 2004] é citada a diferença entre os conceitos de Grade Semântica, *Knowledge Grid* e *Cognitive Grid*. Ao invés de se discutir em detalhes a diferença, é mais útil se falar sobre suas similaridades. Em suma, todos estes termos referem-se à adição de inteligência à grade. O *Knowledge Grid* lida apenas com o compartilhamento de conhecimento, enquanto a Grade Semântica lida com compartilhamento de recursos em geral, não apenas conhecimento, onde um significado bem definido é adicionado aos recursos (como exemplo, podemos citar o uso de ontologias). E por fim o *Cognitive Grid* é as vezes utilizado sem distinção com o termo Grade Semântica e é entendido com o gerenciamento inteligente dos recursos da grade. Ele pode ser definido como uma grade que incorpora Serviços de Grade, ontologias e serviços voltados ao tratamento do conhecimento. A base para o entendimento de todos estes três termos é a Web Semântica. O foco deste trabalho é a Grade Semântica, inclui também conceitos de Cognitive Grid, uma vez que utiliza mecanismos de raciocínio na descoberta de recursos semânticos na grade e também o conceito de Knowledge Grid, uma vez que permite o compartilhamento de conhecimento dos recursos da grade computacional.

Devido a grande demanda por computação de alto desempenho para aplicações científicas, a computação em grade está se tornando um padrão de infra-estrutura que permite o compartilhamento de recursos e a resolução coordenada de proble-

mas em organizações virtuais dinâmicas de múltiplas instituições [Foster et al., 2001]. A necessidade de comunidades científicas para interconexão de aplicações, dados, *expertise* e recursos computacionais é compartilhada por outras áreas de aplicação, tais como negócios, governo, medicina e educação [Foster et al., 2002].

A Grade Semântica é uma iniciativa recente para sistematicamente expor informações ricas semanticamente dos recursos da grade para construir serviços de grade mais inteligentes. Na Grade Semântica os computadores e as pessoas têm que estar aptos a trabalhar em cooperação, lidando com as informações e serviços relacionados com a grade através de um significado bem definido [Roure et al., 2005].

A literatura trata do desafio da Grade Semântica através de diferentes abordagens. Na prática, trabalhar com grade semântica tem como significado principal a introdução de tecnologias da Web Semântica para a grade. O *background* de conhecimento e vocabulário de um domínio pode ser capturado em ontologias, que são modelos de conceitos, seus inter-relacionamentos e suas restrições, que podem ser processados pelas máquinas. Existe um número crescente de projetos recentes que concentram seus esforços em ontologia relacionada à computação em grade e que podemos citar e comparar com nosso trabalho [Tangmunarunkit et al., 2003; Harth et al., 2004; Brooke et al., 2004; Xing et al., 2006; Alper et al., 2006; Somasundaram et al., 2006].

O trabalho de Tangmunarunkit et al. [2003] define um projeto e o protótipo de um selecionador de recursos baseado em ontologia que explora ontologias, o conhecimento em *backgrounds* e as regras para a resolução do casamento de recursos na grade. Os autores utilizam o editor Protégé para criar manualmente as instâncias da ontologia que são então salvas em formato RDF. Eles desenvolveram um protótipo de *matchmaker* para o suporte a serviços de *matchmaking* que não estão incluídos no serviço de *brokering*, pelo menos não neste artigo. Eles utilizaram o sistema de banco de dados dedutivo TRIPLE/XSB [Sintek et al., 2006] para processar as consultas através das regras de *matchmaking*, em combinação com o *background* de conhecimento e ontologias.

Em [Harth et al., 2004] os autores provém acesso dinâmico à capacidade de *matchmaking* construindo um serviço de *matchmaking on-line*. A implementação deles utiliza o Globus Toolkit para o desenvolvimento da grade de serviços e explora o serviço de descoberta e monitoramento na infra-estrutura da grade para a descoberta e atualização dinâmica das informações dos recursos. Eles descrevem a arquitetura do Serviço de *Matchmaker* de Recursos baseado em Ontologia ou *Ontology-based Resource Matchmaker Service* (OMMS).

Em [Brooke et al., 2004] os autores consideram o problema da descrição de recursos no contexto de um *broker* de recursos sendo desenvolvido no Projeto de Inter-Operabilidade da Grade ou *Grid Interoperability Project* (GRIP), que está apto a fazer o *broker* de recursos descrito por diferentes sistemas de *middleware* de grade, tais como GT2, GT3 e Unicore. Os autores consideram necessariamente utilizar o casamento semântico das descrições dos recursos, primeiramente porque existe não atualmente um padrão comum para isto e porque eles desejam criar uma grade transparente no nível da aplicação. Os autores mostram como a abordagem semântica para a descrição de recursos facilita estas questões e apresentam o *broker* do GRIP como um protótipo que trabalha nesta abordagem.

O trabalho recente de Xing et al. [2006] apresenta a Ontologia Core da Grade ou *Core Grid Ontology* (CGO) para prover uma base comum para a representação do conhecimento e de sistemas de grade, incluindo os conceitos básicos e os relacionamentos das entidade da grade e os recursos da grade, de acordo com um modelo de grade abstrata proposto. O CGO foi descrito utilizando a linguagem OWL, que suporta a realização da Grade Semântica, incluindo, além dos recursos da grade, também o *middleware*, os serviços, aplicações e usuários da grade. A ontologia CGO proposta pode ser utilizada para a integração das informações da grade, busca de informações, descoberta de recursos e gerenciamento da alocação de recursos.

A grade semântica demanda de uma arquitetura ou algum tipo de *framework* sistemático para projetar os componentes e as aplicações da Grade Semântica [Goble, 2005]. Alguns trabalhos recentes estão sendo feitos nesta direção. Em [Alper et al., 2006] os autores descrevem a OGSA-Semântica ou *Semantic-OGSA* (S-OGSA), que é um dos resultados mais recentes do projeto Ontogrid [OntoGrid Consortium, 2006] do EU-IST. O S-OGSA é proposto como uma arquitetura de referência para o projeto e está sendo criada com o objetivo de se tornar um framework de referência para a Grade Semântica [Goble et al., 2006]. Em [Gómez-Pérez et al., 2006] nós encontramos uma aplicação da arquitetura definido no projeto Ontogrid (S-OGSA) em um cenário para a análise da qualidade de produtos de missões de satélite.

O trabalho que consideramos estar mais relacionado com nossa proposta é o encontrado em [Somasundaram et al., 2006]. Os autores propõem uma arquitetura de Grade Semântica introduzindo uma camada de conhecimento no topo da arquitetura do *broker* Gridbus [Grid Computing and Distributed Systems Laboratory, 2006] possibilitando assim a descoberta de recursos semanticamente através do *broker*. O componente semântico na camada de conhecimento permite a descrição

semântica dos recursos da grade com a ajuda de uma *template* de ontologia. A diferença é que eles propõem uma arquitetura de cinco camadas que implementa uma camada de conhecimento desenvolvida para suportar escalonamento tanto computacional quanto de dados das aplicações de grade [Venugopal et al., 2004]. Resumindo, a proposta deles concentra-se não apenas na descoberta semântica de recursos, mas também no suporte a escalonamento tanto computacional quanto de dados das aplicações de grade.

Nós projetamos e prototipamos uma arquitetura de grade semântica baseada em um *template* de ontologia e regras para o casamento inteligente de recursos em uma grade. Diferentemente de Tangmunarunkit et al. [2003], nós não incluímos um serviço de *brokering*, uma vez que este não é o foco principal de nosso trabalho.

A ontologia CGO de Xing et al. [2006] provê uma base comum para representação tanto do conhecimento quanto dos sistemas da grade podendo ser utilizado para a integração das informações da grade e busca, descoberta de recursos e gerenciamento de alocação. Diferentemente deste trabalho, nós não focamos na definição da ontologia, uma vez que ela pode ser estendida posteriormente, mas nós trabalhamos com um componente de raciocínio que permite consultar conhecimento semântico relacionados com ambientes de multi grades. Em relação ao *framework* arquitetural da grade semântica, nossa meta não é propor uma arquitetura de referência, tal como Alper et al. [2006] propõe para o S-OGSA, mas projetar uma arquitetura para lidar com um serviço de casamento para os recursos descobertos semanticamente. O protótipo implementado utiliza o Globus *Toolkit* [Foster and Kesselman, 1997] para o desenvolvimento do serviço de grade e explora o serviço de descoberta e monitoramento da infra-estrutura de grade para dinamicamente descobrir e atualizar as informações sobre os recursos, semelhante ao OMMS de Harth et al. [2004].

A tabela 4.1 apresenta um comparativo entre os trabalhos correlatos estudados e esta dissertação de mestrado. A primeira linha refere-se ao *middleware* de grade utilizado. A segunda linha faz referência ao ano de publicação do trabalho. A terceira linha informa se foi utilizado mecanismo de inferência e qual foi. Já a quarta linha mostra qual foi a linguagem de descrição da ontologia utilizada. A quinta linha exibe se o trabalho trata de requisições múltiplas de recursos. E finalmente a sexta linha informa qual foi a ferramenta utilizada como editor da ontologia e de instanciação dos recursos.

É importante observar o objetivo de cada um dos trabalhos apresentados. O trabalho de Tangmunarunkit é um selecionador de recursos baseado em ontologia. Já o trabalho de Harth tem como objetivo fazer um *matchmaking* dinâmico dos

recursos. O trabalho de Brooke tem como finalidade o casamento semântico das descrições dos recursos. O trabalho de Xing trata da ontologia da grade, não apenas dos recursos da grade, mas também o *middleware*, os serviços, aplicações e usuários da grade. Já o trabalho de Alper tem como objetivo a descrição da arquitetura Semantic-OGSA (S-OGSA), de modo a se tornar um *framework* de referência para a Grade Semântica. O trabalho de Somasundaram visa possibilitar a descoberta de recursos semanticamente através do *broker*, que dá suporte ao escalonamento computacional e dos dados das aplicações da grade. Finalmente, o objetivo de nosso trabalho é fazer o casamento entre os recursos computacionais disponíveis na grade e os requisitos da aplicação a ser executada. Este casamento pode ser de forma direta ou semântica, incluindo a descrição simples (de apenas uma característica dos recursos) ou múltipla (várias características dos recursos).

Tabela 4.1: Comparação dos Trabalhos Correlatos.

	Tangmunarunkit	Harth	Brooke	Xing	Alper	Somasundaram	Nosso trabalho
Middleware	OGSA	Globus Toolkit	GT2, GT3 e Unicore	Globus, Unicore, DataGrid, Crossgrid e EGEE	EGEE, OMII e Globus	Globus Toolkit 4	Globus Toolkit 4
Ano de publicação	2003	2004	2004	2006	2006	2006	2006
Mecanismo de inferência	TRIPLE / XSB	TRIPLE / XSB	Não	Não, trabalho futuro	Permite	Algernon	Pellet
Linguagem de descrição da ontologia	RDF	RDF	XML	OWL	OWL	OWL	OWL
Múltiplos Recursos	Sim	Sim	Não	Sim	Não	Não	Sim
Editor de ontologia e instanciação dos recursos	Protégé	Não	PCPack	Protégé	Não	Protégé	Protegé e Ganglia / MDS4

Capítulo 5

Estudo de Caso

Neste trabalho foi proposta uma abordagem flexível e extensível para a realização da seleção de recursos da grade utilizando-se um *matchmaker* baseado em ontologia.

Diferentemente de serviços de descoberta de recursos de grade tradicionais, que descrevem propriedades de recursos e requisições baseados em atributos simétricos (que são muito difícil de se gerenciar quando seu número de atributos cresce), ontologias separadas (por exemplo, descrições semânticas de modelos do domínio) são criadas para descrever declarativamente as requisições dos recursos utilizando uma linguagem de ontologia expressiva. Ao invés do casamento exato de sintaxe apenas, nosso *matchmaker* baseado em ontologia realiza o casamento semântico utilizando os termos definidos nestas ontologias.

Para se avaliar a proposta do Serviço de Descoberta de Recursos de Grade baseado em Semântica foram realizadas algumas consultas, tendo como parâmetros as características do recursos computacionais. Como estudo de caso deste trabalho foram executadas algumas consultas para as seguintes características de recursos: nome do sistema operacional e quantidade de memória RAM disponível. No experimento foram incluídas consultas simples (ou seja, apenas uma característica do recurso) e consultas múltiplas (ou seja, mais de uma característica do recurso). Para estimar o desempenho, considerou-se acerto (caso fosse encontrado algum recurso com essa característica e considerando a existência de recursos similares) ou falha (caso não fosse encontrado) durante a busca por recursos particulares. O desempenho das consultas de descoberta foi avaliado comparando os resultados obtidos por várias consultas, como mostrado na figura 5.8.

5.1 Ambiente de Teste

Como ambiente de teste foram utilizados dois laboratórios de informática do Departamento de Ciência da Computação da Universidade de Brasília: o LABPOS (Laboratório da Pós-Graduação) e o LAICO (LAboratório de sistemas Integrados e COncorrentes). A figura 5.1 ilustra o ambiente de teste, enquanto a tabela 5.1 apresenta o nome dos sistemas operacionais instalados e a quantidade de memória RAM disponível nas máquinas. Estas informações foram obtidas automaticamente pelo Ganglia em um determinado instante.

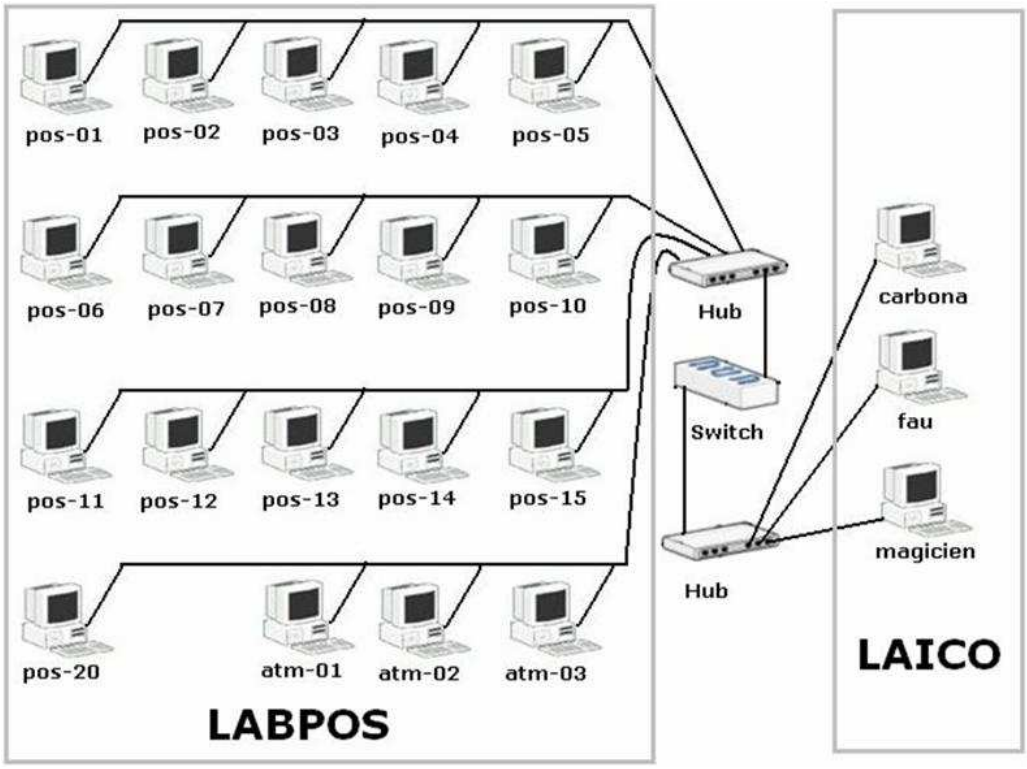

Figura 5.1: Ambiente de Teste.

Instalação do Ambiente

Para que o ambiente de grade pudesse funcionar, foram necessárias as instalações e configurações de algumas ferramentas nas máquinas dos laboratórios de pesquisa. A descrição detalhada destas ferramentas pode ser encontrada no apêndice B. Resumidamente, foi feita a instalação do Globus Toolkit versão 4.0.3 e foram criados certificados de segurança para cada uma das máquinas. Depois instalou-se o Sistema Gerenciador de Banco de Dados (SGBD) PostgreSQL versão 8.1.4, pois o GT4 precisa que ele esteja em execução para executar algumas tarefas. Posteriormente foi instalado o *container* Web Tomcat versão 5.0.28, que

Tabela 5.1: Ambiente de Grade Experimental.

Host	**O.S.**	**RAM Disponível**
pos-01.cic.unb.br	*Linux*	*128*
pos-02.cic.unb.br	Windows	332
pos-03.cic.unb.br	Linux	123
pos-04.cic.unb.br	Windows	333
pos-05.cic.unb.br	Linux	12
pos-06.cic.unb.br	Linux	210
pos-07.cic.unb.br	Linux	87
pos-08.cic.unb.br	Windows	123
pos-09.cic.unb.br	Windows	128
pos-10.cic.unb.br	Linux	7
pos-11.cic.unb.br	Windows	8
pos-12.cic.unb.br	Linux	66
pos-13.cic.unb.br	Windows	44
pos-14.cic.unb.br	*Linux*	*128*
pos-15.cic.unb.br	Linux	122
pos-20.cic.unb.br	Windows	12
atm-01.cic.unb.br	AIX	888
atm-02.cic.unb.br	AIX	243
atm-03.cic.unb.br	*AIX*	*128*
carbona.laico.cic.unb.br	Solaris	568
fau.laico.cic.unb.br	FreeBSD	354
magicien.laico.cic.unb.br	IRIX	564

é responsável por fornecer via Web as informações coletadas sobre os recursos das máquinas, através da interface gráfica conhecida por WebMDS. E finalmente instalou-se e configurou-se o Ganglia versão 3.0.3, que é quem de fato coleta as informações sobre os recursos das máquinas da grade.

5.2 Seleção Direta e Semântica

As consultas foram realizadas no Repositório Semântico especificando-se propriedades simples ou múltiplas de recursos com seleção direta e semântica. Por propriedades simples de recursos entende-se que o usuário tem necessidade por apenas um dos recursos de máquina, por exemplo, sistema operacional Linux (considerando que o usuário desejará executar um algoritmo que só só pode ser executado no Linux, ou sistema operacional compatível). Por outro lado, propriedades múltiplas de recursos referem-se à necessidade de mais de uma característica do recurso, por exemplo, deseja-se executar um algoritmo que demande o

sistema operacional Windows e que tenha no mínimo 512MB de memória RAM disponível.

Definiu-se *seleção direta* o tipo de consulta feita no Repositório de Recursos que retorna apenas quando o casamento é exato, ou seja, se o usuário precisa de um sistema operacional AIX, mas ele digita erroneamente aIX, o sistema não retorna nada. Quando o usuário busca por recursos que são desconhecidos do Repositório de Recursos, uma mensagem de erro é exibida (isto é considerado uma falha e é mostrado no gráfico 5.6 como sendo zero).

Define-se *seleção semântica* quando a consulta ao Repositório de Recursos é feita de um modo relativamente inteligente. No caso do usuário fazer uma busca semântica são exibidos os recursos semanticamente semelhantes, ou seja, os mais parecidos com o que foi solicitado pelo usuário. Para isto, é feita uma verificação na ontologia com o intuito de descobrir as máquinas compatíveis com a requisição.

Note na figura 5.2 que a busca direta com múltiplos recursos de máquina como OS:Linux e RAM:=128 retorna duas máquinas (pos-01 e pos-14) de acordo com a tabela 5.1.

Figura 5.2: Busca Direta: S.O.=Linux e RAM disponível=128.

Já a busca semântica, figura 5.3, retorna três máquinas (pos-01, pos-14 e atm-03), visto que atm-03 é compatível com Unix (AIX).

Considerando-se apenas um recurso, utilizando a busca direta, como OS:AIX, o sistema retorna três máquinas (atm-01, atm-02 e atm-03), conforme a figura 5.4.

Repetindo esta mesma busca, mas de forma semântica, obtém-se treze máquinas, figura 5.5, visto que nove são Linux, três são AIX e uma é IRIX, todas compatíveis com Unix.

Note que na tabela 5.1 existem duas outras máquinas que são compatíveis com Linux: carbona.laico.cic.unb.br com Solaris e fau.laico.cic.unb.br com FreeBSD,

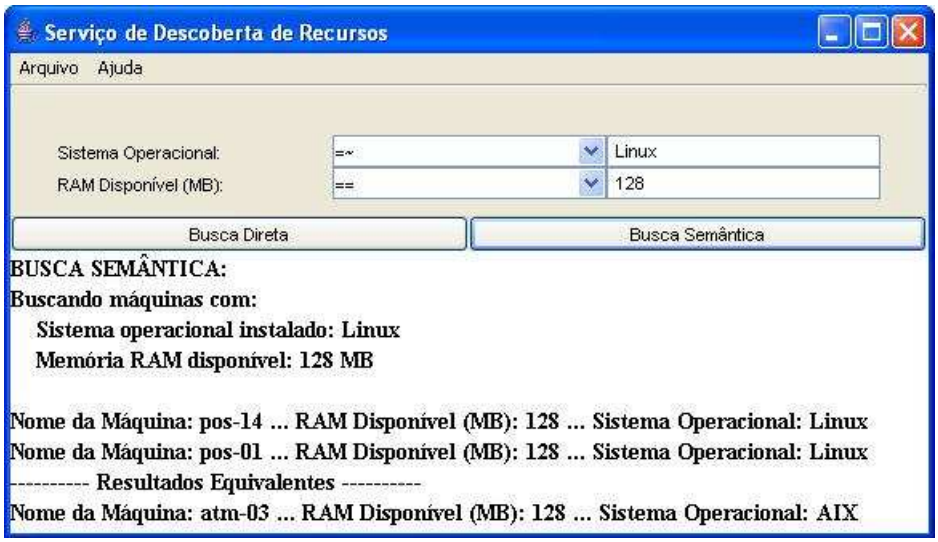

Figura 5.3: Busca Semântica: S.O.=Linux e RAM disponível=128.

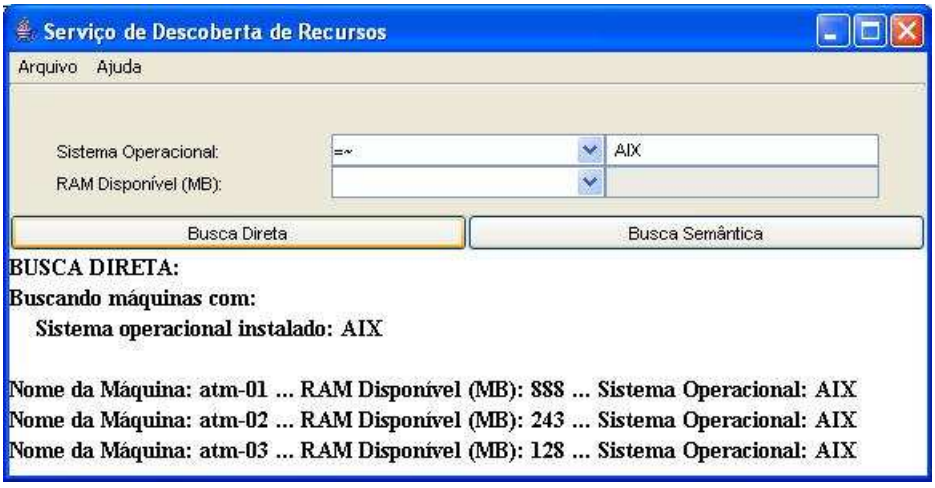

Figura 5.4: Busca Direta: S.O.=AIX.

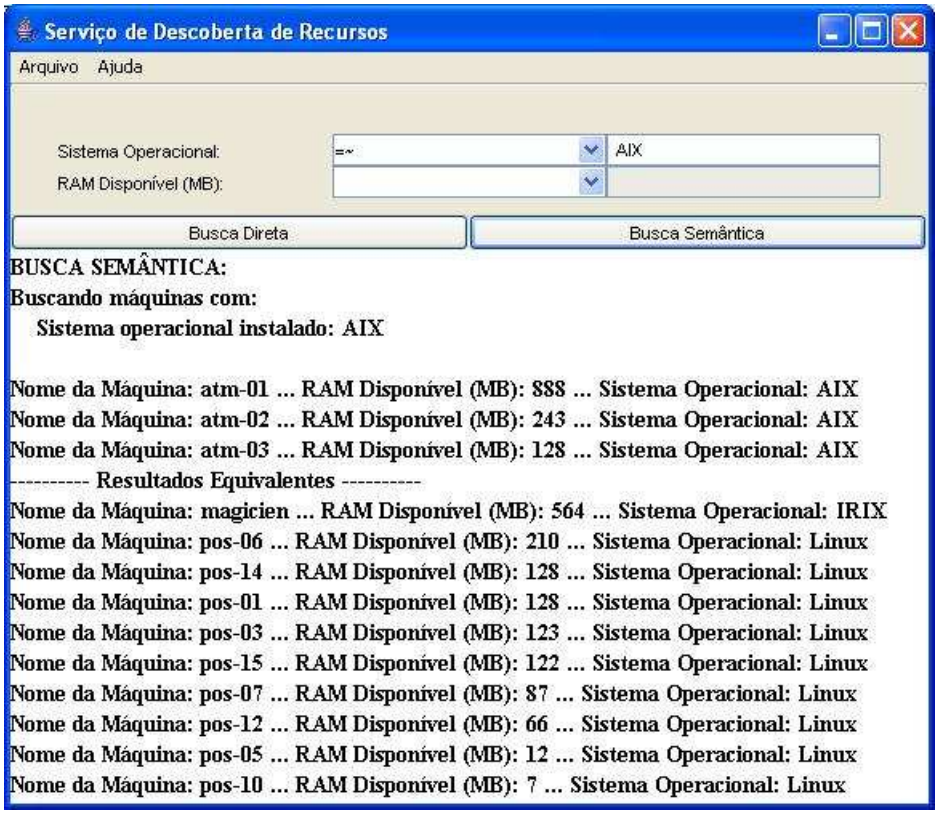

Figura 5.5: Busca Semântica: S.O.=AIX.

mas elas ainda não foram incluídas no gráfico da figura 5.8, uma vez que Solaris e FreeBSD não estão incluídos no *template* da ontologia utilizada na figura 4.2.

Finalmente, considerando a busca direta como OS:aIX que não tem resultados na busca direta, mas como o protótipo trata a não sensibilidade de caixa, as mesmas treze máquinas são retornadas com a busca semântica (vide figura 5.7).

5.3 Análise dos Resultados Obtidos

Analisando os resultados obtidos na busca direta podemos observar que a taxa de acerto é bem baixa, chegando a haver um caso de falha, quando o usuário solicita um sistema operacional cujo nome aIX foi escrito incorretamente, veja a ilustração da figura 5.6. Note que a busca direta retorna estritamente apenas aquilo que foi digitado pelo usuário.

Já na busca semântica a taxa de acerto é bem acentuada e para os testes realizados não houve caso de falha. É interessante observar que o número de máquinas encontrado cujo nome do sistema operacional = AIX é o mesmo de quando o nome é aIX, veja figura 5.7. Isto se deve ao fato do protótipo implementado realizar um tratamento de não sensibilidade de caixa, o que ignora letras

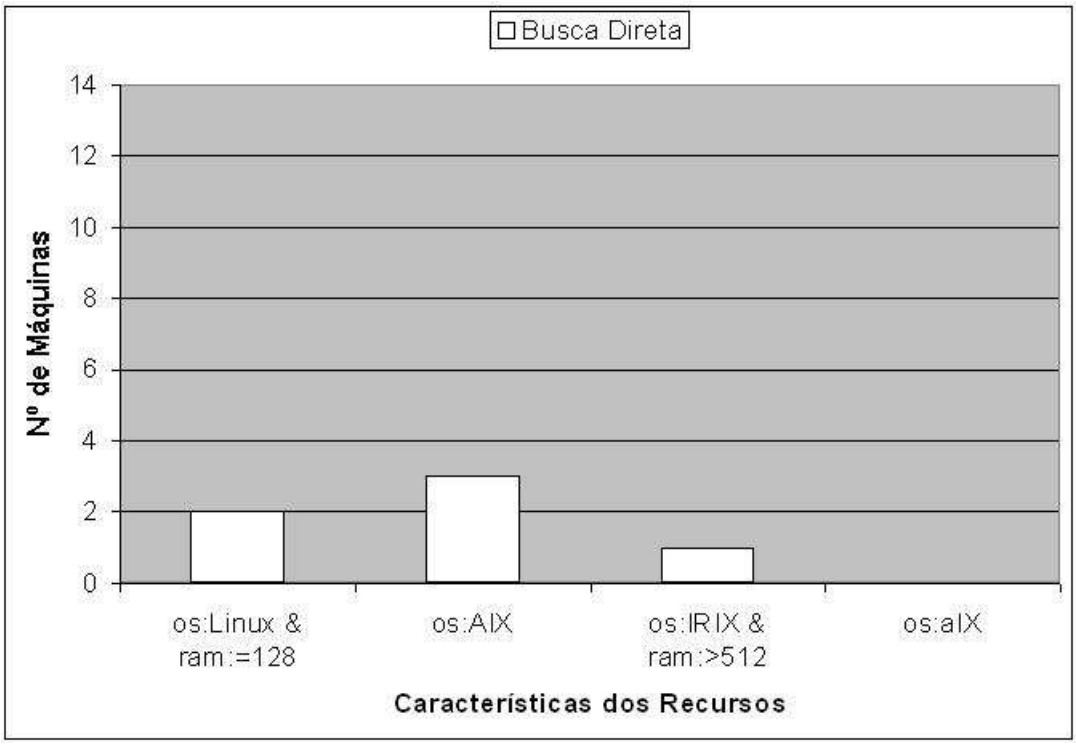

Figura 5.6: Resultados das Busca Direta.

maiúsculas e minúsculas.

Assim podemos comprovar a eficiência do tratamento semântico, uma vez que não só a quantidade de recursos retornada é maior, mas também a qualidade da consulta, já que as máquinas consideradas semanticamente equivalentes também estão aptas para utilização pelo usuário, pois tem as mesmas características semânticas da máquina que foi requerida. Observe na figura 5.8 um comparativo entre o resultado da busca direta e semântica. De acordo com os testes realizados ficou comprovada a eficiência do tratamento semântico.

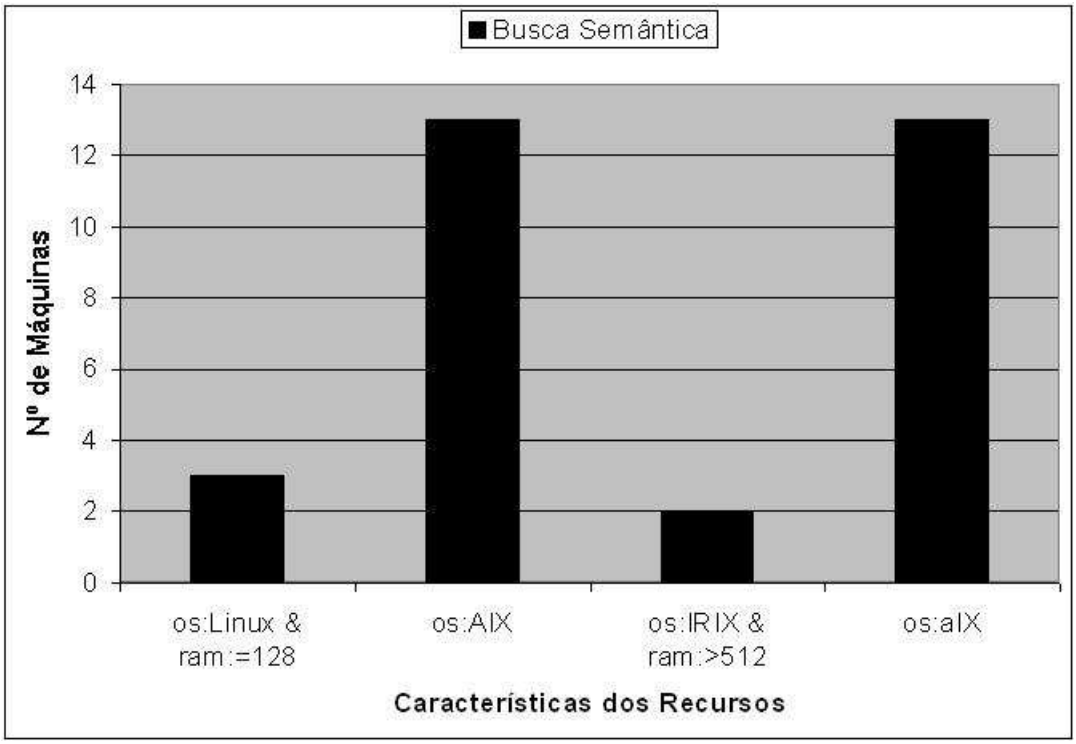

Figura 5.7: Resultados da Busca Semântica.

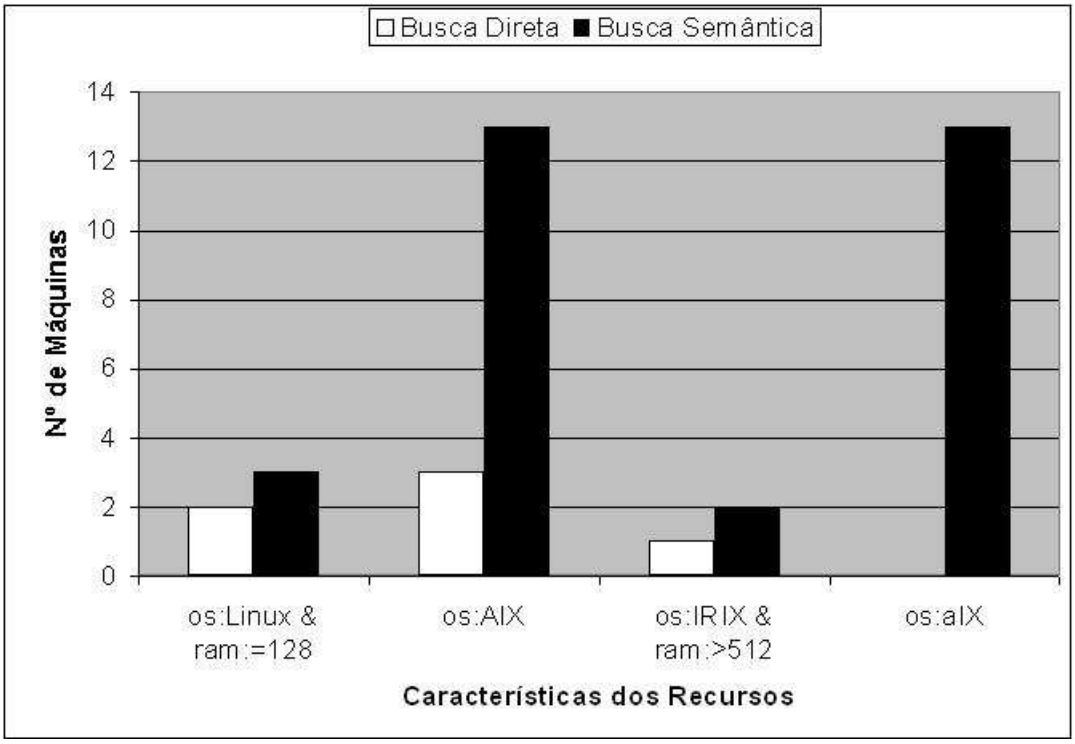

Figura 5.8: Resultados das Buscas Direta vs. Semântica.

Capítulo 6

Conclusão e Trabalhos Futuros

Os serviços de informação dos *middlewares* de computação em grade, como o serviço MDS do Globus, não suportam a descrição semântica dos serviços nem dos recursos da grade. Sendo assim, a abordagem semântica utilizada neste trabalho representa uma alternativa eficiente no processo da escolha dos melhores recursos computacionais do ambiente de execução de aplicações.

A proposta de serviço baseado em semântica para a descoberta de recursos proposto e implementado facilita o gerenciamento dos recursos, pois melhora o processo de busca dos recursos na grade computacional que correspondem aos requisitos da aplicação. O uso do Repositório Semântico, em que as informações sobre os recursos são agregadas dinamicamente, se mostrou uma solução viável, com o uso do *template* ontológico dos recursos.

Na realização deste trabalho notou-se também que o uso da arquitetura de Grade Semântica pode facilitar a implantação de aplicações complexas, nas quais múltiplas organizações estão envolvidas e seus diversos recursos estão compartilhados.

Dos resultados experimentais obtidos, conclui-se que é interessante oferecer a busca semântica no ambiente de grade computacional de modo a simplificar o processo de casamento dos recursos computacionais existentes com os requisitos das aplicações a serem executadas.

Como trabalho futuro, sugerimos testar a abordagem deste trabalho em Serviços de Informação de outros *middlewares* de grade, uma vez que o Repositório de Recursos foi projetado para se utilizar o formato XML.

Outra sugestão é aprimorar as regras semânticas da ontologia utilizada, permitindo a captura de outros recursos de máquina, que não são providos atualmente pelo Ganglia. Para isto seria necessário o estudo de outros sistemas de monitoramento de recursos da grade. Uma outra forma de permitir a adição de novos conceitos no *template* ontológico seria possibilitar que o próprio usuário do sistema

mapeie estes novos recursos de máquina. Deste modo, o Repositório Semântico poderia ser construído de modo colaborativo e teria sua semântica aumentada no mesma proporção em que a comunidade de usuários do sistema também fosse aumentada.

Outro aprimoramento semântico é que as palavras sinônimas de linguagem fossem tratadas pelo *template* ontológico, por exemplo, impressora e *printer* representariam o mesmo conceito ontológico. Assim, espera-se que a descoberta semântica de recursos apresente um desempenho progressivo.

Ainda no sentido de aprimorar o trabalho desenvolvido, pode-se tornar a interface gráfica mais simples e funcional. Um *browser* pode ser criado para a navegação na ontologia dos recursos da grade, facilitando o trabalho do pesquisador, que será usuário final do sistema.

E por fim, fica a sugestão de se testar o sistema em um ambiente de grade computacional com centenas ou milhares de máquinas, que estejam espalhadas geograficamente ao redor do país para se verificar aspectos de escalabilidade.

Referências Bibliográficas

Akkiraju, R., Goodwin, R., Doshi, P., and Roeder, S. (2003). A method for semantically enhancing the service discovery capabilities of uddi. In *Proceedings of IJCAI - Workshop of Information Integration on the Web*, Acapulco, México.

Alchemi (2006). .net grid computing framework. Disponível em: `http://www.gridbus.org/~alchemi/`. Acesso em: dezembro de 2006.

Alper, P., Corcho, O., Kotsiopoulos, I., Missier, P., Bechhofer, S., Kuo, D., and Goble, C. (2006). S-ogsa as a reference architecture for ontogrid and for the semantic grid. In *Proceedings of the* 3^{rd} *GGF Semantic Grid Workshop (GGF16)*, Macedonia.

Andreozzi, S. (2006). Glue schema. Disponível em: `http://glueschema.forge.cnaf.infn.it/`. Acesso em: dezembro de 2006.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P., editors (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge Press. Disponível em: `http://www.cambridge.org/uk/0521781760`. Acesso em: dezembro de 2006.

Baird, I. (2002). Understanding grid computing. Disponível em: `http://www.gridtoday.com/02/0701/100060.html`. Acesso em: dezembro de 2006.

Basiura, R., Batongbacal, M., Bohling, B., Clark, M., Eide, A., Eisenberg, R., Lee, D., Loesgen, B., Miller, C., Reynolds, M., Sempf, B., and Sivakumar, S., editors (2003). *Professional ASP.NET Web Service*. Pearson Education.

Bechhofer, S. and Ng, G. (2006). Oiled. Disponível em: `http://oiled.man.ac.uk`. Acesso em: dezembro de 2006.

Berman, F. (2001). From teragrid to knowledge grid - integrating distributed resources into cohesive, virtual supercomputers. *Communications of the ACM*, 44(11):27–28.

Berman, F., Hey, A. J. G., and Fox, G. C. (2003). *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., England.

Berners-Lee, T. (2006). World wide web consortium. Disponível em: `http://www.w3.org`. Acesso em: dezembro de 2006.

Blackburn, S. (1997). *Dicionário Oxford de Filosofia*. Jorge Zahar, Rio de Janeiro.

Brickley, D. and Guha, R. (2000). Resource description framework (rdf) schema specification 1.0. W3C Candidate Recommendation. Disponível em: `http://www.w3.org/TR/2000/CR-rdf-schema-20000327/`. Acesso em: dezembro de 2006.

Broekstra, J. and Kampman, A. (2004). Serql: An rdf query and transformation language. In *Submitted to the International Semantic Web Conference (ISWC 2004)*.

Brooke, J., Fellows, D., Garwood, K., and Goble, C. (2004). Semantic matching of grid resource description. In *Grid Computing - LNCS*, volume 3165, pages 240–249. Springer-Verlag, Berlin/Heidelberg. ISBN 978-3-540-22888-2.

Buyya, R. (2006). Grid computing. Disponível em: `http://www.gridcomputing.com/`. Acesso em: dezembro de 2006.

Caltech (2006). Monitoring agents using a large integrated services architecture. Disponível em: `http://monalisa.caltech.edu/`. Acesso em: dezembro de 2006.

Cannataro, M., Comito, C., Schiavo, F. L., and Veltri, P. (2004). Proteus, a grid based problem solving environment for bioinformatics: Architecture and experiments. *IEEE Computational Intelligence Bulletin*, 3(1):7–18.

Clifford, B. (2005). Gt4 monitoring and discovery. Disponível em: `http://www.globus.org/toolkit/presentations/GlobusWorld_2005_Session_9c.pdf`. Acesso em: dezembro de 2006.

Congiusta, A., Mastroianni, C., Pugliese, A., Talia, D., and Trunfio, P. (2002). The open grid services architecture: Where the grid meets the web. *IEEE Internet Computing*, 6(6):67–71.

Congiusta, A., Mastroianni, C., Pugliese, A., Talia, D., and Trunfio, P. (2004). Enabling knowledge discovery services on grids. *Proc. of the 2nd European AcrossGrids Conference (AxGrids 2004)*, LNCS 3165:250–259.

Connolly, D., Hendler, J., and Schreiber, G. (2006). Web-ontology (webont) working group. Disponível em: `http://www.w3.org/2001/sw/WebOnt/`. Acesso em: dezembro de 2006.

Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2001). Daml+oil (março 2001) reference description. Disponível em: `http://www.w3.org/TR/daml+oil-reference`. Acesso em: dezembro de 2006.

Czajkowski, K., Foster, I., Kesselman, C., Sander, V., and Tuecke, S. (2002). Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *Proceedings of the* 8^{th} *Workshop on job Scheduling Strategies for Parallel Processing - LNCS*, volume 2537, pages 153–183.

da Figueira Filho, C. S. (2000). Jeops - integração entre objetos e regras de produção em java. Master's thesis, Centro de Informatica, Universidade Federal de Pernambuco, Recife, Pernambuco.

DAML+OIL (2001). Daml+oil language. Disponível em: `http://www.daml.org/2001/03/daml+oil-index.html`. Acesso em: dezembro de 2006.

de Holanda Ferreira, A. B. (1986). *Novo dicionário da língua portuguesa*. Editora Nova Fronteira, Rio de Janeiro, 2th edition.

Dean, M. and Schreiber, G. (2004). Owl web ontology language reference. W3C Recommendation. Disponível em: `http://www.w3.org/tr/owl-ref/`. Acesso em: dezembro de 2006.

Digital Sky Survey (2006). Sloan digital sky survey. Disponível em: `http://www.sdss.org/`. Acesso em: dezembro de 2006.

Djuric, D., Gasevic, D., and Devedzic, V. (2005). Ontology modeling and mda. *Journal of Object Technology*, 4(1):109–128. Disponível em: `http://www.jot.fm/issues/issue\_2005\_01/article3`. Acesso em: dezembro de 2006.

Efstathiadis, S. (2006). A simple ganglia mds information provider. Disponível em: `http://www.star.bnl.gov/STAR/comp/Grid/Monitoring/SimpleGangliaIP.html`. Acesso em: dezembro de 2006.

Foster, I. (2006). Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing - LNCS*, volume 3779, pages 2–13. Springer-Verlag.

Foster, I., Czajkowski, K., Ferguson, D., Frey, J., Graham, S., Maguire, T., Snelling, D., and Tuecke, S. (2005). Modeling and managing state in distributed systems: The role of ogsi and wsrf. *Proceedings of the IEEE*, 93(3):604–612.

Foster, I. and Kesselman, C. (1997). Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128.

Foster, I. and Kesselman, C. (1999). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, New York, NY.

Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). Grid services for distributed system integration. *Computer*, 35(6).

Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organization. *Lecture Notes in Computer Science - Journal of Supercomputer Applications*, 2150.

Fromm, K. R., Polikoff, I., Obrst, L., Daconta, M. C., Murphy, R., and hong Morrison, J. (2005). Introducing semantic technologies and the vision of the semantic web. *Semantic Interoperability Community of Practice (SICoP)*.

Ganglia Development Team (2006). Ganglia monitoring system. Disponível em: `http://ganglia.sourceforge.net/`. Acesso em: dezembro de 2006.

Geldof, M. (2004). The semantic grid: will semantic web and grid go hand in hand? *Grid Technologies Unit of the European Commission DG Information Society*. Disponível em: `http://www.semanticgrid.org/documents/SemanticGridreportpublic.pdf`. Acesso em: dezembro de 2006.

Genesereth, M. R. (2006). Knowledge interchange format (kif). Disponível em: `http://logic.stanford.edu/kif/`. Acesso em: dezembro de 2006.

Genesereth, M. R. and Nilsson, L. (1987). *Logical foundation of AI*. Morgan Kaufman, San Francisco, Los Altos, Califórnia.

Global Grid Forum (2003). Global grid forum. Disponível em: `http://www.gridforum.org`. Acesso em: dezembro de 2006.

Gómez-Pérez, A., Abruña, L. B., Sanchez, M., de los Santos Pérez Hernández, M., Corcho, O., and González-Cabero, R. (2006). Semantic grid applications to complex satellite mission system. In *Proceedings of the 7^{th} IEEE/ACM International Conference on Grid Computing (Grid 2006)*. poster.

Gómez-Pérez, A. and Corcho, O. (2002). Ontology languages for the semantic web. *IEEE Intelligent Systems*, 17(1):54–60.

Goble, C. (2005). Towards a semantic grid architecture. In Goble, C., Kesselman, C., and Sure, Y., editors, *Semantic Grid: The Convergence of Technologies*, number 05271 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany. Disponível em: `http://drops.dagstuhl.de/opus/volltexte/2005/384`. Acesso em: dezembro de 2006.

Goble, C., Kotsiopoulos, I., Corcho, O., Alper, P., and Bechhofer, S. (2006). An overview of s-ogsa: a reference architecture for the semantic grid. *Journal of Web Semantics*, 4(2):102–115.

Google Inc. (2006). Google search. Disponível em: `http://www.google.com/`. Acesso em: dezembro de 2006.

Grid Computing and Distributed Systems Laboratory (2006). The gridbus project. Disponível em: `http://www.gridbus.org/`. Acesso em: dezembro de 2006.

Gruber, T. R. (1993a). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*.

Gruber, T. R. (1993b). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220.

Gruber, T. R. (2005). What is an ontology? Disponível em: `http://www-ksl.stanford.edu/kst/what-is-an-ontology.html`. Acesso em: dezembro de 2006.

Guarino, N. (1998). Formal ontology and information systems. In *Proceedings of FOIS*, pages 3–15, Trento, Itália. FOIS. Amsterdan, IOS Press.

Gunter, D. and Tierney, B. (2003). Netlogger: A toolkit for distributed system performance tuning and debugging. In *Integrated Network Management*, pages 97–100.

Haase, P., Broekstra, J., Eberhart, A., and Volz, R. (2004). A comparison of rdf query languages. *The Semantic Web (ISWC 2004)*, LNCS 3298/2004:502–517.

Harth, A., Decker, S., He, Y., Tangmunarunkit, H., and Kesselman, C. (2004). A semantic matchmaker service on the grid. In *Proceedings of the* 13^{th} *international World Wide Web conference on Alternate track papers & posters (WWW Alt.)*, pages 326–327, New York, USA. ACM Press.

Hawkeye (2006). A monitoring and management tool for distributed systems. Disponível em: `http://www.cs.wisc.edu/condor/hawkeye/`. Acesso em: dezembro de 2006.

Hendricks, M., Galbraith, B., Irani, R., Milberny, J., Modi, T., Tost, A., Tousaaint, A., Basha, J., and Cable, S. (2002). *Professional Java Web Services*. Editora Alta Books, Rio de Janeiro.

Horrocks, I. (1997). *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester.

Horrocks, I. (2006). Fact: Fast classification of terminologies description logic classifier. Disponível em: `http://www.cs.man.ac.uk/~horrocks/FaCT/`. Acesso em: dezembro de 2006.

Howe, D. (2003). Production system from foldoc. Disponível em: `http://www.nue.org/foldoc/foldoc.cgi?query=production+system`. Acesso em: dezembro de 2006.

HP Labs Semantic Web Research (2006). Jena: A semantic web framework for java. Disponível em: `http://jena.sourceforge.net/`. Acesso em: dezembro de 2006.

Isaac, A. and Troncy, R. (2006). Doe - the differential ontology editor. Disponível em: `http://homepages.cwi.nl/~troncy/DOE`. Acesso em: dezembro de 2006.

Jackson, P., editor (1998). *Introduction to Expert Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition.

Jacobs, I. (2005). About the world wide web consortium (w3c). Disponível em: `http://www.w3.org/Consortium/Overview`. Acesso em: dezembro de 2006.

Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., and Scholl, M. (2002). Rql: A declarative query language for rdf. In *11th Intl. World Wide Web Conference (WWW2002)*, pages 592–603.

Kifer, M., Lausen, G., and Wu, J. (1990). Logical foundations of object-oriented and frame-based languages. Technical Report TR-90-003, University of Mannheim.

Knowledge Media Institute (2006). Ocml: Operational conceptual modelling language. Disponível em: `http://kmi.open.ac.uk/projects/ocml/`. Acesso em: dezembro de 2006.

Knowledge Systems AI Laboratory Stanford University (2006a). The chimaera ontology environment. Disponível em: `http://www.ksl.stanford.edu/software/chimaera/`. Acesso em: dezembro de 2006.

Knowledge Systems AI Laboratory Stanford University (2006b). Ontolingua home page. Disponível em: `http://www.ksl.stanford.edu/software/ontolingua/`. Acesso em: dezembro de 2006.

Knublauch, H. (2004). Ontology-driven software development in the context of the semantic web: An example scenario with protege/owl. In *Proceedings of the International Workshop on the Model-Driven Semantic Web*, Monterey, CA. Disponível em: `http://www.knublauch.com/publications.html`. Acesso em: dezembro de 2006.

Koivunen, M. and Miller, E. (2001). W3c semantic web activity. W3C's official internet site. Disponível em: `http://www.w3.org/2001/12/semweb-fin/w3csw`. Acesso em: dezembro de 2006.

Lassila, O. and Swick, R. (1999). Resource description framework (rdf) - model and syntax specification. W3C Recommendation. Disponível em: `http://www.w3.org/TR/REC-rdf-syntax/`. Acesso em: dezembro de 2006.

Legrand, I. (2003). Monalisa: Monitoring agents using a large integrated architecture. Disponível em: `http://chep03.ucsd.edu/files/103.pdf`. Acesso em: dezembro de 2006.

Li, M., Santen, P. V., Walker, D., Rana, O., and Baker, M. (2003). Sgrid: a service-oriented model for the semantic grid. *Future Generation Computer Systems Journal (FGCS)*.

Luger, G. (2002). *Artificial Intelligence - Structures and Strategies for Complex Problem Solving*. Addison Wesley, New York, NY, 4th edition.

Massie, M. L., Chun, B. N., and Culler, D. E. (2004). The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7).

McGuinness, D. L. and van Harmelen, F. (2004). Owl web ontology language overview. Editors' Draft W3C. Disponível em: `http://www.w3.org/tr/2004/REC-owl-features-20040210/`. Acesso em: dezembro de 2006.

Mizoguchi, R. (2003). Tutorial on ontological engineering - part 2: Ontology development, tools and languages. *New Generation Computing*, 22(1):61–96. Disponível em: `http://www.ei.sanken.osaka-u.ac.jp/japanese/tutorial-j.html`. Acesso em: dezembro de 2006.

Niemann, B., Morris, R., Riofrio, H. J., and Carnes, E. (2005). Introducing semantic technologies and the vision of the semantic web. Semantic Interoperability (XML Web Services) Community of Practice (SICoP). Disponível em: `http://web-services.gov`. Acesso em: dezembro de 2006.

OntoGrid Consortium (2006). Ontogrid project. Disponível em: `http://www.ontogrid.net/`. Acesso em: dezembro de 2006.

OntoPrise (2006). Ontostudio. Disponível em: `http://www.ontoprise.de`. Acesso em: dezembro de 2006.

Open Grid Forum (2006). Open grid forum. Disponível em: `http://www.ogf.org/`. Acesso em: dezembro de 2006.

Patel-Schneider, P. F., Hayes, P., and Horrocks, I. (2003). Owl web ontology language semantics and abstract syntax. W3C Candidate Recommendation. Disponível em: `http://www.w3.org/TR/owl-semantics/`. Acesso em: dezembro de 2006.

Racer Systems (2006). Racerpro reasoner. Disponível em: `http://www.racer-systems.com/`. Acesso em: dezembro de 2006.

Rice, P., Longden, I., and Bleasby, A. (2000). Emboss: The european molecular biology open software suite. *Trends in Genetics*, 16(6):276–277. Disponível em: `http://emboss.sourceforge.net/`. Acesso em: dezembro de 2006.

Rich, E. and Knight, K. (1993). *Artificial Intelligence*. McGraw Hill, 2th edition.

Riley, G. (2004). What are expert systems? Disponível em: `http://www.ghgcorp.com/clips/ExpertSystems.html`. Acesso em: dezembro de 2006.

Roure, D., Jennings, N., and Shadbolt, N. (2003). The semantic grid: A future e-science infrastructure. In *Grid Computing - Making the Global Infrastructure*. John Wiley & Sons.

Roure, D. D. (2006). Semantic grid community portal. Disponível em: `http://www.semanticgrid.org/`. Acesso em: dezembro de 2006.

Roure, D. D., Jennings, N. R., and Shadbolt, N. R. (2005). The semantic grid: Past, present and future. In *The Semantic Web: Research and Applications - LNCS*, volume 3532, page 726. Springer-Verlag, Berlin/Heidelberg. ISBN 978-3-540-26124-7.

Russell, S. and Norvig, P., editors (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition.

Óscar Corcho, Fernández-López, M., and Gómez-Pérez, A. (2003). Methodologies, tools and languages for building ontologies: Where is their meeting point? *Data Knowl. Eng.*, 46(1):41–64.

Schopf, J. M., Raicu, I., Pearlman, L., Miller, N., Kesselman, C., Foster, I., and D'Arcy, M. (2006). Monitoring and discovery in a web services framework: Functionality and performance of globus toolkit mds4. In *submitted to HPDC 2006*. available as MCS Preprint ANL/MCS-P1315-0106.

Schor, M. (1986). Declarative knowledge programming: Better than procedural? *In IEEE Expert*, 33:36–43.

Seaborne, A. (2004). *RDQL - A Query Language for RDF*. Disponível em: `http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/`. Acesso em: dezembro de 2006.

Sintek, M. and Decker, S. (2001). Triple - an rdf query, inference, and transformation language.

Sintek, M., Decker, S., and Harth, A. (2006). Triple. Disponível em: `http://triple.semanticweb.org/`. Acesso em: dezembro de 2006.

Sirin, E. and Parsia, B. (2004). Pellet: An owl dl reasoner. In *Proceedings of the International Semantic Web Conference (ISWC)*.

Somasundaram, T. S., Balachandar, R. A., Kandasamy, V., Buyya, R., Raman, R., Mohanram, N., and Varun, S. (2006). Semantic-based grid resource discovery and its integration with the grid service broker. Technical Report GRIDS-

TR-2006-10, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia.

Sowa, J. (1999). *Knowledge representation: logical, philosophical and computational foundations*. Brooks Cole Pub. Co., Pacific Grove, EUA.

Stanford Medical Informatics at the Stanford University School of Medicine (2006). The protégé ontology editor and knowledge acquisition system. Disponível em: `http://protege.stanford.edu/`. Acesso em: dezembro de 2006.

Studer, R., Benjamins, V. R., and Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25(1-2):161–197.

Su, X. and Ilebrekke, L. (2002). A comparative study of ontology languages and tools. *Lecture Notes In Computer Science*, pages 761–765. Disponível em: `http://www.idi.ntnu.no/~xiaomeng/paper/caise02WorkshopCRC.pdf`. Acesso em: dezembro de 2006.

Sun Microsystems (2005). Sun grid engine. Disponível em: `http://gridengine.sunsource.net`. Acesso em: dezembro de 2006.

Sun Microsystems (2006). Java architecture for xml binding. Disponível em: `http://java.sun.com/xml/downloads/jaxb.html`. Acesso em: dezembro de 2006.

Swiss Institute of Bioinformatics (2006). Swiss-prot protein knowledgebase. Disponível em: `http://www.expasy.org/sprot/`. Acesso em: dezembro de 2006.

Talia, D. (2003). Knowledge discovery services and tools on grids. *Symposium ISMIS*.

Tangmunarunkit, H., Decker, S., and Kesselman, C. (2003). Ontology-based resource matching in the grid - the grid meets the semantic web. In *The SemanticWeb - ISWC 2003 - LNCS*, volume 2870, pages 706–721. Springer-Verlag, Berlin/Heidelberg. ISBN 978-3-540-20362-9.

The Globus Alliance (2005). The globus alliance. Disponível em: `http://www.globus.org`. Acesso em: dezembro de 2006.

The Globus Alliance (2006a). Globus toolkit 4 overview. Disponível em: `http://www.globus.org/toolkit/about.html`. Acesso em: dezembro de 2006.

The Globus Alliance (2006b). Gt information services: Monitoring and discovery system (mds). Disponível em: `http://www.globus.org/toolkit/mds`. Acesso em: dezembro de 2006.

The Globus Alliance (2006c). Towards open grid services architecture. Disponível em: `http://www.globus.org/ogsa/`. Acesso em: dezembro de 2006.

The Globus Alliance (2006d). The ws-resource framework. Disponível em: `http://www.globus.org/wsrf/`. Acesso em: dezembro de 2006.

The Globus Project (2000). Globus toolkit 1.1.3 system administration guide. Disponível em: `http://www.globus.org`. Acesso em: dezembro de 2006.

UNICORE Forum (2005). Unicore. Disponível em: `http://www.unicore.org`. Acesso em: dezembro de 2006.

University of Maryland's Mindswap Lab (2006). Pellet owl reasoner. Disponível em: `http://www.mindswap.org/2003/pellet/`. Acesso em: dezembro de 2006.

University of Southern California's Information Sciences Institute (2006). Loom project home page. Disponível em: `http://www.isi.edu/isd/LOOM/`. Acesso em: dezembro de 2006.

University of Virginia (2006). Legion: Worldwide virtual computer. Disponível em: `http://legion.virginia.edu`. Acesso em: dezembro de 2006.

Venugopal, S., Buyya, R., and Winton, L. (2004). A grid service broker for scheduling distributed data-oriented applications on global grids. In *Proceedings of the* 2^{nd} *workshop on Middleware for grid computing (MGC)*, pages 75–80, New York, USA. ACM Press.

World Community Grid (2005). World community grid. Disponível em: `http://www.worldcommunitygrid.org`. Acesso em: dezembro de 2006.

Xing, W., Dikaiakos, M. D., and Sakellariou, R. (2006). A core grid ontology for the semantic grid. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 178–184, Washington, DC, USA. IEEE Computer Society.

Zhang, Y. and Song, W. (2004). Semantic description and matching of grid services capabilities. In *Proceedings of the* 3^{rd} *UK e-Science All Hands Meeting*, UK.

Zhuge, H. (2004). China's e-science knowledge grid environment. *IEEE Intelligent Systems*, 19(1):13–17.

Zou, Y., Finin, T., and Chen, H. (2004). F-owl: an inference engine for the semantic web. *Lecture Notes in Computer Science*.

Apêndice A

Ontologias

A.1 *Template* da Ontologia - josenelson.owl

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.cic.unb.br/josenelson.owl#"
  xml:base="http://www.cic.unb.br/josenelson.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="MainMemory"/>
  <owl:Class rdf:ID="Architecture"/>
  <owl:Class rdf:ID="Host"/>
  <owl:Class rdf:ID="Processor"/>
  <owl:Class rdf:ID="OperatingSystem"/>
  <owl:Class rdf:ID="NetworkAdapter"/>
  <owl:Class rdf:ID="ProcessorLoad"/>
  <owl:Class rdf:ID="FileSystem"/>
  <owl:ObjectProperty rdf:ID="hasFileSystem">
    <rdfs:range rdf:resource="#FileSystem"/>
    <rdfs:domain rdf:resource="#Host"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasProcessor">
    <rdfs:domain rdf:resource="#Host"/>
    <rdfs:range rdf:resource="#Processor"/>
  </owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="hasNetworkAdapter">
  <rdfs:domain rdf:resource="#Host"/>
  <rdfs:range rdf:resource="#NetworkAdapter"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasMainMemory">
  <rdfs:range rdf:resource="#MainMemory"/>
  <rdfs:domain rdf:resource="#Host"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasOperatingSystem">
  <rdfs:domain rdf:resource="#Host"/>
  <rdfs:range rdf:resource="#OperatingSystem"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasArchitecture">
  <rdfs:range rdf:resource="#Architecture"/>
  <rdfs:domain rdf:resource="#Host"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasProcessorLoad">
  <rdfs:domain rdf:resource="#Host"/>
  <rdfs:range rdf:resource="#ProcessorLoad"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="VirtualSize">
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain
   rdf:resource="#MainMemory"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Configured disk-based virtual memory (VM)
   in MB in a computing node.
  </rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ClockSpeed">
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The MHz associated with the CPUS in the subcluster.
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Processor"/>
```

```
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="SMPSize">
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Number of CPUs in an SMP node.</rdfs:comment>
  <rdfs:domain rdf:resource="#Architecture"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="RAMSize">
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Configured physical memory on any one CPU
   in the subcluster in MB.
  </rdfs:comment>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain
   rdf:resource="#MainMemory"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="CacheL2">
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Second-level unified cache size (in kb) of a cpu.
  </rdfs:comment>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#Processor"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="OutboundIP">
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Defines if outbound connectivity is allowed
   from "worker nodes" - can a worked node initiate
   outbound connectivity.
```

```
  </rdfs:comment>
  <rdfs:domain rdf:resource="#NetworkAdapter"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="CacheL1D">
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >First-level data cache size (in kb) of a cpu.
  </rdfs:comment>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#Processor"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Last5Min">
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >5-minute average processor availability for a single
  node (the difference between the available CPUs and
  the average runable task count during that time) X 100.
  </rdfs:comment>
  <rdfs:domain rdf:resource="#ProcessorLoad"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ReadOnly">
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  <rdfs:domain rdf:resource="#FileSystem"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Is the file system readonly?</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Size">
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#FileSystem"/>
```

```
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Total space assigned for this file type (MB).
  </rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="InboundIP">
  <rdfs:domain rdf:resource="#NetworkAdapter"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Defines if inbound connectivity is allowed.
  </rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Root">
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Path name or other information defining the root
   of the file system.
  </rdfs:comment>
  <rdfs:domain rdf:resource="#FileSystem"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Last15Min">
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >15-minute average processor availability for a single
   node (the difference between the available CPUs and
   the average runable task count during that time) X 100.
  </rdfs:comment>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#ProcessorLoad"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="UniqueID">
  <rdfs:comment
```

```
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >A unique identifier for the computing element.
  </rdfs:comment>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Host"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="FSName">
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The name for the file system.</rdfs:comment>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#FileSystem"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="CacheL1I">
  <rdfs:domain rdf:resource="#Processor"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >First-level instruction cache size (in kb) of a cpu.
  </rdfs:comment>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="RAMAvailable">
  <rdfs:domain rdf:resource="#MainMemory"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Unallocated RAM size in MB.</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="OSName">
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#OperatingSystem"/>
  <rdfs:comment
```

```
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Informally names the OS using a vendor-specific
   convention.
  </rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="IPAddress">
  <rdfs:domain rdf:resource="#NetworkAdapter"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Ip address of a network interface.</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="CacheL1">
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >First-level unified cache size (in kb) of a cpu.
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Processor"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Name">
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >A name for this service.</rdfs:comment>
  <rdfs:domain rdf:resource="#Host"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="OSRelease">
  <rdfs:domain rdf:resource="#OperatingSystem"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Informally names the OS release using a
```

```
   vendor-specific convention.
  </rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="NetName">
  <rdfs:domain rdf:resource="#NetworkAdapter"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Names a network interface.</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="VirtualAvailable">
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#MainMemory"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Available virtual memory.</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="InstructionSet">
  <rdfs:domain rdf:resource="#Processor"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Processor instruction set.</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="AvailableSpace">
  <rdfs:comment
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Total available space for this file type (MB).
  </rdfs:comment>
  <rdfs:domain rdf:resource="#FileSystem"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Last1Min">
```

```
    <rdfs:range
     rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#ProcessorLoad"/>
    <rdfs:comment
     rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >1-minute average processor availability for a single
     node (the difference between the available CPUs and
     the average runable task count during that time) X 100.
    </rdfs:comment>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="MTU">
    <rdfs:comment
     rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Maximum transmission unit size (in bytes) for a
     network interface.
    </rdfs:comment>
    <rdfs:range
     rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#NetworkAdapter"/>
  </owl:DatatypeProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 2.1, Build 284)
http://protege.stanford.edu -->
```

A.2 Arquivo XML recuperado pelo Ganglia / MDS4

```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
  <ns1:GLUECE xmlns:ns1="http://mds.globus.org/glue/ce/1.1">
    <ns1:Cluster
     ns1:Name="unspecified"
     ns1:UniqueID="unspecified">
      <ns1:SubCluster ns1:Name="main" ns1:UniqueID="main">
        <ns1:Host ns1:Name="pos-10.cic.unb.br"
                  ns1:UniqueID="pos-10.cic.unb.br">
          <ns1:Processor ns1:CacheL1="0" ns1:CacheL1D="0"
```

```
                ns1:CacheL1I="0" ns1:CacheL2="0"
                ns1:ClockSpeed="995"
                ns1:InstructionSet="x86" />
  <ns1:MainMemory ns1:RAMAvailable="7"
                ns1:RAMSize="233"
                ns1:VirtualAvailable="263"
                ns1:VirtualSize="512" />
  <ns1:OperatingSystem ns1:Name="Linux"
                      ns1:Release="2.4.20-8" />
  <ns1:Architecture ns1:SMPSize="1" />
  <ns1:FileSystem ns1:AvailableSpace="770"
                ns1:Name="entire-system"
                ns1:ReadOnly="false"
                ns1:Root="/"
                ns1:Size="9724" />
  <ns1:NetworkAdapter ns1:IPAddress="164.41.14.90"
                     ns1:InboundIP="true"
                     ns1:MTU="0"
                     ns1:Name="pos-10.cic.unb.br"
                     ns1:OutboundIP="true" />
  <ns1:ProcessorLoad ns1:Last15Min="0"
                    ns1:Last1Min="0"
                    ns1:Last5Min="0" />
</ns1:Host>
<ns1:Host ns1:Name="pos-03.cic.unb.br"
          ns1:UniqueID="pos-03.cic.unb.br">
  <ns1:Processor ns1:CacheL1="0" ns1:CacheL1D="0"
                ns1:CacheL1I="0" ns1:CacheL2="0"
                ns1:ClockSpeed="1095"
                ns1:InstructionSet="x86" />
  <ns1:MainMemory ns1:RAMAvailable="123"
                ns1:RAMSize="233"
                ns1:VirtualAvailable="608"
                ns1:VirtualSize="721" />
  <ns1:OperatingSystem ns1:Name="Linux"
                      ns1:Release="2.4.20-8" />
  <ns1:Architecture ns1:SMPSize="1" />
```

```
  <ns1:FileSystem ns1:AvailableSpace="2334"
                  ns1:Name="entire-system"
                  ns1:ReadOnly="false"
                  ns1:Root="/"
                  ns1:Size="9508" />
  <ns1:NetworkAdapter ns1:IPAddress="164.41.14.83"
                      ns1:InboundIP="true"
                      ns1:MTU="0"
                      ns1:Name="pos-03.cic.unb.br"
                      ns1:OutboundIP="true" />
  <ns1:ProcessorLoad ns1:Last15Min="0"
                     ns1:Last1Min="0"
                     ns1:Last5Min="0" />
</ns1:Host>
<ns1:Host ns1:Name="pos-12.cic.unb.br"
          ns1:UniqueID="pos-12.cic.unb.br">
  <ns1:Processor ns1:CacheL1="0" ns1:CacheL1D="0"
                 ns1:CacheL1I="0" ns1:CacheL2="0"
                 ns1:ClockSpeed="995"
                 ns1:InstructionSet="x86" />
  <ns1:MainMemory ns1:RAMAvailable="66"
                  ns1:RAMSize="233"
                  ns1:VirtualAvailable="557"
                  ns1:VirtualSize="729" />
  <ns1:OperatingSystem ns1:Name="Linux"
                       ns1:Release="2.4.20-8" />
  <ns1:Architecture ns1:SMPSize="1" />
  <ns1:FileSystem ns1:AvailableSpace="3434"
                  ns1:Name="entire-system"
                  ns1:ReadOnly="false"
                  ns1:Root="/"
                  ns1:Size="9573" />
  <ns1:NetworkAdapter ns1:IPAddress="164.41.14.92"
                      ns1:InboundIP="true"
                      ns1:MTU="0"
                      ns1:Name="pos-12.cic.unb.br"
                      ns1:OutboundIP="true" />
```

```
        <ns1:ProcessorLoad ns1:Last15Min="0"
                           ns1:Last1Min="0"
                           ns1:Last5Min="0" />
      </ns1:Host>
      <ns1:Host ns1:Name="pos-14.cic.unb.br"
                ns1:UniqueID="pos-14.cic.unb.br">
        <ns1:Processor ns1:CacheL1="0" ns1:CacheL1D="0"
                       ns1:CacheL1I="0" ns1:CacheL2="0"
                       ns1:ClockSpeed="995"
                       ns1:InstructionSet="x86" />
        <ns1:MainMemory ns1:RAMAvailable="128"
                        ns1:RAMSize="233"
                        ns1:VirtualAvailable="613"
                        ns1:VirtualSize="721" />
        <ns1:OperatingSystem ns1:Name="Linux"
                             ns1:Release="2.4.20-8" />
        <ns1:Architecture ns1:SMPSize="1" />
        <ns1:FileSystem ns1:AvailableSpace="94"
                        ns1:Name="entire-system"
                        ns1:ReadOnly="false"
                        ns1:Root="/"
                        ns1:Size="9508" />
        <ns1:NetworkAdapter ns1:IPAddress="164.41.14.94"
                            ns1:InboundIP="true"
                            ns1:MTU="0"
                            ns1:Name="pos-14.cic.unb.br"
                            ns1:OutboundIP="true" />
        <ns1:ProcessorLoad ns1:Last15Min="0"
                           ns1:Last1Min="0"
                           ns1:Last5Min="0" />
      </ns1:Host>
    </ns1:SubCluster>
  </ns1:Cluster>
</ns1:GLUECE>
```

Apêndice B

Instalações

B.1 Instalação do Globus Toolkit 4

O procedimento abaixo descreve como foi feita a instalação[1] do GT4 em nosso laboratório.

- Baixar o GT4
- Criar o usuário e grupo (e senha): globus
- Criar o diretório de instalação do globus
  - # mkdir /usr/local/globus
  - # chown globus:globus /usr/local/globus
- Descompactar o GT4 em: /home/globus
  - globus$ tar -zvxf gt4.0.3-x86_rhas_3-installer.tar.gz
  - globus$ export GLOBUS_LOCATION=/usr/local/globus-4.0.1
  - globus$ ./configure −−prefix=$GLOBUS_LOCATION
- Fazer o **make** apontando para o diretório de instalação do Globus:
  - globus$ make
- Fazer o **make install**
  - globus$ make install

Tratando da Segurança

[1] http://www.globus.org/toolkit/docs/4.0/admin/docbook/

- Criando um SimpleCA:
  - $GLOBUS_LOCATION/setup/globus/setup-simple-ca "/O=Grid/OU=GlobusTest/OU=simpleCA-pos-08.cic.unb.br/CN=Globus Simple CA"
  - (e-mail = "josenelson@gmail.com") (password = "senha")
- A chave privada da *Certificate Authority* (CA) é armazenada em:
  - /home/globus/.globus/simpleCA//private/cakey.pem
- O certificado público da CA é armazenado em:
  - /home/globus/.globus/simpleCA//cacert.pem

GLOBUS_LOCATION/setup/globus_simple_ca_CA_Hash_setup /setup-gsi-default

Gerar certificados para

- host
  - # grid-cert-request -host 'hostname'
  - Assinando o certificado do host:
    - globus$ cd /etc/grid-security
    - globus$ grid-ca-sign -in hostcert_request.pem -out hostsigned.pem
    - # mv hostsigned.pem /etc/grid_security/hostcert.pem
    - # chown root:root hostcert.pem (certificado é de propriedade do root)
    - # chmod 644 hostcert.pem (certificado é somente leitura para os outros usuários)
- usuário
  - # nelson$ grid-cert-request
  - Assinando o certificado do usuário:
    - # cp /home/nelson/.globus/usercert_request.pem /home/globus
    - globus$ grid-ca-sign -in usercert_request.pem -out signed.pem
    - # mv signed.pem /home/nelson/.globus/usercert.pem

        * # chown nelson:nelson /home/nelson/.globus/usercert.pem (certificado é de propriedade do usuário nelson)
        * # chmod 644 usercert.pem (certificado é somente leitura para os outros usuários)

- container
  - Tornando as credenciais do host acessíveis pelo container:
    * # cd /etc/grid-security
    * # cp hostkey.pem containerkey.pem
    * # cp hostcert.pem containercert.pem
    * # chown globus.globus containerkey.pem containercert.pem

Adicionando autorização

- nelson$ grid-cert-info -subject
- nelson$ whoami
- # vi /etc/grid-security/grid-mapfile

ou

- # $GLOBUS_LOCATION/sbin/grid-mapfile-add-entry -dn "/O=Grid/OU=GlobusTest/OU=simpleCA-pos-08.cic.unb.br/OU=cic.unb.br/CN=Jose Nelson" -ln nelson

É interessante se fazer uma verificação[2] básica da segurança.

- nelson$ grid-proxy-init -verify -debug

Obs: Se não funcionar é porque tem que arrumar o /etc/profile (adicionar os libs).

Inicializar o container

- globus$ globus-start-container

[2] http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch09.html

B.2 Instalação do PostgreSQL

A instalação[3] do PostgreSQL[4] é necessária para o correto funcionamento do GT4. Recomenda-se a utilização da versão 8.1 do PostgreSQL.

- $ tar xvzf postgresql-8.1.4.tar.gz
- $ cd postgresql-8.1.4
- $ ./configure
- $ make
- $ su
- # make install (por padrão, o pgsql será instalado em /usr/local/pgsql)
- # adduser postgres (administrador do pgsql)
- # mkdir /usr/local/pgsql/data
- # chown postgres /usr/local/pgsql/data
- # su - postgres
- $ /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
- $ /usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data >logfile 2>&1 &
- $ /usr/local/pgsql/bin/createdb test
- $ /usr/local/pgsql/bin/psql test

Iniciando o servidor de banco de dados

- $ postmaster -D /usr/local/pgsql/data

Agora é necessária a configuração[5] do RFT.

1. # su - postgres
2. $ vi /usr/local/pgsql/data/pg_hba.conf

[3] http://www.globusconsortium.org/tutorial/
[4] http://www.postgresql.org/
[5] http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch10.html

- Adicionar a linha:
  host rftDatabase “globus” “<server IP address here>” 255.255.255.255 md5

3. $ /usr/local/pgsql/bin/postmaster -i -D /usr/local/pgsql/data&
4. $ /usr/local/pgsql/bin/createuser globus −−pwprompt
   - Setar a senha para o usuário do banco de dados “globus”, e responder sim para a criação dos privilégios no banco de dados.
5. $ su - globus
6. $ /usr/local/pgsql/bin/createdb rftDatabase
7. $ /usr/local/pgsql/bin/psql -d rftDatabase -f $GLOBUS_LOCATION/share/globus_wsrf_rft/rft_schema.sql
8. $ vi $GLOBUS_LOCATION/etc/globus_wsrf_rft/jndi-config.xml
   - encontre o elemento <service name=“ReliableFileTransferService”> e substitua o valor da senha com a senha do usuário “globus”.

B.3 Instalação do Tomcat

Instalando o Tomcat versão 5.0.28:

- Setar o CATALINA_HOME = /usr/local/tomcat
  - # vi /etc/profile
- Criar o arquivo de configuracao, execute o comando:
  - globus$ $GLOBUS_LOCATION/lib/webmds/bin /webmds-create-context-file $CATALINA_HOME/conf/Catalina/localhost
- Se ocorrer algum problema, executar os passos abaixo:
  - Reiniciar o Tomcat. Se ele estiver em execução, pare-o:
    - globus$ $CATALINA_HOME/bin/shutdown.sh
  - Então, inicialize o Tomcat:
    - globus$ $CATALINA_HOME/bin/startup.sh

  - Remova os arquivos duplicados:
    * globus$ rm $GLOBUS_LOCATION/lib/webmds/WEB-INF/lib/puretls.jar
    * globus$ rm $GLOBUS_LOCATION/lib/webmds/WEB-INF/lib/cryptix*.jar
    * globus$ rm $GLOBUS_LOCATION/lib/webmds/WEB-INF/lib/jce-jdk*.jar

Fazendo o deploy do Tomcat:

- Para fazer o deploy da instalacao do Java WS Core no Tomcat execute:
  - globus$ cd $GLOBUS_LOCATION
  - globus$ ant -f share/globus_wsrf_common/tomcat/tomcat.xml deploySecureTomcat -Dtomcat.dir=$CATALINA_HOME
  - globus$ vi $CATALINA_HOME/conf/server.xml
- Adicionar um conector HTTPS na sessão <Service name="Catalina"> e atualize os parâmetros apropriadamente com sua configuração local (Tomcat 5.0.x):

```
<Connector
    className="org.globus.tomcat.coyote.net.HTTPSConnector"
    port="8443" maxThreads="150"
    minSpareThreads="25"
    maxSpareThreads="75"
    autoFlush="true"
    disableUploadTimeout="true"
    scheme="https"
    enableLookups="true"
    acceptCount="10"
    debug="0"
    cert="/etc/grid-security/containercert.pem"
    key="/etc/grid-security/containerkey.pem"/>
```

- Adicione a válvula HTTPS na sessão <Engine name="Catalina" ... >: <Valve className="org.globus.tomcat.coyote.valves.HTTPSValve"/>
- Se ocorrer algum problema, executar os passos abaixo:

- globus$ cd $CATALINA_HOME
- globus$ setenv CLASSPATH $CATALINA_HOME/common/lib /axis-url.jar
- globus$ bin/startup

B.4 Instalação do Ganglia

Para instalação [6] e configuração [7] do sistema de monitoramento Ganglia [8] basta seguir os passos abaixo descritos:

- Baixar o arquivo: ganglia-3.0.3.tar.gz
- Instalando o Ganglia:
  - # tar -zvxf ganglia-3.0.3.tar.gz
  - # mv ganglia-3.0.3 /usr/local/ganglia
  - # chown -R root.root /usr/local/ganglia
  - # cd /usr/local/ganglia
  - # ./configure
  - # make
  - # make install
  - # cp ./gmond/gmond.init /etc/rc.d/init.d/gmond
- Inicializando o Ganglia:
  - # /etc/rc.d/init.d/gmond start
- Parando o Ganglia:
  - # /etc/rc.d/init.d/gmond stop
- Adicionando o GMOND a lista de programas inicializados automaticamente:
  - # /sbin/chkconfig −−add gmond
- Checando se o GMOND esta respondendo bem:

[6] http://www.msg.ucsf.edu/local/ganglia/ganglia_docs/install.html
[7] research.gc.cuny.edu/index.php/Ganglia_installation_and_configuration
[8] http://ganglia.sourceforge.net/

  - # /sbin/chkconfig −−list gmond

- Testando a instalação do GMOND:
  - # telnet localhost 8649

- Ou testando a instalação para um host especifico:
  - # telnet localhost 8649 | grep “hostname”

- Obs: Associando o Globus ao Ganglia:
  - globus$ vi $GLOBUS_LOCATION/etc/globus_wsrf_mds_usefulrp /gluerp.xml