Alexandre Parra Carneiro da Silva

Mecanismo de Matching Semântico de Recursos Computacionais de Grids baseado na Integração Semântica de Múltiplas Ontologias

Florianópolis - SC

2006

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Alexandre Parra Carneiro da Silva

Mecanismo de Matching Semântico de Recursos Computacionais de Grids baseado na Integração Semântica de Múltiplas Ontologias

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

Orientador:

Prof. Mário Antônio Ribeiro Dantas, Dr.

Florianópolis, Setembro de 2006

Mecanismo de Matching Semântico de Recursos Computacionais de Grids baseado na Integração Semântica de Múltiplas Ontologias

Alexandre Parra Carneiro da Silva

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

Prof. Raul Sidnei Wazlawick, Dr.

Coordenador do Curso

Banca Examinadora

Prof. Mário Antônio Ribeiro Dantas, Dr. (Orientador)

Prof. Bruno Richard Schulze, Dr.

Prof. Fernando Álvaro Ostuni Gauthier, Dr.

Prof. Frank Augusto Siqueira, Dr.

Agradecimentos

Gostaria de agradecer primeiramente a Deus, por iluminar-me e abençoar-me sempre.

Aos meus pais, pelo amor, carinho, dedicação, presença em todos os momentos da minha vida, incentivos nos momentos difíceis e apoio incondicional em todos os meus projetos. A minha namorada Ana Flora pelo apoio, incentivo, compreensão e paciência durante o trabalho realizado e à sua família pelo apoio e incentivo.

Ao meu orientador, Prof. Mário Dantas, pela sua orientação, dedicação e disponibilidade, pela credibilidade depositada em mim, pelas conversas incentivadoras ao longo do trabalho realizado e por seus ensinamentos.

Aos colegas do LaPeSD: Vinícius, Alex, Guilherme, Jeferson, Denise pelos momentos de descontração, companherismo, conselhos, sugestões e momentos produtivos no LaPeSD.

À Natalya F. Noy, pesquisadora sênior que trabalha no grupo Protégé no departamento Stanford Medical Informatics da Escola de Medicina da Universidade de Stanford, pelos emails trocados que ajudaram-me a sanar dúvidas importantes à respeito de mapeamentos de ontologias. Aos pesquisadores e desenvolvedores da HP: Ian Dickinson, Dave Reynolds e Chris Dollin, pelos esclarecimentos à respeito do *framework* de manipulação de ontologias Jena. Ao José Geraldo Rodrigues Campos Lopes, recém mestre pela UnB com pesquisa sobre união de ontologias, pelas sugestões e disponibilização das ontologias empregadas neste trabalho.

À Vera Sodré (Verinha) pela simpatia e prestatividade nos instantes de dúvidas.

À CAPES pelo apoio financeiro parcial.

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

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ABox : *Assertional Box*

API : *Application Program Interface*

CGO : *Core Grid Ontology*

CIM : *Common Information Model*

CORBA : *Common Object Request Broker Architecture*

CyCL : *Cyc Language*

DAML : *DARPA Agent Markup Language*

DIG : *DL Implementors Group*

DL : *Description Logic*

DMTF : *Distributed Management Task Force*

DOLCE : *Descriptive Ontology for Linguistic and Cognitive Engineering*

FAFNER : *Factoring via Network-Enabled Recursion*

GGF : *Global Grid Forum*

GIP : *Ganglia Information Provider*

GRAM : *Grid Resource Allocation and Management*

GT3 : *Globus Toolkit 3*

GT4 : *Globus Toolkit 4*

HIP : *Hawkeye Information Provider*

HP : *Hewlett-Packard*

HTML : *HyperText Markup Language*

IP : *Internet Protocol*

IRI : *Internationalized Resource Identifiers*

IST : *Information Sciences and Technology*

I-WAY : *Information Wide Area Year*

LDAP : *Lightweight Directory Access Protocol*

LISP : *List Processing*

MSD : *Monitoring and Discovery System*

OASIS : *Organization for the Advancement of Structured Information Standards*

OCML : *Operational Conceptual Modelling Language*

OFR : Ontologia Funcional de Reputação

OGSA : *Open Grid Specification Architecture*

OGSI : *Open Grid Service Infrastructure*

OIL : *Ontology Inference Layer*

OKBC : *Open Knowledge Base Connectivity*

OMM : *Ontology-based Matchmaker*

OO : *Oriented-Object*

OR : Ontologia de Referência

OV : Organização Virtual

OWL : *Web Ontology Language*

PBS : *Portable Batch System*

QOM : *Quick Ontology Mapping*

QoS : *Quality of Service*

RAM : *Random Access Memory*

RDF : *Resource Description Framework*

RDF(S) : *Resource Description Framework + Schema*

RDQL : *RDF Data Query Language*

RGJ : Raciocinador Genérico Jena

SD : Sistema Distribuído

SDK : *Software Development Kit*

SHOE : *Simple HTML Ontology Extensions*

SMA : Sistema Multi-Agentes

SOA : *Service Oriented Architecture*

SQL : *Structured Query Language*

SRI : *Stanford Research Institute*

SUMO : *Suggested Upper Merged Ontology*

TBox : *Terminological Box*

TI : Tecnologia da Informação

UML : *Unified Modeling Language*

URI : *Uniform Resource Identifier*

URL : *Uniform Resource Location*

VO : *Virtual Organization*

W3C : *World Wide Web Consortium*

WSDL : *Web Services Description Language*

WSRF : *Web Services Resource Framework*

WWW : *World Wide Web*

XML : *Extensible Markup Language*

XOL : *XML Based Ontology Language*

Resumo

O paradigma de grid computacional tem como uma das suas principais características o compartilhamento de recursos heterogêneos dispersos geograficamente por diversas organizações virtuais. No entanto, o processo de *matching* destes recursos torna-se na prática difícil, uma vez que estas organizações podem apresentar visões distintas quanto a forma de descrever seus recursos compartilhados.

A utilização do paradigma de ontologias tem sido considerada, visando à descrição formal de distintas visões de recursos de grid. Em outras palavras, esta metodologia tem como objetivo prover uma base à realização de *matching* semântico. Vários mecanismos de *matching* semântico propostos focam na completa automatização do processo. Por outro lado, inúmeras pesquisas indicam que a completa automação no processo de integração semântica de múltiplas ontologias não consegue capturar ou expressar formalmente todas as relações semânticas possíveis entre as ontologias, sendo necessário à interação humana. Em adição, as consultas realizadas sobre ontologias requerem que o usuário conheça as diferentes sintaxes das linguagens de consulta disponíveis e sobretudo saiba traduzir a estrutura do conhecimento modelado nas ontologias em um formalismo lógico para poder elaborar as consultas. Desta forma, diminui-se o número de usuários aptos a formular consultas efetivas e significativas.

Com o objetivo de superar estas dificuldades, essa dissertação propõe um mecanismo de *matching* semântico de recursos de grid baseado na integração semântica de múltiplas ontologias. A integração é alcançada através do desenvolvimento de uma ontologia que tem o papel de servir de referência aos desenvolvedores das ontologias de recursos das diversas organizações virtuais. Desta forma, os desenvolvedores podem estabelecer relações de equivalência com a ontologia de referência antes de publicar as informações dos seus recursos. Uma outra contribuição deste trabalho é diminuir o grau de dificuldade na interação de usuários comuns com o sistema de *matching* proposto. Esta característica é alcançada criando uma ontologia de pedidos como uma linguagem de consulta de alto nível e um matchmaker baseado nesta linguagem. Esta ontologia foi construída baseada nos termos presentes na ontologia de referência, permitindo que o *matching* semântico seja realizado sobre todas as ontologias conhecidas pelo sistema.

Os experimentos dos estudos de caso realizados indicam que a proposta alcançou com sucesso seus principais objetivos.

Palavras-chave: Grids Computacionais, Ontologia, *Matching* Semântico, Alinhamento de Ontologias, Integração Semântica.

Abstract

The grid computing paradigm has the shared heterogeneous resources approach as the main characteristic for disperse geographically virtual organizations. However, the matching process of these resources becomes in practical difficult. The organizations can present different views on how to describe these shared resources.

To consider these different views have been used recently ontologies to describe formally grid resources as base for executing semantic matching. Several semantic matching mechanisms proposed focus in the complete automatization of the process. However, various researches indicate that the complete automation in the semantic integration process of multiple ontologies does not get capturing or expressing formally all the possible semantic relations between the ontologies, being necessary the interaction human. In addition, the queries realized on ontologies require that the user knows the different syntaxes of the available query languages and especially as to translate the structure of the knowledge modelled in the ontologies in a formalism logical to be able to elaborate the queries. In this way, it reduces the number of apt users to formulate effective and significative queries.

With the objective to overcome these difficulties, this dissertation considers a semantic matching mechanism of grid resources based on the semantic integration of multiple ontologies. The integration is reached through the development of a ontology that has the role to serve of reference so that the grid resources ontologies developers of the diverse virtual organizations can establish equivalence relations with the reference ontology before publishing the information of its resources. One another contribution of this work is to reduce the degree of difficulty in the interaction of common users with the matching system proposed. This characteristic is reached creating a request ontology as a query language of high level and a matchmaker based on this language. This ontology was constructed using terms present in the reference ontology allowing that semantic matching be executed on all the ontologies known for the system.

The experiments of the case studies realized indicate that the proposal reached successfully its main objectives.

Keywords: Grid Computing, Ontology, Semantic Matching, Ontology Alignment, Semantic Integration.

1 Introdução

Uma configuração de grid computacional é uma plataforma para computação geograficamente distribuída, onde os usuários fazem acesso ao ambiente através de uma única interface. Esta interface tem como objetivo fornecer uma visão simplificada para o usuário a respeito do ambiente (FOSTER; KESSELMAN, 2003). Grids computacionais são bastante heterogêneos, tanto no nível de *hardware* quanto de *software*. Esta heterogeneidade ocorre em boa parte pelo fato dos grids serem constituídos por diversas organizações que compartilham os seus mais diversos recursos com as demais organizações no ambiente de grid.

As organizações que integram um grid são conhecidas como *organizações virtuais*, as quais compartilham seus recursos sob políticas próprias (DANTAS, 2005). Como exemplos de organizações, podemos citar: empresas, centros de pesquisa e universidades que compartilham armazenamento de dados, poder de processamento, uso de equipamentos especiais (telescópios eletrônicos) e aplicações (pacotes de *software* de simulação que podem executar com dados fornecidos pelo próprio usuário).

A autonomia das organizações sobre seus recursos pode acarretar diferentes maneiras de como descrever as suas características, dificultando o processo de adequação de recursos baseado em requisitos da aplicação, processo este conhecido na literatura por *resource matching*. Como mencionado em (CZAJKOWSKI et al., 2002), um usuário deveria ser capaz de escolher os recursos mais adequados dentro de uma configuração de grid de acordo com os requisitos da aplicação. Este fato torna importante que o processo de *resource matching* considere o significado das diferentes maneiras de como são descritas as características dos recursos disponibilizados pelos ambientes de grid ao invés da sintaxe dos termos que descrevem essas características.

1.1 Motivação

O emprego de ontologias para a descrição semântica de recursos computacionais como base para realizar *matching* semântico tem crescido nos últimos anos. No entanto, não existe uma ontologia única que represente ambientes de grid (LOPES et al., 2006; BROOKE et al., 2004). Uma ontologia única e consensual para a descrição de recursos em um ambiente de grid é muito difícil de ser definida e provavelmente não haverá esta ontologia, devido à grande diversidade de visões que as organizações podem apresentar sobre seus recursos (LOPES, 2005). Para encontrar similaridades entre diferentes ontologias é preciso integrá-las semanticamente. Pesquisas (FREITAS; STUCKENSCHMIDT; NOY, 2005; NOY, 2004; KALFOGLOU; SCHORLEMME, 2003) apontam que os métodos empregados para automatizar o processo de integração de ontologias não conseguem identificar todas ou a maioria das similaridades entre as diferentes ontologias, necessitando de interação humana. Destaca-se também o fato de que não é de se esperar que qualquer usuário lide facilmente com o formalismo empregado para representar o conhecimento, como é o caso de ontologias. Somente usuários especialistas têm conhecimento necessário e suficiente para interagir com sistemas de conhecimento, como, por exemplo, relacionando ou consultando os conceitos (KALFOGLOU; SCHORLEMME, 2003).

1.2 Objetivos

Pelos motivos expostos na seção anterior, esta dissertação tem como objetivo propor e avaliar um mecanismo de *matching* semântico de recursos de grid baseado na integração semântica de múltiplas ontologias. Esta integração é alcançada desenvolvendo-se uma ontologia de referência que tem como objetivo servir de base para que os desenvolvedores das diferentes ontologias de recursos das organizações virtuais possam estabelecer relações de equivalência semântica. Esta abordagem dentre as demais pesquisadas na literatura apresentou-se a mais vantajosa.

O outro objetivo deste trabalho é diminuir o grau de dificuldade dos usuários comuns ao consultarem recursos de grid modelados em diferentes ontologias escritas na linguagem OWL.

Esta característica é alcançada desenvolvendo uma ontologia de pedidos como uma linguagem de consulta de alto nível e um motor que a interprete. Além disso, uma linguagem de consulta como uma ontologia possibilita o emprego de regras de inferência com o intuito de ampliar as consultas semanticamente e de verificar as suas consistências. Esta ontologia foi construída baseada nos termos definidos na própria ontologia de referência desenvolvida, de forma que o processo de *matching* seja realizado sobre todas as ontologias publicadas no sistema. Cada termo definido na ontologia de pedidos possui metadados associados que facilitam o usuário a compreendê-lo, auxiliando-o na criação de consultas efetivas e significativas. Como contribuição secundária foram desenvolvidas interfaces gráficas que permitem criar facilmente as consultas e a visualizar o resultado da pesquisa no sistema.

1.3 Estrutura do documento

A estrutura desta dissertação está dividida em duas partes. A primeira parte apresenta uma revisão bibliográfica abordando os principais tópicos relacionados a ambientes de grid e a integração semântica de ontologias. A segunda parte apresenta o sistema de *matching* semântico desenvolvido e a avaliação deste sistema através de várias consultas submetidas às diversas ontologias.

Estas duas partes estão organizadas nos capítulos conforme descritos a seguir:

- O capítulo 2 apresenta as principais características de grids computacionais. Adicionalmente, é apresentada a sua arquitetura, o principal *middleware* de construção de ambientes de grid e a evolução dos grids computacionais;
- No capítulo 3, são abordados tópicos referentes às ontologias. Os principais tópicos são: definição do que é ontologia, tipo de ontologias, como projetar ontologias, as principais linguagens de construção de ontologias baseadas em tecnologias Web Semânticas, linguagens de consulta de ontologias baseadas em RDF, a API Jena que possibilita a manipulação de ontologias e por fim diferentes abordagens de integração semântica pesquisadas

na literatura;

- Os principais trabalhos de pesquisa na área e relacionados com este trabalho são descritos de forma resumida no capítulo 4;
- No capítulo 5, o sistema proposto é apresentado em detalhes. Neste capítulo são descritos as principais características do sistema, a arquitetura desenvolvida e o protótipo implementado;
- O capítulo 6 apresenta o ambiente experimental criado para demonstrar o comportamento do sistema proposto através de testes empíricos;
- Finalmente, no capítulo 7 são apresentadas as conclusões e propostas de trabalhos futuros.

2 Grids Computacionais

Ambientes de grids computacionais tiveram sua origem em meados dos anos 90, tendo sido idealizados com o objetivo inicial de obter poder computacional maior do que os fornecidos pelos supercomputadores da época, contudo, com uma relação custo benefício mais atraente. É possível alcançar esta relação pois os grids permitem que computadores comuns e até mesmo clusters e supercomputadores com diversas configurações sejam compartilhados.

A palavra grid tem sua origem no termo *electrical power grid* que denota uma rede de energia elétrica (geração, transmissão e distribuição) (FOSTER; KESSELMAN, 2003). Esta rede é a infra-estrutura que possibilita o uso da energia elétrica (o recurso, neste caso) de forma consistente, generalizada, barata e confiável.

A comparação de um ambiente de grid com uma rede elétrica vem do fato que nesta não se precisa importar com a fonte geradora de energia, o que ocorre igualmente em um ambiente de grid. Portanto, não devemos nos preocupar com a origem dos recursos provenientes do grid e sim se eles estão disponíveis para serem usados pelas aplicações.

O grid computacional pode ser definido então como uma infra-estrutura de *hardware* e *software* que permite, de forma consistente, generalizada, barata e confiável, o acesso a recursos computacionais de alto desempenho.

O papel principal do grid é o de reunir recursos, em grande escala, para atingir determinado fim. Estes recursos podem ser da ordem de centenas, milhares ou até de milhões, dedicados ou não, pertencentes a vários domínios administrativos, geograficamente distribuídos e ligados por uma rede de interconexão que apresenta características bem diversas das normalmente em-

pregadas em clusters, por exemplo. Por isto, fala-se em infra-estrutura de *hardware* e *software*. A primeira para se estabelecer interconexões necessárias para acesso aos recursos e a segunda para monitorar e controlar os mesmos.

Esta infra-estrutura deve permitir o acesso consistente aos recursos, através de serviços padronizados, com interfaces e parâmetros muito bem definidos. O maior desafio é encontrar uma forma de tornar a heterogeneidade dos recursos transparente para os usuários e administradores sem comprometer o alto desempenho. Sem esta padronização fica muito difícil o desenvolvimento de aplicações.

Em (FOSTER; KESSELMAN; TUECKE, 2001) foi descrito um dos problemas reais e específicos que está por trás do conceito de grids. Este problema está relacionado com o compartilhamento e a coordenação de recursos entre diferentes organizações, também conhecidos como organizações virtuais (OVs) (FOSTER; KESSELMAN, 1999; FOSTER; KESSELMAN; TUECKE, 2001). Uma organização virtual é um conjunto de indivíduos, recursos e/ou instituições que definem regras de compartilhamento entre si. O processo de compartilhamento e coordenação não compreende somente arquivos, mas principalmente o acesso a computador, pacotes de *software*, serviços e dispositivos especiais. Estes recursos podem ter diferentes políticas de acesso e de publicação de suas características entre diferentes organizações que compõem o grid. Adicionalmente, usuários e aplicações grid podem requerer diferentes recursos para resolver seus problemas com alto desempenho.

2.1 Características do Ambiente de Grid

Grid computacional de acordo com Buyya (BUYYA, 2002) é um tipo de sistema paralelo e distribuído que permite compartilhamento, seleção e agregação dinâmica em tempo de execução de recursos autônomos geograficamente distribuídos.

Embora existam diversas interpretações do que é um grid computacional, há características semelhantes a respeito dos ambientes de grid (FOSTER; KESSELMAN; TUECKE, 2001):

- São grandes, tanto em termo do número de recursos potencialmente disponíveis quanto em distância geográfica entre as organizações que as constituem;
- São geograficamente distribuídos, portanto a latência envolvida na movimentação dos dados entre os seus recursos é considerável podendo ser fator predominante nas aplicações;
- São heterogêneos, ou seja, a estrutura de *hardware* e de *software* que compõe os recursos e serviços disponibilizados pelas organizações virtuais difere de diversas maneiras;
- Ultrapassam as fronteiras de uma organização humana, fazendo com que políticas de descrição e de acesso dos recursos sejam diferentes entre as organizações que compõem o grid.

Em (SKILLICORN, 2002), são citados quatro tipos de ambientes de grid:

- **Ambientes de grid computacional**: representam a extensão natural de grandes sistemas distribuídos e paralelos. Eles existem para fornecer alto desempenho computacional. São formados por um conjunto de computadores disponíveis que são acessados por usuários através de uma única interface com o objetivo de submeterem aplicações que necessitam de mais de um computador para executar;
- **Grids de acesso**: O enfoque deste tipo de grid é a construção de um ambiente virtual em que usuários, pertencentes a diferentes organizações, possam interagir como se estivessem usando uma única plataforma de hardware dedicada. O desempenho neste tipo de ambiente não é tão importante quanto nos ambientes de grid computacional (FOSTER; KESSELMAN; TUECKE, 2001);
- **Grids de Dados**: Tem como característica admitir que enormes volumes de dados sejam armazenados em repositórios e movimentados com a mesma facilidade que os pequenos arquivos;
- **Grids de centros de dados**: Este tipo de ambiente torna possível acessar e realizar computação sobre enormes repositórios de dados distribuídos que por qualquer motivo não

podem ser coletados em um único local (SKILLICORN, 2001). Este grid é importante para aplicações como *data mining* distribuído.

Os quatro tipos de ambientes de grid citados anteriormente convergem para funcionalidades semelhantes, tais como: descoberta de recurso, plano de execução, autenticação e segurança, e heterogeneidade de servidores e de formatos de dados. No entanto, diferem de acordo com a ênfase atribuída a cada uma destas funções.

2.2 Arquitetura

Como citado, o conceito de grid veio do problema real e específico de compartilhar recursos heterogêneos entre as várias organizações que a compõe. Para conseguir compartilhar os recursos de grid é necessário um grande controle pelos fornecedores de recursos e seus consumidores, ambos definindo clara e cuidadosamente quais recursos devem ser compartilhados, a quem é permitido o compartilhamento e as condições que o compartilhamento deve ocorrer (FOSTER; KESSELMAN; TUECKE, 2001).

Baseado nesse desafio e na forma como os processos são executados nos grids pode-se visualizar a arquitetura de um ambiente de grid como uma arquitetura orientada a serviços (SOA). SOA expressa um conceito arquitetural que define o uso de serviços para suportar exigências de usuários de software (WIKIPEDIA, 2006). Em um ambiente SOA, os nodos em uma rede tornam os recursos disponíveis a outros participantes na rede como serviços independentes que os participantes acessam de forma padronizada. Diferentemente das arquiteturas tradicionais ponto a ponto, as SOAs compreendem serviços de aplicação fracamente acoplados e altamente interoperáveis. A definição de sua interface encapsula a implementação do vendedor de equipamento e da linguagem específica utilizada. Uma SOA é independente de tecnologia de desenvolvimento e os seus componentes de software tornam-se reusáveis quando a sua interface é baseada em padrões. Uma arquitetura orientada a serviço é um dos dois diferentes estilos de construção de Sistemas Distribuídos (SD) (PROJECT; TOOLKIT, 2006).

Na SOA é importante a princípio que os provedores de serviços e recursos definam o con-

junto de termos e condições que devam ser satisfeitos para possibilitar a obtenção do acesso. Posteriomente, é necessária a realização de um contrato de serviço entre o fornecedor e o consumidor, onde irão constar os termos e condições sobre os quais o fornecedor concorda em prover o serviço ao consumidor (ROURE; JENNINGS; SHADBOLT, 2003). O contrato não especifica somente condições para os fornecedores do serviço, mas também para os consumidores.

Para conseguir realizar as tarefas acima citadas, as operações dentro da OV e entre OVs, que integram o ambiente de grid, devem ser capazes de estabelecer compartilhamento de relacionamentos entre quaisquer participantes potenciais. Portanto, identifica-se que a interoperabilidade é uma característica importante neste ambiente. Em uma rede, a interoperabilidade é alcançada através de protocolos comuns, cujas definições especificam como os elementos que compõem o SD irão interagir entre si, de forma a apresentarem um comportamento esperado, e também determinam a estrutura da informação trocada durante as interações (FOSTER; KESSELMAN; TUECKE, 2001).

Objetivando uma arquitetura extensível e de estrutura aberta, a partir da qual são fornecidas soluções para os requisitos chaves desempenhados pela OV, foi apresentado em (FOSTER; KESSELMAN; TUECKE, 2001) uma comparação funcional entre a arquitetura para ambientes de grid e a arquitetura de protocolos da Internet, como apresentada na Figura 1. A equivalência entre as duas arquiteturas é interessante pois auxilia o entendimento funcional das camadas do grid e também demonstra a preocupação de uma padronização (DANTAS, 2005).

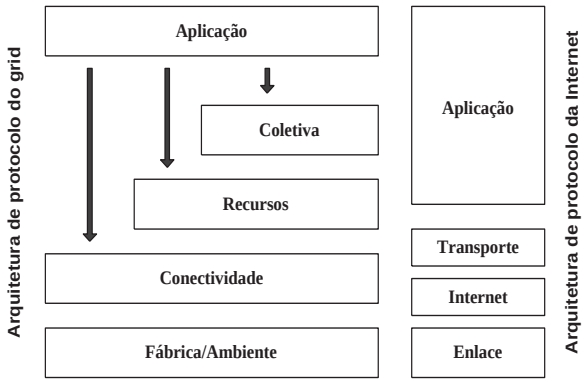


Figura 1: Comparação funcional entre a arquitetura do grid e da Internet

A seguir são apresentadas as características das camadas da arquitetura de grid (DANTAS,

2005):

- **Fábrica/Ambiente**: os componentes dessa camada implementam operações locais específicas que ocorrem em cada recurso como resultado do compartilhamento nas camadas superiores. Dois tipos de mecanismos são implementados: mecanismos de negociação e mecanismos de gerenciamento. Os mecanismos de negociação fazem solicitações para obter informações sobre a estrutura, o estado e as possibilidades dos recursos. Por outro lado, os mecanismos de gerenciamento fornecem formas de monitorar a qualidade de serviço, QoS (Quality of Service);

- **Conectividade**: esta camada tem a função de definir os protocolos básicos para transações de rede específicas do grid, ou seja, protocolos de comunicação e autenticação. Os protocolos de comunicação permitem a troca de dados entre os recursos existentes na camada fábrica/ambiente, enquanto os protocolos de autenticação constroem os serviços de comunicação, provendo mecanismos criptográficos para verificação da identidade de usuários e recursos;

- **Recursos**: a camada define protocolos e APIs (Application Program Interface) que forneçam segurança na negociação, iniciação, monitoramento, controle, geração de relatórios e outros detalhes envolvidos nas operações com recursos individuais. A definição desses protocolos e APIs é baseada nos protocolos de comunicação e autenticação encontrados na camada inferior. As implementações dos protocolos desta camada chamam as funções da camada ambiente para acessar e controlar os recursos locais;

- **Coletiva**: diferente da camada recursos que se preocupam somente com recursos individualmente, os componentes desta camada atuam nas interações entre coleções de recursos. Os componentes dessa camada baseiam-se nas camadas recursos e aplicação implementando uma enorme variedade de serviços, tais como:

  - Serviços de diretório, permitindo aos membros da organização virtual descobrir quais são os recursos desta organização e/ou as suas propriedades;

  - Servidores de autorização comunitários que reforçam a política de acesso aos recursos locais;
  - Serviços de monitoramento e diagnósticos, que são responsáveis por monitorar recursos da OV para identificar falhas, detectar intrusões e sobrecargas;
  - Serviços de co-alocação e escalonamento permitem que participantes da OV requeiram alocação de um ou mais recursos para um determinado propósito e o escalonamento de tarefas para recursos apropriados.

- **Aplicação**: esta camada compreende as aplicações dos usuários que são executados no ambiente da OV. Para a correta execução dessas aplicações, as camadas inferiores provêem serviços essenciais para esta camada.

2.3 Globus Toolkit

Como citado neste capítulo, para tornar uma configuração de grid efetivamente operável há necessidade de uma arquitetura extensível e baseada em padrões abertos. Com esta arquitetura, pretende-se alcançar as seguintes facilidades: interoperabilidade, portabilidade, extensibilidade e compartilhamento de código. Protocolos padrões facilitam a definição de serviços que provêem melhores capacidades. Outro fator que diminui a dificuldade durante o desenvolvimento de um serviço de grid é a construção de APIs e SDKs (Software Development Kits), pois permitem abstrações do ambiente de grid. Desta forma, uma parcela considerável da complexidade envolvida no desenvolvimento de serviços de grid é retirada dos desenvolvedores.

Em conjunto - arquitetura extensível e baseada em padrões abertos, APIs e SDKs - constituem o que geralmente é chamado de *middleware*. O *middleware* fornece serviços necessários para suportar um conjunto comum de aplicações em um ambiente de rede distribuído (AIKEN et al., 2000).

Há diversos *middlewares* para construção de grids atualmente, sendo o *middleware* Globus considerado por muitos - pesquisadores, centros de pesquisa e a indústria - o pacote mais com-

pleto e um padrão de fato para construção de grids. O *middleware* Globus é fruto do projeto Globus, tendo sido desenvolvido inicialmente pelos pesquisadores Ian Foster e Carl Kesselmann que propuseram um conjunto de serviços de modo que fossem utilizados pelas aplicações sem a necessidade de adaptação a um modelo particular de programação (FOSTER; KESSELMAN, 1998).

O projeto Globus teve início a partir de estudos realizados a respeito de problemas apresentados em um projeto anterior, chamado I-WAY (Information Wide Area Year) (FOSTER et al., 1997), conhecido também na época como a Internet do Futuro (DANTAS, 2005). Para viabilizar a construção de grids computacionais, o projeto desenvolveu um conjunto de serviços básicos e bibliotecas de *software* para construção de ferramentas e aplicações voltadas a ambientes de grid, denominado de Globus Toolkit.

Atualmente o Globus Toolkit é desenvolvido pelo *Globus Alliance*, que é um fórum para o desenvolvimento de tecnologias fundamentais necessárias para a construção de grids computacionais. O *Globus Alliance* é formado por uma comunidade de organizações e indivíduos que desenvolvem tecnologias fundamentais para grids. Estas tecnologias possibilitam às pessoas compartilharem poder computacional, bases de dados e outras ferramentas com segurança através de limites corporativos, institucionais e geográficos sem sacrificar a autonomia local (GLOBUS, 2006).

O Globus Toolkit, a partir da sua terceira versão, conhecida como Globus Toolkit 3 (GT3), vem desenvolvendo aplicações e infra-estruturas para computação distribuída de acordo com a perspectiva da arquitetura de software SOA. Nesta versão, os serviços Globus vem sendo definidos como *Grid Services*, em conformidade com as especificações OGSA (Open Grid Specification Architecture). OGSA foi desenvolvido pelo GGF (Global Grid Forum) com o intuito de definir uma arquitetura padrão e aberta para aplicações baseadas em grid. A meta do OGSA é padronizar praticamente todos os serviços que são encontrados em uma aplicação de grid, através da especificação de um conjunto de interfaces para estes serviços.

No entanto, o grupo OGSA percebeu a necessidade de um *middleware* distribuído para

conseguir criar essa nova arquitetura sobre a qual se apoiaria. Apesar de existirem vários *middlewares* distribuídos (por exemplo: CORBA, RMI e RPC, etc) o grupo optou pela arquitetura *Web Service* por esta apresentar melhores opções em relação às demais, sendo as características fracamente acoplado e interoperabilidade os fatores chaves para grids computacionais. Contudo, esta arquitetura ainda tem diversos incovenientes os quais a fazem inadequada para as necessidades OGSA (SOTOMAYOR, 2003). OGSA superou estes obstáculos através da definição de um tipo estendido do *Web Service* chamado *Grid Service*. Um *Grid Service* é simplesmente um *Web Service* com extensões que o fazem adequado para a construção de aplicações de grid conforme as necessidades do OGSA.

Por outro lado, o OGSA sozinho não detalha a descrição dos *Grid Services*. Ele basicamente esboça o que um *Grid Service* deveria ter, porém nada mais. Isto acontece, pois OGSA traz outro padrão conhecido como OGSI (Open Grid Service Infrastructure), também desenvolvido pelo GGF. A função do OGSI é especificar formal e tecnicamente o que é um *Grid Service*. No diagrama da Figura 2, ilustra-se como os *Grid Services* estão relacionados com OGSA, OGSI, *Web Services* e GT3 (SOTOMAYOR, 2003).

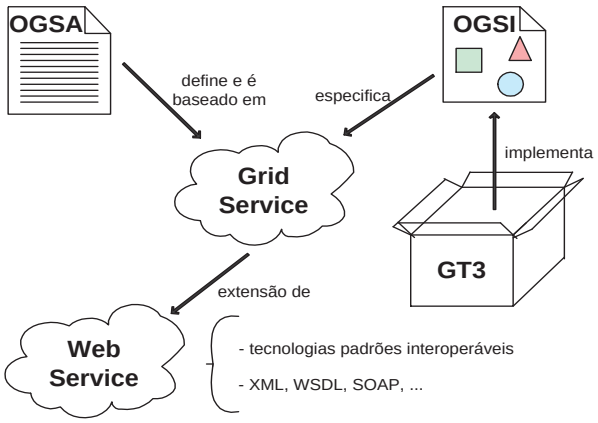


Figura 2: Relacionamentos dos conceitos sobre *Grid Services*

Como mostrado na Figura 2, o *Grid Service* é baseado no OGSA, que por sua vez é especificado pelo OGSI que é implementado pelo GT3. O *Grid Service* é uma extensão da tecnologia *Web Service*, pois a arquitetura *Web Service* contém certas limitações, sendo a principal delas o seu caráter *stateless*, ou seja, seus serviços não retém dados entre invocações. Com esta ca-

racterística, *Web Services* simples (como especificado pelo W3C) não cobririam a maioria das aplicações de grid. Através da infra-estrutura OGSI conseguiu-se aperfeiçoamentos a respeito de *Web Services*, sendo o principal deles a retenção de dados entre invocações. *Web Services* que apresentam esta característica são conhecidos como *Web Services stateful*.

No entanto, o *Grid Service* é especificado pelo OGSI com a esperança de que eventualmente convergisse em *Web Services* e que, de fato, *Web Services* e *Grid Services* tornassem a mesma coisa. Apesar das melhorias trazidas pelo OGSI, ele apresenta diversos inconvenientes que dificultam a convergência para *Web Services*, são eles (SOTOMAYOR, 2003):

- OGSI não trabalha bem com ferramentas *Web Services* atuais;
- OGSI é muito orientado a objetos. Apesar de muitos sistemas *Web Services* terem implementações orientadas a objetos, os próprios *Web Services* não são necessariamente orientados a objetos;
- Apesar de praticamente todas as especificações serem longas e densas, a especificação do OGSI é demasiadamente longa e densa.

2.3.1 Globus Toolkit 4

Para resolver os problemas do OGSI e convergir de vez para *Web Services*, um novo padrão foi apresentado em Janeiro de 2004 para substituir o OGSI, o Web Services Resource Framework (WSRF). WSRF é uma especificação desenvolvida pelo Organization for the Advancement of Structured Information Standards (OASIS) (OPEN, 2006). WSRF especifica como tornar *Web Services stateful* junto com outras características interessantes para ambientes de grid. Este novo padrão é incorporado na versão atual do Globus Toolkit, versão 4, conhecida como GT4. Portanto, neste novo cenário, o OGSA é baseado diretamente sobre *Web Services* através do WSRF ao invés de ser baseado no OGSI. O diagrama da Figura 3 mostra a mudança arquitetural ocorrida do GT3 para o GT4.

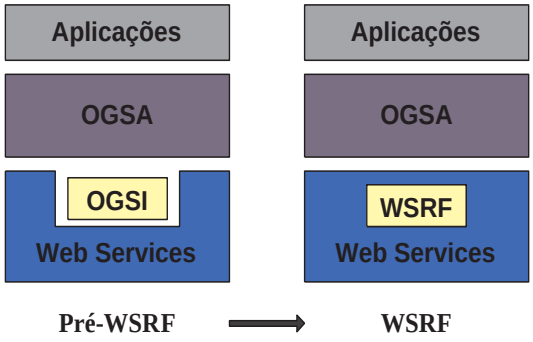


Figura 3: Mudança arquitetural ocorrida entre as versões 3 e 4 do Globus Toolkit

Com esse novo padrão, as inconveniências decorrentes do OGSI são superadas da seguinte forma:

- A especificação WSRF em relação à OGSI é menor e menos densa, sendo composta por 5 documentos;
- O WSRF está em maior conformidade com as objeções propostas pela comunidade *Web Services*, permitindo facilidades no desenvolvimento de ferramentas *Web Services* atuais;
- O WSRF claramente separa o serviço do seu estado, visto que *Web Services* puros não podem ter estado.

O GT4 é composto por diversos componentes de *software*. Estes componentes são divididos em cinco categorias: Segurança, Gerenciamento de Dados, Gerenciamento de Execução, Serviços de Informação e Tempo de Execução Comum. Nesta dissertação será comentado apenas o componente Serviços de Informação, pois ele tem a função de fornecer informações sobre os recursos e serviços existentes em um ambiente de grid.

O Sistema de Informação do GT4 é conhecido como *Monitoring and Discovery System* (MDS) que é um conjunto de *Web Services* para monitorar e descobrir recursos e serviços no grid. Os serviços do MDS fornecem interfaces de consulta e de publicação das informações detalhadas dos recursos. Além disso, há a interface *Trigger* que pode ser configurada para reagir quando condições pré-estabelecidas ocorrem.

O MDS do GT4 (MDS4) é constituído pelo Web Service MDS (WS MDS) e o Pré Web Service MDS (Pré-WS MDS). O WS MDS é baseado no padrão WSRF enquanto o Pré-WS MDS baseia-se no LDAP. A seguir são apresentados os serviços baseados no WS MDS do GT4 (TOOLKIT, 2006):

- ***Index Service***: serviço responsável pela coleta de informações de recursos de grid e a sua publicação.
- ***Trigger Service***: serviço cuja função é coletar informações dos recursos do grid e permitir que o administrador do ambiente defina ações e regras baseado nessas informações. Caso as regras sejam atendidas as ações correspondentes podem ser executadas.
- ***Aggregator Framework***: é um *framework de software* usado para construir serviços que coletam e agregam informações. Os serviços que são construídos por este *framework*, tais como *Index* e *Trigger*, são muitas vezes conhecidos como serviços agregadores.

Os serviços agregadores têm as seguintes características em comum:

- Eles coletam informações via *Aggregator Sources*. Estas fontes são classes Java que implementam uma interface - definida como parte do *Aggregator Framework* - para coletar dados XML (eXtended Markup Language) formatados.
- Usam um mecanismo comum de configuração para manter a informação sobre como usar o *Aggregator Source* e seus parâmetros associados (que geralmente especificam quais dados obter e de que local). O WSDL (Web Service Definition Language) do *Aggregator Framework* define um tipo de entrada de grupo de serviço agregador que controla informações e dados de configuração. Programas de clientes administrativos usam mecanismos de registro de grupo de serviço WSRF padrão para registrar estas entradas de grupos de serviço para o serviço agregador.
- A propriedade "auto limpante", ou seja, cada registro tem um tempo de vida. Se um registro expirar sem ter sido atualizado, ele e seus dados associados serão removidos do servidor.

2.3.1.1 Fornecedores de Informação no GT4

Os ambientes de grids desenvolvidos pelo GT4 têm as suas informações coletadas através de fornecedores ou fontes de informações. Estas fontes são serviços implementados pelo componente MDS4 como *Aggregator Source*. MDS4 inclui as três seguintes fontes de informação:

- ***Query Aggregator Source***: esta fonte consulta informações presentes em propriedades do recurso de um serviço WSRF.
- ***Subscription Aggregator Source***: responsável em coletar as informações de um serviço WSRF via uma subscrição ou notificação WSRF.
- ***Execution Aggregator Source***: tem a função de executar um programa externo fornecido pelo administrador para coletar informações.

O componente MDS4 atualmente fornece algumas fontes de informação, como por exemplo, *Hawkeye Information Provider* (HIP) e o Ganglia Information Provider (GIP). Ambos coletam as mesmas informações, porém de ferramentas distintas. O HIP coleta informações de recursos computacionais gerenciados pelo escalonador Condor (THAIN; TANNENBAUM; LIVNY, 2005) e o GIP de recursos que executam o sistema de monitoramento distribuído Ganglia (MASSIE; CHUN; CULLER, 2004). Outra importante fonte de informação disponível no MDS4 é o WS GRAM. Este serviço WSRF publica informações sobre o escalonador local, tais como: informações sobre a fila, número de processadores disponíveis e livres, quantidade de *jobs*, algumas estatísticas sobre memória, etc.

Dependendo da implementação, um *Aggregator Source* pode usar um componente de software externo (por exemplo, um programa executável), ou mesmo um serviço WSRF pode usar um componente externo para criar e atualizar suas propriedades de recurso. Estas propriedades podem posteriormente ser registradas em um *Index Service* ou outro serviço agregador.

2.4 Evolução do Grid Computacional

Os grids computacionais, desde o seu surgimento, têm tido um processo contínuo de evolução. Contudo, as suas gerações não são rigidamente definidas. Em (ROURE et al., 2003), os autores comentam que a melhor forma de distinguir as gerações é pela sua filosofia ao invés das tecnologias empregadas.

O paradigma de grid computacional foi primeiramente concebido para conectar centros de supercomputação. Nos últimos dez anos, vários ambientes de projetos de grid computacional foram criados. O principal objetivo dos primeiros projetos era fornecer recursos para aplicações que necessitavam de alto desempenho; exemplos são I-WAY (FOSTER et al., 1997) e FAFNER (RSA Factoring-By-Web Project, 2006).

Atualmente, a infra-estrutura grid tem uma meta mais global do que somente conectar poucos números de centros para promover supercomputação. A disponibilidade de tecnologias de rede de alta largura de banda e dispositivos de redes permitiu uma configuração real em escala global para fornecer recursos distribuídos a várias classes diferentes de aplicações (ROURE et al., 2003).

A meta do grid computacional recentemente mudou da solução dedicada para computação de alto desempenho para uma abordagem mais ampla de compartilhamento de recursos (GOBLE; ROURE, 2002). Em outras palavras, a nova abordagem tem o objetivo de cuidar dos aspectos relacionados ao compartilhamento tornando-o coordenado, seguro e flexível entre os indivíduos, instituições e recursos. Esta nova visão enfatiza a importância da informação, essencial no processo de descoberta de recursos e na interoperabilidade entre recursos.

Resumidamente, as três gerações dos grids são (ROURE et al., 2003):

- Sistemas grids da primeira geração envolveram soluções proprietárias para compartilhamento de recursos de alto desempenho;
- Os sistemas da segunda geração introduziram *middlewares* para lidar com escalabilidade e heterogeneidade, focando em poder computacional de larga escala e grande volume de

dados;

- Os sistemas da terceira geração estão empregando uma abordagem orientada a serviço, ou seja, adotam uma visão mais holística da infra-estrutura *e-Science*, onde são permitidos metadados e exibem características independentes.

Nos dias atuais, o grid encontra-se na terceira geração dos sistemas de grid, dirigindo-se para o que denominam de Grid Semântico (ROURE et al., 2003). Esta geração utiliza-se de tecnologias de Web Semântica (W3C - World Wide Web Consortium, 2006; LEE; HENDLER; LASSILA, 2001) como infra-estrutura para aplicações de grid.

A terceira geração está preocupada quanto a forma como a informação é representada, armazenada, acessada, compartilhada e mantida. Informação é entendida como dados contendo significado.

A próxima geração de grids estará interessada com a forma como o conhecimento é adquirido, usado, alcançado, publicado e mantido para auxiliar *e-Scientists* a alcançarem suas metas e objetivos particulares (ROURE et al., 2003). Entende-se como conhecimento as informações aplicadas para alcançar uma meta, resolver um problema ou executar uma decisão.

A metáfora do grid intuitivamente leva-nos à visão da infra-estrutura *e-Science* como um conjunto de serviços que são fornecidos por instituições e indivíduos particulares para serem consumidos por outros (ROURE; JENNINGS; SHADBOLT, 2003). Considerando esta visão, tem-se tornado consenso caracterizar a infra-estrutura de computação como sendo formado pelas seguintes camadas conceituais (ROURE; JENNINGS; SHADBOLT, 2003):

- *Computação ou Dados*: Esta camada lida com a forma com que recursos computacionais são alocados, escalonados e executados e a forma com que os dados são enviados entre os vários recursos de processamento. A camada é caracterizada como sendo capaz de lidar com um grande volume de dados, redes de conexão rápidas e que apresentam diversos recursos como um único metacomputador. Nesta camada os dados são entendidos como bits e bytes não interpretados.

- *Informação*: Nesta camada a informação é tida como dados equipados com significado. Esta camada é responsável com a forma com que a informação é representada, armazenada, acessada, compartilhada e gerenciada. Exemplos de informações são: a caracterização de um inteiro como representando a temperatura de um processo de reação e a identificação de uma string como nome de um indivíduo.

- *Conhecimento*: Esta camada está interessada com a forma como o conhecimento é adquirido, usado, alcançado, publicado e mantido. Como exemplo de conhecimento, pode-se citar o reconhecimento por um operário de fábrica que em um determinado contexto uma reação de temperatura indica a finalização do processo.

Sobre a estrutura das camadas apresentadas na Figura 4 há duas considerações relevantes a serem comentadas. Primeiramente, todos os grids existentes ou que venham a ser construídos tem algum elemento das três camadas. O grau de importância de cada camada a um determinado grid depende do domínio de aplicação do grid. Portanto, em alguns casos, o processamento de grandes quantidades de dados será a preocupação dominante, enquanto em outros os serviços de conhecimento que estão disponíveis serão o assunto prioritário. Em segundo lugar, a visão orientada a serviço aplica-se em todas as camadas.

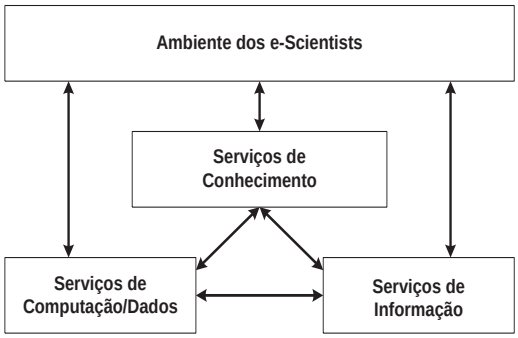


Figura 4: Visão *e-Science* do grid como serviços

Embora a visão seja amplamente aceita, a maioria dos trabalhos de pesquisa realizados na área tem concentrado-se nas camadas de computação/dados e na de informação (ROURE; JENNINGS; SHADBOLT, 2003). Este fato ocorre devido a vários problemas em aberto no gerenciamento de computações massivamente distribuídas de maneira eficiente e no acesso e com-

partilhamento de fontes heterogêneas de informação (ROURE et al., 2003). Todavia, acredita-se que o potencial total do grid pode somente ser concretizado pela completa exploração das funcionalidades e capacidades fornecidas pelos serviços da camada de conhecimento (ROURE et al., 2003).

3 *Ontologias*

Ontologias vêm tornando-se um tópico intensamente investigado por inúmeras comunidades de pesquisa da Inteligência Artificial, incluindo Engenharia do Conhecimento (DAVIES; FENSEL; HARMELEN, 2003), Processamento de Linguagem Natural (ANTONIO; CHANTAL, 2000) e Representação do Conhecimento (FENSEL, 2000). Contudo, o campo de atuação das ontologias vem difundindo-se, com destaque no Comércio Eletrônico (LEGER et al., 2000), Gerenciamento de Conhecimento (KALFOGLOU et al., 2001), Banco de Dados (CANNATARO; COMITO, 2003), Engenharia de Software (AMBROSIO et al., 2004), BioInformática (CANNATARO et al., 2004), dentre outros.

A razão de ontologias tornarem-se tão populares deve-se em grande parte devido ao que elas prometem: um conhecimento compartilhado e comum de domínios que podem ser comunicados entre pessoas e sistemas de informação. Em outras palavras, ontologias são desenvolvidas para fornecer um processamento a nível de máquina dos significados das informações que podem ser comunicadas entre diferentes agentes (*software* e humanos).

Nesta dissertação, o paradigma de ontologia focará no domínio dos ambientes de grids computacionais. Várias pesquisas estão sendo realizadas sobre o assunto, dentre as quais se destacam (BROOKE et al., 2004; ROURE; JENNINGS; SHADBOLT, 2003; LOPES et al., 2006; PERNAS; DANTAS, 2005; HEINE; HOVESTADT; KAO, 2004; CONGIUSTA et al., 2004; TANGMUNARUNKIT; DECKER; KESSELMAN, 2003).

Para melhor compreensão dos conceitos relativos às ontologias presentes ao longo deste capítulo, apresentamos um glossário simples que informa as definições informais que GUARINO & GIARETTA sugeriram como interpretações preferidas dos termos discutidos em (GUARINO;

GIARETTA, 1995). Os termos são:

- **Conceitualização**: uma estrutura semântica intencional, que codifica as regras implícitas, restringindo a estrutura de parte da realidade;
- **Ontologia Formal**: desenvolvimento sistemático, formal e axiomático da lógica de todas as formas e modos de ser;
- **Compromisso Ontológico**: é uma semântica parcial a respeito da conceitualização intencionada de uma teoria lógica;
- **Engenharia Ontológica**: um ramo da Engenharia do Conhecimento que explora os princípios formais da Ontologia para construir ontologias;
- **Teoria Ontológica**: conjunto de fórmulas que devem ser sempre verdadeiras de acordo com uma certa conceitualização;
- **Ontologia**: ramo da Filosofia que lida com a natureza e a organização da realidade;
- **ontologia**: (1º sentido) teoria lógica que fornece uma descrição escrita, porém parcial de uma conceitualização; (2º sentido) sinônimo de conceitualização.

3.1 Definição

A etimologia da palavra "Ontologia" vem do grego onta + logos, que significa "conhecimento do ser", ou metafísica. Ontologia é a parte da Filosofia que trata da natureza do Ser, ou seja, da realidade, das existências dos entes e das questões metafísicas em geral (WIKIPéDIA, 2006).

Guarino & Giaretta em (GUARINO; GIARETTA, 1995), analisando debates ocorridos na comunidade de conhecimento compartilhado, distingue as várias interpretações que pesquisadores têm sobre o termo ontologia. Foram identificadas sete interpretações que foram classificadas em três classes como mostradas na Figura 5.

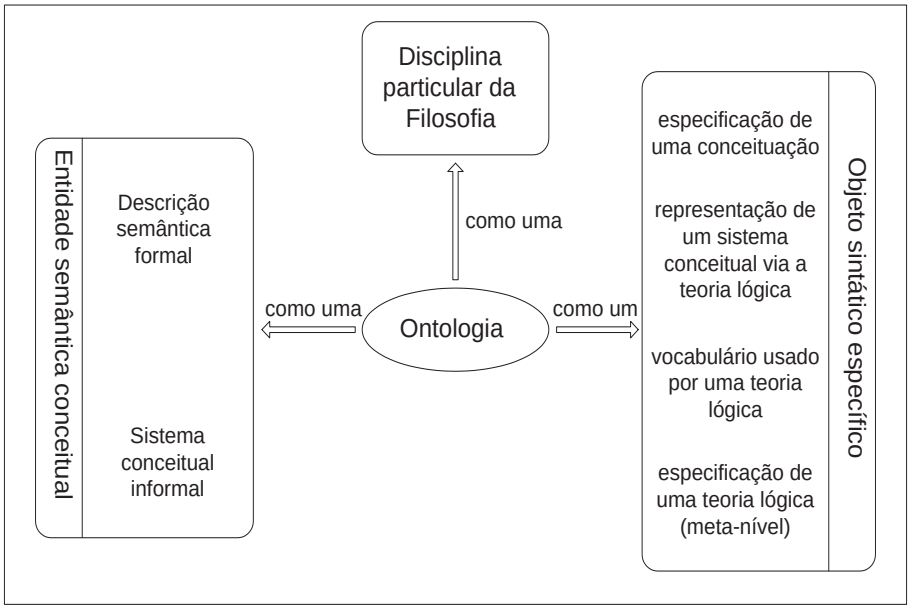


Figura 5: Interpretações do termo "ontologia", segundo (GUARINO; GIARETTA, 1995)

Guarino em (GUARINO, 1998) chama a atenção quanto à necessidade de distinguir os termos "Ontologia" (escrita com letra maiúscula) e "ontologia" (escrita com letra minúscula). De acordo com Guarino, enquanto o primeiro termo parece razoavelmente claro (disciplina particular da Filosofia) o segundo apresenta diferentes sentidos entre as comunidades da Filosofia e da Inteligência Artificial. Na visão da Filosofia, "ontologia" se refere ao sistema particular de categorias de acordo com certa visão do mundo, não tendo uma linguagem específica para representação. Para a comunidade de Inteligência Artificial e, em geral, para toda a comunidade da Ciência da Computação, o termo "ontologia" se refere aos artefatos de engenharia, constituídos por um vocabulário específico usado para descrever certa realidade. Tendo em vista a área abordada por esta dissertação, será adotado o sentido utilizado pela Inteligência Artificial para a palavra "ontologia".

Segundo Gruber & Fensel (FENSEL, 2000; GRUBER, 1993a), ontologia pode ser definida como uma especificação *formal* e *explícita* de um *conceito compartilhado*. Studer et. al menciona em (STUDER; BENJAMINS; FENSEL, 1998) que o termo *conceito* está relacionado a um modelo abstrato de algum fenômeno do mundo que identifica aspectos relevantes do próprio fenômeno. Por outro lado, o *formal* significa que a ontologia pode ser interpretada por má-

quina e o termo compartilhado denota que a ontologia captura conhecimento de um grupo de pessoas e não somente de um único indivíduo. O fato da especificação ser explícita significa que os conceitos descritos nas ontologias e as restrições sobre os seus usos são explicitamente definidos.

Em (GUARINO, 1998) é proposto o refinamento da definição apresentada acima, definindo ontologia como *compromissos ontológicos*. Segundo o autor, ontologia consiste de uma teoria lógica que representa um significado cujo objetivo é definir um vocabulário formal, ou seja, seu compromisso ontológico para uma conceituação particular do mundo.

Os trabalhos (STUDER; BENJAMINS; FENSEL, 1998; USCHOLD; JASPER, 1999) definem que uma ontologia pode empregar uma variedade de formas, mas necessariamente ela incluirá um vocabulário de termos e alguma especificação dos seus significados. Isto inclui definições de conceitos e como eles estão interrelacionados de forma a compartilhar um entendimento sobre um determinado domínio baseado na manifestação de vários participantes desse domínio. Com este acordo, torna-se a comunicação mais precisa e efetiva sobre o significado dos termos, pois restringem as possíveis interpretações do termo, conduzindo a outros como interoperabilidde, reuso e compartilhamento (LOPES, 2005).

Com as definições acima, podemos concluir que ontologias têm como meta capturar conhecimento consensual a respeito de um fenômeno do mundo. Adicionalmente, o conhecimento pode ser reusado e compartilhado entre grupos de pessoas e software, tendo para isso um vocabulário de termos.

Nesta dissertação optamos pela definição de ontologia proposta por (FENSEL, 2000; GRUBER, 1993a), por ser a mais aceita pelos autores da área de Inteligência Artificial, por não contradizer as demais definições mencionadas e por ser adequada ao domínio desta dissertação: o ambiente de grid computacional.

3.2 Finalidade da aplicação de ontologias

Depois de discutidos os conceitos de ontologias, torna-se importante comentar a respeito da finalidade das ontologias. Pode-se chegar à conclusão de que, basicamente, ontologias são aplicadas para possibilitar ou facilitar a comunicação entre diferentes pessoas, aplicações, sistemas, entre outros, os quais fazem parte do mesmo domínio de conhecimento, mas nem sempre compartilham de uma mesma conceituação (visão) a respeito dos componentes deste domínio. Esta falta de entendimento compartilhado leva a (GAVA; MENEZES, 2003):

- Uma comunicação pobre entre as pessoas e as organizações;
- Dificuldades na identificação de requisitos e na definição de uma especificação do sistema;
- Problemas para permitir interoperabilidade e possibilidade de reuso e compartilhamento de conhecimento, o que é extremamente importante tendo-se em vista a grande variedade de métodos, paradigmas, linguagens e ferramentas existentes.

Na Figura 6 encontram-se ilustradas as três categorias de ontologias quanto ao seu uso (USCHOLD; GRÜNINGER, 1996):

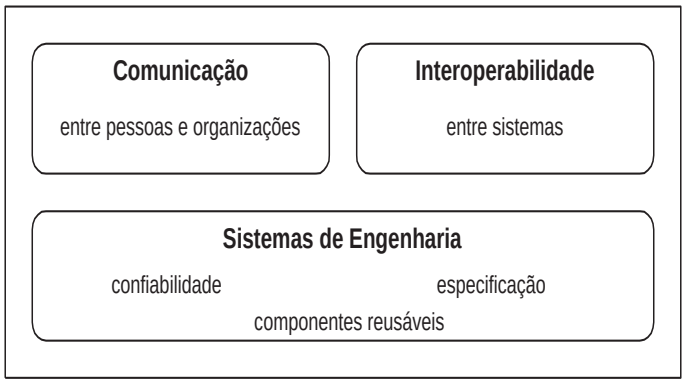


Figura 6: As três principais categorias de uso de ontologias

O emprego de ontologias reduz confusões conceituais e terminológicas dentro de uma organização. Ontologias possibilitam a comunicação e o entendimento compartilhado entre pessoas com diferentes necessidades.

A interoperabilidade entre sistemas é muito buscada, pois diferentes pessoas necessitam acessar, prover e trocar dados através do mesmo ambiente, como é o caso dos ambientes de grids. Para alcançar esta interoperabilidade, ontologias são usadas no desenvolvimento de modelos que expressem o conhecimento de um determinado domínio, formando uma camada de comunicação única e comum a todos os usuários.

A possibilidade de reuso e compartilhamento é possível porque, no momento em que se utilizam ontologias para representação do conhecimento, este se encontra padronizado e expresso em alguma linguagem formal. Desta forma, se torna mais fácil a leitura e interpretação da ontologia por outros domínios, permitindo a sua modificação (inserindo/retirando conceitos, axiomas, relacionamentos, etc) para se adequar a um novo domínio.

A área de Engenharia de Sistemas utiliza-se muito de ontologias para obter a confiabilidade com relação aos conceitos do vocabulário ou linguagem que se está utilizando em um certo ambiente. A representação formal adquirida com a aplicação de ontologias pode tornar possível a automação da verificação de consistência, resultando em ambientes mais confiáveis. O conhecimento compartilhado pode ajudar no processo de identificação de requisitos e na definição de especificações em sistemas de Tecnologia da Informação (TI), principalmente quando os requisitos envolvem grupos diferentes usando terminologias e/ou conceitos distintos em um mesmo domínio (SABATER, 2003).

3.3 Classificação das Ontologias

Pesquisadores classificam as ontologias baseados em diversos fatores. A classificação de ontologias de acordo com Guarino é realizada sob dois aspectos, e são (GUARINO, 1997, 1998):

- Nível de detalhes:
  - **Ontologias de referência** (*off-line*): compreende ontologias que especificam com detalhes o significado pretendido de um vocabulário. Conseqüentemente, eles podem ser usados para estabalecer consenso sobre o compartilhamento deste vocabu-

lário ou sobre uma base de conhecimento que usa este vocabulário;

  - **Ontologias compartilhadas** (*on-line*): são ontologias muito simples e de domínio comum que podem ser desenvolvidas com serviços particulares de inferência com o objetivo de ser compartilhados entre usuários que já concordam sobre a conceitualização básica;

- Nível de dependência sobre uma tarefa particular ou ponto de vista:

  - **Ontologias de alto nível**: descrevem conceitos muito gerais como espaço, tempo, objeto, evento e ação, que são independentes de domínio. Parece razoável, ao menos na teoria, a unificação de ontologias de alto nível para grandes comunidades de usuário;

  - **Ontologias de domínio** e **Ontologias de tarefa**: descrevem, respectivamente, o vocabulário relacionado a um domínio genérico (como medicina ou automóveis), ou a uma tarefa genérica ou atividade (como diagnosticar ou venda de automóveis), através da especialização dos termos introduzidos na ontologia de alto nível. Em outras palavras, ontologias de domínio têm uma visão mais epistemológica do domínio, focando nos conceitos e objetos do universo de discurso enquanto ontologias de tarefas têm visão mais funcional, embora declarativa, do domínio.

- **Ontologias de aplicação**: descrevem conceitos que dependem de domínio e tarefa particular, que são em geral especializações de ambas as ontologias relacionadas. Estes conceitos são geralmente papéis representados por entidades de domínio enquanto executam certa atividade, como unidade substituível ou componente disponível.

Percebe-se que os tipos de ontologias citados estão em ordem decrescente de generalidade, ou seja, representam relacionamentos de especialização, como ilustrado pelas setas da Figura 7.

Para a construção de aplicações não é necessário empregar todos os tipos de ontologias descritos na Figura 7. No entanto, há necessidade de construir ontologias que possam ser reusáveis e compartilhadadas através de múltiplos serviços e métodos (GUARINO, 1997). Gruninger em

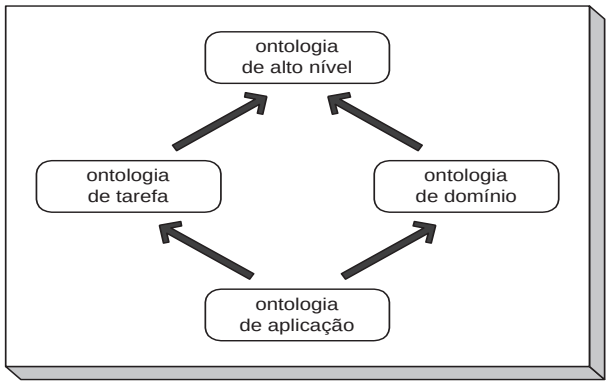


Figura 7: Tipos de ontologias baseados em seu nível de dependência sobre uma tarefa particular ou ponto de vista

(GRUNINGER, 1996), propõe o compartilhamento e reuso de ontologias, usando-as na modelagem de problemas e domínios, provendo uma biblioteca para reutilização fácil de classes e objetos para a modelagem.

Outra forma de classificação das ontologias é quanto a sua finalidade, como já fora mencionado na seção anterior.

Embora existam diferenças dentro das ontologias, há vários elementos em comum entre ontologias. São eles (CHANDRASEKARAN; JOSEPHSON; BENJAMINS, 1999):

- Existem objetos representando elementos do mundo real;
- Objetos possuem propriedades ou atributos que podem capturar valores variados;
- Objetos podem existir participando de várias relações entre outros objetos;
- Propriedades e relações podem mudar com o tempo.

3.4 Projetando Ontologias

Do ponto de vista da Engenharia de Ontologias, uma ontologia é um produto que deve ser projetado para propósitos específicos (LOPES, 2005). Para criar uma ontologia, o primeiro passo a ser realizado é a definição clara do propósito e escopo da sua aplicação. Para alcançar essa

definição clara, o trabalho (USCHOLD; GRÜNINGER, 1996) menciona a necessidade de realizar a captura do conhecimento tratando de:

- identificar os principais conceitos e relacionamentos do domínio de interesse;
- definir textos de forma precisa, descrevendo estes conceitos e relacionamentos encontrados;
- definir os termos usados que se referem aos conceitos e relacionamentos identificados.

Após a captura do conhecimento deve-se partir realmente para o projeto da ontologia. Neste ponto é muito importante ter conhecimento dos principais critérios a serem seguidos para o desenvolvimento do projeto, assim como todos os componentes que deverão ser definidos para integrarem a ontologia criada. Depois de os critérios e componentes serem definidos, é igualmente importante definir a linguagem que será utilizada para construção e o ambiente que será usado para o desenvolvimento. Desta forma, nas próximas seções e subseções, são descritas as questões levantadas, necessárias para o desenvolvimento de ontologias.

3.4.1 Critérios de Projeto

Durante o projeto de ontologias formais é importante escolher como representar os elementos das ontologias. Esta representação faz parte de decisões do projeto. Para guiar e avaliar projetos de ontologias é preciso de critérios objetivos. A seguir são listados critérios preliminares, propostos em (GRUBER, 1993a), que objetivam possibilitar o compartilhamento e a interoperabilidade entre programas baseados em conceitualização compartilhada.

- **Clareza**: uma ontologia deve expressar clara e objetivamente os termos que a definem. A definição deve ser independente de situações sociais ou requisitos computacionais. Uma forma de alcançar essa independência é através do emprego de formalismo utilizando-se de axiomas lógicos quando for possível. Uma definição completa (isto é, um predicado definido por condições necessárias e suficientes) é preferível sobre uma definição parcial

(definido somente por condições necessárias ou suficientes). Todas as definições devem ser documentadas com linguagem natural.

- **Coerência**: a ontologia deve ser coerente, ou seja, deve permitir inferências que sejam consistentes com as definições. No mínimo, a definição dos axiomas deve ser logicamente consistentes;

- **Capacidade de extensão**: a ontologia deve ser projetada de modo a antecipar o uso de um vocabulário compartilhado, fornecendo uma representação que possa ser estendida e especializada. Em outras palavras, a ontologia deve ser capaz de definir novos termos para usos especiais baseados no vocabulário existente, de forma que não requeira a revisão das definições já existentes;

- **Compromisso ontológico mínimo**: uma ontologia deve requerer o mínimo de compromissos ontológicos suficientes para suportar as atividades pretendidas no compartilhamento de atividades. Uma ontologia deve fazer um mínimo de imposições possíveis sobre o mundo que está sendo modelado, permitindo liberdade às partes comprometidas com a ontologia, para possibilitar especialização e instanciação quando necessário.

3.4.2 Componentes de Projeto

Com a finalidade de representar cada elemento presente na ontologia, são empregados certos componentes, também conhecidos como elementos epistemológicos, que são determinados para identificar cada categoria de elementos da ontologia. Com uma visão independente de comunidade específica, Gruber em (GRUBER, 1993b) identifica quatro componentes que devem ser definidos para especificar uma ontologia, que são: classes, relações, funções e axiomas. Estes componentes são definidos da seguinte forma (CORCHO; GOMEZ-PEREZ, 2000):

- **Classes**: também conhecidos como *conceitos*, são usados em sentido amplo. Ou seja, elas podem ser abstratas ou concretas, compostas ou elementares, reais ou fictícias. Um conceito pode ser qualquer coisa sobre a qual alguma coisa é dita. Logo, poderia também ser a descrição de uma tarefa, função, ação, estratégia e processo de raciocínio;

- **Relações**: representam um tipo de interação entre conceitos do domínio. Elas são formalmente definidas como qualquer subconjunto de um produto de *n* conjuntos, matematicamente representado por R: C1 x C2 x C3 x ... x Cn. As relações podem ser unárias ou binárias. Como exemplo de relação binária pode-se considerar "subclasse de";

- **Funções**: são consideradas como um tipo especial de relação onde o valor do último argumento é único para uma lista de valores dos *n-1* argumentos precedentes. Formalmente são representados como: F: C1 x C2 x C3 x ... x Cn-1 → Cn. Exemplos de funções são: *Running_OS*, associando um único sistema operacional a um determinado sistema computacional, e *Mãe_de*, associando um ou vários filhos à sua mãe;

- **Axiomas**: modelam sentenças que são sempre verdadeiras. São inseridos em uma ontologia para diversos propósitos, tais como: restringir suas informações, verificar a exatidão das informações especificadas na ontologia, deduzir novas informações, entre outros.

É importante citar a definição de outros termos, são eles:

- **Taxonomias**: são amplamente usadas para organizar o conhecimento ontológico no domínio usando relacionamentos de generalização/especialização através da aplicação de herança simples ou múltipla. As taxonomias são formadas por termos que, ao lado de suas definições e relações entre eles, formam uma ontologia. Às vezes, uma taxonomia é vista como uma ontologia simples (GUARINO, 1998; NOY; MCGUINNESS, 2001);

- **Slots/Papéis/Propriedades**: representam as várias características e atributos de um conceito (NOY; MCGUINNESS, 2001);

- ***Facets***: são restrições impostas sobre as propriedades (NOY; MCGUINNESS, 2001);

- **Instâncias**: representam elementos específicos de classes (NOY; MCGUINNESS, 2001). Por exemplo, uma classe de vinhos representa todos os vinhos. No entanto, vinhos específicos como *Brunelo Di Montalcino* e *Corton-Charlemagne*, são instâncias desta classe.

3.5 Linguagens para construção de ontologias

São muitas as linguagens desenvolvidas para construção de ontologias. Su & Ilebrekke em (SU; ILEBREKKE, 2002) classificaram as linguagens para construção de ontologias em duas categorias: as linguagens de ontologias tradicionais e as linguagens para *Web*. Esta classificação, mostrada na Figura 8, baseou-se na avaliação sobre linguagens ontológicas realizadas anteriormente em (CORCHO; GOMEZ-PEREZ, 2000).

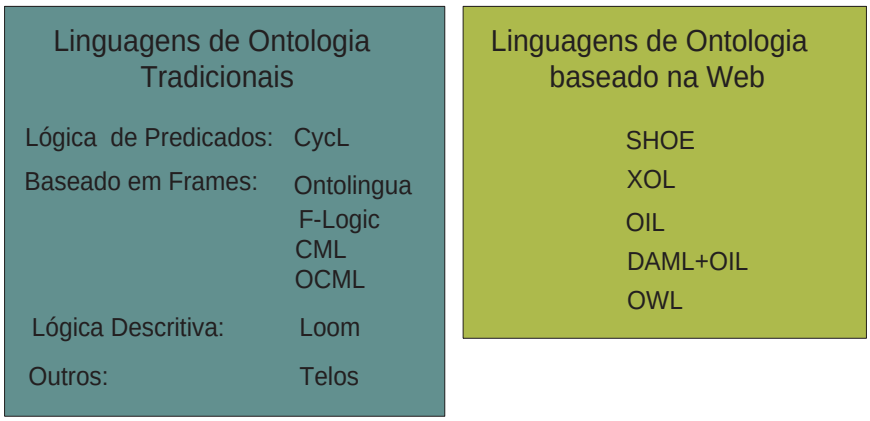


Figura 8: Categorização das linguagens, adaptado de (SU; ILEBREKKE, 2002)

A seguir são descritas as duas categorias:

- **Linguagens de Ontologia Tradicionais**: possuem raiz na Inteligência Artificial ou na Engenharia de Conhecimento. Essa categoria possui várias linguagens quais são: linguagens que já vinham sendo usadas para representar conhecimento dentro de aplicações baseadas em conhecimento, linguagens que foram adaptadas de linguagens de representação de conhecimento e um grupo de linguagens especificamente criadas para a representação de ontologias. Estas linguagens estão em fase estável de desenvolvimento e sua sintaxe consiste de plano texto onde ontologias são especificadas (várias delas têm sintaxe similar à do LISP (MCCARTHY, 1960)). As linguagens podem ser divididas em quatro grupos:

  – No primeiro grupo, encontram-se as linguagens que têm como base a lógica de predicados de primeira ordem, onde a CycL (LENAT; GUHA, 1990) é a linguagem mais representativa desse grupo. Os predicados são o ponto central da modelagem;

  - O segundo grupo é composto por linguagens baseadas em classes como Ontolingua (FARQUHAR; FIKES; RICE, 1997), F-Logic (KIFER; LAUSEN; WU, 1995) e OCML (MOTTA, 1998) cuja modelagem central são as classes (*frames*);
  - Pertencem ao terceiro grupo as linguagens baseadas em Description Logic (DL). A linguagem Loom (MACGREGOR; BURSTEIN, 1991) é um exemplo de linguagem desse grupo. Este grupo caracteriza-se por poder definir conceitos em termos de descrições que especificam as propriedades que os objetos devem satisfazer para pertencer ao conceito;
  - No quarto grupo, têm-se linguagens que não se encaixam nos demais grupos acima. A linguagem Telos (MYLOPOULOS et al., 1990) é um exemplo deste grupo.

- **Linguagens de Ontologia para *Web***: são linguagens baseadas em padrões Web que são usados para facilitar o intercâmbio de informações e seus significados na Internet, constituindo-se na principal camada da Web Semântica. Esta categoria é conhecida também como linguagens de especificação de ontologia baseada na *Web*.

Na seção a seguir apresentaremos de forma mais detalhada as linguagens de especificação de ontologia baseada na *Web* bem como os padrões que elas seguem. Daremos ênfase nesta categoria, pois as ontologias empregadas nesse trabalho foram construídas na linguagem OWL (MCGUINNESS; HARMELEN, 2004). Logo, não serão apresentados detalhes das linguagens que compõem a categoria das ontologias tradicionais.

3.5.1 Tecnologias Web Semântica

Mais recentes que as linguagens de ontologia tradicionais, as linguagens voltadas a *Web* têm tido um grande impacto no contexto *World Wide Web* (WWW). Este impacto deve-se ao desenvolvimento da *Web* Semântica.

A *Web* Semântica é definida pelo W3C (LEE, 2006) como uma extensão da *Web* atual em que a informação tem um significado bem definido, permitindo que pessoas e computadores

trabalhem melhor em cooperação. A *Web* Semântica objetiva ter dados na *Web* bem definidos e ligados de tal modo que possa ser usado mais efetivamente para descoberta, automação, integração e reuso através das várias aplicações. Além disso, dados podem ser compartilhados e processados por ferramentas automatizadas bem como por pessoas.

Para alcançar este objetivo há necessidade de estabelecer padrões de troca de informações que sejam interpretáveis por máquinas. Estes padrões não somente definiriam a sintaxe da informação, mas também o seu significado. A *Web* Semântica, em 2001, tornou-se uma abordagem mais realista com a proposta do W3C. A proposta definiu várias novas camadas para a *Web*, sugerindo linguagens e padrões para as camadas, como a Figura 9 mostra.

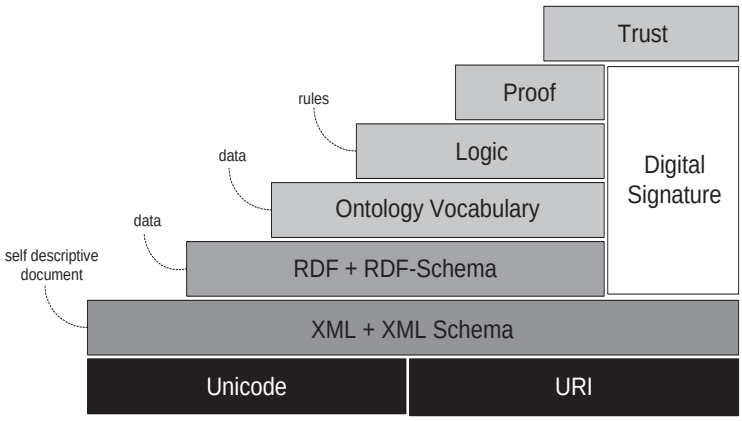


Figura 9: Camadas da *Web* Semântica (KOIVUNEN; MILLER, 2001)

Nas subseções a seguir serão descritas as camadas propostas pelo W3C:

3.5.1.1 A camada Unicode e URI

A primeira camada garante o uso padronizado do mesmo conjuntos de caracteres (Unicode) e uma forma unívoca para a identificação e localização de recursos através do Uniform Resource Identifier (URI).

3.5.1.2 A camada XML

Esta camada constitui-se da tecnologia XML que é uma linguagem de marcação de propósito geral para criar linguagens de marcação de propósito especial, ou seja, é uma meta-

linguagem de editoração. XML é adotada pelo W3C como linguagem padrão para troca de informações na *Web*. A sua meta principal é facilitar o compartilhamento de dados através de diferentes sistemas, principalmente sistemas conectados via a Internet. A idéia é que esta camada descreva a estrutura do documento, deixando para as camadas superiores a definição do seu conteúdo.

3.5.1.3 A camada RDF e RDF-Schema

Modelos de dados comuns e padrões de troca de dados são desejáveis para garantir uma rápida integração de diferentes fontes de dados e para ligar diferentes representações semânticas. O grupo W3C propôs o RDF (*Resource Description Framework*) como um modelo de dados adequado para representar os recursos e suas informações na *Web*, permitindo a integração das informações. RDF é uma linguagem declarativa que é usada para expressar as proposições usando vocabulários formais, particularmente aqueles especificados na linguagem RDF-Schema, para acesso e uso sobre o WWW. Além disso, o RDF fornece a base para linguagens declarativas mais avançadas com um propósito similar (HAYES, 2004).

A definição de uma ontologia em RDF é normalmente vista como uma coleção de triplas. Um conjunto de triplas forma o que é chamado de um grafo RDF (KOKKELINK, 2001). Cada tripla é constituída de três partes: um sujeito *s*, um predicado *p* (também conhecido como propriedade) e um objeto *o*, como ilustrado na Figura 10.

Figura 10: Exemplo de uma tripla representando um significado entre um sujeito e um objeto

Portanto, os nodos de um grafo RDF, representam os sujeitos e objetos das triplas que lhe formam. A direção do arco que liga um sujeito e um objeto possui significado e está sempre apontado para o objeto. A declaração de uma tripla RDF da forma <*s,p,o*> informa que existe algum relacionamento, indicado pelo predicado *p*, entre os recursos denotados pelo sujeito *s* e objeto *o* da tripla, como apresentado na Figura 10. Como um grafo RDF é um conjunto

de triplas, a sua declaração representa a agregação de todas as declarações das triplas contidas nele. Dessa forma, o significado de um grafo RDF é a conjunção lógica dos enunciados correspondentes a todas as triplas que ele contém.

A linguagem RDF não possui mecanismos para descrever relacionamentos entre nodos de diferentes triplas, porém o RDF-Schema possui. RDF-Schema é uma linguagem de descrição do vocabulário RDF, que estende a linguagem RDF semanticamente, provendo um conjunto fixo de primitivas básicas de modelagem para a definição de ontologias (classe, propriedade, e os seguintes relacionamentos *is-a* e *element-of*) e uma maneira padrão de codificá-las em XML. RDF-Schema pode ser empregado como linguagem de especificação de ontologias, apesar deste não ser o seu principal objetivo e sim servir de base para outras linguagens de especificação de ontologias mais expressivas (CORCHO; GOMEZ-PEREZ, 2000; KOIVUNEN; MILLER, 2001).

3.5.1.4 A camada das linguagens de ontologia voltada a Web

Nesta seção são descritas sucintamente as linguagens voltadas a *Web*. Estas linguagens, com exceção da linguagem SHOE, que tem sua sintaxe baseada em HTML (*HyperText Markup Language*), possuem sintaxe baseada em XML (CORCHO; GOMEZ-PEREZ, 2000). Na Figura 11, é apresentada uma pirâmide que define os relacionamentos existentes entre as linguagens de ontologia voltada a *Web*. O acrônimo RDF(S) é usado para referir-se à combinação das linguagens RDF e RDF-Schema.

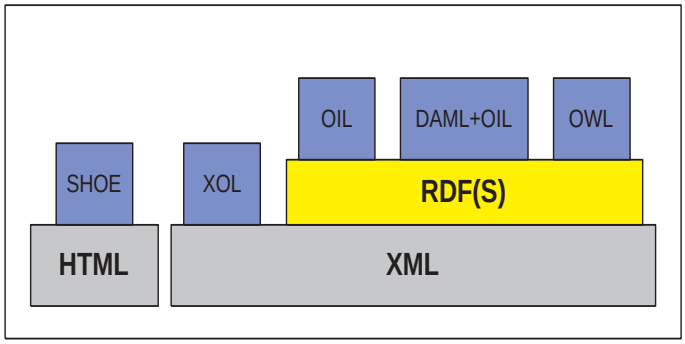


Figura 11: Pirâmide de linguagens de ontologias baseada na Web, adaptado de (CORCHO; GOMEZ-PEREZ, 2000)

A seguir são descritas as linguagens mais relevantes de especificação de ontologias baseadas

na *Web*, são elas:

- **SHOE** (*Simple HTML Ontology Extensions*) (HEFLIN; HENDLER, 2000) é uma extensão do HTML para incorporar conhecimento semântico em documentos *Web* comuns utilizando a anotação de páginas HTML. SHOE fornece *tags* que são necessárias para embutir dados semânticos arbitrários dentro de páginas *Web*. As *tags* são divididas em duas categorias: as de construção de ontologias e as de anotação de documentos *Web*. SHOE foi desenvolvido na Universidade de Maryland, em 1996, permitindo representar conceitos, suas taxonomias, relações n-árias, instâncias e regras de dedução que são usadas pelo seu motor de inferência para obter novos conhecimentos.

- **XOL** (*XML-based Ontology Language*) (KARP; CHAUDHRI; THOMERE, 1999) foi originalmente criada para troca de ontologias de biologia molecular pelo Centro de Inteligência Artificial da SRI (Stanford Research Institute) em 1999. XOL provê uma definição geral que a torna apropriada para troca de outras ontologias do domínio biomédico. As primitivas e semânticas de modelagem são baseadas no OKBC Lite, uma versão simplificada do modelo de conhecimento OKBC (Open Knowledge Base Connectivity) (CHAUDHRI et al., 1997, 1998). A linguagem XOL é muito restrita, pois apenas conceitos, taxonomias e relações binárias podem ser especificadas. Não existe mecanismo de inferência anexada nela, pois foi projetada pricipalmente para troca de ontologias do domínio biomédico.

- **OIL** (*Ontology Inference Layer*) (FENSEL et al., 2000b) foi desenvolvido no contexto do projeto European IST OntoKnowledge. OIL pode ser considerado como um precursor do DAML+OIL (ver próxima seção), ajustado sobre os fundamentos para o projeto de uma linguagem para a web semântica. A linguagem combina primitivas de linguagens baseada em frames, semântica formal e serviços de raciocínio de lógicas de descrição. Para permitir o uso de OIL na *Web*, ele fundamenta-se nos padrões W3C, XML e RDF(S). A descrição da ontologia pela OIL é dividida em três camadas: nível objeto (instâncias concretas), meta-nível primário (definições ontológicas) e o meta-nível secundário (descrevendo características da ontologia). Adicionalmente, OIL fornece definições de classes

e slots, e diversas limitações relacionadas aos axiomas que conseqüentemente limitam a expressividade da linguagem (FENSEL et al., 2000a). OIL possui uma semântica precisa que forma uma base necessária para suporte efetivo de raciocínio.

- **DAML+OIL** (HORROCKS; PATEL-SCHNEIDER; HARMELEN, 2002) é o resultado da união das linguagens DAML (*DARPA Agent Markup Language*) e OIL. DAML+OIL é uma linguagem de marcação semântica para recursos *Web* criada como um esforço das comunidades de ontologia americanas e européias para obter uma linguagem padrão para a *Web* Semântica. Ela foi desenvolvida no contexto da iniciativa DAML. DAML+OIL é construída sobre RDF(S), mas oferece primitivas de modelagem mais ricas, usualmente encontradas em lógicas de descrição (SU; ILEBREKKE, 2002). A maioria das idéias baseadas em frame, fornecidas em OIL, foi removida. O resultado é uma linguagem que trabalha melhor como uma plataforma para entrega de ontologias do que RDF, porém tem limitações no desenvolvimento de ontologias devido à remoção das construções baseadas em frame. DAML-OIL tem o objetivo de descrever a estrutura de um domínio como uma estrutura Orientada a Objetos (OO), sendo o domínio descrito em função de classes e propriedades. Além disso, esta linguagem permite a representação de taxonomias, relações binárias, funções e instâncias.

- **OWL** (*Web Ontology Language*) (MCGUINNESS; HARMELEN, 2004) é a recomendação para especificação de ontologias para a *Web*. Esta linguagem apresenta todos os benefícios de outras linguagens, como por exemplo: DAML+OIL, RDF e RDF-Schema, entre outras. A linguagem OWL consiste de uma revisão que incorpora algumas melhorias necessárias na linguagem DAML+OIL. As melhorias consistem de mecanismos para a representação de classes, hierarquia de classes, propriedades, relacionamento entre classes, restrições em propriedades (cardinalidade, igualdade, etc), características das propriedades (transitiva, funcional, inversa, etc), anotações e instâncias (MCGUINNESS; HARMELEN, 2004). A OWL é dividida em três sub-linguagens que são distinguidas pelo nível de formalidade exigido e oferecido, e a liberdade dada ao usuário para a definição de ontologias, são elas (MCGUINNESS; HARMELEN, 2004):

– **OWL Lite**: é apropriada para usuários que queiram uma hierarquia de classificação e restrições simples. Como exemplo, a linguagem suporta restrições de cardinalidade com valores de 0 ou 1. Esta sub-linguagem é a menos expressiva das três;

– **OWL DL**: fornece aos usuários o máximo de expressividade, assegurando que todas as inferências feitas sobre ontologias construídas nesta sub-linguagem terminem em tempo finito. Esta sub-linguagem faz uso de Lógica de Descrição (Description Logics), de onde vem a extensão DL;

– **OWL Full**: pode ser empregada por usuários que desejam o máximo de expressividade e liberdade sintática do RDF, porém sem qualquer garantia computacional, uma vez que instâncias e propriedades podem ser especificadas como classes, o que é restringido na sub-linguagem OWL DL.

As linguagens menos expressivas (OWL Lite e DL) estão contidas dentro das mais expressivas (OWL DL e Full), como ilustrado na Figura 12. Portanto, uma ontologia definida em uma linguagem menos expressiva é aceita por uma mais expressiva; a recíproca, naturalmente, não é verdadeira.

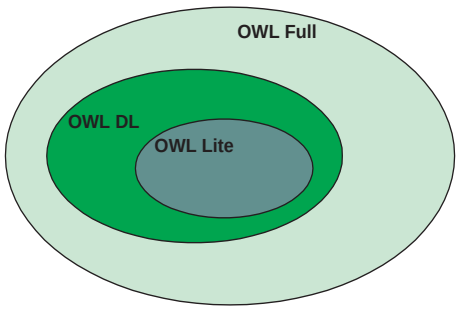


Figura 12: A expressividade das sub-linguagens OWL

Na Tabela 1 é apresentada uma comparação geral entre as linguagens de especificação de ontologias baseadas em tecnologias *Web*.

Tabela 1: Comparativo entre as linguagens voltadas a *Web*, adaptado de (FLEISCHMANN, 2004)

Característica	SHOE	XOL	OIL	DAML+OIL	OWL
Conceito	S	S	S	S	S
Relações binárias	S	S	S	S	S
Funções	N	S	S	S	S
Instâncias	S	S	S/N	S	S
Taxonomias	S/N	S	S	S	S
Axiomas	N	S	N	N	S
Restrições de tipo	S	S	S	S	S
Restrições de integridade	N	?	N	N	S
Mecanismos de inferência	S	N	S	S	S

As linguagens são comparadas de acordo com a existência ou não das seguintes características: conceitos, relações binárias, funções, instâncias, taxonomias, axiomas, restrições de tipo e de integridade, e mecanimos de inferência. As informações presentes nas tabelas significado: "S" denota que tal característica está presente na linguagem; "N" denota que não está; "S/N" denota que a característica não está presente, mas pode ser simulada através de simulações na linguagem e o caractere "?" indica que não foi encontrada nenhuma informação a respeito.

3.5.1.5 As camadas de lógica, prova e confiança

As camadas mais altas propostas pelo W3C ainda não se encontram padronizadas. A camada lógica permite a especificação de regras dentro da própria ontologia, atuando sobre instâncias e recursos, enquanto a camada de prova as executa, e a de confiança avalia se a prova está correta ou não (KOIVUNEN; MILLER, 2001). Para que estas camadas entrem em operação, as camadas inferiores devem estar bem sedimentadas, o que ainda não ocorreu. Além do mais, sob o ponto de vista ontológico, não é interessante antecipar o uso de ontologias com regras, pois isto pode restringir a sua aplicabilidade. Porém, regras podem ter utilidade para restringir atributos e exprimir axiomas (KOIVUNEN; MILLER, 2001).

A camada de lógica já dispõe de protótipos de linguagens, como por exemplo, RuleML (Rule Markup Language) (BOLEY; TABET; WAGNER, 2001).

3.6 API Jena

A API Jena é um *framework* desenvolvido na linguagem de programação Java pela empresa HP (Hewlett-Packard) para construção de aplicações *Web* Semânticas. Inicialmente Jena foi desenvolvida nos laboratórios de pesquisa para *Web* Semântica da HP e posteriomente disponibilizada como projeto de *software* livre. Jena fornece ambiente de programação para RDF, RDF-Schema, DAML+OIL e OWL, incluindo motores de inferência baseados em regras.

Para cada linguagem, existe um parâmetro que permite a construção de URIs para identificar classes e propriedades de ontologias. Cada parâmetro possui uma sintaxe diferente, por exemplo, no parâmetro da linguagem DAML, a URI para indicar uma propriedade do objeto é *daml:ObjectProperty* e na linguagem OWL é *owl:ObjectProperty*. Já no RDF-Schema, o parâmetro é nulo, pois nesta linguagem não se definem propriedades para os objetos. Cada linguagem de ontologia possui suas próprias características, limitadas ao seu modelo de ontologia, mas todas estendem a versão do modelo de classes do Jena.

Jena oferece também uma abstração simples do grafo RDF através de sua interface interna central. A principal contribuição do Jena é a sua API rica que permite a manipulação dos grafos RDF. Além disso, Jena provê várias ferramentas, por exemplo: um *parser* RDF/XML, linguagens de consulta RDF (por exemplo, as linguagens RDQL (SEABORNE, 2004) e SPARQL (AIKEN et al., 2000)), módulos I/O para os formatos de saída N3, N-triple e RDF/XML. Com o Jena o usuário pode escolher o tipo de armazenamento dos grafos - em memória ou em base de dados relacional. Jena fornece funcionalidades adicionais para suportar RDF(S) e OWL.

Os dois objetivos principais da arquitetura Jena são (WILKINSON et al., 2003):

- permitir ao programador da aplicação representações flexíveis e múltiplas dos grafos RDF. Dessa forma, facilita a manipulação dos dados nos grafos e o seu acesso possibilitando ao programador da aplicação navegar nas estruturas de triplas, e
- tornar a visão do grafo RDF simples ao programador do sistema que deseja expor seus dados como triplas.

Fontes de triplas no Jena podem ser disponibilizadas, por exemplo, em bases de dados ou em memória. Adicionalmente, as triplas podem ser alcançadas virtualmente, como resultado de processos de inferência aplicados a outras fontes de triplas (WILKINSON et al., 2003). Uma visão simplificada da arquitetura do Jena é mostrada na Figura 13.

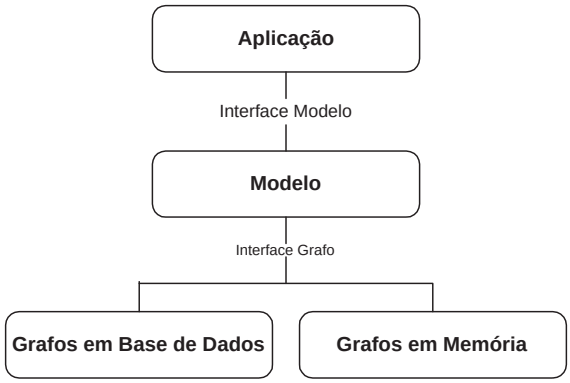


Figura 13: Visão da arquitetura do Jena, adaptado de (WILKINSON et al., 2003)

Aplicações que utilizam Jena geralmente interagem com um modelo abstrato, que traduz operações de mais alto nível em operações de baixo nível sobre triplas armazenadas em um tipo de grafo RDF.

3.6.1 Mecanismo de Inferência

O sistema de inferência do Jena é projetado para permitir que um grande número de motores de inferência seja utilizado. Alguns motores permitem criar novas informações derivadas das informações já existentes nas ontologias. Outros permitem realizar checagens de consistências sobre as ontologias, verificando se todos os axiomas definidos nas próprias ontologias são obedecidos.

A distribuição Jena já inclui alguns raciocinadores pré-definidos, os principais são (TEAM, 2006):

- **Raciocinadores OWL, OWL Mini e OWL Micro**: um conjunto de raciocinadores úteis para checagem de consistência, porém incompletos da linguagem OWL Lite;

- **Raciocinador DAML micro**: usado internamente no Jena para fornecer um mínimo de inferência para ontologias descritas em DAML (legado);

- **Raciocinador de Regra Genérico**: raciocinador baseado em regra que é usado para implementar ambos raciocinadores, RDFS e OWL, mas também está disponível para uso em geral. Suporta inferência baseada em regras sobre grafos RDF que podem ser definidas externamente pelo usuário.

Além dos raciocionadores já incluídos, Jena permite checagem de consistência sobre ontologias OWL DL através do uso de raciocinadores DL externos, tais como Pellet (SIRIN; PARSIA, 2004), Racer (HAARSLEV; MOLLER, 2001) ou FaCT (HORROCKS, 1999). A interface DIG do Jena facilita a conexão de qualquer raciocinador que suporte o padrão DIG (DL Implementors Group) (BECHHOFER, 2006).

A verificação de consistência na terminologia DL, mostrada na Tabela 2, consiste na operação de verificar a consistência de um ABox com respeito a um TBox.

Tabela 2: Terminologia DL (LI; HORROCKS, 2003)

Abreviação	**Por extenso**	**Significado**
ABox	Assertional Box	Contém um exemplo concreto do domínio de conhecimento e axiomas declarados sobre indivíduos, ou seja, um indivíduo é uma instância de um conceito; ou um indivíduo está relacionado a outro por uma função.
TBox	Terminological Box	Define a estrutura do domínio de conhecimento, consistindo de um conjunto de axiomas declarados, ou seja, a definição de um novo conceito em termos de outros conceitos definidos previamente.
KB	Knowledge Box	Uma base de conhecimento DL, ou seja, combinação de ABox e TBox.

3.6.2 Regras de Inferência

O Raciocinador Genérico Jena (RGJ) permite inferência sobre grafos RDF através de regras fornecidas externamente. Estas regras são formadas basicamente por uma lista de termos de corpo (premissas), uma lista de termos de cabeça (conclusões) e um nome opcional que identifica a regra. Cada termo pode ser uma tripla padrão, uma tripla estendida padrão ou uma chamada a uma primitiva embutida (TEAM, 2006). A sintaxe da linguagem de regra fornecida pelo *framework* Jena é mostrada na Figura 14.

```
Rule  :=   bare-rule .
       or   [ bare-rule ]
       or   [ ruleName : bare-rule ]

bare-rule :=   term, ... term -> hterm, ... hterm    // forward rule
           or   term, ... term <- term, ... term       // backward rule

hterm  :=   term
        or   [ bare-rule ]

term   :=   (node, node, node)          // triple pattern
        or   (node, node, functor)       // extended triple pattern
        or   builtin(node, ... node)      // invoke procedural primitive

functor  :=   functorName(node, ... node)      // structured literal

node  :=   uri-ref                  // e.g. http://foo.com/eg
        or   prefix:localname     // e.g. rdf:type
        or   ?varname             // variable
        or   'a literal'              // a plain string literal
        or   'lex'^^typeURI       // a typed literal, xsd:* type names supported
        or   number               // e.g. 42 or 25.5
```

Figura 14: Sintaxe da linguagem de regra reconhecida pelo RGJ

Baseado nessa linguagem o RGJ consegue inferir novas informações das informações já existentes nas ontologias. O raciocinador RGJ pode ter as suas inferências estendidas a partir do desenvolvimento e registro de novas primitivas embutidas.

3.6.3 Linguagens de consulta RDF

RDQL (RDF Data Query Language) e SPARQL são linguagens para a realização de consultas em ontologias descritas em RDF e linguagens derivadas. Atualmente, RDQL tem o *status* de uma submissão W3C (SEABORNE, 2004) enquanto SPARQL é a linguagem de RDF recomendada pelo W3C. Ambas são consideradas somente como linguagens "orientadas a dados", ou

seja, permitem apenas extrair dados de documentos RDF disponíveis na Web ou armazenados em um meio físico qualquer, não possuindo mecanismos de inferência (MILLER; SEABORNE; REGGIORI, 2002).

Para que as consultas possam ser realizadas e as informações possam ser extraídas, a linguagem RDQL e SPARQL disponibilizam uma sintaxe que, ainda que haja algumas particularidades, funciona de maneira similar à linguagem SQL (Structured Query Language). A linguagem SPARQL pode expressar tudo que a linguagem RDQL permite e ainda outras características. Por exemplo, SPARQL possibilita ordenação de seqüências baseado em condições, limitação de seqüências, definir tipo de dados RDF entre outras características. Atualmente a única razão para usar RDQL é por legado.

A seguir são apresentadas as principais cláusulas que compõem a linguagem SPARQL.

- *SELECT*: essa cláusula permite selecionar quais informações serão retornadas como resultado da consulta. As informações são armazenadas em variáveis que são identificadas pelo sinal de interrogação (?);
- *WHERE*: permite especificar as restrições para a realização das consultas. Essas restrições seguem o formato de tripla <sujeito, predicado, objeto>, que podem ser formadas tanto por um objeto quanto por um valor literal;
- *FILTER*: restringe o conjunto de soluções de acordo com uma ou mais expressões. As expressões podem ser funções e operações construídas sintaticamente (PRUD'HOMMEAUX; SEABORNE, 2005). Os operandos dessas funções e operadores são um subconjunto dos tipos de dados do XML Schema (xsd:string, xsd:decimal, xsd:double, xsd:dateTime) e tipos derivados de xsd:decimal;
- *ORDER BY*: captura uma seqüência de solução e aplica sobre ela condições de ordenação. Uma condição de ordenação pode ser uma variável ou a chamada a uma função. A direção de ordenação é ascendente por padrão. Pode-se explicitamente informar a direção de ordenação em ascendente e decrescente, através de ASC e DESC;

- *LIMIT*: limita o número de soluções retornadas. Se o número de soluções reais é maior do que o limite, então no máximo o número limite de soluções será retornado.

A cláusula FROM é implícita, visto que a consulta é realizada sobre a ontologia ou modelo no qual se está trabalhando.

Na Figura 15 é apresentada parte de uma ontologia OWL onde são descritos os conceitos sistema computacional, sistema operacional e processador.

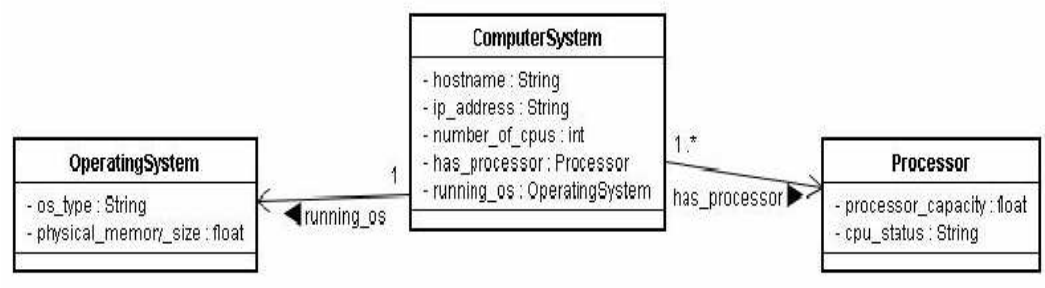


Figura 15: Parte de uma ontologia descrevendo recursos computacionais

O conceito sistema computacional é caracterizado pelas seguintes propriedades: *ip_address*, *hostname*, *number_of_cpus*, *has_processor* e *running_os*. As três primeiras propriedades são os atributos da classe enquanto as duas últimas são relações entre a classe *ComputerSystem* e as classes *Processor* e *OperatingSystem*. A classe *Processor* expressa o conceito processador com as seguintes propriedades: *processor_capacity* e *cpu_status*. A classe *OperatingSystem* representa o conceito sistema operacional que possui as propriedades *os_type* e *physical_memory_size*.

Baseado na ontologia ilustrada na Figura 15, suponha que um usuário pretenda pesquisar recursos que apresentem os seguintes requisitos: no máximo dois recursos identificados pelos seus endereços IPs, tenham sistema operacional Linux e capacidade de processamento maior ou igual a 2800 MHz e que estes recursos possuam a maior capacidade de memória RAM dentre os recursos disponíveis. Na linguagem SPARQL, esta pesquisa pode ser realizada pela consulta descrita na Figura 16.

```
PREFIX grsc: <http://www.owl-ontologies.com/Grid_Resources.owl#>

SELECT ?ip
WHERE {  ?A  grsc:id_address  ?ip .
         ?A  grsc:running_os  ?B .
         ?B grsc:os_type  ?so .
         ?B grsc:physical_memory_size ?mem .
         ?A  grsc:has_processor  ?C .
         ?C  grsc:processor_capacity ?clock .
FILTER ((?so = 'Unix') && (?clock >= 2800))   }
ORDER BY DESC(?mem)
LIMIT 2
```

Figura 16: Exemplo de uma consulta SPARQL

Para o usuário descobrir quais recursos computacionais possuem sistema operacional Linux e capacidade de processamento maior ou igual a 2800 MHz, ele deve traduzir o conhecimento modelado na ontologia em triplas, como mostradas na cláusula WHERE da consulta apresentada na Figura 16.

Os recursos retornados são representados pelos seus endereços IPs através da tripla (?A grsc:ip_address ?ip). Esses recursos (capturados pela variável ?A) devem ter sistema operacional Linux e capacidade de processamento maior ou igual a 2800 MHz.

Para descobrir quais recursos tem Linux é necessário saber os sistemas operacionais de cada recurso. Isto é alcançado através das duas triplas (?A grsc:running_os ?B) e (?B grsc:os_type ?so). A primeira tripla indica que os recursos procurados (?A) têm relações com instâncias da classe *OperatingSystem* através do termo *running_os*, identificadas pela variável ?B. A segunda tripla atribui os sistemas operacionais informados por estas instâncias à variável ?so.

A capacidade de processamento de cada recurso também é alcançado por duas triplas, (?A grsc:has_processor ?C) e (?C grsc:processor_capacity ?clock). A primeira tripla informa que os recursos buscados (?A) têm relações com instâncias da classe *Processor* através do termo *has_processor*, identificadas pela variável ?C. A segunda tripla armazena as capacidades de processamento dos recursos na variável ?clock. Para restringir o conjunto solução desejado, expressões lógicas foram definidas na cláusula FILTER.

Para ordenar o conjunto solução quanto ao tamanho de memória RAM é necessário ter os

valores do tamanho das memórias dos recursos. Estes valores são alcançados pela tripla (?B grsc:physical_memory_size ?mem) armazenando-os na variável ?mem, para servir de condição de ordenação na cláusula ORDER BY na direção decrescente.

A consulta ilustrada na Figura 16, portanto, expressa o desejo de retornar no máximo dois recursos (expresso na cláusula LIMIT) pelos seus endereços IP (?ip), ordenados na ordem decrescente do tamanho das suas memórias RAM e que tenham sistema operacional Linux e capacidade de processamento maior ou igual a 2800 MHz.

3.7 Integração semântica baseada em Ontologia

Uma importante área de aplicação das ontologias é a integração de sistemas e bases de dados existentes. A capacidade de troca de informação em tempo de execução, também conhecida como interoperabilidade, é um tópico importante. Contudo, a tentativa de fornecer interoperabilidade traz problemas similares àqueles associados com a comunicação entre diferentes comunidades de informação. No entanto, uma importante diferença é que as entidades não são pessoas que são capazes de executar abstrações e raciocínios de senso comum sobre os significados dos termos, mas máquinas (FREITAS; STUCKENSCHMIDT; NOY, 2005).

Portanto, para permitir que máquinas se entendam, é necessário explicar o contexto de cada sistema. Esta explicação é descrita em um nível muito alto de formalidade com o intuito de fazer a máquina entendê-la.

3.7.1 Abordagens de integração de bases de informação baseada em ontologias

Ontologias são geralmente usadas como interlínguas para fornecer interoperabilidade, servindo como um formato comum para troca de dados (USCHOLD; GRÜNINGER, 1996). Cada sistema que deseje interagir com outros sistemas necessita transferir suas informações para dentro deste *framework* comum.

Como citado em (FREITAS; STUCKENSCHMIDT; NOY, 2005; VISSER et al., 2000), existem três

diferentes abordagens quanto à integração de fontes de informação baseadas em ontologias, são elas: ontologia única (centralizada), múltiplas ontologias (descentralizada) e abordagem híbrida.

A abordagem centralizada utiliza uma ontologia global que fornece um vocabulário compartilhado para a especificação das semânticas, como ilustrado no item (a) na Figura 17. Todas as fontes de informação estão relacionadas a uma ontologia global. Esta abordagem pode ser aplicada em problemas de integração, onde todas as fontes de informações a serem integradas fornecem uma visão muito próxima de um domínio. Contudo, se uma fonte de informação tem uma visão diferente do domínio, por exemplo, fornecendo outro nível de granularidade, tornará a tarefa de integração difícil, visto que precisará encontrar o compromisso ontológico mínimo (PERES; LOPEZO; CORCHO, 2004). Outra desvantagem desta forma de integração são as mudanças que podem acontecer nas fontes de informação. Elas podem afetar a conceitualização do domínio representado na ontologia. Dependendo da natureza das mudanças em uma fonte, pode implicar em mudanças na ontologia global e nos mapeamentos de outras fontes. Estas desvantagens levaram ao desenvolvimento de modelos de integração sobre múltiplas ontologias.

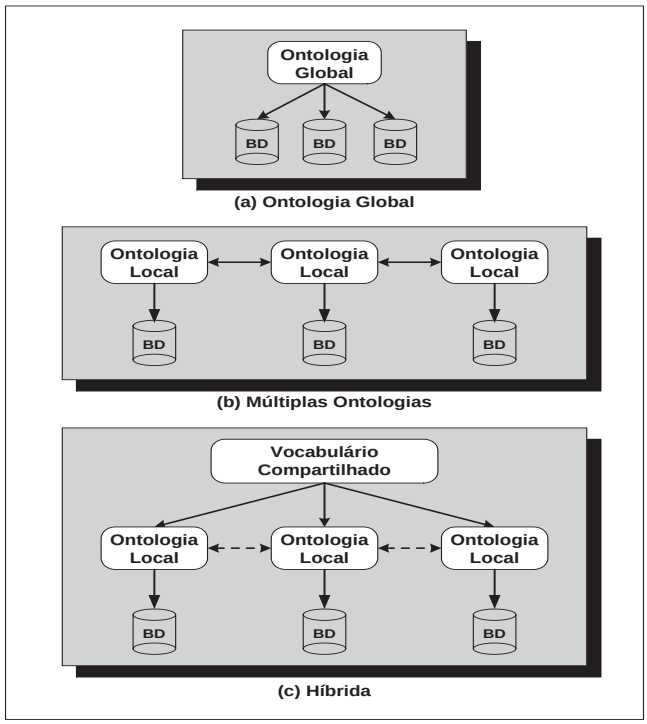


Figura 17: Tipos de abordagens de integração

Diferente da primeira abordagem, na de múltiplas ontologias cada fonte de informação é descrita por sua própria ontologia, como mostrado no item (b) da Figura 17. A princípio, a vantagem desta abordagem é o fato de não necessitar de um compromisso ontológico comum e mínimo sobre uma ontologia global. Cada ontologia de uma fonte pode ser desenvolvida sem se referir às outras fontes ou às suas ontologias. Dessa forma, não havendo uma ontologia comum é imprescindível um acordo entre todas as fontes. Esta abordagem pode simplificar as mudanças, ou seja, modificações em uma fonte de informação ou a adição ou remoção de fontes. Contudo, na prática, esta abordagem traz um dos problemas mais difíceis na pesquisa sobre ontologias, que é o mapeamento entre diferentes ontologias para encontrar similaridades e/ou diferenças entre as ontologias (FREITAS; STUCKENSCHMIDT; NOY, 2005). Na próxima seção, este problema será discutido com maiores detalhes.

A abordagem híbrida, caracterizada no item (c) na Figura 17, foi desenvolvida para superar as desvantagens das abordagens de ontologia global e de múltiplas ontologias. Como na abordagem de múltiplas ontologias, na híbrida as semânticas de cada fonte são descritas pelas suas próprias ontologias. Porém, com o objetivo de tornar as ontologias fontes comparáveis a cada outra elas são construídas sobre um vocabulário compartilhado global. O vocabulário compartilhado contém termos básicos (primitivas) de um domínio. Com a necessidade de construir termos mais complexos de uma ontologia fonte, as primitivas são combinadas por alguns operadores. Devido a cada termo da ontologia fonte ser baseada nestas primitivas, os termos tornam-se mais fáceis de comparar do que na abordagem de múltiplas ontologias. O vocabulário compartilhado pode ser uma ontologia. A vantagem da abordagem híbrida é que novas fontes podem facilmente ser acrescentadas sem ser preciso modificar os mapeamentos ou os termos do vocabulário compartilhado. Adicionalmente, esta abordagem suporta a aquisição e evolução de ontologias fontes. O uso de um vocabulário compartilhado torna as ontologias fonte comparáveis e evita as desvantagens da abordagem descentralizada. No entanto, ontologias já existentes não podem ser reutilizadas facilmente, pois todas as ontologias fontes devem ser construídas baseadas no vocabulário compartilhado.

Na Tabela 3, sumarizamos as vantagens e desvantagens das diferentes abordagens de inte-

gração.

Tabela 3: Vantagens e desvantagens das diferentes abordagens de integração baseada em ontologia

	Centralizada	**Descentralizada**	**Híbrida**
Esforço de implementação	pouco	custoso	razoável
Heterogeneidade Semântica	visão semelhante do domínio	suporta diferentes visões do domínio	suporta diferentes visões do domínio
Alcançar a Interoperabilidade	-	ao fornecer uma nova ontologia fonte, é necessário relacioná-la a outras ontologias	ao fornecer uma nova ontologia fonte, não é necessário relacioná-la a outras ontologias
Comparação de múltiplas ontologias	-	dificuldade devido a falta de um vocabulário comum	simples devido ontologias usarem um vocabulário comum
Permite reutilização de ontologias	-	Sim	Não

3.7.2 Diferentes tipos de mapeamentos de ontologias

Como mencionado, encontrar correspondências entre ontologias, processo este conhecido como mapeamento ou alinhamento ontológico, é um dos problemas mais difíceis na pesquisa sobre ontologias. Mapeamento de ontologias é considerado um morfismo (MESEGUER, 1989) que tipicamente consiste de uma coleção de funções, atribuindo os símbolos usados em um vocabulário a símbolos de outro vocabulário. Contudo, duas ontologias podem ser relacionadas de uma forma mais geral, ou seja, por meio de relações ao invés de funções. O emprego de relações binárias para relacionar símbolos de duas ontologias é conhecido por alinhamento de ontologias (KALFOGLOU; SCHORLEMME, 2003).

Seria ideal que houvessem ontologias padrão para modelos de diferentes domínios, por exemplo, uma única ontologia para as seguintes áreas: Medicina, Bioinformática, Biologia entre outras áreas. No entanto, não somente múltiplas ontologias são desenvolvidas levando à sobreposição dos mesmos domínios, como quanto mais ontologias são desenvolvidas mais con-

teúdos similares ou sobrepostos existirão. Não é esperado que as pessoas tenham a mesma visão sobre um pequeno conjunto de ontologias (FREITAS; STUCKENSCHMIDT; NOY, 2005). Isso acontece por diversas razões, que vão desde as práticas (aplicações diferentes requerem diferentes visões de um domínio) a institucional e social (uma ontologia desenvolvida em outro lugar não pode ser tão adequada àquela que foi desenvolvida para um determinado fim institucional ou social) (FREITAS; STUCKENSCHMIDT; NOY, 2005).

De qualquer forma, aplicações que usam diferentes ontologias para descrever os seus domínios ainda necessitam de interoperabilidade. Dessa forma, é necessário encontrar correspondências entre diferentes ontologias. Dadas duas ontologias, é preciso distinguir as similaridades e diferenças presentes e principalmente expressar essas correspondências de uma forma que a máquina possa processá-los. Para entendimento deste trabalho, focaremos nos diferentes tipos específicos que podem existir de mapeamentos e sobre as diferentes formas de especificá-los.

A primeira classe de ferramentas para o mapeamento de ontologias lida com o caso onde duas ontologias a ser mapeadas compartilham os conceitos de uma ontologia de referência comum, como mostrado no item (a) na Figura 18. Algumas ontologias de alto nível, tais como SUMO (NILES, 2001) e DOLCE (GANGEMI et al., 2003) são desenvolvidas especialmente com o intuito de facilitar o compartilhamento do conhecimento.

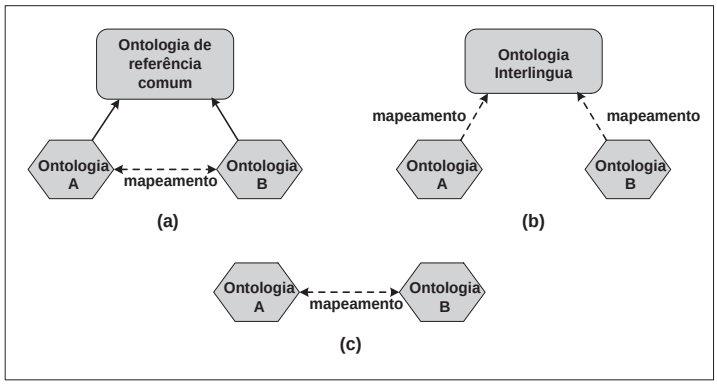


Figura 18: Diferentes tipos de mapeamentos de ontologias

Grüninger and Kopena em (GRUNINGER; KOPENA, 2005), propuseram uma outra abordagem para integração de ontologias que é baseada especificamente na idéia de uma interlíngua compartilhada. Nesta abordagem não há mapeamento direto entre as ontologias, mas somente

a interlíngua, como ilustrado no item(b) na Figura 18.

Quando uma ontologia compartilhada não está disponível, é necessário confiar em ferramentas que ajudem a encontrar mapeamentos diretos entre as ontologias, como apresentado no item (c) na Figura 18. Estas ferramentas usam outros tipos de informação, tais como: informação léxica e estrutural, entrada do usuário ou recursos externos. As ferramentas desenvolvidas por Hovy (HOVY, 1998) são provavelmente as mais representativas usando informação léxica, tais como nomes de conceitos e definições, sua estrutura léxica, distância entre cadeias de caracteres, etc.

A maioria das ferramentas para mapeamentos de ontologias usa alguma espécie de informação de definição ou de estrutura para descobrir os mapeamentos. Estas informações incluem determinados elementos, tais como relacionamentos subclasse-superclasse, domínios e limites de propriedades, análise da estrutura do grafo da ontologia, etc. Algumas ferramentas desta categoria incluem QOM (EHRIG; STAAB, 2004), Similarity Flooding (MELNIK; GARCIA-MOLINA; RAHM, 2002) e ferramentas Prompt (NOY; MUSEN, 2003).

Entrada de informação pelo usuário é uma outra importante fonte. Pesquisadores da área acreditam que o processo de mapeamento de ontologias completamente automático está além de ser alcançado e, portanto alguma interação humana é requerida (FREITAS; STUCKENSCHMIDT; NOY, 2005; NOY, 2004; KALFOGLOU; SCHORLEMME, 2003; MCGUINNESS et al., 2000; MITRA; WIEDERHOLD; KERSTEN, 2000; SHETH; LARSON, 1990). Esta interação pode incluir o envio do algoritmo de mapeamento com um conjunto inicial de pares correspondentes, verificar a comparação que um algoritmo produziu ou configurar os comparadores específicos usados (NOY; MUSEN, 2003; MCGUINNESS et al., 2000; MITRA; WIEDERHOLD; KERSTEN, 2000).

Várias fontes externas disponíveis na forma eletrônica fornecem informações úteis para a descoberta de mapeamentos. O algoritmo S-match (GIUNCHIGLIA; SHVAIKO; YATSKEVICH, 2004), por exemplo, usa anotações da WordNet (MILLER, 1995) para ajudar a reconhecer os mapeamentos entre ontologias. WordNet é uma base de dados léxica eletrônica da língua inglesa projetada para ser utilizada por programas.

O trabalho de pesquisa em (KALFOGLOU; SCHORLEMME, 2003), fornece uma revisão completa e extremamente compreensiva do estado da arte sobre mapeamento de ontologias (NOY, 2004). Esta revisão analisou 35 trabalhos relacionados com mapeamento de ontologias e, após análise pragmática, comenta que o uso de heurísticas é a técnica mais empregada no processo de mapeamento. Ele ainda afirma que tal constatação não é surpreendente, visto que heurísticas são fáceis de desenvolver e de automatizar. Contudo, mesmo as heurísticas mais bem construídas são facilmente anuladas. Em sua pesquisa, Kalfaglou & Schorlemme destacaram um trabalho que após analisá-lo identificaram falhas na heurística empregada ao executar um caso similar ao qual o autor do trabalho garantia a sua corretude. As falhas encontradas na heurística desenvolvida em (MITRA; WIEDERHOLD, 2002) é comum a trabalhos que utilizam heurísticas. Isto ocorre, pois todas as heurísticas estudadas apóiam-se em características sintáticas, dicas lingüísticas e similaridades estruturais. Por outro lado, notou-se que nenhum dos trabalhos analisados usou as semânticas pretendidas dos conceitos a serem mapeados.

Kalfoglou & Schorlemme comentam que geralmente estas semânticas não conseguem ser capturadas por um formalismo base, sendo desta maneira necessário um especialista humano para dar o seu significado preciso. Muitos trabalhos na área escondem a hipótese de que a intervenção de um ser humano é altamente bem vinda. Os proponentes desta abordagem afirmam que este humano deveria ser o componente central do sistema, exercendo os seguintes papéis: validar e confirmar resultados de ferramentas de mapeamentos semi-automáticos, atualizar regras de mapeamento e inspecionar as ontologias e os domínios de entrada (KALFOGLOU; SCHORLEMME, 2003).

Sheth & Larson (SHETH; LARSON, 1990) argumentam que a automação completa não é possível quando se necessita de mais informações do que atualmente é fornecido pelos esquemas das bases de dados. Desta forma, não há captura adequada do mundo real a ser modelado devido à falta de similaridades estruturais entre esquemas ou dados ilustrativos em aplicações alvo, tornando a automatização do processo de *matching* ou de integração difícil na prática. Além disso, Sheth & Larson argumentam que a automação total implica em explosão combinatória, visto que estes processos sofrem de um crescimento exponencial do número de mapeamentos

possíveis.

Kalfoglou & Schorlemme aproveitam o trabalho de pesquisa de Sheth & Larson para mostrar o alto grau de complexidade na automação do processo de integração semântica de ontologias visto que é fato que as próprias ontologias são mais complexas que os esquemas de base de dados.

Ehrig & Sure em (EHRIG; SURE, 2004) consideram que os métodos de similaridade entre entidades (rótulos, URIs, instâncias, propriedades, instâncias conectadas entre outros) são a base para o mapeamento de ontologias. Baseado nisso, eles propuseram uma metodologia para combinar diferentes medidas de similaridade para encontrar candidatos de mapeamento entre duas ontologias. O objetivo é encontrar resultados melhores de mapeamentos a partir de abordagens inteligentes que combinem diferentes medidas de identificação de similaridades, ao invés da utilização de somente uma única medida como foi realizado em outras pesquisas. Estes métodos foram determinados através de regras que foram codificadas manualmente por especialistas em ontologias. No total foram codificadas 15 regras (R1 a R15).

A metodologia empregada utilizou cinco estratégias para combinar os métodos de similaridade (EHRIG; SURE, 2004). Para permitir que as comparações não ocorram somente entre a série de testes realizados em (EHRIG; SURE, 2004), mas também com a literatura existente, Ehrig & Sure focaram no uso de métricas padrões de retorno de informação. As métricas utilizadas foram *precision* e *recall* (MAULDIN, 1991), que foram adaptadas pelos autores para se ajustarem ao cenário de avaliação.

Os resultados dos testes mostraram que as duas métricas utilizadas alcançaram os seus maiores valores com a estratégia mais complexa. Adicionalmente, os resultados permitiram afirmar que há melhora significativa no mapeamento com o aumento da complexidade das estratégias empregadas. No entanto, Ehrig & Sure destacam que mesmo a abordagem tendo alcançado bons resultados em comparação a outros na literatura, cerca de 80%, os resultados não chegam próximo do ideal. A justificativa apresentada é que o processo de mapeamento totalmente automático e de ajuste sobre este processo pode trazer resultados errados que comprometam

o valor de todo o processo de mapeamento. Os autores comentam ainda que o mapeamento semi-automático, há interação de humanos no processo, é uma abordagem comum e indicada para contornar este problema. Outra dificuldade é o problema geral de fazer comparações, especialmente com ontologias grandes, onde a complexidade dos cálculos de similaridades podem crescer dramaticamente. Na abordagem empregada, os autores comentam que esperam uma complexidade da ordem de O(n^2 x log^2(n)), sendo *n* o número de entidades a serem comparadas. A complexidade calculada é derivada de: O(log(n)) para acesso à entidade, O(log(n)) para a complexidade do método e O(n^2) para a comparação de todos os pares possíveis (EHRIG; SURE, 2004).

4 *Trabalhos Correlatos*

O emprego de ontologias em grids computacionais para realizar *matching* semântico é recente. Como observado em (LOPES, 2005), os trabalhos da área datam em sua maioria após o ano 2001. No entanto, a cada ano o número de publicações a respeito vem crescendo. O uso de ontologias vem do fato que atualmente têm-se muitos dados, porém pouco se sabe a respeito deles (CONGIUSTA et al., 2004). A seguir, apresentamos de forma resumida os trabalhos de pesquisa na área e relacionados que esta dissertação se baseou.

4.1 Abordagem de Tangmunarunkit, Decker & Kesselman (OMM)

Tangmunarunkit e outros (TANGMUNARUNKIT; DECKER; KESSELMAN, 2003) propuseram em sua pesquisa o OMM (Ontology-based Matchmaker), que é um serviço selecionador de recursos para ambientes de grid. Este sistema procura ser flexível e extensível para solucionar o problema de *matching* de recursos em ambientes de grid usando tecnologias *Web* Semântica, usufruindo-se de ontologias e regras baseadas em F-Logic (KIFER; LAUSEN; WU, 1995) e na lógica de Horn (LLOYD, 1987). Para isso, foi desenvolvido um protótipo de um *matchmaker* de recursos de grid baseado em ontologias. Além de utilizar ontologias emprega conhecimento prévio de domínio e regras de comparação para resolver o problema de *matching* em ambientes de grid.

Uma das justificativas apresentadas pelos autores para o desenvolvimento do OMM é o fato dos sistemas tradicionais de seleção de recursos, como, por exemplo, Condor Matchmaker (RAMAN; LIVNY; SOLOMON, 1998) e Portable Batch System (GROUP, 2006), executarem o *matching*

baseado na simetria de atributos (mesma sintaxe). Nestes sistemas, os valores dos atributos publicados pelos recursos são comparados com aqueles requeridos pelas tarefas ou aplicações. Para que a comparação seja significativa e efetiva, os fornecedores e consumidores de recursos necessitam estabelecer um acordo prévio quanto aos atributos e valores a serem usados para que o *match* exato possa ocorrer. A comparação sintática exata e o estabelecimento de acordo entre fornecedores e consumidores tornam tais sistemas inflexíveis e difíceis de estender a novas características ou conceitos, à medida que o número de atributos dos recursos no grid cresce, podendo torná-los não gerenciáveis (TANGMUNARUNKIT; DECKER; KESSELMAN, 2003). Além disso, os autores argumentam que em ambientes heterogêneos multi-institucionais, como é o caso dos ambientes de grid, é difícil forçar que as descrições dos recursos compartilhados neles tenham a mesma sintaxe e semântica.

Nesta abordagem foram criadas ontologias que representam os seguintes domínios: recursos, políticas e pedidos. Estas ontologias foram modeladas de maneira independente e descritas sintática e semanticamente utilizando a linguagem RDF-Schema. A comparação entre as ontologias é realizada através de regras de comparação, denominadas de *matchmaking rules*. Os autores não informam quem é responsável em criar estas regras. No entanto, devido a enorme quantidade de combinações e tipos de restrições que os pedidos de recursos podem apresentar, acreditamos que as regras devem ser criadas pelos usuários conforme o seu pedido de recurso.

A linguagem TRIPLE (DECKER; SINTEK, 2002) foi utilizada para representar as regras de comparação. TRIPLE foi escolhida por ser especialmente projetada para consulta, transformação e raciocínio com dados RDF. TRIPLE é baseada em lógica de Horn e possui várias características da lógica F-Logic. Embora a linguagem TRIPLE não possua suporte nativo para linguagens de representação de conhecimento, ela pode ser configurada por axiomas para suportar linguagens de modelagem,como por exemplo, RDF-Schema.

O *matchmaker* desenvolvido consiste de três componentes:

- **Ontologias**: capturam o modelo do domínio e o vocabulário para expressar as características dos recursos e os seus pedidos;

- **Conhecimento Prévio de Domínio**: capturam conhecimento adicional sobre o domínio;
- **Regras de Comparação**: definem quando um recurso satisfaz um pedido.

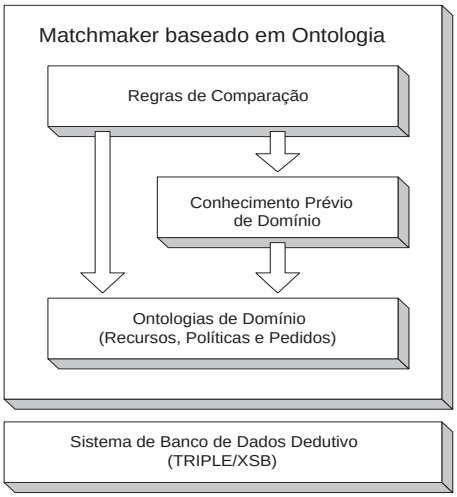


Figura 19: Ontology-based Matchmaker (OMM)

A Figura 19 mostra o relacionamento entre os três componentes que integram o serviço de *matchmaking*.

O módulo Conhecimento Prévio de Domínio captura conhecimento adicional sobre as ontologias de domínio (usualmente a nível de instância) que não é capturado pelas ontologias (TANGMUNARUNKIT; DECKER; KESSELMAN, 2003). Este conhecimento é usado durante o processo de seleção, sendo este capturado em termos de regras. Estas regras usam os vocabulários definidos nas ontologias para acrescentar axiomas que não puderam ser expressos pela linguagem de ontologia utilizada e que geralmente influência o raciocínio sobre as instâncias. A Figura 20 mostra algumas regras de conhecimento prévio escritas para o OMM. As regras definem quais sistemas operacionais são compatíveis e definem *compatibleWith* como transitivo, reflexivo e simétrico. Também definem *substitutes* em função de *compatibleWith* significando que sistemas operacionais podem ser substituídos por outro. Um exemplo de conhecimento capturado é o de sistemas operacionais, como descrito na Figura 20. O conhecimento modelado informa que os sistemas operacionais Debian e Red Hat são da família Linux, SunOS da família Unix e que um Linux é considerado um Unix.

```
@gridBackground { // specifies grid background knowledge
    Linux[rdfs:subClassOf->GR:OperatingSystem] .
    Unix[rdfs:subClassOf->GR:OperatingSystem] .
    Debian[rdf:type->Linux]. Redhat[rdf:type->Linux] .
    SunOS[rdf:type->Unix]. Linux[rdf:type->Unix] .

    // transitivity axiom
    FORALL X,Y,Z X[compatibleWith->Z]<- X[compatibleWith->Y] AND Y[compatibleWith->Z] .

    // identity axiom
    FORALL X X[compatibleWith->X].

    //symmetry axiom
    FORALL X,Y X[compatibleWith->Y]<- Y[compatibleWith->X] .

    FORALL X,Y,Z X[substitutes->Z] <- (Y[rdf:type->Z]  and
        X[substitutes->Y]) or X[compatibleWith->Z]  .
}
```

Figura 20: Conhecimento prévio do domínio de sistemas operacionais definido no OMM

O módulo Regras de Comparação define regras que restringem a comparação semântica entre pedidos e recursos. A linguagem TRIPLE pode raciocinar sobre as propriedades dos objetos e entre os seus relacionamentos descritos nas ontologias de domínios e nas informações capturadas pelo módulo Conhecimento Prévio de Domínio, como ilustrado pelas setas na Figura 19. A Figura 21 descreve as seguintes entradas requeridas nas regras: o conjunto de informações publicadas dos recursos (representado por *Data*), conhecimento prévio (*Background*) e ontologia de domínio (*Ontology*). A primeira regra define a propriedade *matches* que diz quando um *JobRequest* é compatível com uma publicação de um *ComputerSystem*. Esta regra é definida em função de outras regras, por exemplo, *matchesOS* e *matchesFS*.

```
FORALL Data, Background @match(Data,Background,Ontology)  {
  FORALL X,Y X[matches->Y] <-
            X [rdf:type->GR:JobRequest]@Data
        and  Y [rdf:type->GR:ComputerSystem]@rdfschema(Data,Ontology)
        and   ((X.GR:RequestResource.GR:RequiredMemory)@Data)[matchesMEM->(Y.GR:RunningOS)@Data]
        and   ((X.GR:RequestResource.GR:RequiredOS)@Data)[matchesOS->(Y.GR:RunningOS)@Data]
        and   ((X.GR:RequestResource.GR:RequiredFS)@Data)[matchesFS->(Y.GR:HostedFileSystem)@Data]
        and   ((X.GR:RequestResource.GR:RequiredCPU)@Data)[matchesCPU->Y] .

 // checking OperatingSystem requirement
  FORALL X,Y X[matchesOS->Y] <-
            X [rdf:type->GR:OSRequirement]@Data
        and Y [rdf:type->GR:OperatingSystem]@Data
        and  ((X.GR:OSType)@Data)[substitutes->(Y.GR:OSType)@Data]@Background.

// checking FileSystem Requirement
 FORALL X,Y X[matchesFS->Y] <-
            X [rdf:type->GR:FSRequirement]@Data
       and Y [rdf:type->GR:FileSystem]@rdfschema(Data,Ontology)
       and (X.GR:MinDiskSpace)@Data =< (Y.GR:AvailableSpace)@Data  .
}
```

Figura 21: Parte das regras de comparação definidas para o OMM

As características apresentadas pelo OMM são:

- **Descrição assimétrica de recursos e pedidos**: As descrições dos recursos e pedidos são modelados em ontologias distintas. Comparações semânticas entre as duas ontologias são fornecidas entre eles;

- **Restrições Bilaterais**: Tanto o recurso quanto o pedido podem independentemente especificar as suas restrições. A descrição de um pedido pode especificar restrições em termos dos requisitos computacionais enquanto o recurso em termos de políticas de acesso e de uso;

- **Capacidade de expressar preferência de *matching***: O pedido pode especificar preferência quando vários recursos atendem as restrições definidas no pedido;

- **Checagem de Integridade**: O *matchmaker* utiliza-se do conhecimento do domínio para checar se os pedidos enviados possuem inconsistências. Como exemplo de inconsistência pode-se citar quando um usuário especifica no pedido que deseja recursos que executem sistema operacional "Windows 2000" e tenham processadores da família "SPARC";

- **Flexibilidade e Extensibilidade**: Novos termos podem ser facilmente acrescentados dentro de uma ontologia. O *matchmaker* é capaz de suportá-los somente criando novas regras de comparação.

4.2 Lopes, Melo, Dantas & Ralha

Lopes et. al em (LOPES et al., 2006) propuseram um mecanismo de *matching* semântico com o objetivo de estender a proposta apresentada em (HEINE; HOVESTADT; KAO, 2004), que foi restringida para facilitar a ampliação do conhecimento. Esta ampliação foi alcançada atribuindo equivalências entre conceitos (classes) de diferentes ontologias que são representados por termos com a mesma sintaxe para posteriormente serem unidos em uma única ontologia. Igualmente em (HEINE; HOVESTADT; KAO, 2004), Lopes et. al utilizaram-se da união de múltiplas ontologias para construir automaticamente uma ontologia global, onde a pesquisa por recursos é realizada. Uma justificativa apresentada por Lopes et. al e compartilhada por (HEINE;

HOVESTADT; KAO, 2004) é a falta de uma padronização na descrição de recursos de grid.

No processo de criação da ontologia global, Lopes et. al acrescentaram as seguintes funcionalidades à proposta apresentada em (HEINE; HOVESTADT; KAO, 2004):

- Adição de relações de equivalência entre conceitos (classes) com sintaxe distinta, devido a diferenças entre letras maiúsculas e minúsculas, visto que se utilizou a linguagem OWL que é sensível ao caso;
- Acréscimo de relações de equivalência entre conceitos (classes), através do uso de um dicionário de sinônimos;
- União de classes de recursos com sintaxes distintas, porém com mesmo significado;
- União das propriedades das classes com a mesma sintaxe.

Em (HEINE; HOVESTADT; KAO, 2004) foi criado um mecanismo de união de ontologias através da união de nodos dos grafos, ou seja, união dos conceitos (classes), conceitos estes com idêntica sintaxe. A abordagem de Lopes et. al (LOPES et al., 2006) além de unir conceitos com mesma sintaxe utilizaram-se de um banco de dados com sinônimos para permitir que conceitos com sintaxe distinta, todavia com mesmo significado, possam ser considerados semanticamente equivalentes, como ilustrado na Figura 22.

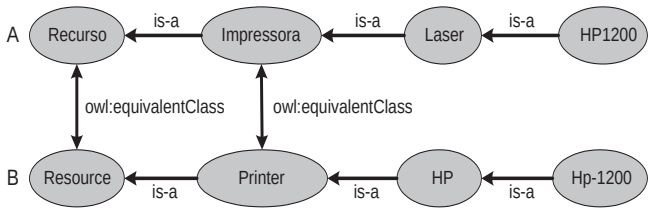


Figura 22: Ontologia global expressando equivalências entre as classes Recurso-Resource e Impressora-Printer, provenientes das ontologias A e B

Como citado em (LOPES, 2005), conceitos descritos em idiomas diferentes como *PersonalComputer* e *ComputadorPessoal* e até abreviações, como por exemplo, *memória* e *mem*

podem ser inseridos como sinônimos no banco de dados. A inclusão destes sinônimos na ontologia global, criada a partir da união de outras ontologias, ocorre utilizando a propriedade *owl:equivalentClass*, presente na linguagem OWL, que é usada para indicar que duas classes são equivalentes. Na Figura 23, é ilustrado a descrição da ontologia global após o processo de união das ontologias A e B mostradas na Figura 22, destacando a propriedade de equivalência de classes OWL definidas entre as classes Printer-Impressora e Recurso-Resource.

```
1..9
10      <owl:Class rdf:ID="Printer">
11          <rdfs:subClassOf>
12              <owl:Class rdf:ID="Resource"/>
13          </rdfs:subClassOf>
14          <owl:equivalentClass>
15              <owl:Class  rdf:ID="Impressora"/>
16          </owl:equivalentClass>
17      </owl:Class>
18      <owl:Class rdf:about="#Impressora">
19          <rdfs:subClassOf rdf:resource="#Recurso"/>
20          <owl:equivalentClass rdf:resource="#Printer"/>
21      </owl:Class>
22      <owl:Class rdf:ID="Recurso">
23    <owl:equivalentClass>
24          <owl:Class rdf:about= "#Resource"/>
25    </owl:equivalentClass>
26      </owl:Class>
27      <owl:Class rdf:about="#Resource">
28          <owl:equivalentClass rdf:resource="#Recurso"/>
29      </owl:Class>
30      <owl:Class rdf:ID="Laser">
31          <rdfs:subClassOf rdf:resource="#Impressora"/>
32      </owl:Class>
33      <owl:Class rdf:ID="HP">
34          <rdfs:subClassOf rdf:resource="#Printer"/>
35      </owl:Class>
36      <owl:Class rdf:about="#Impressora">
37          <rdfs:subClassOf rdf:resource="#Recurso"/>
38      </owl:Class>
39      <owl:Class rdf:ID="Hp-1200">
40          <rdfs:subClassOf rdf:resource="#HP"/>
41      </owl:Class>
42      <owl:Class rdf:about="#HP1200">
43          <rdfs:subClassOf rdf:resource="#Laser"/>
44      </owl:Class>
...
```

Figura 23: Ontologia global resultante da união das ontologias A e B

Portanto, admitindo que um fornecedor de recursos A modele os seus recursos em português e outro fornecedor (B) em inglês e caso o banco de dados de sinônimos seja abrangente o suficiente, a pesquisa por um recurso Impressora obterá instâncias de ambas as classes, *Impressora* e *Printer* (LOPES, 2005).

A pesquisa sobre a ontologia global em busca de recursos de grid é executada pelo usuário através de consultas elaboradas na linguagem RDQL. RDQL baseia-se na propriedade *owl:equivalentClass* para retornar instâncias de classes consideradas equivalentes.

4.3 Casare & Sichman

Casare & Sichman propuseram em (CASARE; SICHMAN, 2005) a Ontologia Funcional de Reputação (OFR) com o intuito de ser usada como um conhecimento comum sobre reputação compartilhado por agentes em um Sistema Multi-Agentes (SMA). A OFR agrega conhecimento amplo sobre reputação produzido em algumas áreas de interesse tais como Psicologia e Inteligência Artificial. Os autores observaram que existe um grande número de trabalhos sobre reputação de agentes, contudo cada trabalho define os seus próprios conceitos básicos do domínio. Avançando em suas observações, eles afirmam que algumas vezes diferentes significados são associados ao mesmo termo (caracterizando um homônimo perfeito (Wikipédia, 2006)) e em outras ocasiões o mesmo significado é relacionado a diferentes termos, ou seja, sinônimos.

Casare & Sichman declaram também que o conhecimento de reputação estruturado como uma ontologia pode ser usada para permitir a integração a nível semântico envolvido na interoperação de agentes de software utilizando diferentes modelos de reputação.

O objetivo da OFR como uma ontologia comum, é dar suporte para a integração semântica entre agentes de software utilizando diferentes modelos de reputação (CASARE; SICHMAN, 2005). Os modelos de reputação utilizados foram: Cognitive Reputation Model (CONTE; PAOLUCCI, 2002), Tipology of Reputation (MUI; HALBERSTADT; MOHTASHEMI, 2002) e ReGret System (SABATER, 2003).

Com o intuito de permitir a integração visando a interoperabilidade dos diversos agentes em um SMA, Casare & Sichman basearam-se nas três abordagens de integração citadas em (VISSER et al., 2000), maiores detalhes consultar seção 3.7.1. Analisando as abordagens na ótica dos agentes de uma SMA, os autores chegaram às seguintes conclusões (CASARE; SICHMAN, 2005):

- A adoção da abordagem centralizada não é uma escolha adequada, visto que ela considera uma única visão sobre o domínio reputação e a idéia principal é considerar os diferentes modelos de reputação que os agentes em um SMA pode apresentar;

- Por outro lado, uma abordagem puramente descentralizada levaria a outro problema. Visto que uma ontologia global mais abstrata não existe, deve-se fornecer para cada *n* modelos diferentes de reputação (*n-1*) processos de alinhamento com o objetivo de permitir interoperabilidade;

- A abordagem híbrida, como a descentralizada, não limita a diversidade de modelos de reputação em um ambiente SMA heterogêneo, visto que cada modelo opera com a sua ontologia particular. Todavia, esta abordagem não envolve procedimento complexo de alinhamento de ontologias, pois os vocabulários destas diferentes ontologias são relacionados somente aos da OFR. Desta forma, é preciso de *n* processos de alinhamento para permitir interoperabilidade.

Casare & Sichman optaram pela abordagem híbrida, pois esta permite a integração semântica de agentes usando diferentes modelos de reputação de forma menos complexa, visto que novas fontes de informação podem ser facilmente integradas e comparadas às outras fontes de informação. Adicionalmente, esta abordagem é extensível, já que a OFR desenvolvida permite estender outras noções relacionadas à reputação, como por exemplo, Trust (CASTELFRANCHI; FALCONE; PEZZULO, 2003).

Os três passos seguidos para realizar o alinhamento semântico das diferentes ontologias de reputação no SMA foram (CASARE; SICHMAN, 2005):

- **1º Passo**: Coletar informações sobre o modelo de reputação a ser integrado, com o intuito de identificar os conceitos principais relacionados à reputação;

- **2º Passo**: Definir estes conceitos em OWL DL com o objetivo de criar uma ontologia;

- **3º Passo**: Relacionar o vocabulário desta ontologia ao da ontologia comum, OFR.

O segundo passo consistiu na realidade em criar uma ontologia para cada um dos três modelos estudados, devido a estes modelos de reputação a princípio não serem descritos em termos ontológicos. Os autores comentam que o terceiro passo pode ser suportado por ferramentas

semi-automáticas de alinhamento de ontologias, como o PROMPT, com o objetivo de acelerar o processo de relacionamento dos vocabulários. Adicionalmente, os autores comentam que o processo de relacionamento deve envolver um operador humano que tome a decisão final sobre se aceita ou rejeita as correspondências produzidas pela ferramenta ou se edita correspondências que não foram reconhecidas pela ferramenta. Contudo, Casare & Sichman não utilizaram o PROMPT, definindo os relacionamentos manualmente.

Vale ressaltar que tanto a ontologia OFR quanto as ontologias que descrevem os três modelos de reputação modelados ontologicamente foram descritos no mesmo idioma, o Inglês.

4.4 Quadro Comparativo

A Tabela 4 apresenta as características de cada um dos trabalhos analisados.

Os trabalhos analisados, Tangmunarunkit et. al (TANGMUNARUNKIT; DECKER; KESSELMAN, 2003) e Lopes et. al (LOPES et al., 2006), propuseram mecanismos de *matching* semântico de recursos de grid, o primeiro baseado em regras de comparação e o último empregando um dicionário de sinônimos.

O primeiro trabalho, apesar de argumentar que seja difícil de forçar uma descrição sintática e semântica de recursos entre as diferentes organizações que integram um grid, construiu o seu mecanismo baseado em uma única forma de descrever o domínio dos recursos no ambiente de grid, ou seja, uma única visão sobre os recursos (uma ontologia de recursos e de políticas). A pesquisa por recursos é realizada através de pedidos definidos na ontologia de pedidos, que tem o seu vocabulário previamente relacionado com os vocabulários da ontologia de recursos e de políticas através de regras de comparação. Além das relações de equivalência entre termos definidos em ontologias distintas, as regras estabelecem as restrições que os valores atribuídos aos termos dos vocabulários da ontologia de recursos e de política devem atender. Um ponto negativo desta abordagem é a obrigatoriedade de a cada novo pedido de recurso, o usuário se ver obrigado a criar uma nova regra de comparação para que expresse adequadamente as novas restrições no pedido. Outro ponto negativo é a linguagem de especificação de ontologias

Tabela 4: Quadro comparativo dos trabalhos analisados

	Tangmunarunkit et. al	**Lopes et. al**	**Casare & Sichman**
Voltado a grids	Sim	Sim	Não
Objetivo	*Matching* Semântico de Recursos	*Matching* Semântico de Recursos	Interoperabilidade em sistemas multi-agentes
Quantidade de fontes de informação	Várias	Várias	Várias
Representação das fontes de informação	Uma única ontologia para as fontes de informação	Uma ontologia para cada fonte de informação	Uma ontologia para cada fonte de informação
Mecanismo de consulta	Baseado em regras de comparação	Baseado na linguagem RDQL	-
Verificação de Consistência	Sim (consistência das consultas)	Sim (consistência das informações dos recursos após a união das ontologias)	Não
Mecanismo inferência	TRIPLE/XSB	PELLET	RACER
Ano de publicação	2003	2006	2005
Ontologia(s) Disponível(is)	Não	Sim	Não
API para manipular ontologias	-	Jena	-
Linguagem de ontologia	RDF(S)	OWL	OWL

utilizada, RDF(S), visto que ela é pouca expressiva, não sendo capaz de expressar relações complexas entre conceitos.

Entre os dois trabalhos de *resource matching* pesquisados, o segundo se mostrou mais realista, pois considera as diferentes visões das organizações sobre os seus recursos compartilhados no grid. No entanto, a união de várias ontologias através das equivalências semânticas entre conceitos (classes), para automaticamente criar uma única ontologia que represente essas equivalências é limitada. A limitação ocorre, pois uma busca mais detalhada por recursos é geralmente realizada sobre suas características específicas que naturalmente foram modeladas como propriedades dessas classes e não como classes. Como mencionado, Lopes et. al em sua

abordagem considera a equivalência de propriedades de classes somente se possuírem a mesma sintaxe, o que limita a sua proposta. O sistema é validado elaborando consultas que buscam por recursos através somente dos termos que representam os conceitos da ontologia global. A abordagem utiliza a linguagem OWL, considerada dentre as linguagens de especificação de ontologias baseada na *Web* a mais expressiva.

Na linguagem OWL, é possível declarar que duas propriedades são equivalentes através da propriedade *owl:equivalentProperty*. Duas propriedades (atributos) pertencentes a classes diferentes são consideradas equivalentes em OWL se estas classes são equivalentes, caso contrário há incoerência (MCGUINNESS; HARMELEN, 2004).

Como exemplo da dificuldade de expressar a equivalência entre ontologias utilizando dicionário de sinônimos, consideremos duas ontologias a serem integradas, onde na primeira foi definida a classe *MainMemory* com as seguintes propriedades (atributos): *physical_memory_size* e free_physical_memory; e na segunda, é definida a classe *OperatingSystem* que tem as propriedades *os_type*, *main_memory_size* e *free_main_memory*. As duas classes não são equivalentes, visto que os termos que as representam não são sinônimos, implicando que as duas classes conceituam entidades distintas. No entanto, as propriedades *physical_memory_size* e *free_physical_memory* são equivalentes às propriedades *main_memory_size* e *free_main_memory* de acordo com o dicionário. Isto pode ocorrer, pois o desenvolvedor da última ontologia, ao modelá-la, considerou o fato de que sistemas operacionais informam a capacidade de memória principal e a quantidade de memória principal livre no sistema computacional e, portanto, indiretamente, apresentam essas características. Por outro lado, o desenvolvedor da primeira ontologia modelou essas características como pertencentes ao conceito memória principal. Este exemplo ilustra de forma simples a dificuldade encontrada para expressar as equivalências semânticas corretamente devido às diferentes interpretações do domínio. Como mencionado acima, não é possível expressar as equivalências apontadas acima em OWL devido às propriedades equivalentes pertencerem a conceitos distintos, gerando incoerências.

Situações similares, como descritas no exemplo acima, não permitem que ferramentas ba-

seadas em informações de definição ou de estrutura capturem automaticamente essas correspondências e as expresse formalmente, através de axiomas, fazendo com que haja perda de informações e conseqüentemente interoperabilidade. Nestes casos, deve-se expressar as relações de outra forma, sem utilizar axiomas que denotam equivalência, como é o caso das propriedades *owl:equivalentClass* e *owl:equivalentProperty*. Além disso, em um ambiente onde podem existir ontologias que descrevem um mesmo domínio, porém escritas em diferentes idiomas, tornam ainda mais complexo, moroso e impreciso o emprego de ferramentas automáticas para encontrar as relações de equivalência, pois precisam capturar similaridades entre termos baseados em várias lingüísticas para junto com o seu contexto analisarem se são equivalentes.

Outra limitação da abordagem de Lopes et. al diz respeito ao alto grau de conhecimento exigido dos futuros usuários do sistema de *matching* semântico proposto. Os usuários, para pesquisarem recursos no sistema, precisam formular consultas utilizando a linguagem RDQL. Para que a consulta seja significativa e efetiva, o usuário necessita estar familiarizado com a sintaxe RDQL, como se referir a cada vocábulo descrito na ontologia a ser consultada, bem como percorrer corretamente sua estrutura formal baseado no formato <sujeito,predicado,objeto>. Como já mencionado, não é de se esperar que qualquer usuário lide facilmente com o formalismo empregado para representar o conhecimento de um domínio. Além disso, as linguagens de consulta RDQL e SPARQL não permitem inferência, impedindo a expansão das consultas.

O trabalho de Casare & Sichman baseou-se na abordagem híbrida para a integração semântica de vários modelos de reputação. Para superar a limitação de não poder reutilizar ontologias de reputação, Casare & Sichman propuseram que os alinhamentos semânticos fossem realizados entre as diferentes ontologias de reputação a serem integradas com a ontologia global de reputação desenvolvida. Dessa forma, as ontologias de reputação não precisam ser remodeladas para usarem somente os termos (ou suas combinações) de um vocabulário compartilhado ou de uma ontologia global. A abordagem de Casare & Sichman mostra-se menos complexa em relação à abordagem descentralizada, pois precisa-se de somente *n* alinhamentos semânticos, sendo *n* o número de ontologias a serem integradas, enquanto na descentralizada é necessário *n* vezes (*n*-1) alinhamentos para tornar todas as ontologias interoperáveis. A forma de alinha-

mento realizado assemelha-se à abordagem proposta em (GRUNINGER; KOPENA, 2005), apesar desta semelhança não ter sido citada pelos autores. Em adição, os autores ainda comentam que nos processos de alinhamento deve haver a participação de humanos para que todos os possíveis alinhamentos existentes entre as ontologias sejam reconhecidos ou a sua grande maioria, tornando-as mais interoperáveis.

Observamos que as abordagens de Tangmunarunkit et. al e Lopes et. al não levam em conta as características levantadas acima. Portanto, consideramos que uma abordagem contendo estas características seja mais completa e interoperável, e de fácil uso para qualquer usuário. Esta abordagem é apresentada detalhadamente no próximo capítulo e será o foco principal desta dissertação.

5 *Projeto e Implementação do Mecanismo de Matching Semântico baseado na Integração Semântica de Múltiplas Ontologias*

Este capítulo descreve a arquitetura do sistema de *matching* semântico desenvolvido. O sistema foi projetado baseado em uma ontologia global desenvolvida, denominada de Ontologia de Referência (OR), para permitir a integração semântica de várias ontologias de recursos. Adicionalmente, foi desenvolvida a ontologia de pedidos, através da qual são definidas as consultas por recursos grid. Com uma linguagem de consulta construída como ontologia, o processo de *matching* proposto pode verificar a consistência das consultas, ampliar as consultas e permitir que usuários comuns facilmente definam quais recursos desejam pesquisar no sistema. O mecanismo de *matching* semântico abordado nesta dissertação procura diminuir as limitações apontadas nas abordagens citadas no final do capítulo anterior. O objetivo é construir um sistema de *matching* semântico que possibilite consultar as diferentes visões que os recursos compartilhados em um ambiente de grid podem apresentar de uma forma mais completa e interoperável. Mais completo, pois consideramos não somente relações de equivalência entre conceitos, mas também entre propriedades (atributos) desses conceitos; e mais interoperável porque o processo de alinhamento semântico é realizado por humanos, capazes de executar abstrações e raciocínios mais complexos que permitam identificar a maioria das equivalências semânticas existentes entre termos de diferentes ontologias. Adicionalmente, busca-se abstrair do usuário a necessidade de conhecer como traduzir a estrutura do conhecimento modelado nas ontologias em um formalismo lógico para ser empregado nas consultas sobre as ontologias. O

sistema proposto tem as seguintes características:

- A OR construída define os conceitos, vocabulários e relacionamentos fundamentais do domínio grid. Esta ontologia tem como função servir de referência para permitir a integração semântica das diferentes ontologias de recursos definidas pelas OVs;
- As equivalências semânticas são estabelecidas entre os termos (que representam classes ou suas propriedades) da ontologia de recursos de uma OV particular com os termos da OR compartilhada. As correspondências são relações binárias entre conceitos (classes) ou entre as propriedades que caracterizam esses conceitos. Estas relações são criadas e informadas pelas OVs durante o processo de integração de suas ontologias perante o sistema;
- Foi construída uma ontologia de pedidos para servir de linguagem de consulta reconhecida pelo *matchmaker* desenvolvido. Os termos definidos na ontologia de pedidos que representam as características de recursos do grid são os mesmos que foram definidos na OR;
- Criamos regras baseadas na estrutura de conhecimento modelado na OR e indivíduos definidos na OR para ampliar consultas por recursos de grid e para checar a consistência das consultas enviadas ao *matchmaker*.

Esta última característica foi inspirada no módulo de conhecimento prévio, modelado por regras, apresentado em (TANGMUNARUNKIT; DECKER; KESSELMAN, 2003).

5.1 Visão Geral

Como mencionado no capítulo 4, existe a necessidade de um sistema de *resource matching* para ambientes de grid que seja capaz de trabalhar com várias ontologias de descrição dos recursos. Essa característica é importante, pois na realidade é difícil de existir um consenso na definição de uma ontologia de recursos única por parte das organizações virtuais, fruto da

alta subjetividade na definição de tal estrutura conceitual. Adicionalmente, constatamos a dificuldade de consultar sistemas baseados em ontologias devido à dificuldade de navegar nas estruturas dos conhecimentos modelados.

O sistema de *matching* semântico proposto nessa dissertação considera a diversidade de visões que as organizações virtuais de um grid podem apresentar através de suas ontologias, como ilustrado na Figura 24. O sistema desenvolvido é baseado em ontologias descritas na linguagem OWL para permitir a integração. Neste trabalho, as ontologias utilizadas e construídas foram especificadas na linguagem OWL, por esta linguagem ser a mais expressiva de todas as linguagens de ontologias baseada na *Web*, além de ser uma recomendação para a construção de ontologias.

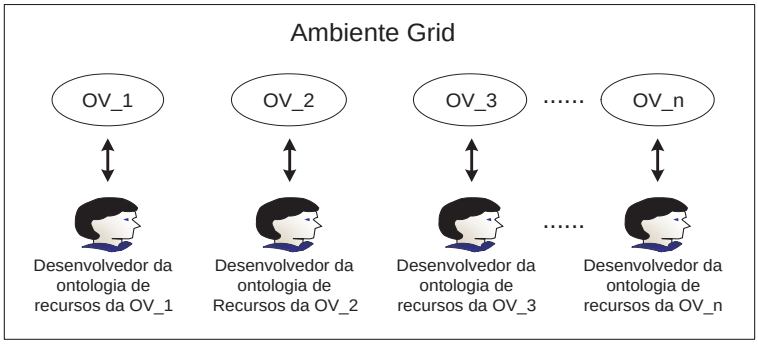


Figura 24: Diferentes ontologias de recursos projetadas pelas organizações virtuais que compõem um grid

Optamos em seguir a abordagem empregada em (CASARE; SICHMAN, 2005), visto que ela traz a menor complexidade no processo de alinhamento de ontologias. Para seguirmos esta abordagem, desenvolvemos primeiramente a Ontologia de Referência. Esta ontologia, como o próprio nome indica, serve de referência para as OVs estabelecerem os relacionamentos semânticos entre os termos de suas ontologias de recursos com os da OR e os publiquem no sistema de *matching* semântico, como ilustrado na Figura 25.

Para o sistema proposto foi desenvolvida outra ontologia, a ontologia de pedidos. Esta ontologia é usada no sistema como uma linguagem de consulta que permite empregar regras de inferência com o objetivo de ampliar as consultas semanticamente e verificar a consistência

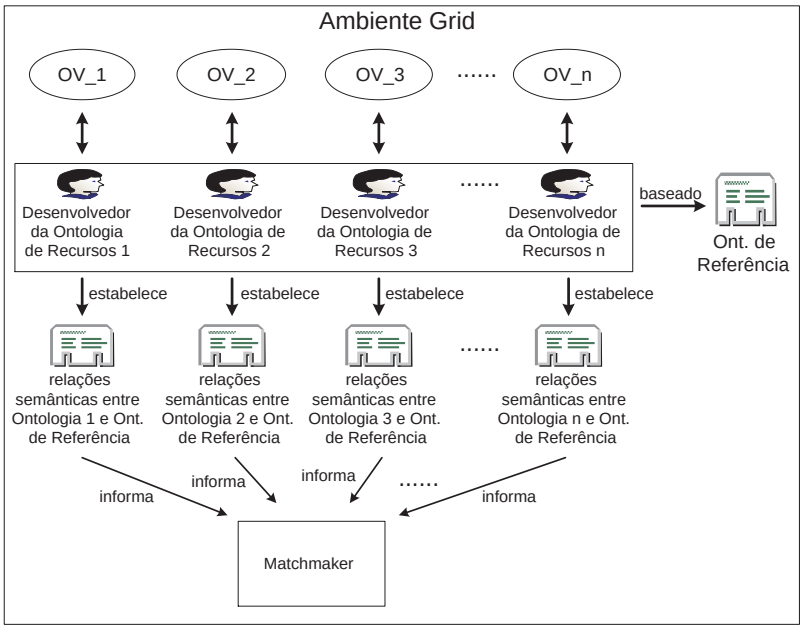


Figura 25: Estabelecendo relações semânticas entre os conceitos presentes nas ontologias de recursos das OVs com a OR e as informando ao *matchmaker* proposto

das consultas. Além disso, qualquer usuário pode encontrar os recursos de grid desejados, sem necessariamente precisar conhecer a estrutura conceitual das ontologias e a sua representação formal. Esta transparência é alcançada, pois no *matchmaker* foi desenvolvido um algoritmo que permite navegar nas diferentes estruturas das ontologias escritas na linguagem OWL sendo necessário conhecer somente as relações semânticas entre as ontologias e a OR, e as características dos recursos que satisfazem as restrições expressas na consulta. Desta forma, a ontologia de pedidos foi projetada utilizando os termos que denotam as características dos recursos definidos na própria OR, como mostrado na Figura 26.

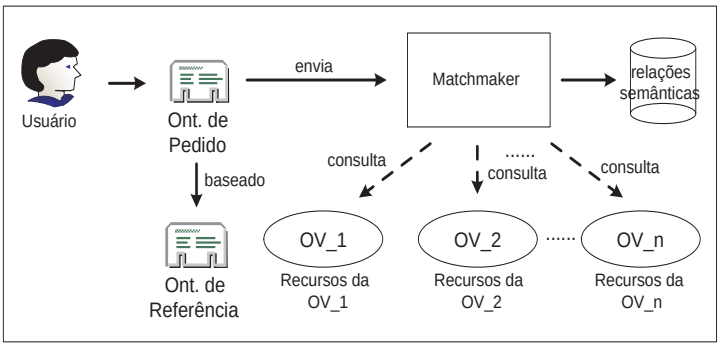


Figura 26: Consultando os recursos das organizações virtuais através da ontologia de pedidos

5.2 Ontologia de Referência e de Pedidos

A Ontologia de Referência desenvolvida nesta pesquisa baseou-se na ontologia Core Grid Ontology (CGO) (XING; DIKAIAKOS; SAKELLARIOU, 2006) e no modelo Common Information Model (CIM) (DMTF, 2006). A ontologia CGO foi projetada para fornecer um *framework* de mais alto nível em que todos os conceitos dos grids podem ser representados de uma forma semanticamente coerente e consistente. CGO foi desenvolvida em OWL para capturar e modelar os conceitos e conhecimentos básicos dos grids (XING; DIKAIAKOS; SAKELLARIOU, 2006), tais como: VO (Virtual Organization), GridMiddleware, GridService, GridApplication e GridResource. A flexibilidade e extensibilidade do CGO permitem que esta seja usada para, entre outros fins, integração de informação do grid, busca de informação, descoberta de recurso e gerenciamento de alocação de recurso (XING; DIKAIAKOS; SAKELLARIOU, 2006).

Tendo em vista o objetivo desta dissertação, a Ontologia de Referência desenvolvida focou em parte na ontologia CGO e foi estendida para modelar as características dos recursos mais comuns em sistemas computacionais. A modelagem destas características baseou-se no modelo CIM. CIM é um esquema conceitual que define os elementos gerenciados em um ambiente de TI, por exemplo, computadores ou redes de armazenamento. CIM usa um modelo baseado em UML para definir o seu esquema, sendo este a base para a maioria dos outros padrões especificados pelo Distributed Management Task Force (DMTF) (DMTF, 2006).

A Ontologia de Referência apresenta três partes principais: (1) conceitos e a hierarquia dos conceitos; (2) as propriedades, caracterizam os conceitos ou estabelecem relações entre eles e (3) metadados que descrevem os conceitos e as suas propriedades utilizando linguagem natural.

Na Ontologia de Referência desenvolvida foram modelados os conceitos básicos relevantes a este trabalho, como ilustrado na Figura 27.

Estes conceitos são representados pelas seguintes classes: *GridMiddleware*, *Policy*, *GridResource* e *GridService*. A classe *GridResource* tem como subclasses diretas as classes *NetworkResource*, *StorageResource*, *ComputingResource* e *Dataset*. Respectivamente elas definem os

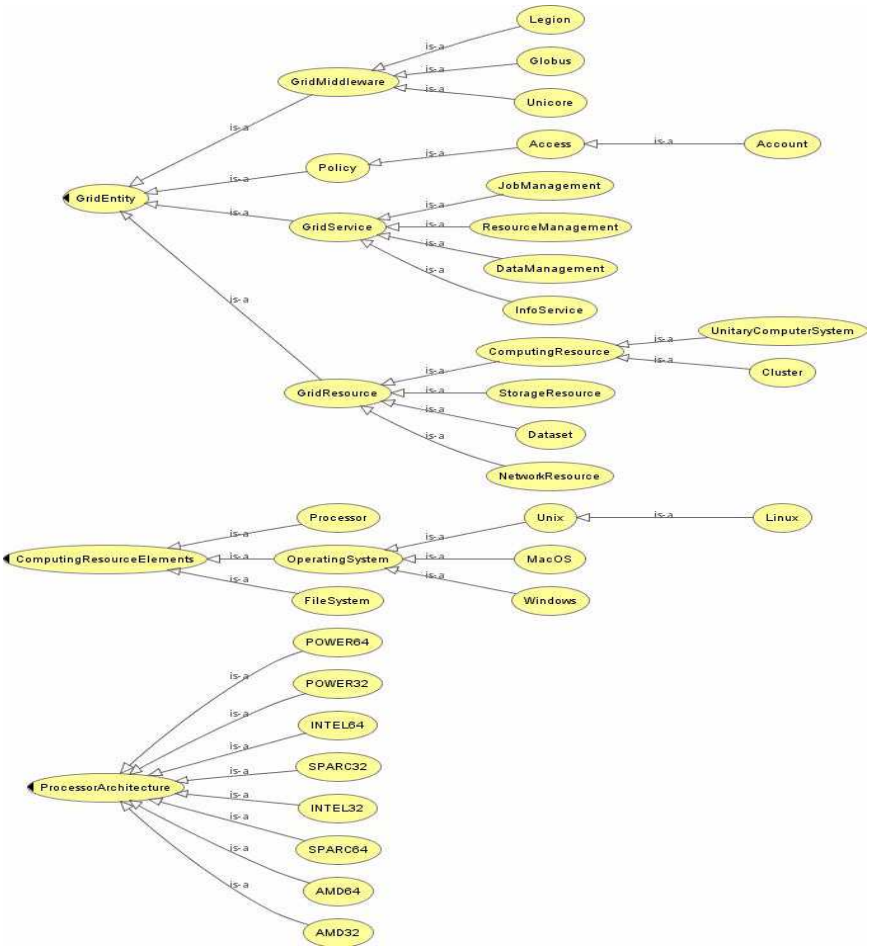


Figura 27: A Ontologia de Referência desenvolvida

recursos de rede, de armazenamento, de computação e conjunto de dados. A classe *ComputingResource* especializa-se em *Cluster* (recurso de computação formado por vários elementos de processamento fisicamente separados por redes de interconexão) e *UnitaryComputerSystem* (recurso de computação único, que não apresenta redes de interconexão entre os seus elementos de processamento).

As subclasses diretas ou indiretas da superclasse *ComputingResourceElements* modelam o conhecimento sobre os principais elementos que constituem recursos computacionais. Estas subclasses foram modeladas baseadas em grande parte no modelo CIM. A Tabela 5 ilustra a classe *OperatingSystem* e algumas das suas propriedades (atributos).

As classes *Cluster* e *UnitaryComputerSystem* apresentam relacionamentos com as subclasses diretas da superclasse *ComputingResourceElements*. Por exemplo, a classe *UnitaryCompu-*

Tabela 5: A descrição da classe *OperatingSystem* e suas propriedades

Classe	Descrição	Propriedades
OperatingSystem	This concept represents the logical resource operating system. Operating system is a manager of resources which integrate a computer, e.g. it manages processors, main memories, I/O devices and files.	os_type, version, free_main_memory, free_virtual_memory, total_main_memory_size

terSystem, ilustrada na Tabela 6, tem a propriedade *running_os* que define a relação entre uma instância desta classe com uma instância da classe *OperatingSystem*.

Tabela 6: Descrição parcial da classe *UnitaryComputerSystem*

Classe	Descrição	Propriedades
UnitaryComputerSystem	This concept represents unitary computer systems, in other words, which do not have interconnection networks among their processing elements. For example: PC, Server, WorkStation and SMP.	hostname, ip_address, number_of_cpus, computer_system_description, running_os, has_processor

As propriedades mais relevantes representando as características das classes e relações entre classes na Ontologia de Referência podem ser visualizadas no Apêndice B. A ontologia de pedidos, ilustrada na Figura 28, foi construída para desempenhar o papel de linguagem de consulta a ser submetida ao *matchmaker* desenvolvido. Ela utiliza os termos definidos na OR que representam as principais características dos recursos. Para que os usuários expressem os requisitos que os recursos do grid devem satisfazer, foram criados conceitos que denotam alguns operadores binários lógicos, como mostrado na Figura 28. Estes operadores indicam ao *matchmaker* como comparar os valores atribuídos aos termos na ontologia de pedidos com os valores atribuídos aos termos que representam as características dos recursos de grid modelados nas ontologias de recursos das OVs durante o processo de *matching* semântico.

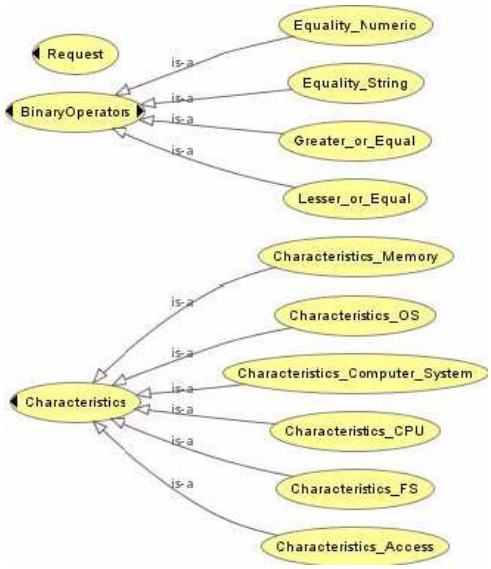


Figura 28: Ontologia de pedidos desenvolvida

Na classe *Request* foram definidas as seguintes propriedades, como podem ser observadas na Figura 29:

- ***query_id***: identificador da consulta;
- ***decrescent_order***: diretiva de pesquisa que indica o critério de ordenação dos recursos a serem retornados após *matching* semântico. O critério é realizado sobre uma das características de recursos (característica de valor numérico) definidas nas subclasses da classe *Characteristics*. A ordenação dos recursos retornados é decrescente, ou seja, do maior para o menor quanto aos valores numéricos atribuídos à característica do recurso escolhido como critério de ordenação;
- ***number_resources_return***: diretiva de pesquisa que indica a quantidade máxima de recursos a ser retornado pelo *matchmaker* após *matching* semântico;
- ***requirements***: propriedade que relaciona instâncias da classe *Request* com as instâncias da classe *BinaryOperators*. Esta relação informa quais restrições a consulta possui.

A classe *BinaryOperators* possui a propriedade *on* que relaciona os operadores às características de recursos computacionais definidos nas subclasses da classe *Characteristics*, como

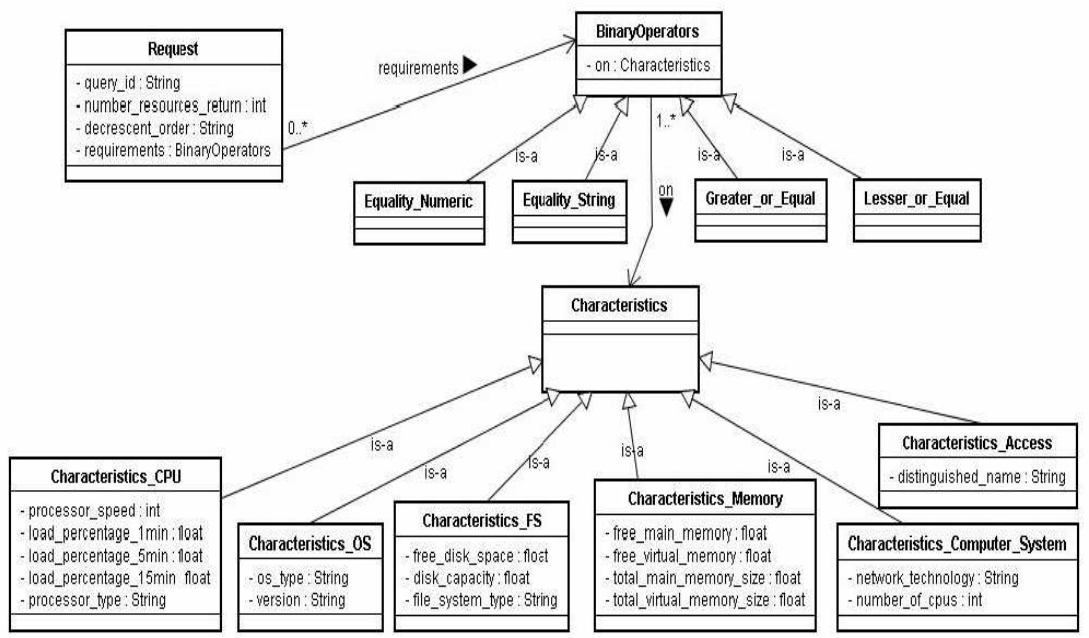


Figura 29: Propriedades das classes e relações entre classes na ontologia de pedidos

mostrado na Figura 29.

As subclasses da classe *Characteristics* especializam as características dos recursos pelo tipo de recurso. As subclasses *Characteristics_CPU*, *Characteristics_OS*, *Characteristics_FS*, *Characteristics_Computer_System*, *Characteristics_Memory* e *Characteristics_Access* respectivamente agregam as características dos seguintes elementos que constituem os recursos computacionais: processadores, sistema operacionais, sistemas de arquivos, memórias principais, sistemas computacionais e de política de acesso dos sistemas computacionais. Os termos utilizados para representar estas características pertencem ao vocabulário da OR.

5.3 Arquitetura do sistema de *matching* semântico

Nesta seção é descrito mais detalhadamente o sistema de *matching* semântico proposto. O sistema engloba os principais componentes: o portal de integração de ontologias, o provedor de informações e o serviço *matchmaker*. A Figura 30 apresenta a arquitetura do sistema.

O portal de integração de ontologias é a interface por onde a OV pode realizar a integração semântica da sua ontologia de recursos no sistema de *matching* semântico. Após a integração,

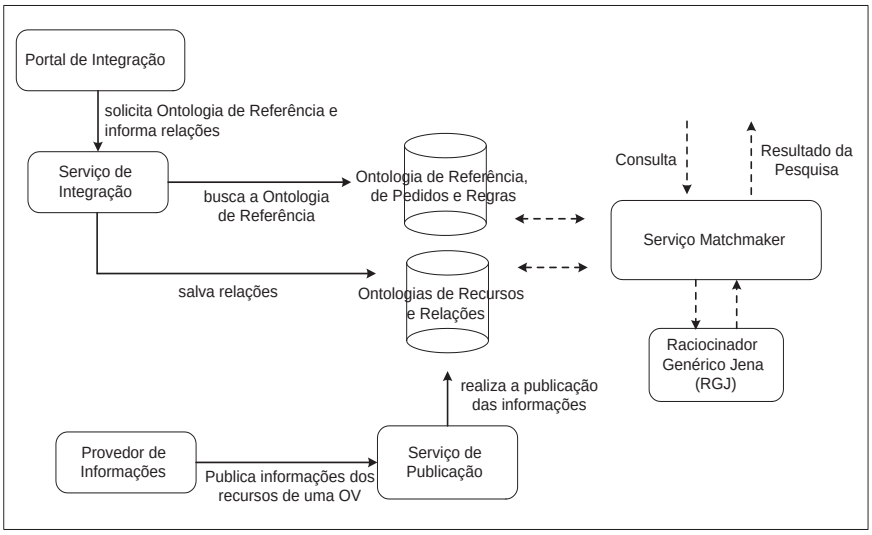


Figura 30: Arquitetura do sistema de *matching* semântico

os provedores de informações da OV podem publicar as informações dos seus recursos periodicamente no sistema para que sejam consultadas através do serviço *matchmaker*. A OR é compartilhada no ambiente de grid pelo portal. A ontologia de pedidos é compartilhada com os usuários do sistema através da interface de consulta desenvolvida. As regras de verificação de consistência e de ampliação das consultas são elaboradas baseadas na OR e na ontologia de pedidos e disponibilizadas ao serviço *matchmaker*.

Nas próximas seções, os principais componentes do sistema desenvolvido são apresentados com maiores detalhes.

5.3.1 Portal de Integração

No sistema proposto, para que uma OV publique as informações dos seus recursos modelados ontologicamente, ela deve primeiramente integrar a sua ontologia no sistema, através do portal de integração. Para isso, a OV precisa indicar equivalências semânticas que os termos da sua ontologia possuem com os termos da OR compartilhada. Após a OV estabelecer as relações de equivalência, elas são enviadas ao sistema, junto com o modelo terminológico da ontologia da OV (TBox), através do portal até o serviço de integração, como ilustrado na Figura 31.

O serviço de integração tem a função de verificar se as relações informadas estão corretas,

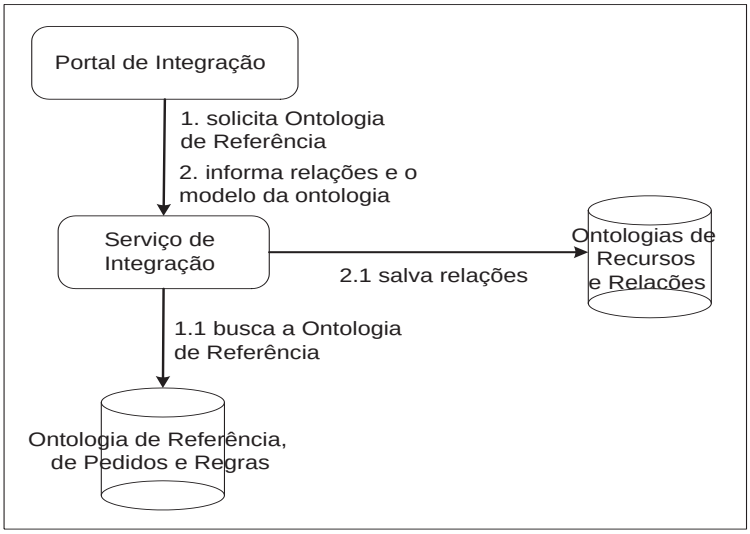


Figura 31: Organização virtual integrando semanticamente sua ontologia de recursos no sistema

ou seja, se elas relacionam termos que representam classe-classe ou propriedade-propriedade. Esta checagem é possível, pois a ontologia contém informações indicando o que cada termo na ontologia representa. Caso as relações estejam incorretas (classe-propriedade ou vice-versa), será enviada uma mensagem reportando os erros à OV. Caso contrário, as relações fornecidas pela OV serão salvas no banco de dados denominado Ontologias de Recursos e Relações, como ilustrado na Figura 31.

O processo de integração semântica pode ser visualizado melhor pelo diagrama de seqüência ilustrado na Figura 32.

No diagrama, observa-se que a OV através do portal solicita ao serviço de integração a OR, invocando o método *getOntRef* (passo 1). Este serviço busca a OR no banco de dados Ontologia de Referência, Pedidos e Regras (representado no diagrama como BD2, passo 1.1), e a retorna a OV. Após indicar as equivalências semânticas da sua ontologia com a OR, a OV envia essas equivalências ao sistema para serem salvas (passo 2). Além das relações de equivalência semântica, é informado a ontologia (TBox) e o identificador da OV. O serviço de integração baseado no modelo terminológico da ontologia checa as relações (passo 2.1), e caso haja erro (erro != null) é enviado à OV uma mensagem informando os erros. Caso contrário (passo 3), o serviço acessa o banco de dados Ontologias de Recursos e Relações, representado no diagrama

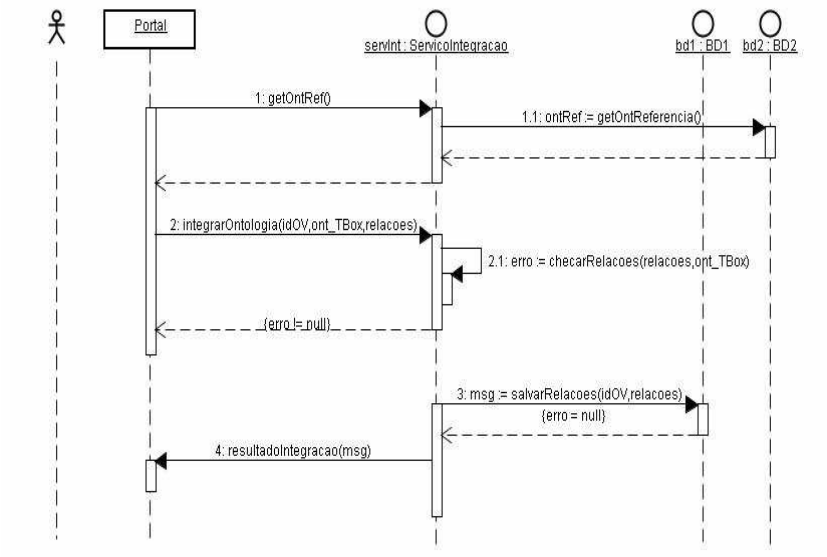


Figura 32: Integração semântica de uma ontologia de recursos de uma OV no sistema de *matching* semântico

como BD1, e salva as relações atribuindo-as ao identificador da organização fornecido. O resultado do processo de integração semântica da ontologia no sistema é disponibilizado para a OV pelo portal, (passo 4).

5.3.2 Provedor de Informações

Uma vez que o modelo ontológico que descreve os recursos de grid de uma OV foi integrado ao sistema, o Provedor de Informações que coleta informações dos recursos da OV pode publicá-las no sistema através do serviço de publicação, como mostrado na Figura 33.

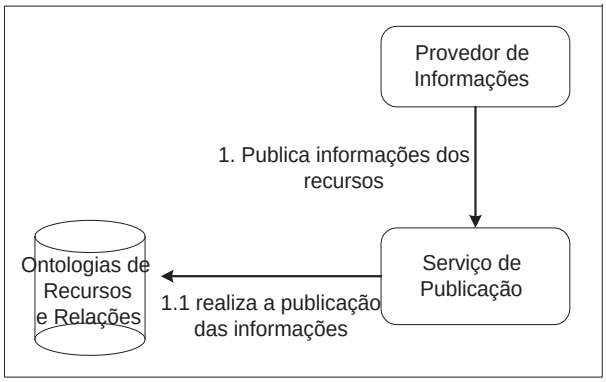


Figura 33: Procedimento de publicação de recursos de uma OV no sistema

O diagrama da Figura 34 ilustra de forma mais detalhada os eventos que ocorrem quando uma OV publica as informações dos seus recursos através dos seus provedores de informações.

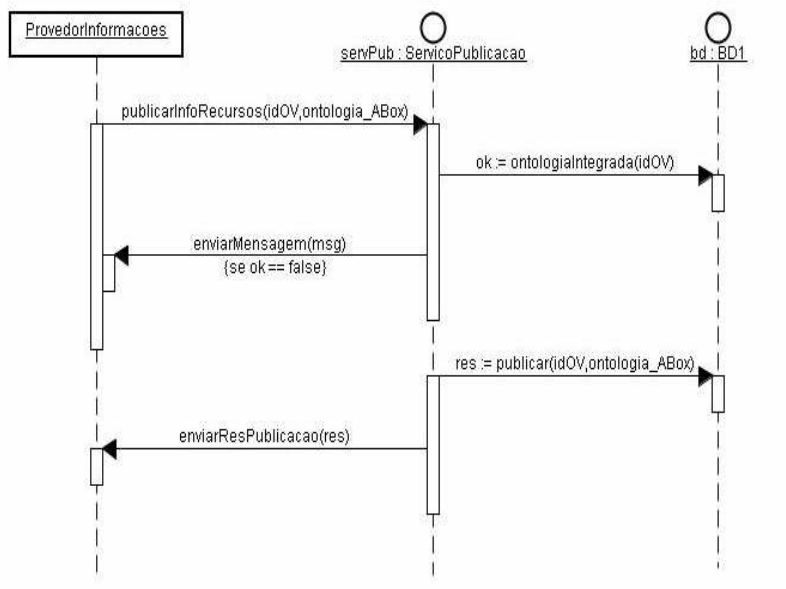


Figura 34: Provedor de informações de uma organização virtual publicando informações dos seus recursos no sistema

O provedor de informação invoca o método *publicarInfoRecursos* fornecendo o seu identificador e as informações mais recentes dos recursos (ABox) da OV. O serviço de publicação ao receber o identificador da OV checa se a sua ontologia já fora integrada no sistema (*ontologiaIntegrada*). Caso não tenha sido integrada, é enviada uma mensagem notificando a necessidade de integração da ontologia no sistema. Caso contrário, as informações dos recursos são salvas pelo serviço de publicação no banco de dados (BD1). Após á publicação dos recursos, o serviço de publicação dos recursos envia o resultado da operação para o provedor de informações.

5.3.3 Serviço Matchmaker

O processo de consulta por recursos de grid no sistema de *matching* semântico proposto pode ser compreendido de maneira macro através do esquema apresentado na Figura 35.

Para realizar consultas no sistema, o usuário deve descrever os requisitos que os recursos devem satisfazer na ontologia de pedidos para submetê-los ao serviço *matchmaker* desenvolvido. O serviço interage com duas bases de informação: a primeira contém as relações de

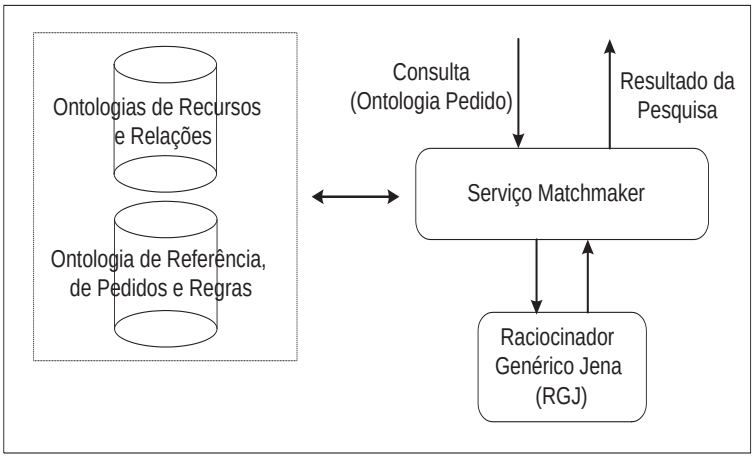


Figura 35: Consultando recursos através do serviço *matchmaker*

equivalência e as informações dos recursos (descritas por ontologias) publicadas pelas OVs e a outra base contém a OR, a ontologia de pedidos e dois tipos de regras, quais são: verificação de consistência e ampliação semântica da consulta, baseado no conhecimento de domínio modelado e nas informações definidas na OR. O *matchmaker* utiliza o raciocinador baseado em regras (RGJ) para avaliar os dois tipos de regras. O *matchmaker* foi desenvolvido baseado na ontologia de pedidos, nas regras e relações de equivalência semântica. As regras, por sua vez, baseiam-se na estrutura de conhecimento modelado na OR e nos indivíduos criados nela.

O diagrama de seqüência UML da Figura 36 mostra o processo de consulta de recursos com maiores detalhes. Nele pode-se observar que o usuário submete uma consulta ao serviço *matchmaker* através da interface de consulta (passo 1). O *matchmaker* ao receber a consulta verifica se ela não possui alguma inconsistência. Esta checagem é executada utilizando regras de verificação de consistência disponibilizadas no banco de dados BD2. Depois de alcançar as regras (passo 1.1), o *matchmaker*, com o auxílio do raciocinador RGJ, executa as regras sobre a consulta, a estrutura de conhecimento e indivíduos descritos na OR para detectar possíveis inconsistências, (passo 1.2). Um exemplo de verificação de inconsistência na consulta é mostrado na próxima seção. O método *validarConsulta* retornando *false*, indica ao *matchmaker* que encontrou inconsistências, que serão reportadas ao usuário que enviou a consulta (passo 1.3). Caso contrário, o *matchmaker* continua executando o restante dos passos do processo de *matching* semântico sobre os recursos do grid.

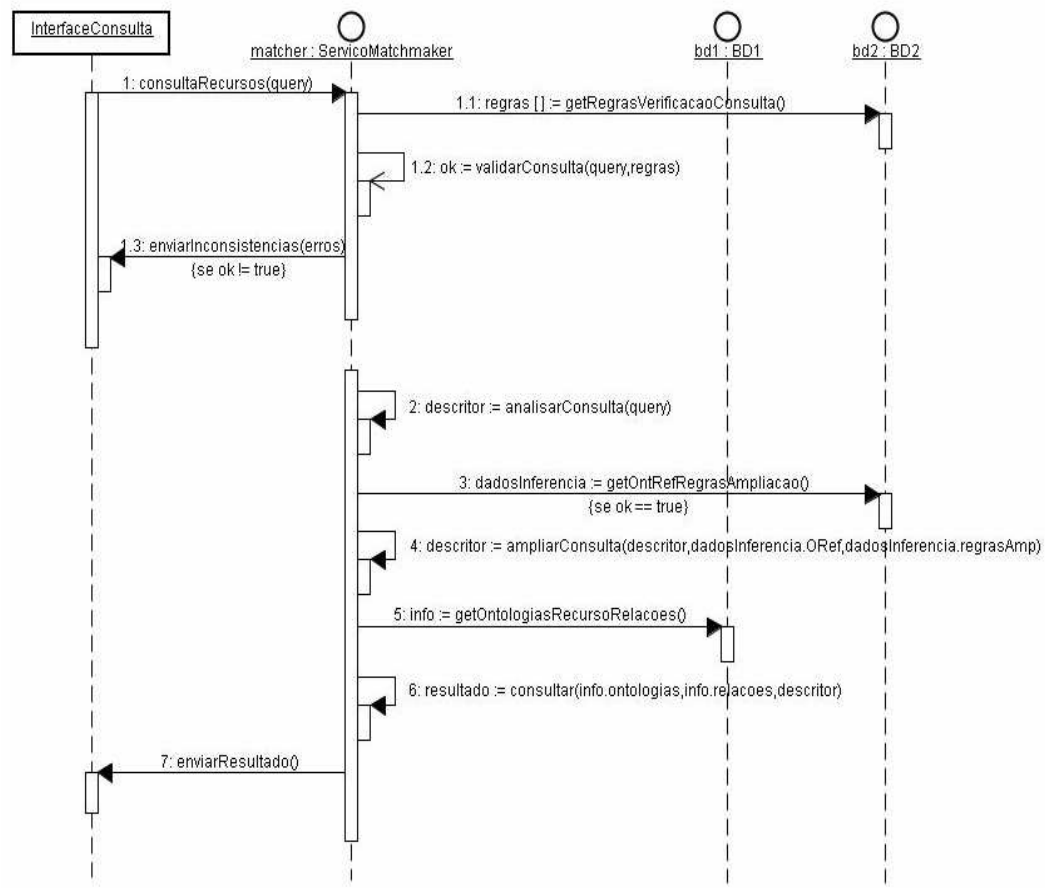


Figura 36: Diagrama de seqüência do processo de *resource matching* semântico

Após a validação da consulta, o *matchmaker* analisa a consulta recebida para detectar e coletar quais termos que representam as características dos recursos foram restringidos, os valores atribuídos a esses termos e as restrições que cada termo deve ser submetido (passo 2). Essas informações são armazenadas em um *Descritor*. No passo 3, o *matchmaker* busca no BD2 a OR e as regras para serem utilizadas no processo de ampliação semântica das consultas (passo 4). Baseado nas informações coletadas na consulta, no conhecimento modelado na OR e indivíduos definidos, e nas regras de inferência, o serviço *matchmaker* amplia a consulta e as novas informações capturadas são armazenadas no *Descritor*. Exemplo detalhando o processo de ampliação semântica das consultas por recursos é apresentado na próxima seção.

Ampliadas as informações presentes na consulta, o *matchmaker* busca as relações de equivalência semântica e as descrições dos recursos fornecidas pelas OVs ao sistema (passo 5). A consulta ampliada, as descrições dos recursos publicadas e as relações são os dados de entrada

que permitem ao *matchmaker* pesquisar todas as informações dos recursos descritas pelas diferentes ontologias publicadas no sistema (passo 6). A pesquisa é possível, uma vez que o *matchmaker* utiliza as relações de equivalência indicadas pelas OVs para saber quais termos das ontologias têm correspondência com os termos definidos na consulta. Conhecido os termos correspondentes com um determinado termo, o *matchmaker* compara o valor atribuído a este termo com os valores dos termos das ontologias semanticamente relacionados. A comparação entre os valores é feita de acordo com o operador binário atribuído ao termo na consulta. Após o *matching* semântico, o *matchmaker* envia à interface de consulta o resultado da pesquisa (passo 7).

5.3.3.1 Regras de verificação de consistência e de ampliação semântica de consultas

Como mencionado, o *matchmaker* proposto utiliza-se de dois tipos de regras. Ambos baseiam-se nas informações descritas na consulta e na estrutura do conhecimento e indivíduos modelados na OR. O primeiro tipo verifica a consistência das consultas. Estas regras são importantes para que não sejam desperdiçados recursos e tempo durante o processo de *matching* semântico sobre consultas sem coerência. Um exemplo de consulta inconsistente é mostrado na Tabela 7. Esta consulta requer recursos que atendam todos os seguintes requisitos: tipo de sistema de arquivos igual a 'EXT3', quantidade de espaço em disco rígido livre maior ou igual a 2 GB e maior ou igual a 5 GB, sistema operacional 'Windows XP' e o número de processadores igual a 4. Este último requisito declara que deve-se comparar a quantidade de processadores dos recursos com o valor 4 pelo operador (*Equality_String*) que tem como significado comparar cadeias de caracteres.

Tabela 7: Exemplo de uma consulta de recursos inconsistente

Requisitos	**Valores**
Request.query_id	'query_1'
Request.Equality_String.Characteristics_OS.os_type	'Windows XP'
Request.Equality_String.Characteristics_FS.file_type_system	'EXT3'
Request.Equality_String.Characteristics_Computer_System.number_of_cpus	'4'

A consulta apresenta três inconsistências:

- Requer recursos que ao mesmo tempo tenha o sistema operacional Windows XP e sistema de arquivos EXT3. Pelo conhecimento prévio, sabe-se que sistemas operacionais da família Windows não suportam o sistema de arquivos EXT3;
- Solicita recursos que tenham, ao mesmo tempo, quantidade de espaço em disco livre maior ou igual a 2 GB e a 5 GB. No entanto, para evitar essas incoerências é permitido que haja somente uma restrição por característica de recurso;
- Solicita recursos que tenham 4 processadores, porém o operador binário utilizado para indicar ao *matchmaker* como deve comparar este valor com as descrições publicadas dos recursos está incoerente.

As regras são avaliadas pelo raciocinador RGJ, e caso as premissas que expressam uma inconsistência na consulta sejam verdadeiras, é gerado, como conclusão da regra, uma mensagem informando a inconsistência. Esta mensagem é apresentada ao usuário que enviou a consulta. Na Figura 37 são apresentadas as três regras que, após serem avaliadas sobre a consulta descrita na Tabela 7, indicaram as inconsistências citadas acima.

```
<rules>

  <validation_ rule> [ checkingConsistencyEXT3withOSsFamilyWindows:  (?A greq:file_system_type ?FS)  (?B greq:os_type ?OS)
     (?C rdf:type owl:Class)  equal(?FS 'EXT3')  equalURI(?C Windows')  (?D rdf:type ?C)  (?D ont_ref:os_type ?SO1) equalURI(?OS
    ?SO1)-> (?D error Resource__with__operating__system__of__family__'Windows'__and__file__system__'EXT3'__not__exists__! )  ]
  </validation_rule  >

  <validation_ rule> [ checkingCardinalityTermFree_Main_Memory:  (?A owl:onProperty greq:free_main_memory)  (?A owl:cardinality
     ?card1) countLiteralValues(?B greq:free_main_memory ?qtd1) (?C greq:characteristic 'free_main_memory')
     countLiteralValues(?C greq:characteristic ?qtd2)  sum(?qtd1 ?qtd2 ?total) greaterThan(?total ?card1)          -> (?D error It__is__
     accepted__only__ one__value__for__the_term__' free_main_memory'__for__each__query __! )  ]
  </validation_rule  >

  <validation_ rule> [ checkingTypeTermNumber_of_CPUS:  (?A greq:on ?B)  (?A rdf:type ?C)  countLiteralValues(?B
    greq:number_of_cpus ?qtd)  greaterThan(?qtd 0)  equalURI(?C 'Equality_String')       -> (?C error The__value__attributed__to__ term__
    'number_of_cpus'__ must__be__related __with__the__binary__operators__that__compare__values__numerics__instead__ o f __
     strings__!) ]
  </validation_rule  >
...
```

Figura 37: Regras de verificação de consistência que reconheceram inconsistências na consulta query_A

A primeira regra ilustrada na Figura 37, captura a inconsistência de que não há recursos que possuam sistema operacional da família Windows com sistema de arquivos igual a EXT3.

A primeira tripla desta regra (?A greq:file_system_type ?FS) atribui o valor correspondente ao tipo de sistema de arquivos definido na consulta à variável ?FS. O namespace *greq* representa a URI da ontologia de pedidos, ou seja, das consultas. A segunda tripla atribui o valor que corresponde ao sistema operacional (propriedade *os_type*), informado na consulta, à variável *?OS*. Os sistemas operacionais que pertencem à família Windows são capturados através da estrutura e dos indivíduos (instâncias) definidos na OR (representado pelo namespace *ont_req*). Parte das instâncias geradas a respeito dos sistemas operacionais na OR podem ser visualizadas na Tabela 8. Estes indivíduos são alcançados percorrendo a estrutura de conhecimento modelado na OR. A princípio, deve-se encontrar a classe que representa a família Windows. Na OR ela é encontrada através das seguintes triplas (?C rdf:type owl:Class) (?C 'Windows'), expressa pela variável *?C*. Após encontrar a classe que representa os sistemas operacionais da família Windows, o passo seguinte é buscar os indivíduos que expressam esta classe, indicadas na regra pela variável *?D*. Os sistemas operacionais indicados por estas instâncias são alcançados através da tripla (?D ont_ref:os_type ?SO1), armazenando-os na variável *?SO1*. Caso o sistema de arquivos requisitado na consulta seja igual a EXT3, capturado na regra por equal(?FS 'EXT3') e o sistema operacional informado seja um dos descritos na OR, detectado pela tripla equalURI(?OS ?SO1), então a regra conclui que há inconsistência e gera a mensagem apresentada na primeira regra mostrada na Figura 37.

O segundo tipo de regras definido no sistema permite capturar o conhecimento modelado na OR para ampliar a consulta enviada ao *matchmaker*. As regras baseiam-se nas instâncias e na estrutura de conhecimento descritas na OR. Por exemplo, na OR, ilustrada na Figura 27, o conceito de sistema operacional é representado pela classe *OperatingSystem*. Esta classe foi especializada nos seguintes tipos de sistemas operacionais: Windows, Unix e MacOS, sendo a classe Unix, por sua vez, especializada no sistema operacional Linux. Instâncias foram criadas para representar estes tipos de sistemas operacionais, como observado na Tabela 8. Baseado nestas instâncias e no conhecimento modelado pelas relações de subclasse que os tipos de sistemas operacionais possuem, é possível ampliar as consultas.

Para ilustrar a característica de ampliação semântica das consultas no sistema proposto,

Tabela 8: Algumas instâncias que expressam sistemas operacionais específicos na OR

Classe	Instância da Classe
Unix	os_type = SunOS
Unix	os_type = AIX
Unix	os_type = FreeBSD
Linux	os_type = Debian
Linux	os_type = Slackware
Linux	os_type = Fedora Core
Windows	os_type = Windows 2000
Windows	os_type = Windows XP
Windows	os_type = Windows Server 2003

suponhamos que uma consulta requeira recursos que tenham sistema operacional igual a Unix. Com a regra de ampliação de consulta, apresentada na Figura 38, é possível buscar por recursos que possuam os sistemas operacionais SunOS, AIX e FreeBSD, instâncias da classe *Unix*, além dos recursos com sistema operacional igual a Unix. Junto com esses recursos serão retornados também os que tiverem os sistemas operacionais Debian, Slackware e Fedora Core, instâncias da classe *Linux*, visto que a relação de subclasse modelada na OR, entre a classe *Linux* e *Unix*, é por definição transitiva. Recursos que tiverem sistema operacional igual a Linux também serão retornados.

```
- <queries_extension_rules  >

 <rule > [ GetOSsLikeUNIX: (?A greq:os_type ?SO) (?B rdf:type owl:Class) equalURI(?B 'Unix')
           equalURI(?B ?SO) (?D rdf:type ?B) (?D ont_ref:os_type ?SO1) (?E rdfs:subClassOf ?B)
           equalURI(?E 'Linux') (?F rdf:type ?E) (?F ont_ref:os_type ?SO2) ->(?G gres:result_value ?SO1)
           (?H gres:result_value ?SO2) (?I gres:result_value 'Unix') (?F gres:result_value 'Linux') ]
 </rule >
....
```

Figura 38: Regra de ampliação de consulta baseada na consulta enviada (*greq*) e na OR desenvolvida (*ont_ref*)

5.3.3.2 Navegando nas diferentes estrututras de conhecimento

Como já mencionado, foi desenvolvido um algoritmo para navegar na estrutura das ontologias OWL baseado nas descrições de recursos que atendem as restrições definidas nas consultas e nas relações de equivalência informadas ao sistema. Este algoritmo tem como objetivo tor-

nar transparente ao usuário a necessidade de conhecer as diferentes estruturas que descrevem os recursos do grid definidas em diversas ontologias OWL e como representar estas estruturas formalmente.

Para a melhor compreensão do algoritmo desenvolvido, explicaremos através de um exemplo como o algoritmo atua sobre dois recursos computacionais descritos em ontologias distintas.

Parte da primeira ontologia, denominada de A, tem os seguintes conceitos: sistema computacional, sistema operacional e processador, como pode ser visto na Figura 39.

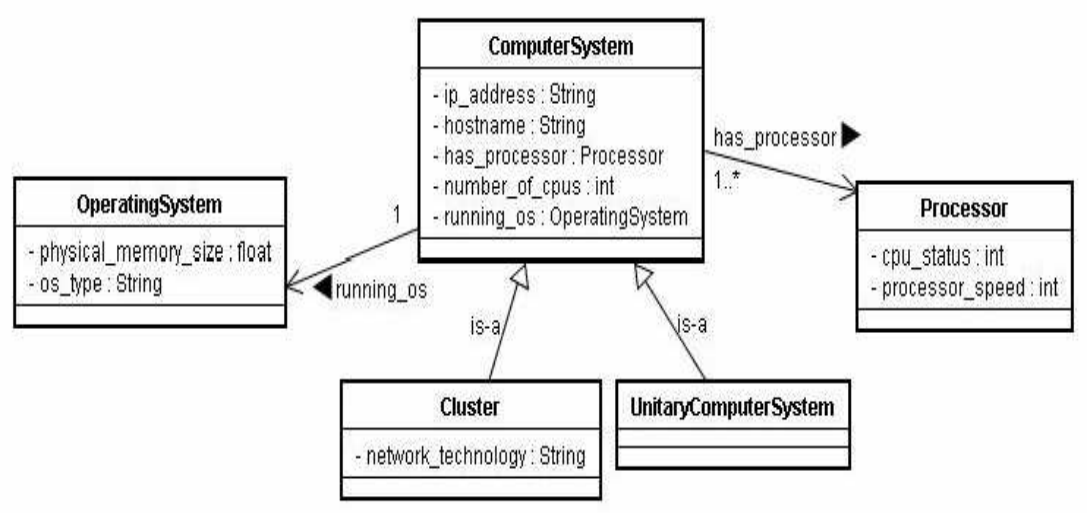


Figura 39: Ontologia A descrevendo recursos computacionais e alguns elementos que os constituem

O conjunto de triplas que representam parcialmente os conceitos da ontologia A é apresentado na Figura 40.

As duas primeiras triplas expressam que uma classe *UnitaryComputerSystem* é uma especialização da classe *ComputerSystem* e que a instância *UnitaryComputerSystem_06* é uma instância da classe *UnitaryComputerSystem*. Esta instância representa um recurso que possui as propriedades hostname, *ip_address* e *number_of_cpus* (triplas 3, 4 e 5). Além disso, *UnitaryComputerSystem_06* possui relações com as instâncias das classes *Processor* e *OperatingSystem* indicadas pelas propriedades *has_processors* e *running_os*, expressas pelas triplas 7 e 11. A instância *Processor_06* possui as propriedades *cpu_status* e *processor_speed* defi-

```
1 - (grsc:UnitaryComputerSystem rdfs:subClassOf grsc:ComputerSystem)
2 - (grsc:UnitaryComputerSystem_06 rdf:type grsc:UnitaryComputerSystem)

3 - (grsc:UnitaryComputerSystem_06 grsc:hostname 'agentgrid.lrg.ufsc.br'^^http://www.w3.org/2001/XMLSchema#string)
4 - (grsc:UnitaryComputerSystem_06 grsc:ip_address '150.162.63.7'^^http://www.w3.org/2001/XMLSchema#string)
5 - (grsc:UnitaryComputerSystem_06 grsc:number_of_cpus '1'^^http://www.w3.org/2001/XMLSchema#int)

6 - (grsc:Processor_06 rdf:type grsc:Processor)
7 - (grsc:UnitaryComputerSystem_06 grsc:has_processor grsc:Processor_06)
8 - (grsc:Processor_06 grsc:cpu_status 'running'^^http://www.w3.org/2001/XMLSchema#string)
9 - (grsc:Processor_06 grsc:processor_speed '2400'^^http://www.w3.org/2001/XMLSchema#int)

10 - (grsc:OperatingSystem_06 rdf:type grsc:OperatingSystem)
11 - (grsc:UnitaryComputerSystem_06 grsc:running_os grsc:OperatingSystem_06)
12 - (grsc:OperatingSystem_06 grsc:physical_memory_size '512'^^http://www.w3.org/2001/XMLSchema#int)
13 - (grsc:OperatingSystem_06 grsc:os_type 'Fedora Core'^^http://www.w3.org/2001/XMLSchema#string)
...
```

Figura 40: Representação parcial, na forma de triplas, dos conceitos e instâncias modelados na ontologia A

nidas pelas triplas 8 e 9. Por fim, a instância *OperatingSystem_06* apresenta as propriedades *physical_memory_size* e *os_type* (triplas 12 e 13).

A segunda ontologia (B), apresentada na Figura 41, descreve os seguintes conceitos: *TipodeMaquina* e *SistemaOperacional*.

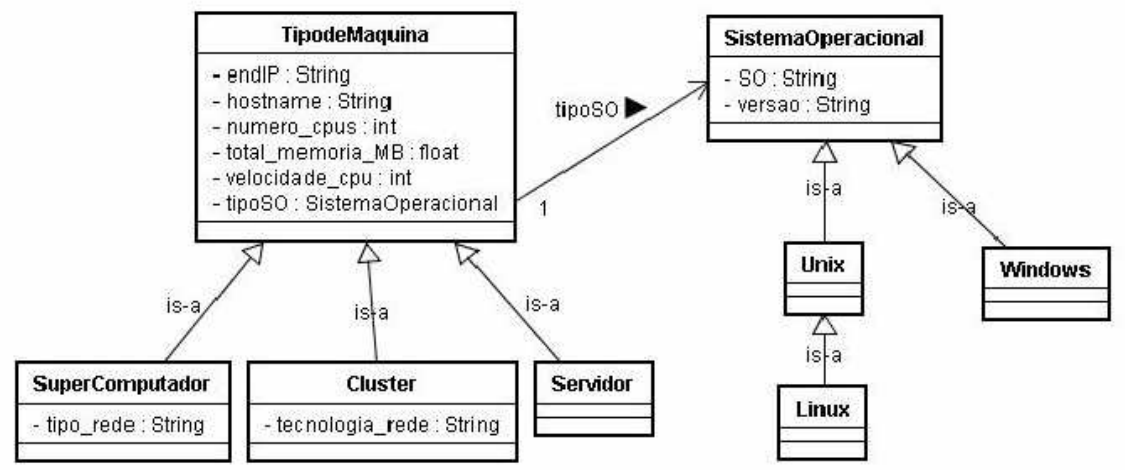


Figura 41: Ontologia B descrevendo recursos computacionais e alguns elementos que os constituem

O conjunto de triplas que expressam parcialmente os conceitos da ontologia B é ilustrado na Figura 42.

As duas primeiras triplas definem a classe *Servidor* como sendo uma especialização da

```
1 - (grsc:Servidor rdfs:subClassOf grsc:TipodeMaquina)
2 - (grsc:Servidor_2 rdf:type grsc:Servidor)

3 - (grsc:Servidor_2 grsc:hostname 'server1.fis.ufpr.br'^^http://www.w3.org/2001/XMLSchema#string)
4 - (grsc:Servidor_2 grsc:endIP '140.68.85.1'^^http://www.w3.org/2001/XMLSchema#string)
5 - (grsc:Servidor_2 grsc:numero_cpus '1'^^http://www.w3.org/2001/XMLSchema#int)
6 - (grsc:Servidor_2 grsc:velocidade_cpu '2100'^^http://www.w3.org/2001/XMLSchema#int)
7 - (grsc:Servidor_2 grsc:total_memoria_MB '1024.0'^^http://www.w3.org/2001/XMLSchema#float)

8 - (grsc:Servidor_2 grsc:tipoSO grsc:Linux_2)
9 - (grsc:Linux_2 rdf:type grsc:Linux)
10 - (grsc:Linux_2 grsc:SO 'Fedora Core'^^http://www.w3.org/2001/XMLSchema#string)
11 - (grsc:Linux_2 grsc:versao '3'^^http://www.w3.org/2001/XMLSchema#string)
...
```

Figura 42: Representação parcial, na forma de triplas, dos conceitos e instâncias modelados na ontologia B

classe *TipodeMaquina* e *Servidor_2* como uma instância da classe *Servidor*. O recurso Servidor_2 apresenta as seguintes propriedades: nome do host (*hostname*), endereço IP (*endIP*), número de processadores que possui (*numero_cpus*) e capacidade do processamento desses processadores (*velocidade_cpu*) e de sua memória RAM (*total_memória_MB*) expressas pelas triplas 3, 4, 5, 6 e 7. Adicionalmente, *Servidor_2* tem relação com a instância da classe *Linux* (*Linux_2*), através da propriedade tipoSO, tripla 8. Esta instância tem as propriedades *SO* e *versão*, que podem ser vistas nas triplas 10 e 11.

O algoritmo desenvolvido pode ser visualizado a seguir, na Figura 43. Para cada termo (que representa uma característica de recurso) restringido na consulta é verificado qual conjunto de triplas atendem a essa restrição. Cada tripla desse conjunto corresponde à característica que um recurso satisfaz. Por exemplo, suponhamos que uma consulta busque por recursos que tenham capacidade de processamento maior ou igual a 1800 MHz. Essa característica é expressa na consulta pelo termo *processor_speed* cuja condição é >= 1800 MHz. Considerando que as propriedades *processor_speed* (ontologia A) e *velocidade_cpu* (ontologia B) são tidas no sistema como equivalentes ao termo *processor_speed* (consulta), o primeiro passo do *matchmaker* é verificar quais triplas que descrevem os recursos publicados no sistema possuem estas propriedades. Na ontologia A, a tripla 9 e na ontologia B, a tripla 6 apresentam estas propriedades. A seguir o *matchmaker* checa se os valores atribuídos a essas triplas satisfazem a condição >= 1800MHz. As triplas que atenderem a condição são guardadas para serem utilizadas no método

getRecursos, linhas 1 a 17, descrito na Figura 43. Neste exemplo, as triplas 9 (ontologia A) e 6 (ontologia B) atendem a condição e são armazenadas no vetor **parcial**, na posição 1, para serem posteriormente utilizadas na consulta.

```
1     getRecursos(indiceTermo)
2     Inicio
3           armazena ◄— vazio
4           Ler(parcial[indiceTermo])
5           Ler(n)
6           i ◄— 1
7           repetir
8                 enquanto (parcial[indiceTermo] <> vazio) fazer
9                       tripla ◄— parcial[indiceTermo].proximo
10                      suj ◄— tripla.getSujeito
11                      buscar(suj,armazena)
12                      parcial[indiceTermo] - tripla
13                fim enquanto
14                i ◄— i +1
15          até (i <= n)
16          retorna armazena
17    Fim

18    buscar(nodo,armazena)
19    Inicio
20          Ler(ontologias)
21          Ler(identificadores)
22          Ler(qtdIdentificadores)
23          j ◄— 1
24          encontrou ◄— false

25          repetir
26                pred ◄— identificadores[j]
27                se (ontologias C <nodo,pred,?>) entao
28                      tripla ◄— ontologias.getTripla(<nodo,pred,?>)
29                      armazena U tripla
30                      encontrou ◄— true
31                      retornar
32                fim_se
33                j ◄— j + 1
34          até (j <= qtdIdentificadores)

35          se (encontrou = false) entao
36                vetorTriplas ◄— ontologias.listaTriplas(<?,?,nodo>)
37                enquanto (vetorTriplas <> vazio) fazer
38                      sentenca ◄— vetorTriplas.proximo
39                      nodo1 ◄— sentenca.getSujeito
40                      buscar(nodo1,armazena)
41                      vetorTriplas - sentenca
42                fim enquanto
43          fim se
44    Fim
```

Figura 43: Algoritmo de busca de recursos nas ontologias

A cada termo *Ti* restringido na consulta é invocado o método getRecursos, passando o índice (*i*) que representa o termo. Este índice serve para buscar o conjunto de triplas que obedecem a restrição imposta sobre *Ti* que foram guardadas anteriormente no vetor **parcial** (linha 4). Neste exemplo, o único termo (*T1*) corresponde a *processor_speed*, portanto, as triplas armazenadas foram:

- <grsc:Processor_06, grsc:processor_speed, 2400> (tripla_A)
- <grsc:Servidor_2, grsc:velocidade_cpu, 2100> (tripla_B)

Previamente, é criado um conjunto vazio (***armazena***) que armazenará as triplas que infor-

mam quais recursos atendem a restrição a qual uma determinada característica de recurso foi submetida (linha 3). Na linha 4, são lidos os conjuntos que contém as triplas que satisfazem cada restrição definida na consulta. Além disso, é lida a quantidade de restrições (***n***) definidas na consulta (linha 5). Em nosso exemplo, há somente uma restrição sobre o termo (característica do recurso) processor_speed, ou seja, ***n*** é igual a 1. Cada tripla do conjunto das triplas referente ao termo *processor_speed* (linha 9), é consultada para pegar o seu sujeito (linha 10). O método ***buscar*** é invocado passando o sujeito ***suj*** e o conjunto ***armazena*** (linha 11).

O método buscar é responsável em encontrar quais recursos satisfazem todas as restrições. Na linha 20 são buscadas as descrições dos recursos publicadas no sistema (***ontologias***). As linhas 21 e 22 mostram a leitura dos identificadores que representam unicamente os recursos computacionais (***identificadores***) e da quantidade de identificadores (***qtdIdentificadores***). O identificador estabelecido foi o endereço IP que os recursos computacionais possuem. Na ontologia de pedidos o termo que se refere ao endereço IP é *ip_address* e os equivalentes nas ontologias A e B são *ip_address* e *endIP*. Para cada identificador (linha 26), é testada se ontologias contém a tripla <nodo,pred,?>, ou seja, que apresente como sujeito o valor atribuído ao nodo e como propriedade (ou predicado) o identificador analisado ***pred*** (linha 27). Caso exista, caracteriza-se que foi encontrado o recurso - representado pelo sujeito (***nodo***) - que atende a restrição em questão. Esta tripla é alcançada (linha 28), para que seja inserida no conjunto ***armazena*** (linha 29). Quando é encontrado um recurso, a repetição em torno dos identificadores é encerrada (linha 31). Caso não exista a tripla <nodo,pred,?> o laço continua testando com outro identificador atribuído a ***pred*** (linhas 26 e 27). Não encontrando nenhum recurso pelos identificadores únicos estipulados (encontrou = false), o algoritmo vai para o bloco de instruções entre as linhas 35 a 43, que busca o recurso alternando o sujeito pelo objeto e vice versa.

No segundo bloco são retornadas de ***ontologias*** as triplas que tenham como objetos o sujeito ***suj***, passado como nodo ao método ***buscar*** (linha 36). Sobre cada tripla encontrada é utilizado o seu sujeito para servir como parâmetro para a nova busca, invocando recursivamente o método ***buscar*** (linhas 38, 39 e 40). O critério de parada desta recursão é quando se encontra o recurso que atende uma determinada restrição (linha 27).

Aplicando o algoritmo descrito sobre as duas ontologias exemplificadas na Figura 39 e Figura 41, temos:

- A tripla_A tem o seu sujeito *grsc:Processor_06* enviado ao método ***buscar***, linha 11. Neste método, a princípio é verificado se o sujeito é um recurso computacional, observando se este apresenta alguma propriedade que seja um dos identificadores informados (por exemplo, *ip_address* e *endIP*). Após a checagem (linha 27), constatou-se que esse sujeito não é um recurso. Então, são procuradas as triplas que têm como objeto o sujeito *grsc:Processor_06* (linha 36). A tripla <grsc:UnitaryComputerSystem_06, grsc:has_processor, grsc:Processor_06> é a única resultante da busca e o seu sujeito *grsc:UnitaryComputerSystem_06* é passado como parâmetro ao método ***buscar*** para verificar se ele representa o recurso computacional desejado (linha 40). Na linha 27, o algoritmo capta que o sujeito informado apresenta o identificador *ip_address*. Portanto, a tripla <grsc:UnitaryComputerSystem_06, grsc:ip_address, 150.162.63.7> é guardada no conjunto ***armazena*** (linha 29).

- A próxima tripla enviada ao método ***buscar*** (linha 11), é a tripla_B que tem o sujeito *grsc:Servidor_2*. O método verifica se este sujeito possui um dos identificadores que indicam unicamente um recurso computacional. Como apresentado na Figura 42, o sujeito *grsc:Servidor_2* tem o identificador *endIP*. Portanto, a tripla <grsc:Servidor_2, grsc:endIP, 140.68.85.1> é guardada no conjunto ***armazena*** (linha 29).

Após encontrar os dois recursos que atendem a restrição *processor_speed* >= 1800 MHz estes são retornados pelo método ***getRecursos*** (linha 16) e armazenados para posterior visualização. Em uma consulta onde há mais de uma restrição, o procedimento de busca de recursos é executado para cada restrição. Os recursos que satisfazem cada restrição são guardados separadamente com o objetivo de sabermos quais recursos atendem todas as restrições. De acordo com a diretiva de busca, estes recursos ou parte deles serão retornados como o resultado do *matching* semântico.

Na Figura 44 é ilustrada a estrutura em grafo das principais triplas que se referem aos recursos *UnitaryComputerSystem_06* (Ontologia A) e *Servidor_2* (Ontologia B).

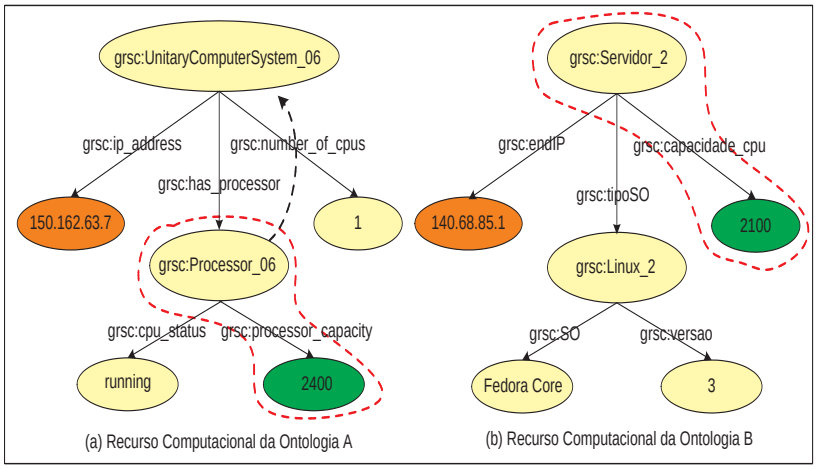


Figura 44: Grafos expressando parcialmente os recursos UnitaryComputerSystem_06 e Servidor_2

O algoritmo criado baseia-se primeiramente nas triplas que atendem a restrição *processor_speed* >= 1800 MHz, circundadas pelas linhas tracejadas. Os sujeitos dessas triplas são analisados para saber se representam os recursos computacionais desejados. Para identificá-los, essas triplas devem possuir predicados que expressem o identificador único dos recursos (representado respectivamente nas ontologias A e B pelos termos *ip_address* e *endIP*). Caso essas triplas possuam os identificadores, os seus sujeitos são considerados os recursos computacionais procurados.

Para o sujeito da tripla <grsc:Servidor_2, grsc:velocidade_cpu, 2100>, descrito na ontologia B, nota-se que há uma tripla da forma <grsc:Servidor_2, grsc:endIP, ?>. Portanto, o sujeito *grsc:Servidor_2* representa um recurso que atende a restrição definida, sendo guardada a tripla <grsc:Servidor_2, grsc:endIP, 140.68.85.1>. Já na tripla <grsc:Processor_06, grsc:processor_speed, 2400>, o seu sujeito não possui uma tripla igual a <grsc:Processor_06, grsc:ip_address, ?>. Logo, o sujeito *grsc:Processor_06* não representa o recurso desejado. Dessa forma, uma nova busca é realizada para encontrar a tripla que tem o nodo *grsc:Processor_06* como um objeto e não como um sujeito. Nessa etapa são pesquisadas as triplas que tenham a forma <?, ?, grsc:Processor_06>. A única tripla que tem esse objeto é a tripla <grsc:UnitaryComputerSystem

grsc:has_processor, grsc:Processor_06>. O próximo passo é analisar se o sujeito dessa tripla (*grsc:UnitaryComputerSystem_06*) tem o predicado *grsc:ip_address*. Como pode ser visto na Figura 44, existe a tripla < grsc:UnitaryComputerSystem_06, grsc:ip_address, 150.162.63.7>, portanto o sujeito *grsc:UnitaryComputerSystem_06* representa outro recurso que satisfaz a restrição, sendo esta tripla armazenada.

A partir das triplas armazenadas é possível construir interfaces que apresentem os recursos que atendam as restrições definidas nos pedidos, bastando navegar na estrutura das ontologias à procura de suas propriedades.

5.3.4 Interface de Consulta

Para facilitar a busca por recursos no sistema criado, foi desenvolvida uma interface gráfica, como mostrado na Figura 45. Esta interface permite ao usuário (após conexão com o serviço *matchmaker*) elaborar consultas, enviar consultas ao *matchmaker*, e visualizar as correspondências semânticas estabelecidas e o resultado da pesquisa.

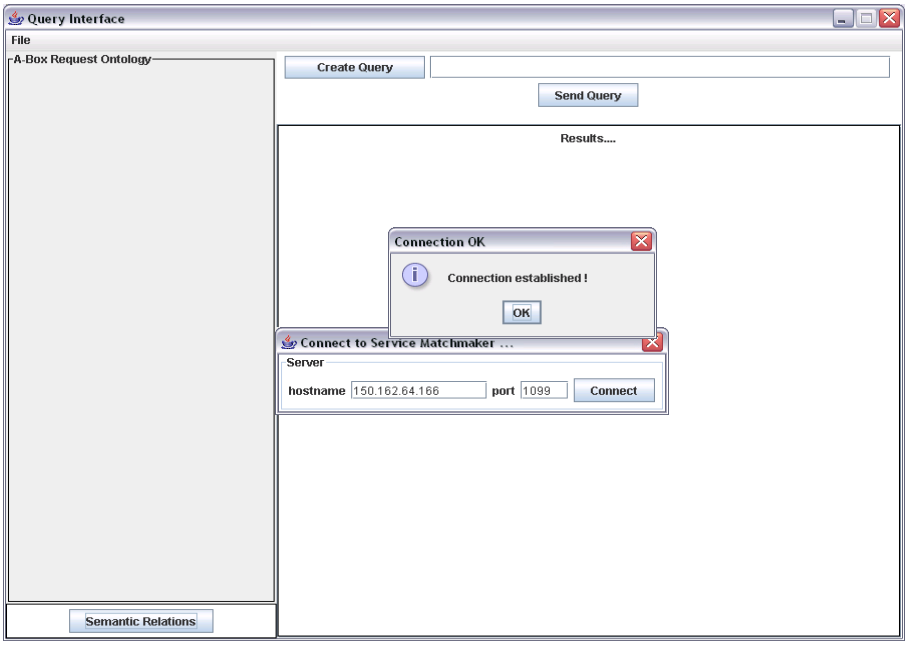


Figura 45: Usuário estabelecendo conexão com o *matchmaker* através da interface de consulta

Para definir os requisitos que os recursos devem atender, o usuário deve clicar no botão

"*Create Query*" para abrir a tela de edição de consultas, mostrada na Figura 46.

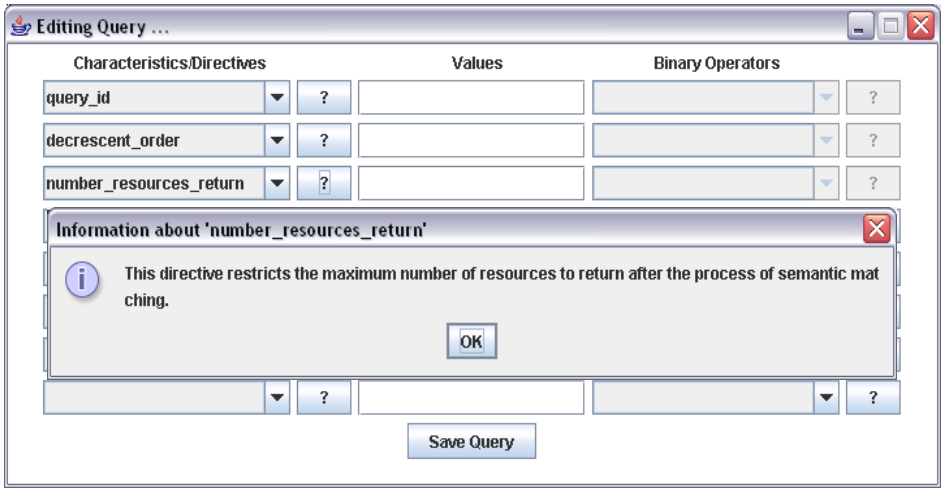


Figura 46: Informação sobre a diretiva *number_resources_return*

O usuário, pela tela de edição de consultas, pode criar consultas que restringem as características dos recursos e seus valores junto com o tipo de comparação à qual estes valores devem ser submetidos. Na primeira coluna, estão destacadas as diretivas *decrescent_order* e *number_resources_return* que respectivamente indicam ao *matchmaker* o critério de ordenação dos recursos e a quantidade de recursos a serem retornados após a pesquisa. A característica *query_id* é onde se atribui um identificador para a consulta.

Ao escolher uma diretiva ou característica, o usuário pode visualizar o seu significado através dos botões "?" correspondentes, como mostrado na Figura 46. O mesmo ocorre com os operadores binários usados para construir restrições sobre as características, como ilustrado na Figura 47. Dessa forma, o usuário tem informações que o permitem elaborar uma consulta conforme a sua necessidade. As informações são provenientes de metadados definidos na própria ontologia de pedidos.

A Figura 48 apresenta a definição de uma consulta identificada por *query_1*. Esta consulta requer recursos que possuam sistema operacional do tipo Linux, sejam monoprocessados, com capacidade de processamento superior ou igual a 1800 MHz e que permitam acessos pelo identificador único do grid (*distinguished_name*) igual a "/O=UFSC/OU=INE/CN=parra". Esta consulta é elaborada na interface de edição de consulta mostrada na Figura 48 e salva no arquivo

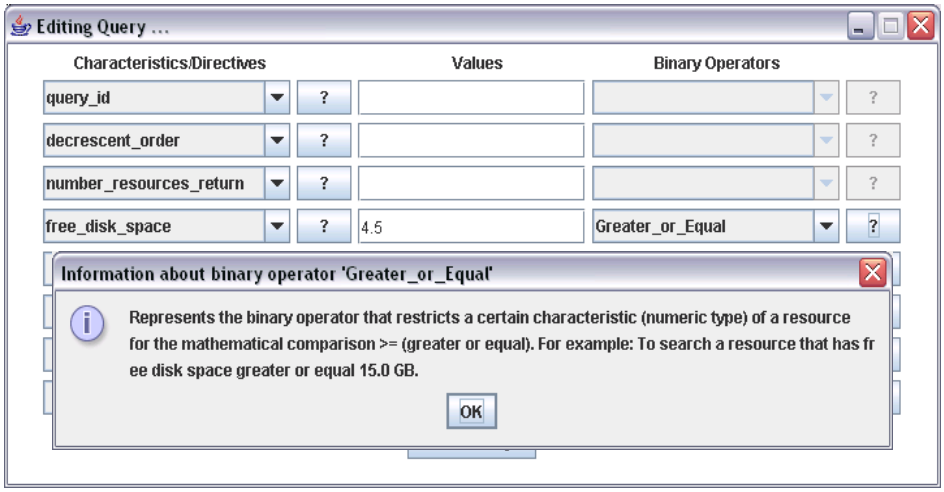


Figura 47: Informação sobre o operador binário *Greater_or_Equal*

query_1.owl.

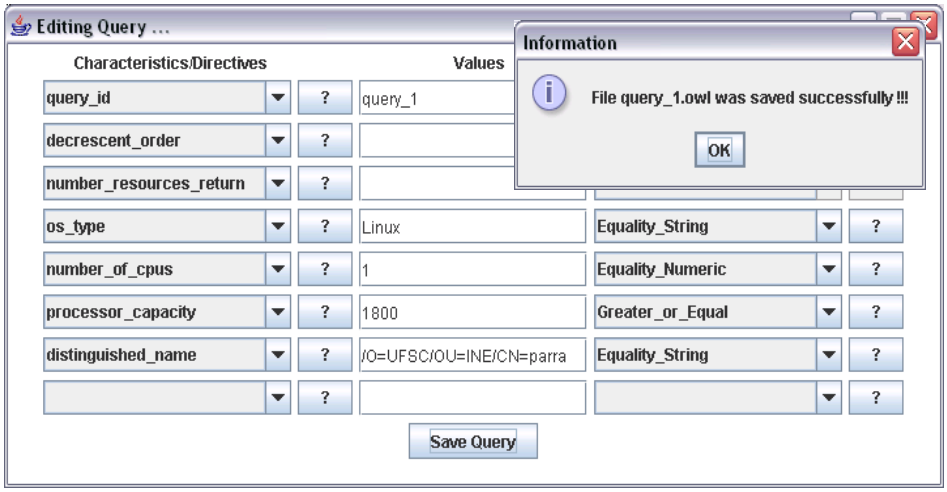


Figura 48: Criando a consulta query_1

Após salvar a consulta query_1, ela é apresentada em forma de uma estrutura de árvore no lado esquerdo da interface principal, destacando as restrições definidas previamente, como ilustrado na Figura 49. As restrições são impostas sobre os termos que denotam as características de recursos definidas na OR. Uma restrição é formada pelo tipo de comparação (informado entre parênteses) cujo valor atribuído à característica deve ser submetido durante a pesquisa.

Para submeter query_1 ao serviço *matchmaker*, o usuário deve clicar no botão "Send Query". Após o processo de *matching* semântico, o *matchmaker* retorna o resultado que é apresentado

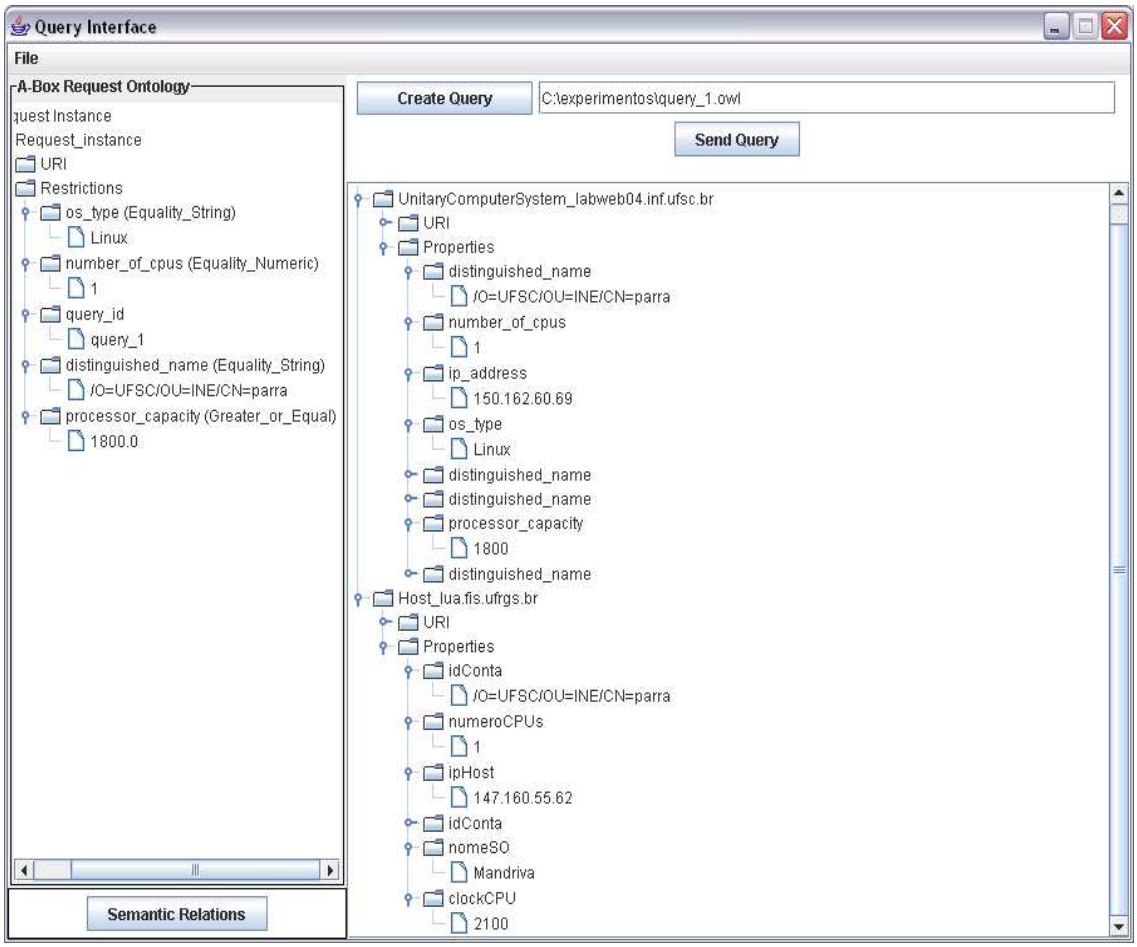
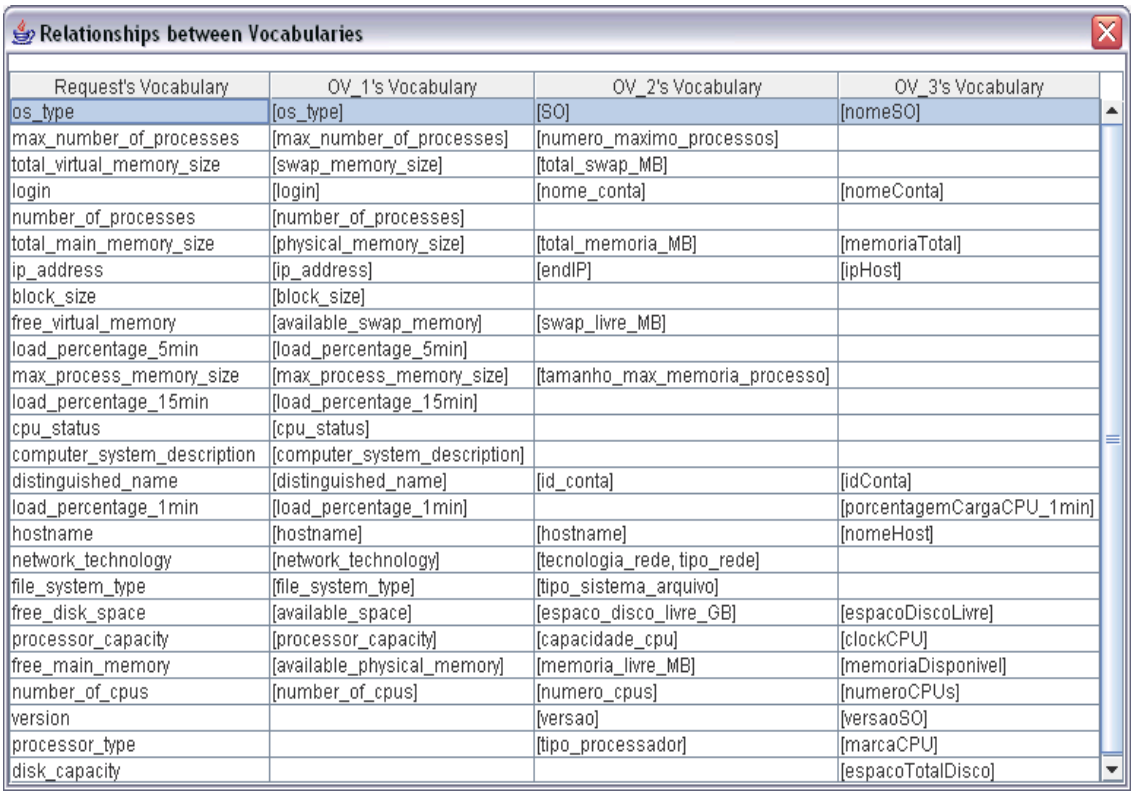


Figura 49: Resultado da consulta por recursos baseado na query_1

no lado direito da interface principal, como mostrado na Figura 49. O resultado também é visualizado na forma de árvore, onde as raízes representam o identificador local dos recursos. Cada raiz possui duas folhas: a primeira denominada URI, que representa o identificador único do recurso no grid, e a segunda folha (*Properties*), representa as propriedades do recurso. Os valores atribuídos às propriedades do recurso também são mostrados.

Nota-se que a busca resultou em dois recursos que apresentam termos distintos, deixando claro que pertencem a diferentes organizações virtuais. O usuário após a busca pode consultar a relação dos termos das diversas ontologias de recursos integradas no sistema com os termos definidos na ontologia de pedidos. Desta forma, o usuário pode avaliar se o resultado atende às restrições desejadas. Para consultar as relações, o usuário deve clicar no botão "*Semantic Relations*" para que apareça uma tela informando as relações de equivalência, como ilustrado

na Figura 50. Optamos por mostrar as características dos recursos retornados pelos seus termos originais para evidenciar as relações semânticas estabelecidas pelas organizações virtuais com os termos que constituem o vocabulário da OR.



Relationships between Vocabularies

Request's Vocabulary	OV_1's Vocabulary	OV_2's Vocabulary	OV_3's Vocabulary
os_type	[os_type]	[SO]	[nomeSO]
max_number_of_processes	[max_number_of_processes]	[numero_maximo_processos]	
total_virtual_memory_size	[swap_memory_size]	[total_swap_MB]	
login	[login]	[nome_conta]	[nomeConta]
number_of_processes	[number_of_processes]		
total_main_memory_size	[physical_memory_size]	[total_memoria_MB]	[memoriaTotal]
ip_address	[ip_address]	[endIP]	[ipHost]
block_size	[block_size]		
free_virtual_memory	[available_swap_memory]	[swap_livre_MB]	
load_percentage_5min	[load_percentage_5min]		
max_process_memory_size	[max_process_memory_size]	[tamanho_max_memoria_processo]	
load_percentage_15min	[load_percentage_15min]		
cpu_status	[cpu_status]		
computer_system_description	[computer_system_description]		
distinguished_name	[distinguished_name]	[id_conta]	[idConta]
load_percentage_1min	[load_percentage_1min]		[porcentagemCargaCPU_1min]
hostname	[hostname]	[hostname]	[nomeHost]
network_technology	[network_technology]	[tecnologia_rede, tipo_rede]	
file_system_type	[file_system_type]	[tipo_sistema_arquivo]	
free_disk_space	[available_space]	[espaco_disco_livre_GB]	[espacoDiscoLivre]
processor_capacity	[processor_capacity]	[capacidade_cpu]	[clockCPU]
free_main_memory	[available_physical_memory]	[memoria_livre_MB]	[memoriaDisponivel]
number_of_cpus	[number_of_cpus]	[numero_cpus]	[numeroCPUs]
version		[versao]	[versaoSO]
processor_type		[tipo_processador]	[marcaCPU]
disk_capacity			[espacoTotalDisco]

Figura 50: Equivalência entre os termos das ontologias das três organizações virtuais com os da ontologia de pedidos

A linha destacada na Figura 50 mostra a relação semântica do termo pertencente ao vocabulário da ontologia de pedidos (os_type), denotando o tipo de sistema operacional, com os termos os_type, SO e nomeSO, correspondentes às ontologias das três organizações que compõem o grid, OV_1, OV_2 e OV_3. Neste exemplo, os dois recursos retornados, identificados pelos IPs 150.162.60.69 e 147.160.55.62, pertencem respectivamente à OV_1 e OV_3, pois suas características são representadas por termos que fazem parte dos vocabulários dessas ontologias.

Para definir a consulta query_1 na linguagem SPARQL, ilustrada na Figura 51, o usuário precisa conhecer como traduzir a estrutura do conhecimento modelado na ontologia para o formato de triplas para gerar as restrições.

Comparativamente com a linguagem SPARQL, a ontologia de pedidos e o motor de con-

```
PREFIX grsc: <http://www.owl-ontologies.com/Grid_Resources.owl#>

SELECT ?ip, ?so, ?qtdPropcs, ?capProc, ?login
WHERE { ?A grsc:ip_address ?ip .
          ?A grsc:number_of_cpus ?qtdProcs .
          ?A grsc:running_os ?B .
          ?B grsc:os_type ?so .
          ?A grsc:has_processor ?C .
          ?C grsc:processor_capacity ?capProc .
          ?A grsc:authorized_account ?D .
          ?D grsc:distinguished_name ?login .
FILTER ( (?so = 'Linux') && (?qtdProcs = 1) && (?capProc >= 1800) &&
         (?login = '/O=UFSC/OU=INE/CN=parra') )  }
```

Figura 51: Representando a consulta query_1 na linguagem SPARQL

sulta que a reconhece torna transparente ao usuário como navegar na estrutura das ontologias. Adicionalmente, o usuário não precisa conhecer a sintaxe da linguagem SPARQL para poder elaborar consultas.

Além disso, como mencionado na seção 3.6.3, SPARQL não possui mecanismo de ampliação da consulta. Portanto, ao enviar a consulta expressa na Figura 51, o resultado do *matching* é somente o recurso 150.162.60.69, pois o recurso 147.160.55.62 não tem sistema operacional igual a Linux, como pode ser visto na Figura 49.

6 *Resultados Experimentais*

O sistema de *matching* semântico descrito no capítulo 5 foi implementado utilizando as seguintes tecnologias: linguagem de programação Java 1.5.0_06, o editor de ontologias Protégé (KNUBLAUCH; MUSEN; RECTOR, 2004), versão 3.1.1, para criar as ontologias; a API Jena, versão 2.3, para manipulação de ontologias OWL; e a API Protégé-OWL, versão 2.1, para geração automática das consultas baseada na ontologia de pedidos.

Para a validação do sistema proposto foram realizados alguns experimentos sobre ontologias de terceiros. O objetivo de usar ontologias de terceiros é mostrar a diversidade das visões que os recursos de grid podem apresentar. Foram obtidas duas ontologias de terceiros. No entanto, devido à dificuldade de obter ontologias de recursos de grid, foi desenvolvida uma ontologia simples no Protégé, para descrever sistemas de computação e seus principais componentes. Portanto, os experimentos foram realizados sobre três ontologias de recursos distintas.

Na seção a seguir, apresentamos as três ontologias que correspondem às três organizações virtuais que integram o ambiente de grid simulado. O processo de integração semântica das ontologias no sistema é exemplificado pela integração da ontologia da OV_1 no sistema. O resultado da integração das demais ontologias no sistema é apresentado para posteriormente realizarmos os experimentos.

6.1 Ontologias de recursos de grid

Para a realização dos testes, simulamos um ambiente de grid composto de três organizações virtuais, denominadas de OV_1, OV_2 e OV_3, que possuem as suas próprias ontologias de re-

cursos. Apesar da OR desenvolvida permitir a descrição de vários tipos de recursos de grid, nos limitamos a descrever somente os recursos computacionais mais comuns. Essa limitação ocorre devido à falta de modelos abstratos e de ontologias que representem detalhadamente vários outros tipos de recursos encontrados em grids. Para criar um portal pela qual a OV pode integrar a sua ontologia de recursos no sistema, foi desenvolvido um aplicativo. Através do aplicativo, qualquer OV pode estabelecer e informar as relações semânticas da sua ontologia com a OR, permitindo a integração das diferentes ontologias de recursos no sistema. O aplicativo criado é apresentado na Figura 52.

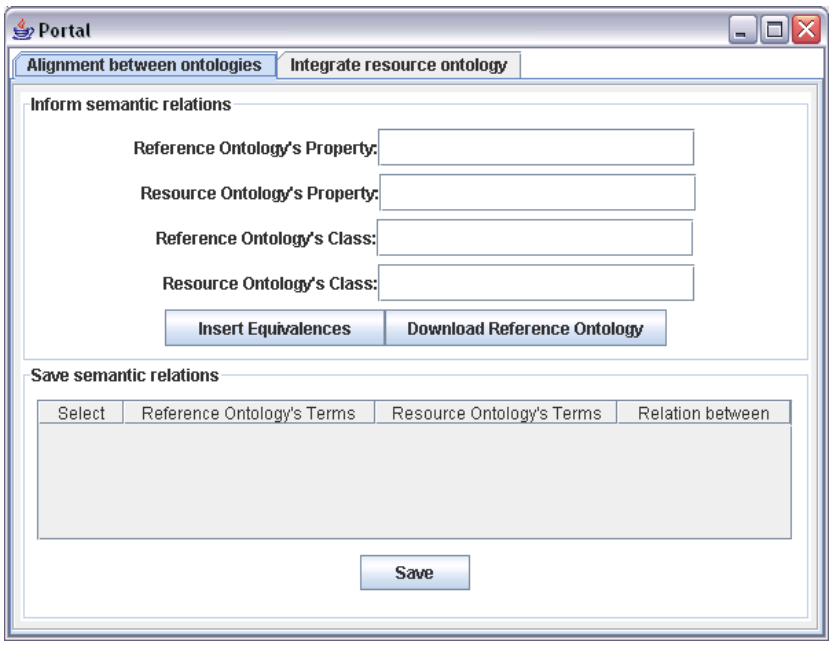


Figura 52: Aplicativo que representa o portal de integração

Através deste aplicativo uma OV pode realizar as seguintes interações com o sistema desenvolvido:

- Solicitar a OR ao sistema para servir como base de integração;
- Estabelecer as relações de equivalência entre termos definidos em sua própria ontologia de recursos com a OR;
- Integrar a sua ontologia enviando as relações ao sistema.

A ontologia de recursos simples desenvolvida neste trabalho foi inspirada no modelo CIM, como ilustrado na Figura 53. Esta ontologia foi desenvolvida no idioma inglês para representar a visão que a OV_1 tem sobre os seus recursos compartilhados no grid.

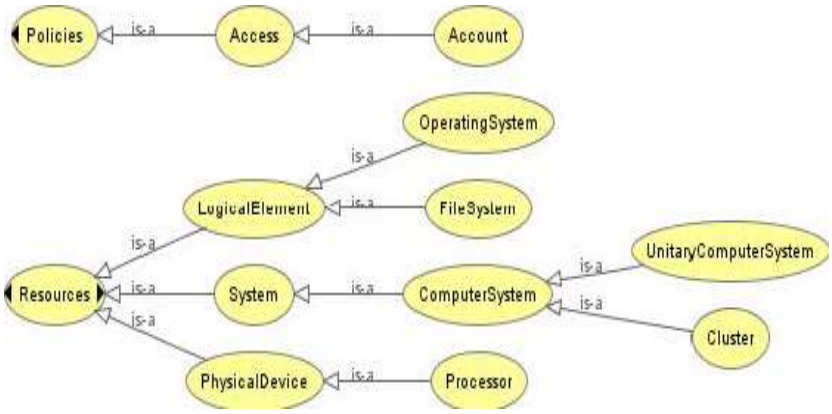


Figura 53: A ontologia de recursos da OV_1

Os conceitos (classes) na ontologia da OV_1 expressam recursos e políticas de acesso às quais esses recursos estão sujeitos. A Tabela 9 apresenta algumas características desses recursos. Todas as propriedades das classes e relações entre classes modeladas nesta ontologia podem ser observadas no Apêndice B.

Tabela 9: Características de recursos e seus significados expressos na ontologia da OV_1

Classe	**Propriedade**	**Descrição Propriedade(metadados)**
FileSystem	available_space file_system_type	Available hard disk space (Unit GB) File system type (acronym)
OperatingSystem	available_physical_memory os_type	Amount of available main memory (Unit MB) Operating system (e.g.: Unix, Fedora Core and Windows XP)

Para a OV_1 integrar a sua ontologia, ela deve verificar as relações de equivalência que a sua ontologia tem com a OR e informá-las ao sistema. Para isso, a OV_1 precisa analisar a estrutura conceitual e os metadados da OR. A OR é compartilhada através do serviço de integração que pode ser acessada a partir do portal. A OR pode ser melhor visualizada por um editor de ontologias, por exemplo, o Protégé como mostrado na Figura 54, para permitir o estabelecimento de relações de equivalência mais precisas.

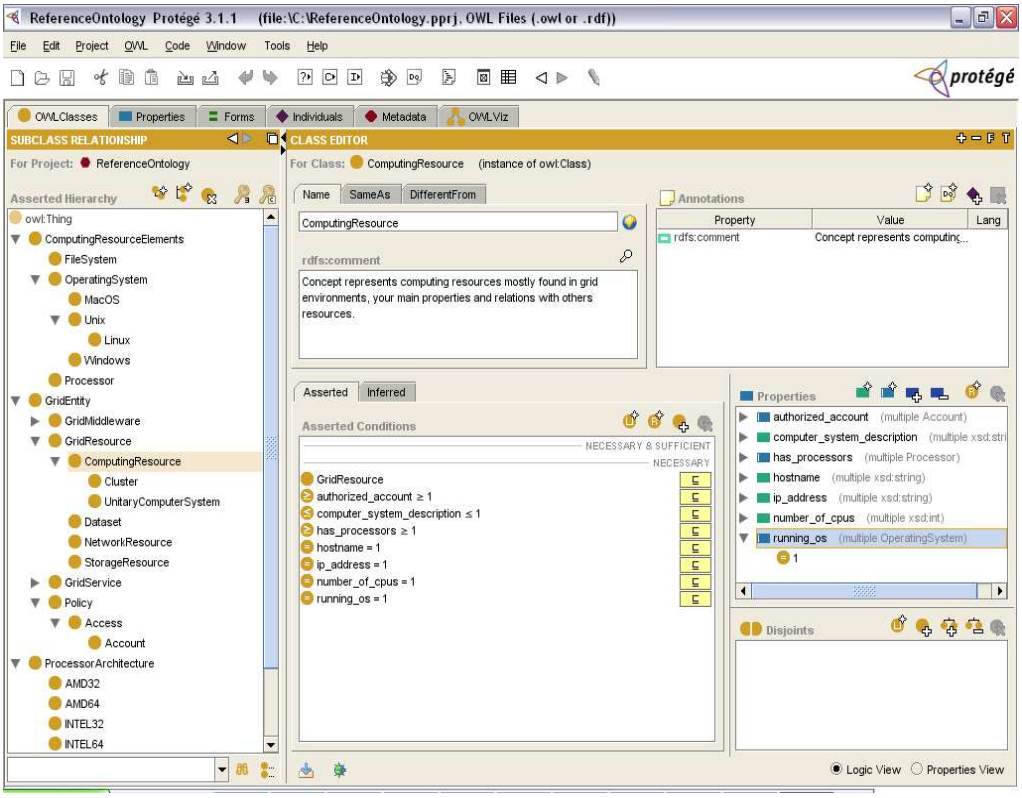


Figura 54: Ontologia de Referência visualizada no Protégé

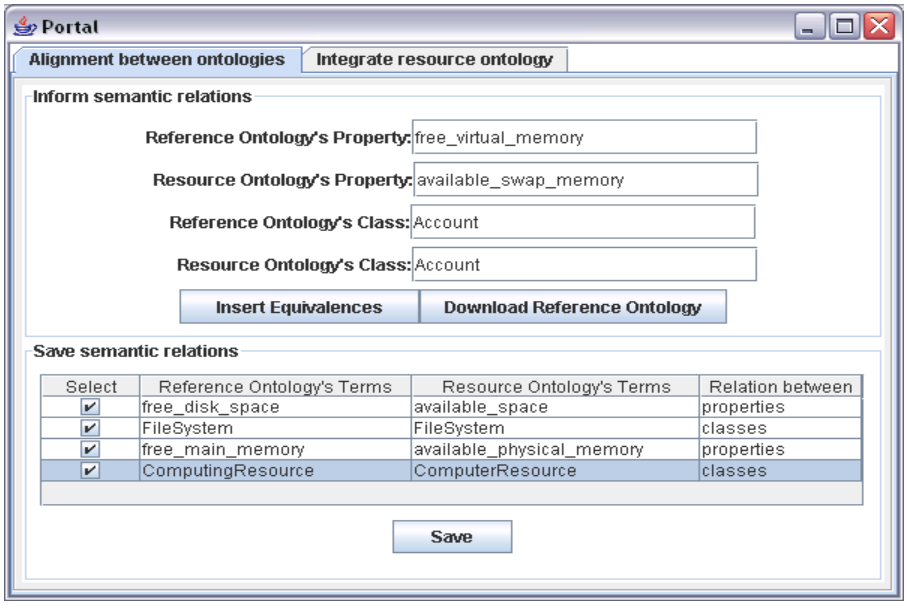


Figura 55: OV_1 estabelecendo relações entre propriedades e entre classes da sua ontologia de recursos (*Resource Ontology*) com a OR (*Reference Ontology*)

As relações podem ser especificadas no portal e enviadas através dele, como ilustradas na Figura 55 e Figura 57. Depois de ter estabelecido as relações, a OV_1 as salva em um arquivo XML para informá-los ao integrar a sua ontologia de recursos no sistema, como pode ser visto na Figura 57. Antes de integrar, a OV_1 deve obter referência à interface do serviço de integração, cuja função é receber as ontologias e as relações semânticas para disponibilizá-las corretamente no sistema. A partir do momento que recebe a referência, ilustrado na Figura 56, a OV_1 pode interagir com o serviço de integração. Este serviço é executado na máquina 150.162.64.166 e para acessá-lo é necessário informar o IP da máquina hospedeira e a porta na qual aguarda as conexões.

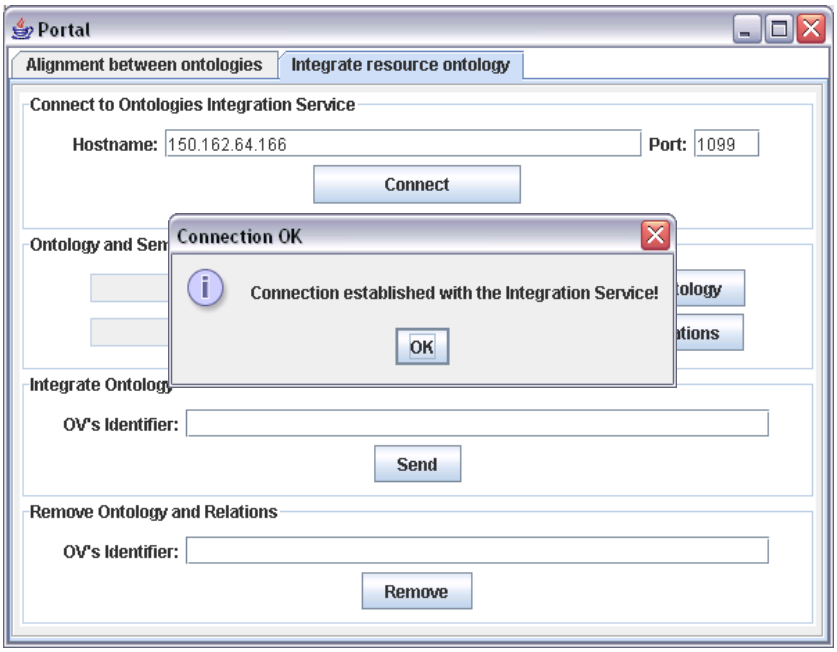


Figura 56: OV_1 conectando-se no serviço de integração

Após estabelecer conexão, a OV_1 precisa informar o arquivo da ontologia que contém a sua terminologia (TBox), as relações e um identificador que represente a organização, como mostrado na Figura 57. Caso a OV_1 venha a se retirar do ambiente de grid, ela pode requisitar a remoção da ontologia e das relações também através do portal, bastando informar o identificador utilizado quando integrou sua ontologia e clicar no botão "Remove".

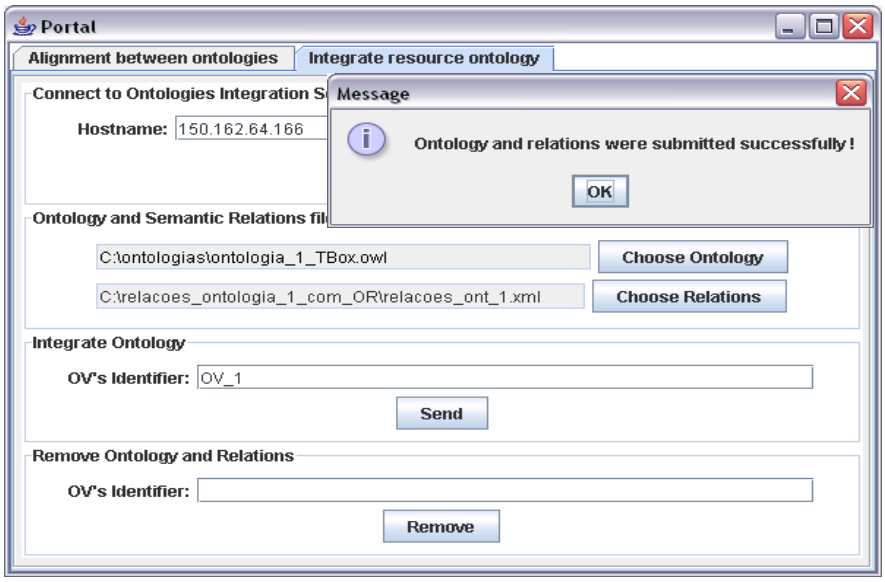


Figura 57: OV_1 integrando a sua ontologia no sistema

Foi desenvolvido um aplicativo que simula os fornecedores de informações de recursos das OVs. Na Figura 58 é apresentado um fornecedor da OV_1 publicando os seus recursos compartilhados no grid.

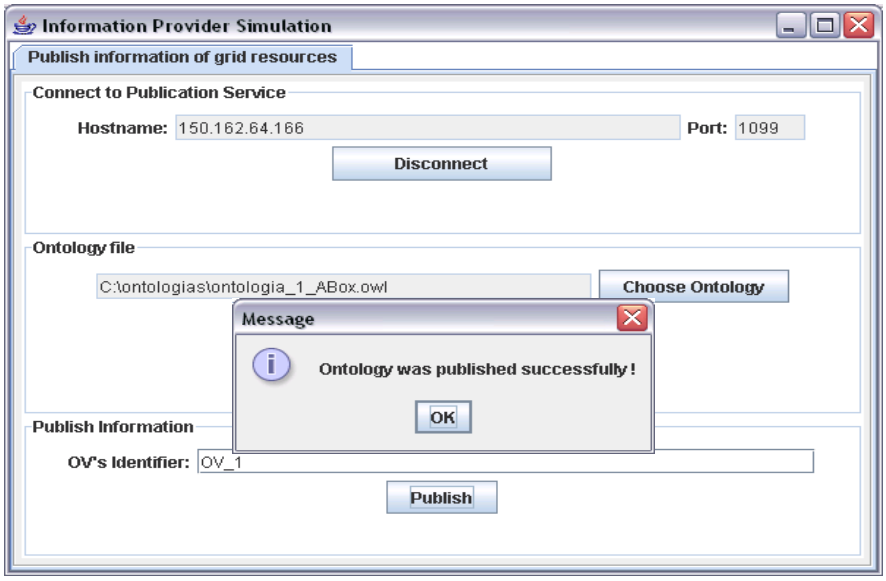
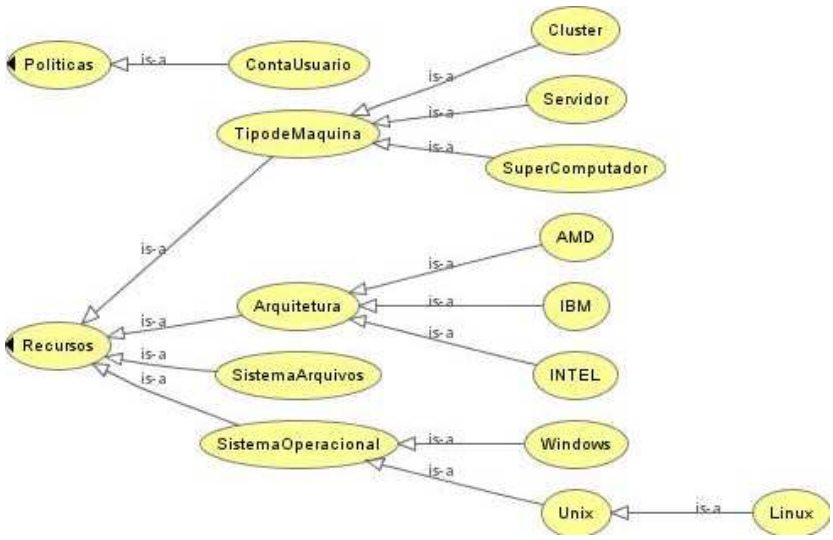
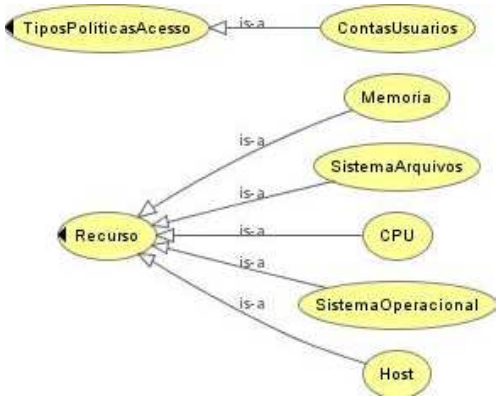


Figura 58: Publicando informações dos recursos da OV_1 pelo fornecedor de informações

As outras duas organizações, OV_2 e OV_3, tiveram os seus recursos expressos respectivamente pelas ontologias desenvolvidas por Pernas & Dantas (PERNAS; DANTAS, 2005) e Lopes et. al (LOPES et al., 2006). As duas ontologias foram estendidas para expressar políticas de acesso. A última ontologia é uma adaptação do trabalho de descoberta de recursos do grid proposto por Ramos (RAMOS, 2006), onde a partir do esquema XML desenvolvido para a representação de recursos do grid foi construída uma ontologia. Estas duas ontologias podem ser vistas na Figura 59.



(a) Ontologia de recursos da OV_2



(b) Ontologia de recursos da OV_3

Figura 59: As ontologias de recursos das organizações OV_2 e OV_3

As Tabelas 10 e 11 ilustram algumas características dos recursos definidos nas ontologias da OV_2 e OV_3. Propriedades dos conceitos e relacionamentos entre conceitos das duas ontologias encontram-se no Apêndice B.

Tabela 10: Características de recursos e seus significados expressos na ontologia da OV_2

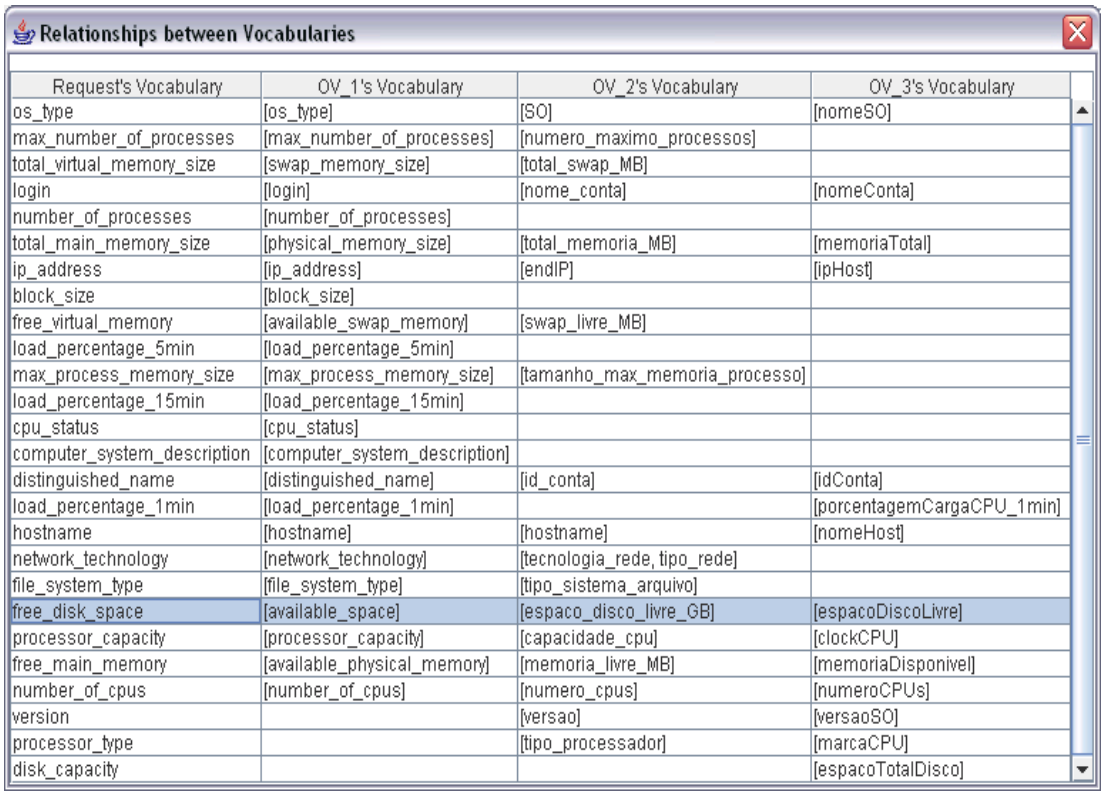
Classe	Propriedade	Descrição da Propriedade (metadados)
SistemaArquivos	espaco_disco_livre_GB	Espaço em disco rígido disponível, expresso em GB.
	tipo_sistema_arquivo	Tipo de sistema de arquivos (ex: EXT3, NTFS and ZFS)
SistemaOperacional	SO	Sistema operacional (ex: Debian, AIX, SunOS and Linux.)
TipodeMaquina	memoria_livre_MB	Quantidade de memória RAM disponível, expresso em MB.
	total_disco_GB	Capacidade do disco rígido, expresso em GB.

Tabela 11: Características de recursos e seus significados expressos na ontologia da OV_3

Classe	Propriedade	Descrição da Propriedade (metadados)
SistemaArquivos	espacoDiscoLivre	Espaço em disco disponível em GB.
SistemaOperacional	nomeSO	Nome do sistema operacional.
Memoria	memoriaDisponivel	Quantidade de memória principal disponível em MB.
	memoriaTotal	Tamanho da memória principal em MB.
Host	ipHost	Número IP do host.
	numeroCPUs	Quantidade de processadores do host.

As organizações OV_2 e OV_3, igualmente como ocorreu com a OV_1, integraram as suas ontologias no sistema desenvolvido. Posteriormente, publicaram os seus recursos sobre os quais serão realizados os experimentos de *matching* semântico. A Figura 60 mostra a visão que o *matchmaker* tem após a integração das três ontologias.

A linha destacada na Figura 60 ilustra a equivalência semântica entre os termos *available_space*, *espaco_disco_livre_GB* e *espacoDiscoLivre* respectivamente definidos nas ontologias das organizações OV_1, OV_2 e OV_3 com o termo *free_disk_space* da OR. Esta relação informa ao *matchmaker* que ao receber uma consulta com uma restrição sobre o termo *free_disk_space*, ele deverá verificar se as descrições dos recursos com respeito às características representadas pelos termos *available_space*, *espaco_disco_livre_GB* e *espacoDiscoLivre* atendem a restrição.



Request's Vocabulary	OV_1's Vocabulary	OV_2's Vocabulary	OV_3's Vocabulary
os_type	[os_type]	[SO]	[nomeSO]
max_number_of_processes	[max_number_of_processes]	[numero_maximo_processos]	
total_virtual_memory_size	[swap_memory_size]	[total_swap_MB]	
login	[login]	[nome_conta]	[nomeConta]
number_of_processes	[number_of_processes]		
total_main_memory_size	[physical_memory_size]	[total_memoria_MB]	[memoriaTotal]
ip_address	[ip_address]	[endIP]	[ipHost]
block_size	[block_size]		
free_virtual_memory	[available_swap_memory]	[swap_livre_MB]	
load_percentage_5min	[load_percentage_5min]		
max_process_memory_size	[max_process_memory_size]	[tamanho_max_memoria_processo]	
load_percentage_15min	[load_percentage_15min]		
cpu_status	[cpu_status]		
computer_system_description	[computer_system_description]		
distinguished_name	[distinguished_name]	[id_conta]	[idConta]
load_percentage_1min	[load_percentage_1min]		[porcentagemCargaCPU_1min]
hostname	[hostname]	[hostname]	[nomeHost]
network_technology	[network_technology]	[tecnologia_rede, tipo_rede]	
file_system_type	[file_system_type]	[tipo_sistema_arquivo]	
free_disk_space	[available_space]	[espaco_disco_livre_GB]	[espacoDiscoLivre]
processor_capacity	[processor_capacity]	[capacidade_cpu]	[clockCPU]
free_main_memory	[available_physical_memory]	[memoria_livre_MB]	[memoriaDisponivel]
number_of_cpus	[number_of_cpus]	[numero_cpus]	[numeroCPUs]
version		[versao]	[versaoSO]
processor_type		[tipo_processador]	[marcaCPU]
disk_capacity			[espacoTotalDisco]

Figura 60: Relações de equivalência entre as ontologias das organizações e a OR

Cabe ressaltar que as células vazias encontradas na tabela ilustrada na Figura 60 indicam que não há termos equivalentes nas três ontologias de recursos das OVs integradas (representados pelas colunas 2, 3 e 4) com os termos definidos na ontologia de pedidos (primeira coluna).

6.2 Estudos de Caso

As consultas por recursos foram realizadas sobre as descrições dos recursos publicadas pelas três organizações que compõem o grid. Cada organização publicou dez descrições de recursos computacionais, simulando no total trinta recursos de grid. Juntamente com esses recursos foram informadas as equivalências semânticas estabelecidas por cada organização ao integrar a sua ontologia no sistema. Essas equivalências são encontradas na Figura 60. Os testes foram realizados sobre duas máquinas: a primeira (M1) é de onde foram elaboradas e enviadas as consultas (lado do cliente) e a segunda (M2) onde o serviço *matchmaker* está sendo executado aguardando pelas consultas. As configurações das duas máquinas, M1 e M2, podem

ser encontradas na Tabela 12.

Tabela 12: Configuração das máquinas usadas nos testes

Máquina	Sistema Operacional	Memória	Processador
M1	Windows XP	256 MB	Duron (1200 MHz)
M2	Debian 3.1	512 MB	Pentium IV (1800 MHz)

As consultas criadas para os experimentos a seguir foram submetidas ao *matchmaker* visando mostrar as principais características do sistema de *matching* semântico desenvolvido.

6.2.1 Ampliando a consulta por recursos

Como citado, o mecanismo de *matching* semântico desenvolvido além de basear-se na integração semântica das ontologias publicadas no sistema, utiliza-se de regras de inferência sobre a estrutura de conhecimento modelada e nos indivíduos definidos na OR para ampliar as consultas. A consulta definida na Figura 61, busca por recursos que dentre os vários requisitos devem ter processadores AMD e sistema operacional Unix.

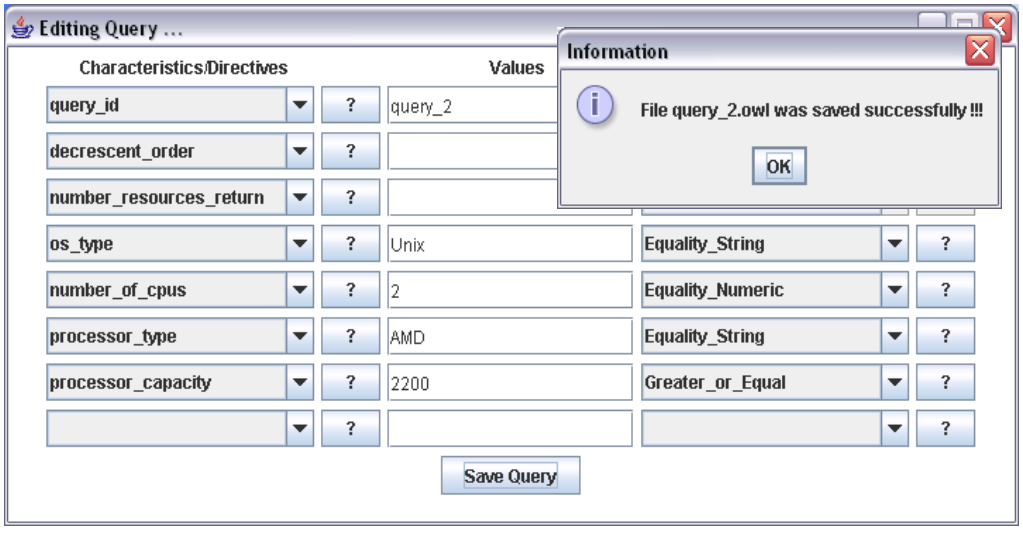


Figura 61: Definindo a consulta identificada por query_2

Em um sistema de *matching* sintático exato, teríamos como resultado somente recursos que tivessem processadores e sistemas operacionais com valores exatamente iguais a "AMD" e "Unix". Para evidenciarmos a característica de ampliação da busca no mecanismo de *matching*

semântico proposto, retiramos as regras de inferência, ilustradas na Figura 62, do sistema antes de enviarmos a consulta query_2 ao *matchmaker*.

```
<inference_rules >

   <rule > [ GetOSsLikeUNIX: (?A greq:os_type ?SO)(?B rdf:type owl:Class) equalURI(?B 'Unix') equalURI(?B ?SO) (?D
      rdf:type ?B) (?D ont_ref:os_type ?SO1) (?E rdfs:subClassOf ?B) equalURI(?E 'Linux') (?F rdf:type ?E) (?F
      ont_ref:os_type ?SO2) -> (?G gres:result_value ?SO1) (?H gres:result_value ?SO2) (?I gres:result_value 'Unix')
      (?F gres:result_value 'Linux') ]
   </rule>

    <rule > [GetArchProcessorsAMD: (?A greq:processor_type ?proc1) (?B ont_ref:architecture_type ?C) (?D
      ont_ref:architecture_type ?E) (?C rdf:type ?F) (?E rdf:type ?G) equalURI(?proc1 'AMD') equalURI(?F 'AMD32')
      equalURI(?G 'AMD64') (?B ont_ref:processor_type ?proc2) (?D ont_ref:processor_type ?proc3) -> (?D
      gres:result_value ?proc2) (?D gres:result_value ?proc3) ]
   </rule >
...
```

Figura 62: Regras de ampliação da consulta por recursos com sistemas operacionais Unix e processadores AMD

A consulta query_2 busca por recursos que tenham dois processadores AMD com capacidade de processamento superior ou igual a 2200 MHz e sistema operacional Unix. O resultado desta consulta é "No Match !!!", ou seja, não há recursos no grid que possuam essas características, como mostrado na Figura 63.

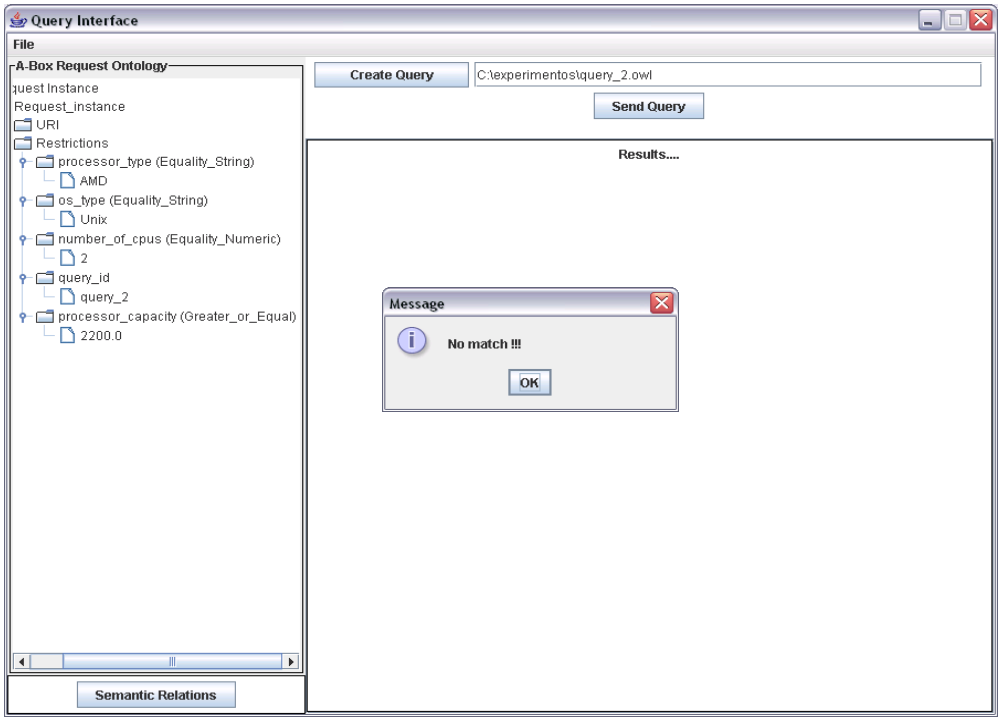


Figura 63: Resultado da pesquisa conforme a query_2 sem as duas regras

A mesma consulta é novamente enviada ao *matchmaker*, no entanto as regras retiradas

foram inseridas no sistema. Após o processo de *matching* semântico, o *matchmaker* retorna dois recursos, conforme a Figura 64 ilustra.

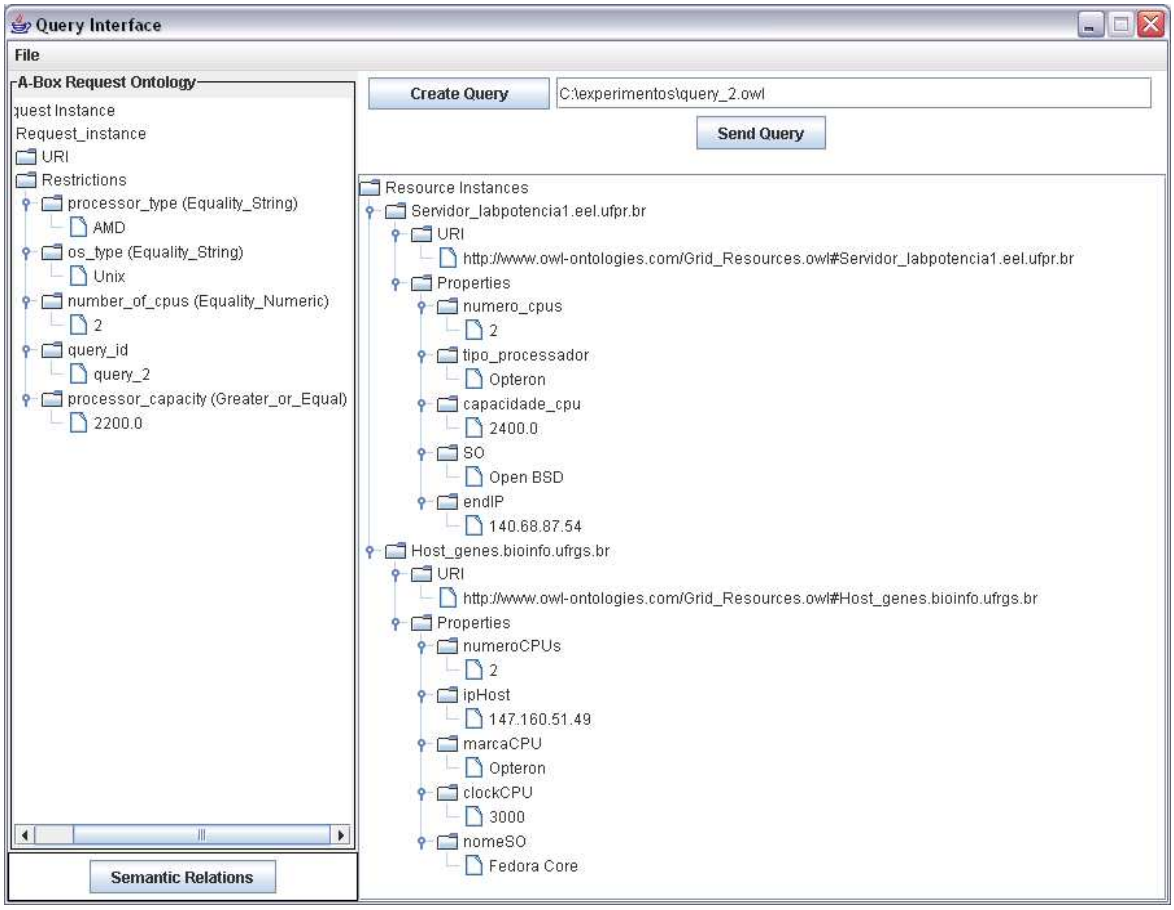


Figura 64: Resultado da pesquisa conforme a query_2 com as duas regras

O primeiro recurso retornado da consulta query_2 tem o processador AMD, Opteron; e sistema operacional Unix, Open BSD. O segundo tem processador Opteron e sistema operacional Fedora Core. Portanto, a modelagem do conhecimento (através de regras) sobre processadores e sistemas operacionais e as instâncias (indivíduos) definidas na OR, como mostrado no Apêndice C, permitem ampliar o processo de *matching*, não se restringindo a uma comparação sintática exata.

6.2.2 Verificando a consistência das consultas

O sistema de *matching* semântico desenvolvido utiliza-se também de regras para executar verificações de consistência sobre as consultas enviadas ao *matchmaker*. Estas verificações

são importantes, pois permitem ao *matchmaker* desconsiderar consultas incoerentes, economizando recursos do sistema. A Figura 65 apresenta a consulta query_3 que busca por recursos que tenham sistema operacional Windows XP, com sistema de arquivos EXT2, quantidade de processadores superior ou igual a 16 sendo estes processadores UltraSparc II e com capacidade superior ou igual a 600 MHz. O resultado da busca deve conter no máximo 2 recursos que atendam todas estas restrições e que sejam retornados na ordem decrescente da sua quantidade de memória RAM livre.

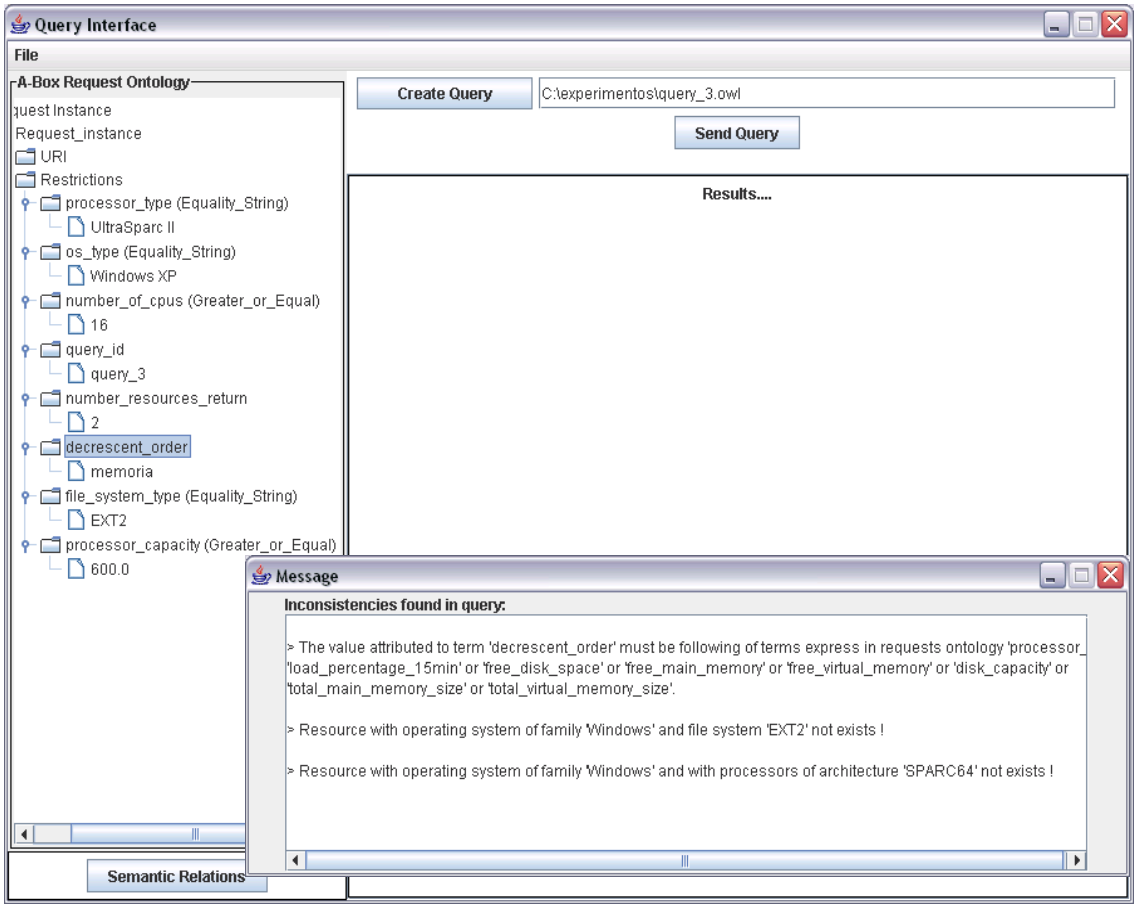


Figura 65: Resultado da pesquisa mostrando as inconsistências na consulta query_3

Após o envio da query_3, o *matchmaker* retornou mensagens informando três inconsistências encontradas na consulta. A primeira avisa que o valor (memoria) atribuído à diretiva *decrescent_order* deve ser um dos termos que representam as características de recursos (de tipo numérico) descritos na ontologia de pedidos. Neste exemplo, ao desejar ordenar o resultado pela quantidade de memória RAM disponível, o usuário deveria utilizar o termo *free_main_memory*.

A segunda indica que não há recursos com sistema operacional da família Windows e com sistema de arquivos EXT2. A terceira informa que não existem recursos com sistema operacional da família Windows e com processadores da arquitetura SPARC64. Estas mensagens são as conclusões das regras, depois da avaliação das suas premissas apontarem que as condições que caracterizam as inconsistências são verdadeiras. A Figura 66 apresenta as três regras que capturam as três inconsistências encontradas na consulta anterior.

```
<rules >
 <validation_ rule > [ [ checkingPossibleValuesTermDecrescent_Order: (?A greq:decrescent_order ?reqOrdenacao)
    notEqual(?reqOrdenacao 'processor_capacity') notEqual(?reqOrdenacao 'number_of_cpus')
    notEqual(?reqOrdenacao 'load_percentage_1min') notEqual(?reqOrdenacao 'load_percentage_5min')
    notEqual(?reqOrdenacao 'load_percentage_15min') notEqual(?reqOrdenacao 'free_disk_space')
    notEqual(?reqOrdenacao 'free_main_memory') notEqual(?reqOrdenacao 'free_virtual_memory')
    notEqual(?reqOrdenacao 'disk_capacity') notEqual(?reqOrdenacao 'total_main_memory_size')
    notEqual(?reqOrdenacao 'total_virtual_memory_size') -> (?A error The__value__attributed__to__term__
    'decrescent_order'__must__be__following__of__terms__express__in__ request__ontology__'processor_capacity'
    __or__'number_of_cpus'__or__'load_percentage_1min'__or__' load_percentage_5min'__or__
    'load_percentage_15min' __or__'free_disk_space'__or__'free_main_memory'__or__'free_virtual_memory'__or __
    'disk_capacity'__or__'total_main_memory_size'__or__'total_virtual_memory_size'.) ]
 </validation_r ule>
 <validation_rule > [ checkingConsistencyEXT2withOSsFamilyWindows: (?A greq:file_system_type ?FS) (?B
    greq:os_type ?OS) (?C rdf:type owl:Class) equal(?FS 'EXT2') equalURI(?C 'Windows') (?D rdf:type ?C) (?D
    ont_ref:os_type ?SO1) equalURI(?OS ?SO1) -> (?D error Resource__with__operating__system__of__family__
    'Windows'__and__file__system__'EXT2'__not__exists__!)
 </validation_rule >
 <validation_rule > [ checkingConsistencyArchProcessorsSPARC64withOSsFamilyWindows: (?A greq:processor_type
    ?source_proc) (?B greq:os_type ?source_os) (?Windtype rdf:type owl:Class) equalURI(?arch 'SPARC64')
    equalURI(?Windtype 'Windows') (?D rdf:type ?Windtype) (?D ont_ref:os_type ?target_os) (?E rdf:type ?arch) (?F
    ont_ref:architecture_type ?E) (?F ont_ref:processor_type ?target_proc) equalURI(?source_proc ?target_proc)
    equalURI(?source_os ?target_os) -> (?F error Resource__with__operating__system__of__family__'Windows'__
    and__with__processors__of__architecture__'SPARC64'__not__exists__!) ]
 </validation_rule >
...
```

Figura 66: Regras que detectaram as inconsistências presentes na consulta query_3

6.2.3 Demais características do sistema de *matching* semântico

Como citado, a ontologia de pedidos desenvolvida permite ao usuário do sistema, através de suas diretivas, informar ao *matchmaker* o número limite de recursos e/ou a condição de ordenação dos recursos a serem retornados após pesquisa.

Na consulta apresentada na Figura 48, foram estipuladas as seguintes restrições de pesquisa: recursos que tenham sistema operacional Linux com um único processador de capacidade superior ou igual a 1800 MHz e que possa se ter acesso através do identificador único no grid "/O=UFSC/OU=INE/CN=parra". Os recursos 147.160.55.62 e 150.162.60.69 satisfazem essas restrições como pode ser observado na Figura 49. O resultado da consulta query_1 servirá de base para demonstrar a funcionalidade das diretivas descritas na ontologia de pedidos. As próximas duas consultas, query_4 e query_5, enviadas ao *matchmaker* têm as mesmas restrições definidas na query_1.

A query_4 foi definida, como pode ser visto na Figura 67, para mostrar como o *matchmaker* considera uma consulta que expressou um limite máximo de recursos a retornar. A query_4 é submetida ao *matchmaker* e como resultado tem-se o recurso com o endereço IP 150.162.60.69. Nota-se que a condição de retornar somente um único recurso number_resources_return = 1 foi atendida pelo *matchmaker*, que dentre os dois recursos 147.160.55.62 e 150.162.60.69 possíveis retornou aleatoriamente um deles.

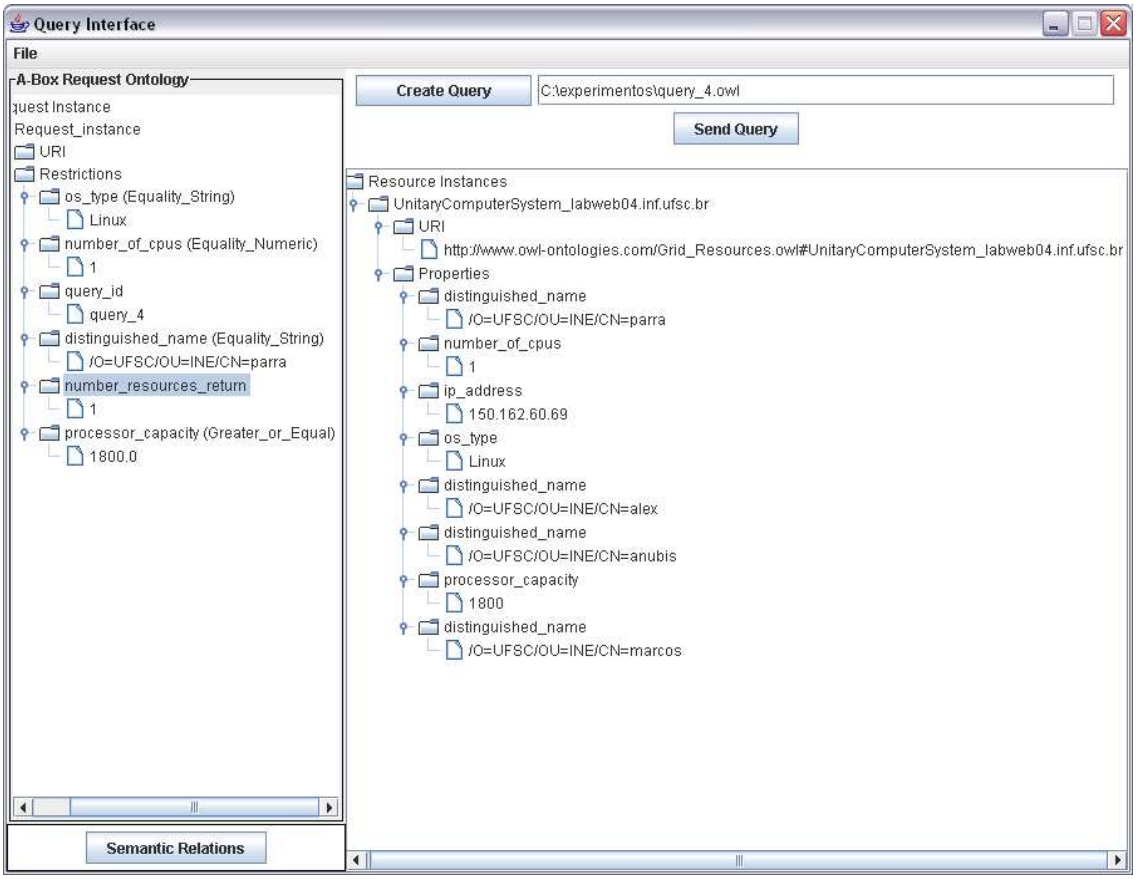


Figura 67: Resultado da pesquisa de recursos de acordo com a consulta query_4

A interface de consulta ilustrada na Figura 68 mostra a query_5 destacando o critério de ordenação decrescente baseado na capacidade de processamento (decrescent_order = processor_capacity) e o limite de recursos a retornar igual a 1 (number_resources_return = 1). Em outras palavras, busca-se um recurso que tenha a maior capacidade de processamento dentre todos os recursos que satisfaçam as restrições. O resultado indica que o recurso 147.160.55.62 apresentou a maior capacidade de processamento em relação ao recurso 150.162.60.69 retor-

nado na consulta anterior. O recurso 147.160.55.62 tem um processador de 2100 MHz contra 1800 MHz do recurso 150.162.60.69. As descrições dos dois recursos podem ser encontradas na Figura 49.

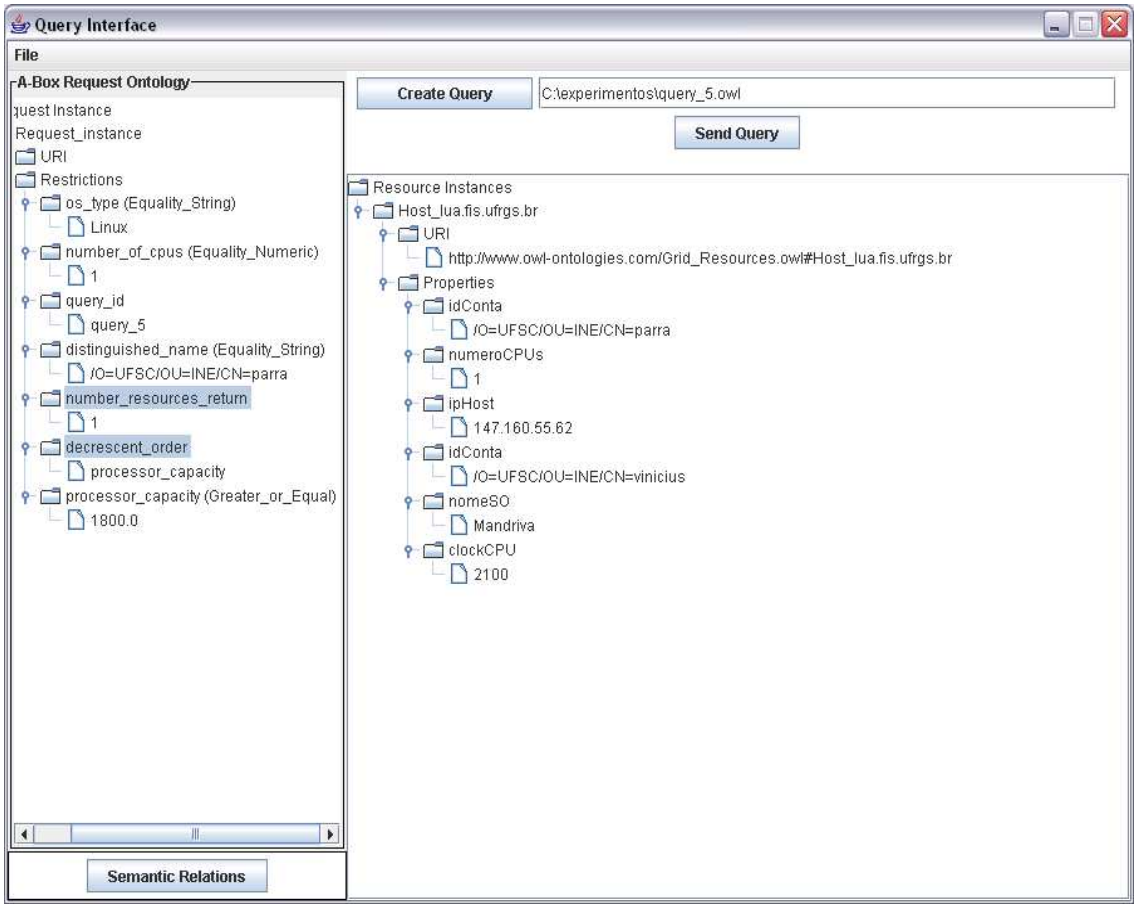


Figura 68: Resultado da pesquisa por recursos de acordo com a consulta query_5

7 *Considerações Finais e Trabalhos Futuros*

Nesta dissertação, foi proposto e implementado um mecanismo de *matching* semântico para a localização de recursos computacionais em ambientes de grid descritos por diferentes ontologias. Pesquisas realizadas no capítulo 4 mostraram a necessidade de tal mecanismo, visto que não temos uma ontologia padrão para a descrição de recursos de grid e por acreditarmos que tal ontologia dificilmente existirá. O mecanismo considera ainda a dificuldade encontrada por usuários comuns em interagir com sistemas baseados em conhecimento.

O processo de integração de ontologias para a realização de *matching* semântico apresentou-se mais completo em relação ao trabalho de Lopes et. al. (LOPES et al., 2006). Esta característica foi alcançada permitindo que diferentes ontologias tenham suas equivalências semânticas não somente estabelecidas sobre os seus conceitos, mas também pelas propriedades que os caracterizam permitindo dessa forma realizar pesquisas mais detalhadas. Optamos em estabelecer as equivalências através da interação de humanos, pois pesquisas apontam que atualmente as ferramentas automáticas não conseguem reconhecer e/ou exprimir formalmente a maioria das equivalências semânticas existentes entre ontologias, ocorrendo perda de informações e conseqüentemente falta de interoperabilidade. Outra característica que torna o processo de integração mais completo é a utilização de regras para ampliar semanticamente as consultas e reconhecer as suas inconsistências. O emprego de regras torna ainda o processo de *matching* semântico flexível e extensível.

Os resultados experimentais realizados e apresentados na seção 6.2 mostram a importância do mecanismo proposto. Estes resultados foram realizados sobre ontologias de diferentes ta-

manhos, diferentes estruturas conceituais e escritas em diferentes línguas, português e inglês. A ontologia de referência desenvolvida mostrou-se capaz de servir de base para alinhamentos semânticos de diferentes ontologias de recursos do grid. Estes alinhamentos denotam a equivalência entre as ontologias de recursos de grid e a ontologia de referência.

O desenvolvimento de uma ontologia (ontologia de pedidos) como uma linguagem de consulta utilizando o vocabulário da ontologia de referência permitiu facilitar a interação de usuários comuns com o sistema de *matching* semântico proposto. Esta facilidade foi alcançada abstraindo do usuário a necessidade de saber como expressar formalmente os conhecimentos sobre recursos de grid modelados nas ontologias OWL para poder consultá-las. Esta abstração ocorre, pois foi criado um motor de consulta que navega nas estruturas das ontologias baseado nas restrições descritas na consulta e nas equivalências estabelecidas e informadas previamente pelos desenvolvedores das ontologias das organizações virtuais que compõem o grid. Em adição, tendo uma ontologia como uma linguagem de consulta pode-se usar os metadados presentes nela para facilitar ao usuário a elaboração das consultas.

As interfaces criadas para editar consultas e consultar os recursos do grid, mostraram-se bastante simples, porém claras quanto aos seus propósitos.

Da presente pesquisa, pode-se visualizar algumas direções para trabalhos futuros, quais são:

- Utilização do mecanismo em um ambiente de grid real;
- Estender a ontologia de referência, caracterizando os seguintes conceitos: recursos de rede, de armazenamento entre outros;
- Ampliar a capacidade de restrição das consultas, estendendo a ontologia de pedidos para que esta expresse outros operadores binários, tais como: menor (<), maior (>), diferente (!=), multiplicidade (*), entre outros;
- Modelar conhecimentos através de regras para identificar inconsistências nas descrições dos recursos do grid antes de aceitá-los como recursos disponíveis para consulta;

- Criar mecanismos que considerem a atualização somente dos dados dinâmicos das descrições dos recursos do grid.

Referências

AIKEN, R.; CAREY, M.; CARPENTER, B.; FOSTER, Ian; LYNCH, C.; MAMBRETTI, J.; MOORE, R.; STRASNNER, J.; TEITELBAUM, B. *Network Policy and Services: A Report of a Workshop on Middleware*. [S.l.], 2000. Technical Report, RFC 2768, IETF.

AMBROSIO, Ana Paula; SANTOS, Dirson C. de; LUCENA, Fabio N. de; SILVA, Joao Carlos da. Software engineering documentation: An ontology-based approach. *WebMedia & LA-Web 2004 Joint Conference 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress*, IEEE Computer Society Digital Library, p. 38–40, 2004. Último acesso: 10/02/2006.

ANTONIO, Moreno; CHANTAL, Perez. Reusing the mikrokosmos ontology for concept-based multilingual terminology databases. *2nd International Conference on Language Resources & Evaluation (LREC)*, May 2000. Último acesso: 10/02/2006.

BECHHOFER, Sean. *Interface DIG - DL Implementors Group*. Junho 2006. Último acesso: 13/06/2006.

BOLEY, Harold; TABET, Said; WAGNER, Gerd. Design rationale for ruleml: A markup language for semantic web rules. In: *Semantic Web Working Symposium*. [S.l.: s.n.], 2001. p. 381–401. Disponível em http://www.semanticweb.org/SWWS/program/full/paper20.pdf, Último acesso: 10/02/2006.

BROOKE, John; FELLOWS, Donal; GARWOOD, Kevin L.; GOBLE, Carole A. Semantic matching of grid resource descriptions. *2nd European Across Grids Conference*, p. 240–249, 2004.

BUYYA, R. *Grid Computing Info Centre (GRID Infoware)*. 2002. Disponível em: http://www.gridcomputing.com/, Último acesso: 17/06/2006.

CANNATARO, Mario; COMITO, C. A datamining ontology for grid programming. *1st International Workshop on Semantics in PeertoPeer and Grid Computing*, p. 113–134, May 2003. Último acesso: 10/02/2006.

CANNATARO, Mario; COMITO, Carmela; GUZZO, Antonella; VELTRI, Pierangelo. Integrating ontology and workflow in proteus, a grid-based problem solving environment for bioinformatics. *International Conference on Information Technology: Coding and Computing (ITCC)*, IEEE Computer Society Digital Library, v. 2, p. 90–94, April 2004. Último acesso: 10/02/2006.

CASARE, Sara; SICHMAN, Jaime Simão. Using a functional ontology of reputation to interoperate different agent reputation models. *Journal of the Brazilian Computer Society - Special Issue on Ontology Issues and Applications*, Brazilian Computer Society, v. 11, n. 2, p. 19 – 94, November 2005.

CASTELFRANCHI, C.; FALCONE, R.; PEZZULO, G. Trust in information sources as a source for trust: A fuzzy approach. In: *Proceedings of 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*. Melbourne, AUS: ACM, 2003. p. 89–96.

CHANDRASEKARAN, Balakrishnan; JOSEPHSON, J. R.; BENJAMINS, V. R. Ontologies: What are they, and why do we need them? *IEEE Intelligent Systems and their applications*, v. 14, n. 1, p. 20–26, January 1999.

CHAUDHRI, V. K.; FARQUHAR, A.; FIKES, R.; KARP, P. D.; RICE, J. P. *Open Knowledge Base Connectivity 2.0*. [S.l.], 1997. Technical Report KSL-98-06.

CHAUDHRI, Vinay K.; FARQUHAR, Adam; FIKES, Richard; KARP, Peter D.; RICE, James P. Okbc: a programmatic foundation for knowledge base interoperability. In: *Proceedings of the 15th national/tenth conference on Artificial Intelligence/Innovative applications of artificial intelligence*. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1998. p. 600–607.

CONGIUSTA, Antonio; MASTROIANNI, Carlo; PUGLIESE, Andrea; TALIA, Domenico; TRUNFIO, Paolo. Enabling knowledge discovery services on grids. *2nd European Across Grids Conference*, Spring-Verlag, v. 3165, p. 250–259, 2004.

CONTE, R.; PAOLUCCI, M. *Reputation in Artificial Societies: Social Beliefs for Social Order*. Norwell, MA, USA: Kluwer Acadenic Publishers, 2002.

CORCHO, Oscar; GOMEZ-PEREZ, Asuncion. A roadmap to ontology specification languages. In: *12th International Conference on Knowledge Engineering and Knowledge Management*. [S.l.]: Springer-Verlag GmbH, 2000. p. 80–96.

CZAJKOWSKI, K.; FOSTER, Ian; KESSELMAN, Carl; SANDER, V.; TUECKE, Steven. A protocol for negotiating service level agreements and coordinating resource management in distributed systems. *8th Workshop on Job Scheduling Strategies for Parallel Processing*, v. 2537, p. 153–183, 2002.

DANTAS, Mario Antonio Ribeiro. *Computação Distribuída de Alto Desempenho: Redes, Clusters E Grids Computacionais*. [S.l.]: Axcel Books do Brasil Editora, 2005.

DAVIES, John; FENSEL, Dieter; HARMELEN, Frank van. *Towards the Semantic Web: Ontology-driven Knowledge Management*. [S.l.]: John Wiley and Sons Ltd, 2003.

DECKER, Stefan; SINTEK, Michael. Triple - a query, inference, and transformation language for the semantic web. *In Proceedings of the 13th International Semantic Web Conference (ISWC)*, Springer Verlag, n. 2342, p. 364 – 378, 2002. Lecture Notes in Computer Science.

DMTF. *Common information model (CIM) standards*. Agosto 2006. Último acesso: 13/08/2006.

DüRST, M.; SUIGNARD, M. *Internationalized Resource Identifiers (IRIs)*. [S.l.], January 2005.

EHRIG, M.; STAAB, S. Qom - quick ontology mapping. *In 3rd International Semantic Web Conference (ISWC)*, Springer-Verlag, p. 683–696, 2004.

EHRIG, Marc; SURE, York. Ontology mapping - an integrated approach. In: *Proceedings of the First European Semantic Web Symposium (ESWS)*. [S.l.]: Springer-Verlag, 2004. v. 3053, p. 76–91.

FARQUHAR, Adam; FIKES, Richard; RICE, James. Tools for assembling modular ontologies in ontolingua. *14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference*, MIT Press, p. 436–441, July 1997.

FENSEL, Dieter. Ontologies: Silver bullet for knowledge management and electronic commerce. *Springer-Verlag*, 2000. Último acesso: 10/02/2006.

FENSEL, D.; CRUBEZY, M.; HARMELEN, F. van; HORROCKS, I. Oil & upml: A unifying framework for the knowledge web. In: *In Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence (ECAI)*. [S.l.: s.n.], 2000. Citeseer.ist.psu.edu/443527.html.

FENSEL, Dieter; HORROCKS, Ian; HARMELEN, Frank van; DECKER, Stefan; ERDMANN, Michael; KLEIN, Michel C. A. Oil in a nutshell. In: *Proceedings of the European Knowledge Acquisition Conference*. [S.l.]: Springer-Verlag, 2000. p. 1–16. Lecture Notes in Artificial Intelligence (LNAI).

FLEISCHMANN, Ana Marilza Pernas. *Ontologias Aplicadas à Descrição de Recursos em Grids Computacionais*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina (UFSC), 2004.

FOSTER, Ian; GEISLER, J.; NICKLESS, W.; SMITH, W.; TUECKE, Steven. Software infrastructure for the i-way high performance distributed computing experiment. *5th IEEE Symposium on High Performance Distributed Computing*, p. 562–571, 1997.

FOSTER, Ian; KESSELMAN, Carl. The globus project: A status report. *In Proceedings of 17th IEEE Heterogeneous Computing Workshop (HCW)*, IEEE Computer Society Press, v. 15, n. 5-6, p. 4–18, March 1998. Disponível em: citeseer.ist.psu.edu/foster98globus.html, Último acesso: 16/02/2006.

FOSTER, Ian; KESSELMAN, Carl. *The Grid: Blueprint for a New Computing Infrastructure*. [S.l.]: Morgan Kaufmann, 1999.

FOSTER, Ian; KESSELMAN, Carl. *The Grid 2: Blueprint for a new Computing Infrastructure*. [S.l.]: John Wiley and Sons Ltd, 2003.

FOSTER, Ian; KESSELMAN, Carl; TUECKE, Steven. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, v. 3, n. 15, 2001.

FREITAS, Fred; STUCKENSCHMIDT, Heiner; NOY, Natalya F. Ontology issues and applications guest editors' introduction. *Journal of the Brazilian Computer Society - Special Issue on Ontology Issues and Applications*, Brazilian Computer Society, v. 11, n. 2, p. 5–16, November 2005.

GANGEMI, A.; GUARINO, N.; MASOLO, C.; OLTRAMARI, A. Sweetening wordnet with dolce. *Artificial Intelligence Magazine*, v. 24, n. 3, p. 13 – 24, 2003.

GAVA, Tânia; MENEZES, Crediné. Especificação de software baseada em ontologias. *III Escola de Informática*, Sociedade Brasileira de Computação, p. 167–205, 2003.

GIUNCHIGLIA, F.; SHVAIKO, P.; YATSKEVICH, M. Semantic matching. *In 1st European Semantic Web Symposium (ESWS)*, p. 61 – 75, 2004.

GLOBUS, Alliance. *About the Globus Alliance*. 2006. Disponível em: http://www.globus.org/ , Último acesso: 17/02/2006.

GOBLE, Carole; ROURE, D. De. The semantic web and grid computing. *Real World Semantic Web Applications - Frontiers in Artificial Intelligence and Applications*, IOS Press, v. 92, 2002.

GROUP, PBS. *The portable batch system*. 2006. Disponível em: http://pbs.mrj.com, Último acesso: 18/04/2006.

GRUBER, Thomas R. Towards principles for the design of ontologies used for knowledge sharing. *International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer Academic Publishers, August 1993. Último acesso: 10/02/2006.

GRUBER, Thomas R. A translation approach to portable ontology specifications. *Knowledge Acquisition*, Academic Press Ltd., v. 5, n. 2, p. 199–220, June 1993.

GRUNINGER, Michael. Designing and evaluating generic ontologies. *In Proceedings of the 12th European Conference of Artificial Intelligence*, 1996.

GRUNINGER, Michael; KOPENA, J. Semantic integration through invariants. *AI Magazine*, v. 26, n. 1, p. 11–20, 2005.

GUARINO, Nicola. Semantic matching: Formal ontological distinctions for information organization, extraction and integration. *In Lecture Notes in Computer Science*, Springer-Verlag, v. 1299, p. 139–170, June 1997. Último acesso: 10/02/2006.

GUARINO, Nicola. Formal ontology and information systems. *1st International Conference on Formal Ontologies in Information Systems, FOIS*, p. 3–15, June 1998. Último acesso: 10/02/2006.

GUARINO, Nicola; GIARETTA, Pierdaniele. Ontologies and knowledge bases: Towards a terminological clarification. In: MARS, N (Ed.). *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing (KBKS)*. [S.l.]: IOS Press, 1995. p. 25–32.

HAARSLEV, V.; MOLLER, H. Racer system description. *International Joint Conference on Automated Reasoning*, p. 701 – 705, 2001.

HAYES, Patrick. *RDF Semantics, W3C Recommendation*. 2004. Disponível em: http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ e Último acesso: 14/05/2006.

HEFLIN, Jeff; HENDLER, James. Searching the web with shoe. In: *18th American Association for Artificial Intelligence (AAAI) in Workshop on AI for Web Search*. [S.l.]: AAAI Press, 2000. p. 35–40.

HEINE, Felix; HOVESTADT, Matthias; KAO, Odej. Towards ontology-driven p2p grid resource discovery. *5th IEEE/ACM International Workshop on Grid Computing (GRID'04)*, p. 76–83, 2004. Último acesso: 10/02/2006.

HORROCKS, Ian. Fact and ifact. *Proceedings of the International Workshopon Description Logics*, p. 133 – 135, 1999.

HORROCKS, Ian; PATEL-SCHNEIDER, Peter F.; HARMELEN, Frank Van. Reviewing the design of daml+oil: An ontology language for the semantic web. In: *18th American Association for Artificial Intelligence (AAAI)*. [S.l.: s.n.], 2002. p. 792–797.

HOVY, E. Combining and standardizing large-scale, practical ontologies for machine translation and other uses. *In The First International Conference on Language Resources and Evaluation (LREC)*, p. 535–542, 1998.

KALFOGLOU, Yannis; DOMINGUE, John; MOTTA, Enrico; VARGAS-VERA, Maria; SHUM, Simon Buckingham. myplanet: An ontology-driven web based personalized news service. *Proceedings of the IJCAI - Workshop on Ontologies and information Sharing*, MIT Press, p. 44–52, 2001.

KALFOGLOU, Y.; SCHORLEMME, M. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, v. 18, n. 1, p. 1–31, 2003.

KARP, R.; CHAUDHRI, V.; THOMERE, J. *XOL: An XML-Based Ontology Exchange Language*. [S.l.], 1999. Technical Report.

KIFER, Michael; LAUSEN, Georg; WU, James. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, v. 42, p. 741–843, July 1995.

KNUBLAUCH, H.; MUSEN, M.; RECTOR, A. Editing description logic ontologies with the protégé owl plugin. *17th International Workshop on Description Logics*, Whistler, British Columbia, Canada, 2004.

KOIVUNEN, Marja-Riitta; MILLER, Eric. *W3C Semantic Web Activity*. [S.l.]: HIIT Publications, November 2001. 27-44 p. Disponível em: http://www.w3.org/2001/12/semweb-fin/w3csw.

KOKKELINK, Stefan. *RDF Graphs*. 2001. Disponível em: http://zoe.mathematik.uni-osnabrueck.de/QAT/Transform/RDFTransform/node1.html e Último acesso: 10/02/2006.

LEE, Tim Berners. *World Wide Web Consortium (W3C)*. 2006. Disponível em: http://www.w3.org/Consortium/ e Último acesso: 10/02/2006.

LEE, Tim Berners; FIELDING, R.; MASINTER, L. *Uniform Resource Identifier (URI): Generic Syntax*. [S.l.], January 2005.

LEE, Tim Berners; HENDLER, James; LASSILA, Ora. The semantic web. *Scientific American*, v. 5, n. 284, p. 34–43, May 2001. Último acesso: 10/02/2006.

LEGER, Alain; MICHEL, Géraldine; BARRETT, Peter; GITTON, Sylvain; GOMéZ-PéREZ, Asuncion; LEHTOLA, Aarno; MOKKILA, Kristiina; RODRIGEZ, Santiago; SALLANTIN, Jean; VARVARIGOU, Theodora; VINESSE, Jérôme. Ontology domain modeling support for multi-lingual services in e-commerce: Mkbeem. *14th European Conference on Artificial Intelligence (ECAI)*, 2000. Último acesso: 10/02/2006.

LENAT, Douglas B.; GUHA, R. V. *Building Large Knowledge-based Systems*. [S.l.]: Addison-Wesley Pub (Sd), 1990.

LI, Lei; HORROCKS, Ian. A software framework for matchmaking based on semantic web technology. In: *Proceedings of the 12th International Conference on World Wide Web*. New York, NY, USA: ACM Press, 2003. p. 331–339.

LLOYD, J. W. *Foundations of logic programming*. 2. ed. New York, NY, USA: Springer-Verlag New York, Inc., 1987. 212 p.

LOPES, José Geraldo Rodrigues Campos. *Matching Semântico de Recursos Computacionais em Grades Computacionais com Múltiplas Ontologias*. Dissertação (Mestrado) — Universidade de Brasília (UnB), Dezembro 2005.

LOPES, J. G. R. C.; MELO, Alba Cristina Magalhaes Alves; DANTAS, M. A. R.; RALHA, Celia Ghedini. A proposal and evaluation of a mechanism for grid ontology merge. *High Performance Computing System and Applications*, IEEE Computer Society, v. 0, p. 2, 2006.

MACGREGOR, Robert; BURSTEIN, Mark H. Using a description classifier to enhance knowledge representation. *7th IEEE Conference on AI Applications*, v. 6, n. 3, p. 41–46, February 1991.

MASSIE, Matthew L.; CHUN, Brent N.; CULLER, David E. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, v. 30, n. 5-6, p. 817–840, 2004.

MAULDIN, Michael L. *Conceptual Information Retrieval: A case study in adaptative partial parsing.* [S.l.]: Kluwer Academic Publishers, 1991.

MCCARTHY, John L. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, v. 3, n. 4, p. 184–195, 1960.

MCGUINNESS, D. L.; FIKES, R.; RICE, J.; WILDER, S. *An environment for merging and testing large ontologies*. San Francisco, CA: Morgan Kauffman Publishers, 2000. 483-493 p. Principles of Knowledge Representation and Reasoning: Proceedings of the 17th International Conference (KR2000).

MCGUINNESS, Deborah L.; HARMELEN, Frank Van. *OWL Web Ontology Language Overview*. 2004. Disponível em: http://www.w3.org/TR/2004/REC-owl-features-20040210 e Último acesso: 03/02/2006.

MELNIK, S.; GARCIA-MOLINA, H.; RAHM, E. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. *In 18th International Conference on Data Engineering (ICDE)*, IEEE Computing Society, 2002.

MESEGUER, J. General logics. *In Logic Colloquium*, p. 275 – 239, 1989.

MILLER, George A. Wordnet: A lexical database for english. *Communications of the ACM*, v. 38, n. 11, p. 39–41, 1995.

MILLER, Libby; SEABORNE, Andy; REGGIORI, Alberto. Three implementations of squishql, a simple rdf query language. In: *Proceedings of the First International Semantic Web Conference on The Semantic Web*. London, UK: Springer-Verlag, 2002. v. 2342, p. 423–435.

MITRA, P.; WIEDERHOLD, G. Resolving terminological heterogeneity in ontologies. *In Proceedings of the ECAI'02 workshop on Ontologies and Semantic Interoperability*, 2002.

MITRA, P.; WIEDERHOLD, G.; KERSTEN, M. A graph-oriented model for articulation of ontology interdependencies. *In Proceedings of the 7th International Conference on Extending Database Technology (EDBT)*, Stringer-Verlag, London, UK, p. 86–100, 2000.

MOTTA, E. An overview of the ocml modelling language. *8th Workshop on Knowledge Engineering: Methods & Languages (KEML)*, January 1998.

MUI, L.; HALBERSTADT, A.; MOHTASHEMI, M. Notions of reputation in multi-agents systems: A review. *1st International Joint Conference on Autonomous Agents and Multi-Agents Systems (AAMAS)*, ACM, p. 280–287, July 15-19 2002.

MYLOPOULOS, John; BORGIDA, Alexander; JARKE, Matthias; KOUBARAKIS, Manolis. Telos: Representing knowledge about information systems. *In ACM Transaction on Information Systems*, v. 8, n. 4, p. 325–362, 1990.

NILES, I. Towards a standard upper ontology. *In 2nd International Conference on Formal Ontology in Information Systems (FOIS)*, ACM Press, New York, NY, USA, p. 2–9, 2001.

NOY, Natalya F. Semantic integration: a survey of ontology-based approaches. *ACM SIGMOD Record, Special Issue on Semantic Integration*, v. 33, n. 4, p. 65 – 70, 2004.

NOY, Natalya F.; MCGUINNESS, Deborah L. Ontology development 101: A guide to creating your first ontology. *Technical Report SMI-2001-0880, Stanford Medical Informatics*, p. 1–25, 2001.

NOY, Natalya F.; MUSEN, M. A. Ontology issues and applications. *International Journal of Human-Computer Studies*, v. 59, n. 6, p. 983–1024, 2003.

OPEN, OASIS. *Organization for the Advancement of Structured Information Standards (OASIS)*. 2006. Disponível em: http://www.oasis-open.org/, Último acesso: 17/02/2006.

PERES, Asuncion Gomez; LOPEZO, Mariano Fernadez; CORCHO, O. *Ontological Engineering*. [S.l.]: Springer Verlag, 2004.

PERNAS, Ana Marilza; DANTAS, Mario A. R. Grid computing environment using ontology based service. *5th International Conference on Computational Science (ICCS), Lecture Notes in Computer Science*, v. 3516, p. 858–861, May 2005.

PROJECT, Globus; TOOLKIT, Globus. *GT 4.0 Common Runtime Components: Key Concepts - Service Oriented Architecture*. 2006. Disponível em: http://www.globus.org/toolkit/docs/4.0/common/key/; Último acesso: 10/02/2006.

PRUD'HOMMEAUX, Eric; SEABORNE, Andy. *SPARQL query language for RDF*. [S.l.], July 2005.

RAMAN, Rajesh; LIVNY, Miron; SOLOMON, Marvin H. Matchmaking distributed resource management for high throughput computing. *In Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, July 1998.

RAMOS, A. C. M. A. Melo Tania G. An extensible resource discovery mechanism for grid computing environments. *6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, IEEE Computer Society, Los Alamitos, CA, USA, v. 1, p. 115–122, 2006.

ROURE, David De; BAKER, M.; JENNINGS, N. R.; SHADBOLT, N. The evolution of the grid. *Grid Computing - Making the Global Infrastructure a Reality*, John Wiley and Sons Ltd., p. 65–100, 2003.

ROURE, David De; JENNINGS, N. R.; SHADBOLT, N. The semantic grid: A future e-science infrastructure. In: BERMAN, F.; FOX, G.; HEY, A. J. G. (Ed.). *Grid Computing - Making the Global Infrastructure a Reality*. [S.l.]: John Wiley and Sons Ltd., 2003. p. 437–470.

RSA Factoring-By-Web Project. *FAFNER*. Maio 2006. Último acesso: 13/05/2006.

SABATER, J. *Trust and reputation for agent societies*. Tese (Doutorado) — Institut d' Investigacion en Intelligence Artificial, 2003.

SEABORNE, Andy. *RDQL - A Query Language for RDF, W3C member submission*. 2004. Disponível em: http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/ e Último acesso: 10/08/2006.

SHETH, A.; LARSON, J. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *In Proceedings of the ECAI'02 workshop on Ontologies and Semantic Interoperability*, v. 22, n. 3, p. 183 – 230, September 1990.

SIRIN, Evren; PARSIA, Bijan. Pellet: An owl dl reasoner. In: *In Proceedings of the 2004 International Workshop on Description Logics*. [S.l.]: CEUR-WS.org, 2004. v. 104, p. 1–16. Http://dblp.uni-trier.de.

SKILLICORN, D. B. The case for datacentric grids. *Technical Reports - Queen's University, Departament of Computing and Information Science*, November 2001.

SKILLICORN, D. B. Motivating computational grids. *Proceedings of the 2nd IEEE/ACM Internacional Symposium on Cluster Computing and the Grid*, March 2002.

SOTOMAYOR, Borja. *The Globus Toolkit version 3 Programmer's Tutorial*. 2003. Disponível em: http://www.cse.buffalo.edu/t̃ina/cse486/fall2005/progtutorial_0.4.3.pdf , Último acesso: 17/02/2006.

STUDER, Rudi; BENJAMINS, V. Richard; FENSEL, Dieter. Knowledge Engineering: Principles and Methods. *IEEE Transactions on Data and Knowledge Engineering*, v. 25, n. 1-2, p. 161–197, 1998.

SU, Xiaomeng; ILEBREKKE, Lars. A comparative study of ontology languages and tools. In: *Advanced Information Systems Engineering, 14th International Conference*. [S.l.]: Springer-Verlag GmbH, 2002. p. 761–765. Http://link.springer.de/link/service/series/0558/bibs/2348/23480761.htm.

TANGMUNARUNKIT, Hongsuda; DECKER, Stefan; KESSELMAN, Carl. Ontology-based resource matching in the grid - the grid meets the semantic web. In: *International Semantic Web Conference*. [S.l.]: Springer, 2003. p. 706–721. Disponível em citeseer.ist.psu.edu/577737.html, Último acesso: 10/02/2006.

TEAM, The Jena Development. *Inference Engine User Manual*. 2006. Disponível em: http://jena.sourceforge.net/inference/index.html, Último acesso: 18/07/2006.

THAIN, Douglas; TANNENBAUM, Todd; LIVNY, Miron. Distributed computing in practice: the condor experience. *In Concurrency - Practice and Experience*, v. 17, n. 2-4, p. 323–356, 2005.

TOOLKIT, The Globus. *GT 4.0: Information Services*. 2006. Disponível em: http://www.globus.org/toolkit/docs/4.0/info/, Último acesso: 15/08/2006.

USCHOLD, Mike; GRÜNINGER, Michael. Ontologies: Principles, methods, and applications. *Knowledge Engineering Review*, v. 11, n. 2, p. 93–155, 1996.

USCHOLD, Mike; JASPER, R. A Framework for Understanding and Classifying Ontology Applications. *In proceedings of the IJCAI - Workshop on Ontologies and Problem-Solving Methods*, August 1999.

VISSER, U.; STUCKENSCHMIDT, H.; WACHE, H.; VOGELE, T. Enabling technologies for interoperability. *14th Workshop on International Symposium of Computer Science for Environmental Protection*, p. 35–46, 2000.

W3C - World Wide Web Consortium. *Semantic Web Activity Statement*. Fevereiro 2006. Disponível em: http://www.w3.org/2001/sw/Activity e Último acesso: 10/02/2006.

WIKIPéDIA. *Ontologia - Wikipédia, a enciclopédia livre*. 2006. Disponível em: http://pt.wikipedia.org/w/index.php?title=Ontologia&oldid=1354784 e Último acesso: 03/02/2006.

Wikipédia. *Wikipédia,a enciclopédia livre*. Julho 2006. Último acesso: 01/07/2006.

WIKIPEDIA. *Service Oriented Architecture (SOA - Wikipédia, a enciclopédia livre*. 2006. Disponível em: http://en.wikipedia.org/wiki/Service-oriented_architecture, Último acesso: 13/02/2006.

WILKINSON, Kevin; SAYERS, Craig; KUNO, Harumi A.; REYNOLDS, Dave. Efficient rdf storage and retrieval in jena2. In: *In Proceedings of VLDB Workshop on Semantic Web and Databases*. [S.l.: s.n.], 2003. p. 131–150. Disponível em http://www.hpl.hp.com/techreports/2003/HPL-2003-266.pdf, Último acesso: 31/07/2006.

XING, Wei; DIKAIAKOS, Marios D.; SAKELLARIOU, Rizos. A core grid ontology for the semantic grid. *6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 178–184, 2006.

APÊNDICE A – Publicações

A.1 Trabalhos Publicados

Título: Seleção de Recursos em Ambientes Grid baseada em Ontologia

Evento: Escola Regional de Alto Desempenho (ERAD)

Local: Ijuí - RS

Data: 10/01/06 - 14/01/06

Autores: Alexandre Parra Carneiro da Silva e Mário Antônio Ribeiro Dantas

Título: An Efficient Approach for Resource Set-Matching in Grid Computing Configurations

Evento: 20th International Symposium on High Performance Computing Systems and Applications (HPCS)

Local: St. John's - Canadá

Data: 14/05/06 - 17/05/06

Autores: Alexandre Parra Carneiro da Silva e Mário Antônio Ribeiro Dantas

Título: Seleção de Múltiplos Recursos em Configurações de Grade Computacional baseada em Ontologia

Evento: Seminário Integrado de Software e Hardware (SEMISH)

Local: Campo Grande - MS

Data: 17/07/06 - 20/07/06

Autores: Alexandre Parra Carneiro da Silva e Mário Antônio Ribeiro Dantas

APÊNDICE B – Ontologias

Neste apêndice são apresentadas de forma mais detalhada as descrições das ontologias utilizadas no sistema de *matching* semântico proposto.

Estas descrições compreendem as propriedades expressando as características dos conceitos (definidos como classes) e relacionamentos entre os conceitos descritos nas ontologias. As ontologias são: Ontologia de Referência (Figura 69), Ontologia da OV_1 (Figura 70), Ontologia da OV_2 (Figura 71) e Ontologia da OV_3 (Figura 72).

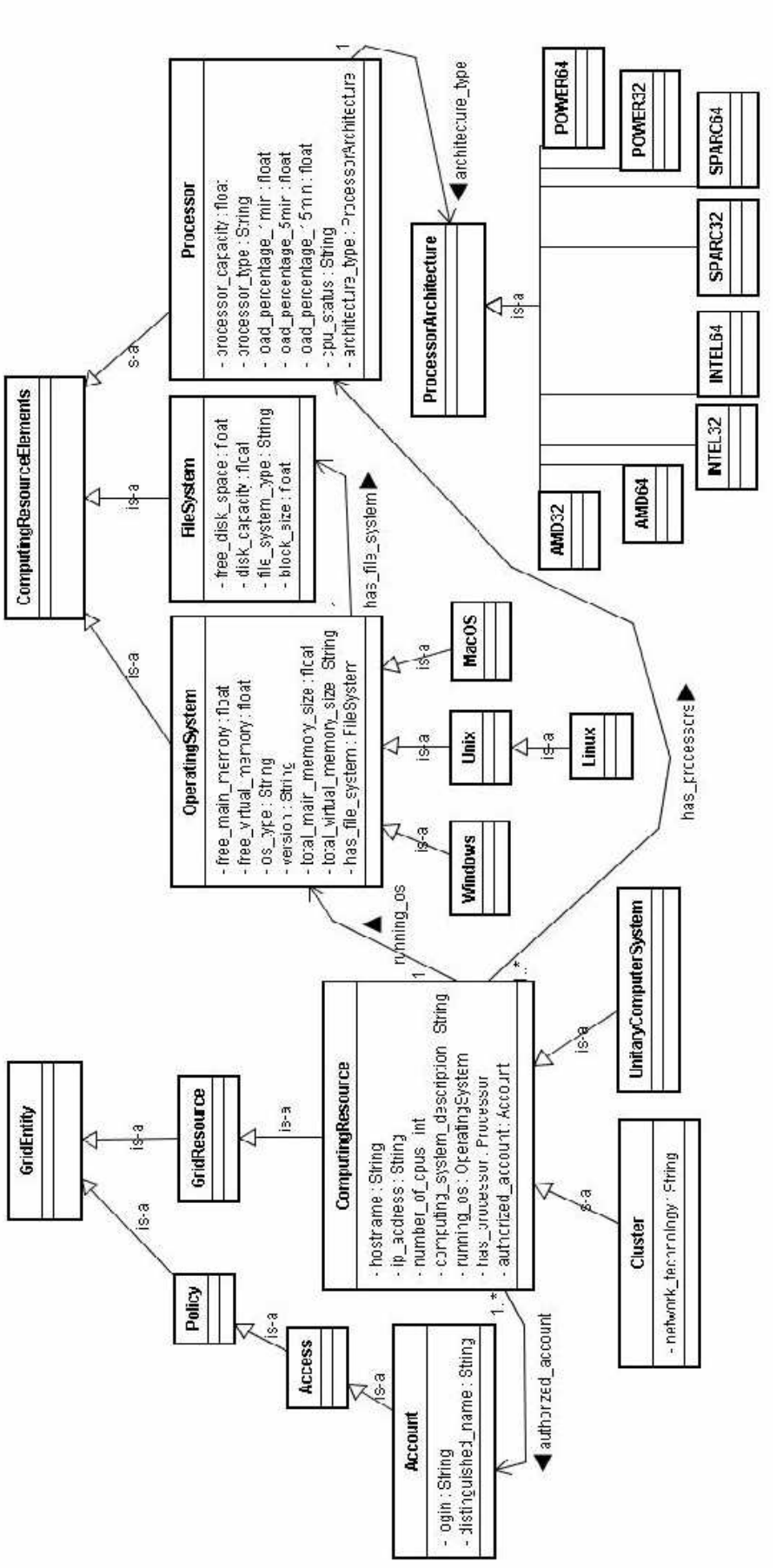


Figura 69: Classes, propriedades e relações entre classes da Ontologia de Referência

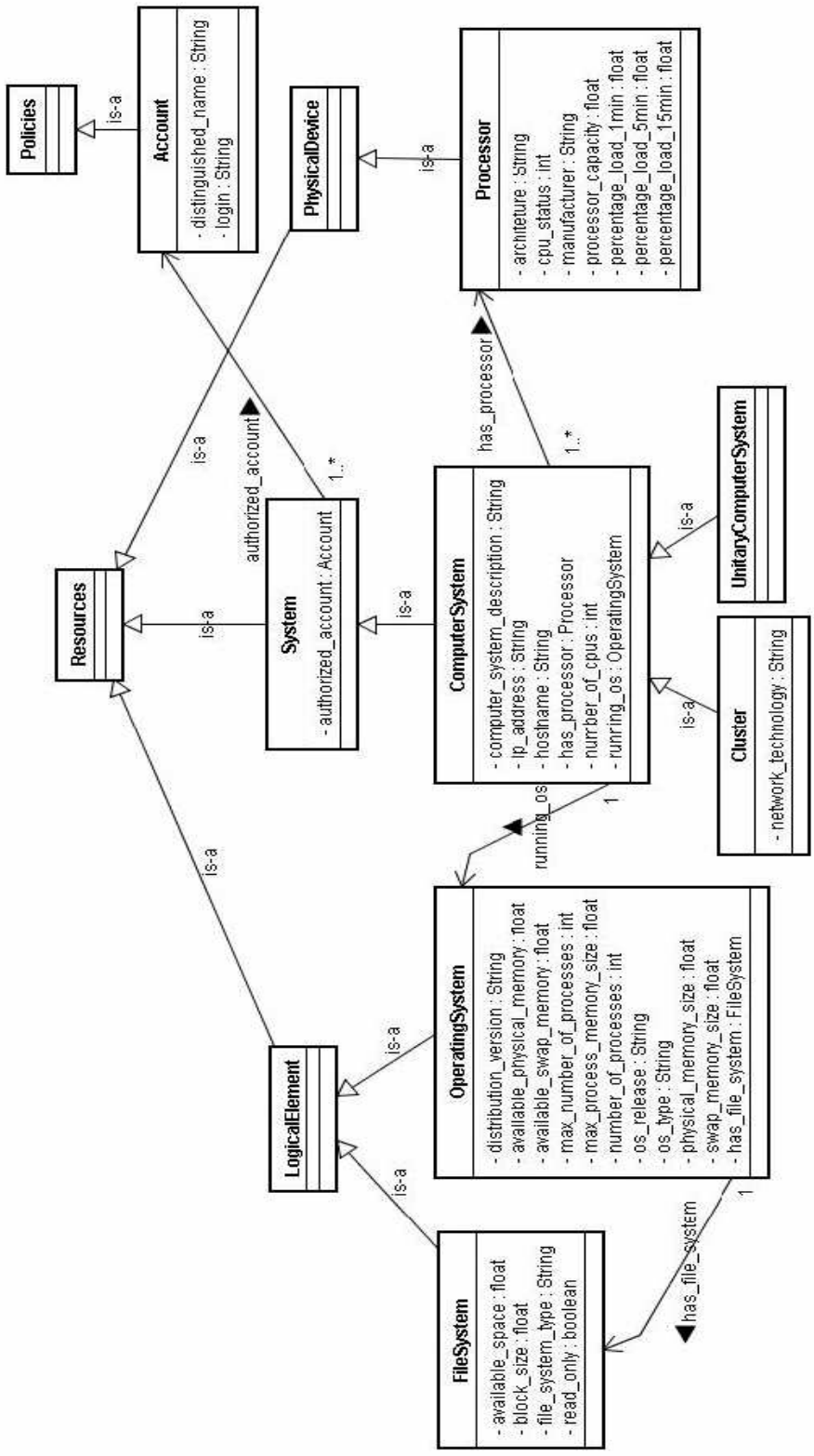


Figura 70: Classes, propriedades e relações entre classes da ontologia da OV_1

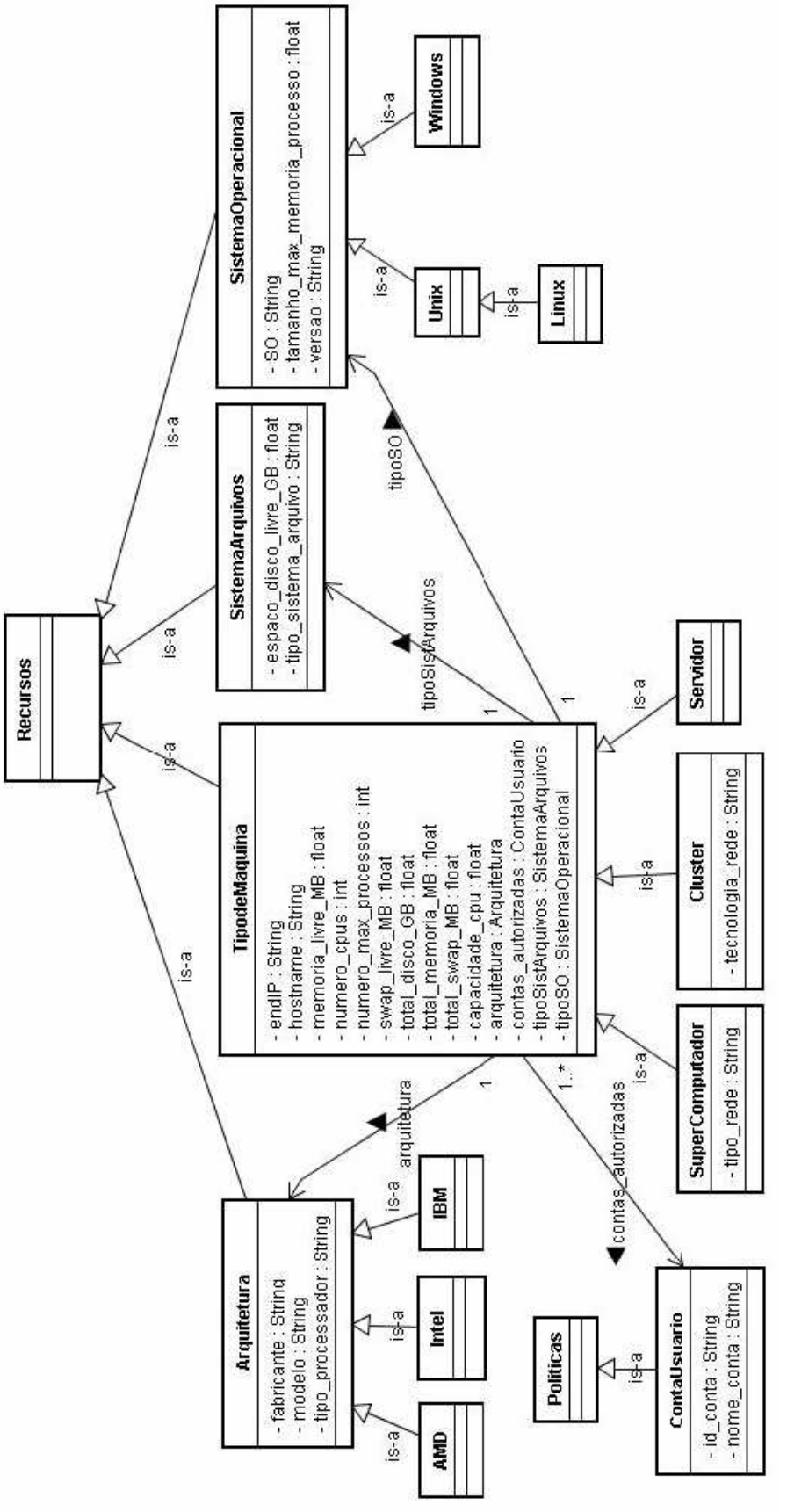


Figura 71: Classes, propriedades e relações entre classes da ontologia da OV_2

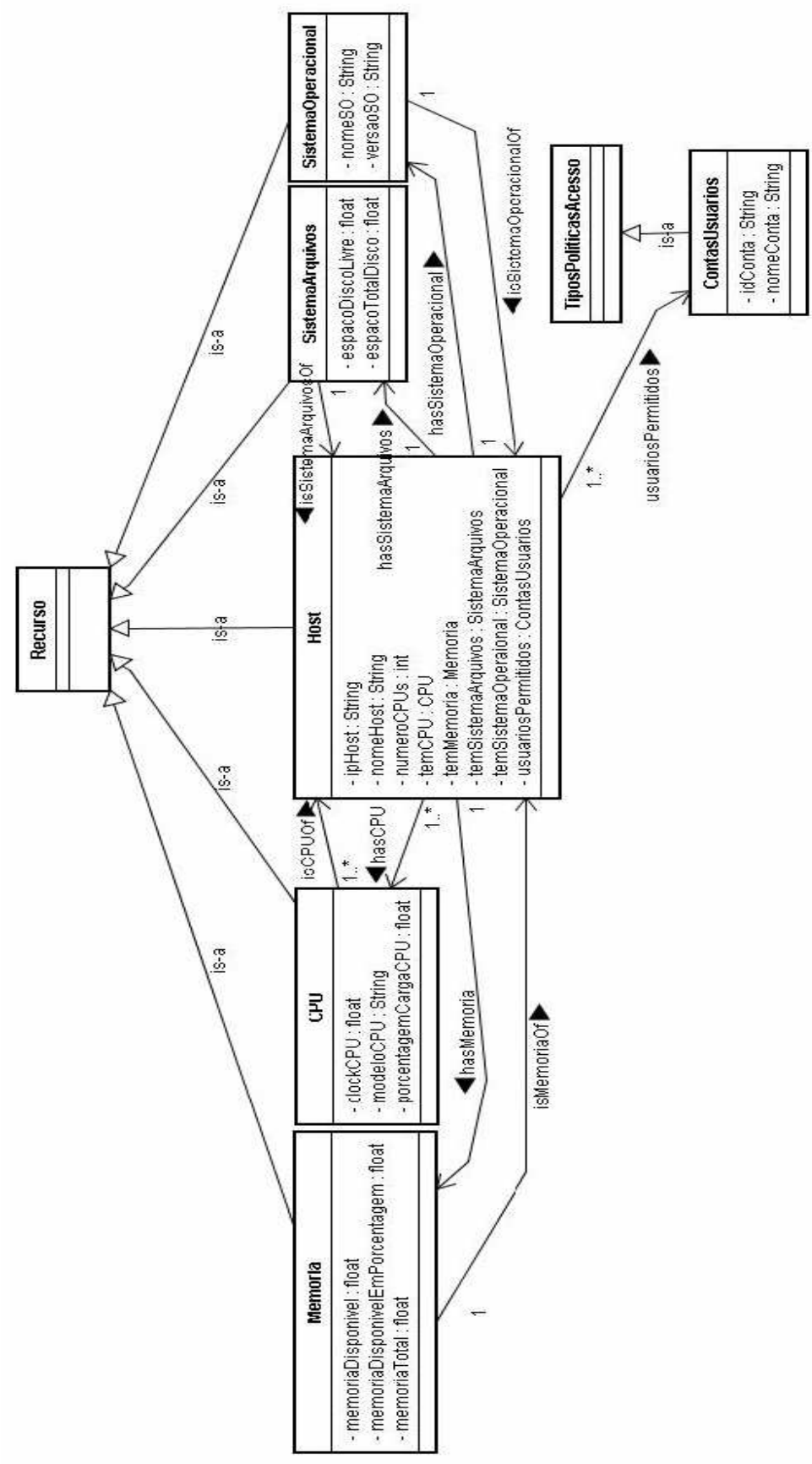


Figura 72: Classes, propriedades e relações entre classes da ontologia da OV_3

APÊNDICE C – Regras de ampliação de consulta e verificação de consistência

Neste apêndice são apresentadas as regras de ampliação de consultas e de verificação de consistência das consultas criadas para este trabalho. Pressupomos que as regras utilizadas no sistema foram definidas por especialistas que conhecem as ontologias (OR e de pedidos) e as tecnologias envolvidas, ou seja, representação formal da linguagem OWL e a linguagem de regra fornecida pelo *framework* Jena. Ambas as regras utilizam prefixos como mecanismo de abreviação para *namespaces* IRIs (Internationalized Resource Identifiers) (DüRST; SUIGNARD, 2005). IRIs são a generalização de URIs (LEE; FIELDING; MASINTER, 2005) e são totalmente compatíveis com URIs e URLs. As IRIs empregadas nas regras são ilustradas na Tabela 13.

Tabela 13: Abreviações de IRIs utilizadas nas regras

Prefixos	**IRIs**
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
owl	http://www.w3.org/2002/07/owl#
greq	http://www.owl-ontologies.com/Requests.owl#
ont_ref	http://www.owl-ontologies.com/ReferenceOntology.owl#
gres	http://www.owl-ontologies.com/Results.owl#

As regras de ampliação de consulta, são apresentadas na Figura 73 e as de verificação de consistência mostradas nas Figuras 74 e 75.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <queries_extension_rules   >
  - <par>
      <term >os_type </term >
      <rule >[GetOSsLikeLinux: (?A greq:os_type ?SO)(?B rdf:type owl:Class) equalURI(?B 'Linux')
        equalURI(?B ?SO) (?D rdf:type ?B) (?D ont_ref:os_type ?SO1) -> (?D gres:result_value ?SO1)
        (?D gres:result_value ?B) (?E gres:result_value 'Linux')]
      </rule >
    </par>
  - <par>
      <term >os_type </term >
      <rule >[GetOSsLikeUNIX: (?A greq:os_type ?SO)(?B rdf:type owl:Class) equalURI(?B 'Unix')
        equalURI(?B ?SO) (?D rdf:type ?B) (?D ont_ref:os_type ?SO1) (?E rdfs:subClassOf ?B)
        equalURI(?E 'Linux') (?F rdf:type ?E) (?F ont_ref:os_type ?SO2) ->(?G gres:result_value ?SO1)
        (?H gres:result_value ?SO2) (?I gres:result_value 'Unix') (?F gres:result_value 'Linux') ]
      </rule >
    </par>
  - <par>
      <term >processor_type</term>
      <rule >[GetArchProcessorsAMD: (?A greq:processor_type ?proc1) (?B ont_ref:architecture_type
        ?C) (?D ont_ref:architecture_type ?E) (?C rdf:type ?F) (?E rdf:type ?G) equalURI(?proc1 'AMD')
        equalURI(?F 'AMD32') equalURI(?G 'AMD64') (?B ont_ref:processor_type ?proc2) (?D
        ont_ref:processor_type ?proc3) -> (?D gres:result_value ?proc2) (?D gres:result_value ?proc3) ]
      </rule >
    </par>
  - <par>
      <term >processor_type  </term >
      <rule >[GetArchProcessorsAMD32: (?A greq:processor_type ?proc1) (?B ont_ref:architecture_type
        ?C) (?C rdf:type ?D) equalURI(?D 'AMD32') equalURI(?proc1 'AMD32') (?B ont_ref:processor_type
        ?proc2) -> (?D gres:result_value ?proc2) ]
      </rule >
    </par>
  - <par>
      <term >processor _type </term >
      <rule >[GetArchProcessorsAMD64: (?A greq:processor_type ?proc1) (?B ont_ref:architecture_type
        ?C) (?C rdf:type ?D) equalURI(?D 'AMD64') equalURI(?proc1 'AMD64') (?B ont_ref:processor_type
        ?proc2) -> (?D gres:result_value ?proc2) ]
      </rule >
    </par>
  - <par>
      <term >processor _type </term >
      <rule >[GetArchProcessorsINTEL: (?A greq:processor_type ?proc1) (?B ont_ref:architecture_type
        ?C) (?D ont_ref:architecture_type ?E) (?C rdf:type ?F) (?E rdf:type ?G) equalURI(?proc1 'INTEL')
        equalURI(?F 'INTEL32') equalURI(?G 'INTEL64') (?B ont_ref:processor_type ?proc2) (?D
        ont_ref:processor_type ?proc3) -> (?D gres:result_value ?proc2) (?D gres:result_value ?proc3) ]
      </rule >
    </par>
  - <par>
      <term >processor _type </term >
      <rule >[GetArchProcessorsINTEL32: (?A greq:processor_type ?proc1) (?B ont_ref:architecture_type
      ?C) (?C rdf:type ?D) equalURI(?D 'INTEL32') equalURI(?proc1 'INTEL32') (?B ont_ref:processor_type
        ?proc2) -> (?D gres:result_value ?proc2) ]
      </rule >
    </par>
  - <par>
      <term >processor _type </term >
      <rule >[GetArchProcessorsINTEL64: (?A greq:processor_type ?proc1) (?B ont_ref:architecture_type
      ?C) (?C rdf:type ?D) equalURI(?D 'INTEL64') equalURI(?proc1 'INTEL64') (?B ont_ref:processor_type
        ?proc2) -> (?D gres:result_value ?proc2) ]
      </rule >
    </par>
...
```

Figura 73: Regras de ampliação de consulta criadas para o sistema de *matching* semântico desenvolvido

```
<?xml version="1.0" encoding="UTF-8" ?>
- <rules >
    <validation_rule  > [ checkingCardinalityTermFree_Main_Memory: (?A owl:onProperty
      greq:free_main_memory) (?A owl:cardinality ?card1) countLiteralValues(?B greq:free_main_memory
      ?qtd1) (?C greq:characteristic 'free_main_memory') countLiteralValues(?C greq:characteristic ?qtd2)
      sum(?qtd1 ?qtd2 ?total) greaterThan(?total ?card1) -> (?D error It__is__accepted__only__one__value
      __for__the__term__'free_main_memory'__for__each__query__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingCardinalityTermFree_Virtual_Memory: (?A owl:onProperty
      greq:free_virtual_memory) (?A owl:cardinality ?card1) countLiteralValues(?B
      greq:free_virtual_memory ?qtd1) (?C greq:characteristic 'free_virtual_memory') countLiteralValues(
      ?C greq:characteristic ?qtd2) sum(?qtd1 ?qtd2 ?total) greaterThan(?total ?card1) -> (?D error It__is__
      accepted__only__one__value__for__the__term__'free_virtual_memory'__for__each__query__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingCardinalityTermTotal_Main_Memory_Size: (?A owl:onProperty
      greq:total_main_memory_size) (?A owl:cardinality ?card1) countLiteralValues(?B
      greq:total_main_memory_size ?qtd1) (?C greq:characteristic 'total_main_memory_size')
      countLiteralValues(?C greq:characteristic ?qtd2) sum(?qtd1 ?qtd2 ?total) greaterThan(?total ?card1)
      -> (?D error It__is__accepted__only__one__value__for__the__term__'total_main_memory_size'__ for
      __ each__query__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingCardinalityTermTotal_Virtual_Memory_Size: (?A owl:onProperty
      greq:total_virtual_memory_size) (?A owl:cardinality ?card1) countLiteralValues(?B
      greq:total_virtual_memory_size ?qtd1) (?C greq:characteristic 'total_virtual_memory_size')
      countLiteralValues(?C greq:characteristic ?qtd2) sum(?qtd1 ?qtd2 ?total) greaterThan(?total ?card1)
      -> (?D error It__is__accepted__only__one__value__for__the__term__'total_virtual_memory_size'__
      for__each__query__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingCardinalityTermDisk_Capacity: (?A owl:onProperty greq:disk_capacity) (?A
      owl:cardinality ?card1) countLiteralValues(?B greq:disk_capacity ?qtd1) (?C greq:characteristic
      'disk_capacity') countLiteralValues(?C greq:characteristic ?qtd2) sum(?qtd1 ?qtd2 ?total)
      greaterThan(?total ?card1) -> (?D error It__is__accepted__only__one__value__ for__the__term__
      'disk_capacity'__for__each__query__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingCardinalityTermFree_Disk_Space: (?A owl:onProperty
      greq:free_disk_space) (?A owl:cardinality ?card1) countLiteralValues(?B greq:free_disk_space
      ?qtd1) (?C greq:characteristic 'free_disk_space') countLiteralValues(?C greq:characteristic
      ?qtd2) sum(?qtd1 ?qtd2 ?total) greaterThan(?total ?card1) -> (?D error It__is__accepted__only__one
      __ value__for__the__term__'free_disk_space'__for__each__query__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingCardinalityTermLoad_Percentage_1min: (?A owl:onProperty
      greq:load_percentage_1min) (?A owl:cardinality ?card1) countLiteralValues(?B
      greq:load_percentage_1min ?qtd1) (?C greq:characteristic 'load_percentage_1min')
      countLiteralValues(?C greq:characteristic ?qtd2) sum(?qtd1 ?qtd2 ?total) greaterThan(?total
      ?card1) -> (?D error It__is__accepted__only__one__value__for__the__term__ 'load_percentage_1min'
       for__ each__query__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingCardinalityTermNumber_of_CPUs: (?A owl:onProperty
      greq:number_of_cpus) (?A owl:cardinality ?card1) countLiteralValues(?B greq:number_of_cpus
      ?qtd1) (?C greq:characteristic 'number_of_cpus') countLiteralValues(?C greq:characteristic
      ?qtd2) sum(?qtd1 ?qtd2 ?total) greaterThan(?total ?card1) -> (?D error It__is__accepted__only__one
      __value__for__the__term__'number_of_cpus'__for__each__query__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingCardinalityTermDistinguished_Name:
       (?A owl:onProperty greq:distinguished_name) (?A owl:cardinality ?card) countLiteralValues(?B
       greq:distinguished_name ?qtd) notEqual(?qtd ?card) -> (?C error It__is__accepted__only__one__
       value__for__the__term__'distinguished_name'__for__each__query__!) ]
    </validation_rule  >
    ...
```

Figura 74: Regras de verificação de consulta mais relevantes criadas para o sistema de *matching* semântico desenvolvido

```
...
    <validation_rule  > [ checkingPossibleValuesTermDecrescent_Order: (?A greq:decrescent_order ?reqOrdenacao)
       notEqual(?reqOrdenacao 'processor_capacity') notEqual(?reqOrdenacao 'number_of_cpus')
       notEqual(?reqOrdenacao 'load_percentage_1min') notEqual(?reqOrdenacao 'load_percentage_5min')
       notEqual(? reqOrdenacao  'load_percentage_15min')  notEqual(?reqOrdenacao 'free_disk_space')
       notEqual(?reqOrdenacao 'free_main_memory')   notEqual(?reqOrdenacao 'free_virtual_memory')
       notEqual(?reqOrdenacao 'disk_capacity')  notEqual(?reqOrdenacao 'total_main_memory_size')
       notEqual(?reqOrdenacao 'total_virtual_memory_size') -> (?A error The__value__attributed__to__term__
       'decrescent_order'__must__be__following__of__terms__express__in__ request__ontology__
       'processor_capacity'__or__'number_of_cpus'__or__'load_percentage_1min'__or__' load_percentage_5min'
       __or__  'load_percentage_15min'__or__'free_disk_space'__or__'free_main_memory'__or__
    'free_virtual_memory'__or__'disk_capacity'__or__'total_main_memory_size'__or__'total_virtual_memory_size'.) ]
    </validation_rule  >
    <validation_rule  > [ checkingValueTermNumber_Resources_Return: (?A  greq:number_resources_return
       ?amount) le(?amount 0) -> (?A error The__value__ attributed__to__term__'number_resources_return' __must
       __be__greater__or__equal__1__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingValueTermFreeMainMemory: (?A greq:free_main_memory ?value)  le(?value 0)
       ->(?A error The__value__attributed__to__term__'free_main_memory'__must__be__greater__than__ zero__!)  ]
    </validation_rule  >
    <validation_rule  > [ checkingConsistencyEXT3withOSsFamilyWindows: (?A greq:file_system_type ?FS)
       (?B greq:os_type ?OS) (?C rdf:type owl:Class) equal(?FS 'EXT3') equalURI(?C 'Windows')  (?D rdf:type ?C)
       (?D ont_ref:os_type ?SO1) equalURI(?OS ?SO1) -> (?D error Resource__with__ perating__system__ of__
       family__'Windows'__and__file__system__'EXT3'__not__exists__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingConsistencyEXT2withOSsFamilyWindows: (?A greq:file_system_type ?FS)
       (?B greq:os_type ?OS) (?C rdf:type owl:Class) equal(?FS 'EXT2') equalURI(?C 'Windows')  (?D rdf:type ?C)
       (?D ont_ref:os_type ?SO1) equalURI(?OS ?SO1) -> (?D error Resource__with__operating__system__of__
       family__'Windows'__and__file__system__'EXT2'__not__exists__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingConsistencyArchProcessorsSPARC32withOSsFamilyWindows:
       (?A greq:processor_type ?source_proc) (?B greq:os_type ?source_os) (?Windtype rdf:type owl:Class)
       equalURI(?arch 'SPARC32') equalURI(?Windtype 'Windows') (?D rdf:type ?Windtype) (?D  ont_ref:os_type
       ?target_os) (?E rdf:type ?arch) (?F ont_ref:architecture_type ?E) (?F  ont_ref:processor_type ?target_proc)
       equalURI(?source_proc ?target_proc) equalURI(?source_os ?target_os) -> (?F error  Resource__with__
       operating__system__of__family__'Windows'__and__with__processors__of__architecture__'SPARC32'__not
        __exists__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingConsistencyArchProcessorsSPARC64withOSsFamilyWindows:
       (?A greq:processor_type ?source_proc) (?B greq:os_type ?source_os) (?Windtype rdf:type  owl:Class)
       equalURI(?arch 'SPARC64') equalURI(?Windtype 'Windows') (?D rdf:type ?Windtype) (?D  ont_ref:os_type
       ?target_os) (?E rdf:type ?arch) (?F ont_ref:architecture_type ?E) (?F  ont_ref:processor_type ?target_proc)
       equalURI(?source_proc ?target_proc) equalURI(?source_os  ?target_os) -> (?F error Resource__with__
      operating__system__of__family__'Windows'__and__with__processor__of__architecture__'SPARC64'__not
       __exists__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingConsistencyArchProcessorsPOWER32withOSsFamilyWindows:
       (?A greq:processor_type ?source_proc) (?B greq:os_type ?source_os) (?Windtype rdf:type  owl:Class)
       equalURI(?arch 'POWER32') equalURI(?Windtype 'Windows') (?D rdf:type ?Windtype) (?D  ont_ref:os_type
       ?target_os) (?E rdf:type ?arch) (?F ont_ref:architecture_type ?E) (?F  ont_ref:processor_type ?target_proc)
       equalURI(?source_proc ?target_proc) equalURI(?source_os  ?target_os) -> (?F error Resource__with__
       operating__system__of__family__'Windows'__and__with__processors__of__architecture__'POWER32'__not
       __exists__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingConsistencyArchProcessorsPOWER64withOSsFamilyWindows:
       (?A greq:processor_type ?source_proc) (?B greq:os_type ?source_os) (?Windtype rdf:type  owl:Class)
       equalURI(?arch 'POWER64') equalURI(?Windtype 'Windows') (?D rdf:type ?Windtype) (?D  ont_ref:os_type
       ?target_os) (?E rdf:type ?arch) (?F ont_ref:architecture_type ?E) (?F  ont_ref:processor_type ?target_proc)
       equalURI(?source_proc ?target_proc) equalURI(?source_os  ?target_os) -> (?F error  Resource__with__
       operating__system__of__family__'Windows'__and__with__processors__of__architecture__'POWER64'__not
       __exists__!) ]
    </validation_rule  >
    <validation_rule  > [ checkingTypeTermNumber_of_CPUS: (?A greq:on ?B)  (?A rdf:type ?C)
       countLiteralValues(?B greq:number_of_cpus ?qtd) greaterThan(?qtd 0) equalURI(?C  'Equality_String') -> (?C
       error The__value__attributed__to__term__'number_of_cpus'__ must__be__related __with__the__binary __
       operators__that__compare__values__numerics__instead__o f __strings__!) ]
    </validation_rule  >
...
```

Figura 75: Continuação das regras de verificação de consulta