

UNIVERSIDADE CATÓLICA DE PELOTAS
CENTRO POLITÉCNICO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

**Uma Proposta para Descoberta de
Recursos na Computação Ubíqua com
Suporte Semântico**

por
Renato Marques Dilli

Dissertação apresentada como
requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Orientador: Prof. Dr. Adenauer Corrêa Yamin

DM-2010/2-003

Pelotas, setembro de 2010

AGRADECIMENTOS

Ao Instituto Federal Sul-rio-grandense pelo incentivo e apoio.

Em especial, ao Prof. Dr. Adenauer Corrêa Yamin, meu orientador, pela sua dedicação e disposição para acompanhar e auxiliar-me em todos os momentos deste trabalho.

Agradeço aos colegas do PPGINF, em especial à Nelsi e ao Luthiano pelos vários momentos de estudo que me auxiliaram no desenvolvimento deste trabalho.

Aos funcionários e professores do Centro Politécnico da Universidade Católica de Pelotas pela seriedade e compromisso.

SUMÁRIO

LISTA DE FIGURAS	6
LISTA DE TABELAS	8
LISTA DE LISTAGENS	9
LISTA DE ABREVIATURAS E SIGLAS	10
RESUMO	12
ABSTRACT	13
1 INTRODUÇÃO	14
1.1 Tema	15
1.2 Motivação	15
1.3 Objetivos	16
1.4 Estrutura do Texto	16
2 ESCOPO DE PESQUISA	17
2.1 Computação Ubíqua: Principais Conceitos	17
2.2 Middleware EXEHDA	19
2.3 Descoberta de Recursos: Fundamentos e Requisitos	23
2.3.1 Objetivos Gerais	23
2.3.2 Principais Requisitos e Funcionalidades	24
2.3.3 Gerenciamento da Informação	29
2.3.4 Suporte à Mobilidade	30
2.3.5 Descoberta com Semântica	31
2.4 Revisando Tecnologias de Processamento Semântico	31
2.4.1 Lógica de Descrição	33
2.4.2 Ontologias	33
2.5 Construção e Processamento de Ontologias	34
2.5.1 XML	35
2.5.2 RDF/RDF-Schema	35
2.5.3 OWL	36
2.5.4 Protégé	37
2.5.5 SPARQL	37
2.5.6 API Jena	38
2.5.7 Raciocinadores	40

2.6	Considerações Sobre o Capítulo	41
3	TRABALHOS EM DESCOBERTA DE RECURSOS	42
3.1	Baseados em Sintaxe	42
3.1.1	INS/TWINE	42
3.1.2	UPnP	42
3.1.3	Allia	43
3.1.4	Jini	43
3.1.5	SLP	45
3.1.6	Globus Toolkit	45
3.1.7	PerDis	46
3.1.8	Condor	47
3.1.9	Salutation	48
3.2	Baseados em Semântica	49
3.2.1	OMM (Ontology-Based MatchMaker)	49
3.2.2	A Grid Service Discovery Matchmaker Based on Ontology Description	50
3.2.3	Ontologias Aplicadas à Descrição de Recursos em Ambientes Grid	51
3.2.4	Serviço Baseado em Semântica para Descoberta de Recursos em Grade Computacional	52
3.2.5	DReggie	53
3.3	Considerações Sobre o Capítulo	54
4	EXEHDA-SD: CONCEPÇÃO E MODELAGEM	56
4.1	Características Envolvidas no Projeto do EXEHDA-SD	56
4.2	Modelagem da Arquitetura de Software	60
4.2.1	CC - Componente Cliente	60
4.2.2	CR - Componente Recurso	61
4.2.3	CD - Componente Diretório	61
4.3	Processador Semântico	66
4.3.1	Definições da OntUbi	68
4.3.2	Definições da OntSD	71
4.3.3	Consulta sem Raciocinador	74
4.3.4	Consulta com Raciocinador	75
4.3.5	Verificação da Ontologia	76
4.3.6	Persistência de Dados	77
4.3.7	Definição de Regras	78
4.4	Descrição e Consulta de Recursos	79
4.4.1	Representação e Manutenção de Recursos	80
4.4.2	Especificação da Consulta por Recursos	80
4.5	Preferências de Pesquisa do Usuário	85
4.6	Ativação do EXEHDA-SD	87
4.7	Considerações Sobre o Capítulo	88
5	EXEHDA-SD: ESTUDOS DE CASO	89
5.1	Embrapa Clima Temperado	89
5.1.1	Casa de Vegetação	90
5.1.2	Tipos de Recursos Envolvidos	91
5.1.3	Ambiente de Teste	92

5.2	Cenário 1 - Pesquisa por Recursos com Regras Inferência Definidas pelo Cliente	92
5.3	Cenário 2 - Descoberta e Seleção de Nodos Através de Raciocinadores .	96
5.4	Cenário 3 - Exemplo do Fluxo de Dados da Arquitetura do EXEHDA-SD	98
5.5	Considerações Sobre o Capítulo	101
6	CONSIDERAÇÕES FINAIS	102
6.1	Discussão dos Trabalhos Relacionados ao EXEHDA-SD	103
6.2	Principais Contribuições da Pesquisa Realizada	105
6.3	Publicações Realizadas	106
6.4	Trabalhos Futuros	107
	REFERÊNCIAS	108

LISTA DE FIGURAS

Figura 2.1	Middleware EXEHDA (YAMIN, 2004)	19
Figura 2.2	Ambiente Ubíquo (YAMIN, 2004)	20
Figura 2.3	Descoberta de Recursos: Arquitetura de 2 Componentes	26
Figura 2.4	Descoberta de Recursos: Arquitetura de 3 Componentes	27
Figura 2.5	Algoritmos de Matching para Consulta por Recursos	30
Figura 2.6	Camadas da Web Semântica Proposta pela W3C	32
Figura 2.7	Construtores da Lógica de Descrição	33
Figura 2.8	Representação RDF em Grafo	36
Figura 2.9	Hierarquia de Interfaces da API Jena (VERZULLI, 2001)	39
Figura 2.10	Camadas da API Jena (DICKINSON, 2008)	39
Figura 3.1	Arquitetura INS/Twine (BALAZINSKA; BALAKRISHNAN; KARGER, 2002)	43
Figura 3.2	Jini - Descoberta e Join (JINI, 2009)	44
Figura 3.3	Jini - Lookup e Cliente (JINI, 2009)	44
Figura 3.4	Globus Toolkit (TOOLKIT, 2009)	46
Figura 3.5	Arquitetura PerDis (SCHAEFFER, 2005)	47
Figura 3.6	Arquitetura Condor (THAIN; TANNENBAUM; LIVNY, 2005)	48
Figura 3.7	Arquitetura de Pernas e Dantas (PERNAS; DANTAS, 2004)	51
Figura 4.1	Subsistema do EXEHDA Diretamente Relacionado ao EXEHDA-SD	57
Figura 4.2	Definição de Classe, Subclasse e Superclasse	58
Figura 4.3	EXEHDA-SD: Ambiente Celular de Descoberta	59
Figura 4.4	EXEHDA-SD: Arquitetura do Modelo Proposto	60
Figura 4.5	EXEHDA-SD: Indisponibilidades - Seleção de Nodo	63
Figura 4.6	EXEHDA-SD: Indisponibilidades - Intervalos de Tempo	63
Figura 4.7	EXEHDA-SD: Comunicador P2P	64
Figura 4.8	EXEHDA-SD: OntUbi	67
Figura 4.9	EXEHDA-SD: OntSD	68
Figura 4.10	EXEHDA-SD: Classes da OntSD	72
Figura 4.11	EXEHDA-SD: Ontologia Armazenada no Banco de Dados	77
Figura 4.12	EXEHDA-SD: Cadastro de Recursos - Nodos	80
Figura 4.13	EXEHDA-SD: Cadastro de Recursos - Periféricos	81
Figura 4.14	EXEHDA-SD: Consulta por Recursos - Nodos	83
Figura 4.15	EXEHDA-SD: Consulta por Recursos - Periféricos	84
Figura 4.16	EXEHDA-SD: Consulta por Recursos - Regras	85
Figura 4.17	EXEHDA-SD: Preferências do usuário	87

Figura 5.1	CENÁRIO: Organização Celular	90
Figura 5.2	CENÁRIO: Slots Instanciados	92
Figura 5.3	CENÁRIO 1: Preferências do Cliente	93
Figura 5.4	CENÁRIO 1: Interface de Consulta por Slots	94
Figura 5.5	CENÁRIO 1: Resultado da Consulta (1)	95
Figura 5.6	CENÁRIO 1: Resultado da Consulta (2)	95
Figura 5.7	CENÁRIO 1: Resultado da Consulta (3)	96
Figura 5.8	CENÁRIO 2: Nodos Instanciados	96
Figura 5.9	CENÁRIO 2: Resultado da Consulta	97
Figura 5.10	CENÁRIO 3: Anúncio de Recursos	99
Figura 5.11	CENÁRIO 3: Consulta por Recursos	100

LISTA DE TABELAS

Tabela 2.1	Comparativo de Descoberta de Recursos Baseado em Sintaxe e Semântica	31
Tabela 3.1	Comparativo dos Trabalhos Relacionados	54
Tabela 4.1	Lista de Operadores Válidos para Definição de Critérios	82
Tabela 6.1	Comparativo dos Trabalhos Relacionados com o EXEHDA-SD . . .	104

LISTA DE LISTAGENS

4.1	SPARQL: Consulta 1	75
4.2	SPARQL: Consulta 2	75
4.3	JENA: Verificação da Ontologia	76
4.4	OWL: Representação de Nodo	76
4.5	JENA: Persistência de Dados	77
4.6	JENA: Consulta com Banco de Dados	78
4.7	JENA: Definição de Regras (Ontologia.rules)	78
4.8	JENA: Código Java com Regras	78
4.9	JENA: Resultado da Consulta	79
4.10	XML: Consulta	81
4.11	SPARQL: Consulta	82
4.12	XML: Profile de Ativação CC e CR	87
5.1	CENÁRIO 1 - Regra do Cliente	93
5.2	CENÁRIO 1: XML	94
5.3	CENÁRIO 1: SPARQL	94
5.4	CENÁRIO 2: XML	96
5.5	CENÁRIO 2: SPARQL	97

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
AVU	Ambiente Virtual do Usuário
CC	Componente Cliente
CD	Componente Diretório
CR	Componente Recurso
CIB	Cell Information Base
DAML	DARPA Agent Markup Language
DL	Description Logic
DNS	Domain Name System
EXEHDA	Execution Environment for High Distributed Applications
EXEHDA-SD	EXEHDA Semantic Discovery
G3PD	Grupo de Pesquisa em Processamento Paralelo e Distribuído
GPS	Global Positioning System
GRR	Generic Rule Reasoner
HP	Hewlett-Packard
HTML	HyperText Markup Language
ISAM	Infraestrutura de Suporte à Aplicações Móveis
ISAMpe	ISAM Pervasive Environment
JVM	Java Virtual Machine
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MANETs	Mobile Ad hoc Networks
MDS	Monitoring and Discovery System
OIL	Ontology Inference Layer
OWL	Web Ontology Language

P2P	Peer-to-Peer
PDA	Personal Digital Assistant
PerDis	Pervasive Discovery Service
plenUS	plentiful of Ubiquitous Systems
RDF	Resource Description Framework
RDFS	RDF Schema
RDQL	RDF Data Query Language
RMI	Remote Method Invocation
SGML	Standard Generalized Markup Language
SLP	Service Location Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
UBICOMP	Ubiquitous Computing
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
URI	Universal Resource Identifier
URL	Universal Resource Locator
USE	Ubiquitous System Environment
VO	Virtual Organization
W3C	World Wide Web Consortium
WSDL	Web Service Definition Language
WAN	Wide Area Network
XML	eXtensible Markup Language

RESUMO

Em ambientes ubíquos os recursos devem estar compartilhados para que possam ser acessados de qualquer lugar e a qualquer momento. Nesta abordagem o processo de descoberta de recursos assume um importante papel em satisfazer adequadamente as requisições por recursos. Este trabalho apresenta a proposta do EXEHDA-SD, EXEHDA-*Semantic Discovery*, que consiste em um mecanismo de descoberta de recursos, para computação ubíqua, que agrega tecnologias em sua arquitetura para o processamento semântico de requisições por recursos, aumentando a expressividade na representação e consulta. O mecanismo prevê a dinamicidade em que os recursos entram e saem do ambiente e persegue aspectos como escalabilidade e preferências do usuário. Neste sentido, o trabalho desenvolvido contempla um estudo sobre mecanismos de descoberta de recursos no contexto da computação ubíqua e das tecnologias para processamento semântico com a intenção de fundamentar as escolhas arquiteturais do mecanismo. O EXEHDA-SD foi modelado para ser prototipado na forma de um serviço para o *middleware* EXEHDA, possibilitando a realização de pesquisas por recursos localizados nas células do ambiente ubíquo, em uma perspectiva escalável, através da utilização de ontologias e ferramentas para sua manipulação. O processamento de ontologias expande a capacidade de representação dos recursos do ambiente, bem como sua localização. A utilização de semântica na descrição dos recursos, dentre outros aspectos, facilita a localização de recursos similares ao solicitado. O modelo foi comparado com o estado da arte, e seu diferencial foi discutido. Outrossim, suas funcionalidades foram avaliadas por três estudos de caso.

Palavras-chave: Computação Ubíqua, Descoberta de Recursos, Processamento Semântico.

TITLE: “A PROPOSAL FOR RESOURCE DISCOVERY IN UBIQUITOUS COMPUTING WITH SUPPORT SEMANTIC”

ABSTRACT

In ubiquitous environments resources should be shared so that they can be accessed from anywhere and anytime. In this approach the process of resource discovery plays an important role in adequately satisfy the requests for resources. This paper presents the proposed EXEHDA-SD, EXEHDA-Semantic Discovery, which consists of a mechanism for resource discovery for ubiquitous computing, which combines technologies into its architecture for the semantic processing of requests for resources, increasing expressiveness in representation and consultation. The mechanism provides for the dynamics in which resources enter and leave the environment and pursue issues such as scalability and user preferences. In this sense, the work includes a study on mechanisms of resource discovery in the context of ubiquitous computing and technology for semantic processing with the intention of substantiating the architectural choices of the mechanism. The EXEHDA-SD was modeled to be prototyped as a service to the EXEHDA middleware, enabling the execution of search for resources located in the cells of the ubiquitous environment, in a scalable perspective, through the use of ontologies and tools for handling. Processing ontology expands the capacity for representation of environmental resources and their location. The use of semantic description of resources, among other things, facilitates the location of features similar to those requested. The model was compared with the state of the art, and its differential was discussed. Moreover, their functions were evaluated by three case studies.

Keywords: Ubiquitous Computing, Discovery Resources, Semantic Processing.

1 INTRODUÇÃO

A computação ubíqua teve uma das suas primeiras citações feita por Mark Weiser, cientista chefe do Centro de Pesquisa da Xerox, em 1991. Em seu histórico artigo "O Computador para o Século Vinte e um" (WEISER, 1991) Weiser previa que no futuro o foco dos usuários ficaria voltado para a realização de seu trabalho, e não para a ferramenta utilizada, utilizando-se de computação sem perceber ou necessitar de conhecimentos técnicos.

Atualmente o acesso a redes de computadores, cuja interconexão é global, está presente em praticamente qualquer lugar, desde escolas, universidades, cafés, aeroportos e até em praças. A evolução tecnológica atual tem concebido equipamentos de informática portáteis cada vez menores, com autonomias estendidas, possibilitando ao usuário ficar conectado por várias horas ininterruptas, facilitando sua mobilidade. Pode-se dizer que atualmente estamos com recursos tecnológicos para prover a computação transparente, presente em todo lugar e a qualquer momento. A fim de prover serviços ao usuário final de forma transparente, independentemente de tempo e espaço, sistemas ubíquos exploram os sensores e redes disponíveis (COSTA; YAMIN; GEYER, 2008).

Vários esforços de pesquisa voltados para computação ubíqua estão focados para resolver problemas inerentes da área, como heterogeneidade, escalabilidade, segurança, privacidade e confiança, interação espontânea, mobilidade, consciência de contexto, gerência de contexto, interação do usuário transparente e invisibilidade .

A abstração deste ambiente computacional ubíquo pode ser feita através de uma camada de software chamada *middleware*. Este software realiza o gerenciamento das aplicações, bem como o contexto atual e as adaptações necessárias ao contexto, devido às alterações que ocorreram no ambiente ubíquo.

Uma das funções dos *middlewares* é gerenciar os recursos disponíveis e dispersos pelo ambiente. Os recursos podem ser serviços e/ou dispositivos que podem entrar e sair do ambiente a qualquer momento. Cabe ao mecanismo de descoberta de recursos auxiliar nesta gerência.

A descoberta de recursos em um ambiente ubíquo é de extrema importância para o funcionamento do ambiente, pois o *middleware* precisa saber quais recursos estão presentes no contexto e quais atendem as necessidades da aplicação para que possam ser alocados e descartados com menor interferência possível do usuário.

O maior desafio da descoberta de recursos na computação ubíqua é a integração de dispositivos computacionais com pessoas. Quando estes dispositivos estão integrados com os usuários, várias informações pessoais são expressas na forma digital. Serviços de descoberta de recursos podem obter estas informações através das redes de comunicação

diretamente ou por inferência (ZHU; MUTKA; NI, 2005).

O EXEHDA-SD está sendo concebido para localizar recursos disponíveis nas células do *middleware* EXEHDA (LOPES; PILLA; YAMIN, 2007), tendo como maior contribuição o aumento de expressividade na representação e consulta por recursos através da utilização de tecnologias que promovem o processamento semântico do mecanismo. Os recursos descobertos pelo mecanismo visam atender os requisitos dos componentes das aplicações em execução do *middleware*. Este processo de seleção de recursos baseado nos requisitos da aplicação é conhecido como casamento de recursos ou *resource matching*.

1.1 Tema

Este trabalho tem como tema a descoberta de recursos na UBICOMP. Neste sentido, para modelagem e concepção do EXEHDA-SD é realizado o estudo de mecanismos de descoberta de recursos, revisando principalmente sua arquitetura, linguagem de descrição e consulta de recursos.

A computação ubíqua utiliza-se de recursos dispersos e heterogêneos, de forma muito dinâmica. Estas características fazem da descoberta de recursos um desafio a ser pesquisado para atender as diversidades do ambiente ubíquo.

Também serão estudadas neste trabalho as tecnologias para agregar recursos semânticos em mecanismos de descoberta de recursos para promover um *matching* semântico, estendendo a capacidade de interpretar consultas sobre recursos existentes no ambiente.

1.2 Motivação

O grande número de recursos que devem ser gerenciados em um ambiente ubíquo é fortemente heterogêneo e dinâmico (SOLDATOS et al., 2006).

A abundância de recursos permite aos usuários aproveitar o poder computacional, armazenamento, ferramentas e aplicações que não estão disponíveis em seus dispositivos locais. Além disso, esses dispositivos nem sempre estão presos a uma mesa, possibilitando usuários continuarem trabalhando e interagindo remotamente com pessoas e aplicações mesmo quando eles são móveis (ROBINSON, 2006).

O compartilhamento de recursos e informações em ambientes distribuídos faz parte das particularidades da computação ubíqua. Neste sentido, a maneira que estes recursos são descobertos e disponibilizados tem sido alvo de pesquisa em vários trabalhos atuais (SCHAEFFER, 2005), (LOPES, 2005), (ALLEMAND, 2006). Estudos atuais têm apontado o uso de tecnologias para processamento semântico no aprimoramento dos serviços em um ambiente ubíquo, entre eles a consciência e adaptação de contexto e a descoberta de recursos. O serviço de descoberta tem se mostrado uma função chave para implementar aplicações conscientes de contexto para usuários móveis em ambientes de computação ubíqua (ZHU; MUTKA; NI, 2005).

Ontologias estão sendo utilizadas para prover um entendimento semântico entre duas ou mais partes, expandindo a compreensão de termos sub-entendidos, através das relações entre as palavras.

Nesta perspectiva, nota-se a importância de integrar tecnologias utilizadas na Web Semântica ao processo de descoberta de recursos da computação ubíqua.

1.3 Objetivos

O objetivo central deste trabalho é a concepção de uma proposta para descoberta de recursos com suporte semântico, no contexto da computação ubíqua, avaliando suas principais características e desafios de pesquisa.

Os objetivos específicos são:

- organizar os fundamentos teóricos sobre descoberta de recursos, caracterizando as tecnologias relacionadas;
- sistematizar as funcionalidades dos principais trabalhos relacionados a descoberta de recursos na UBICOMP;
- estudar o *middleware* EXEHDA, revisando seus fundamentos e a concepção dos diversos módulos de sua arquitetura;
- modelar, implementar e avaliar as funcionalidades do EXEHDA-SD;
- definir as diferentes tecnologias que irão integrar a proposta EXEHDA-SD.

A seguir, a estrutura do texto na qual é feita a discussão dos conceitos, metodologias e tecnologias a serem exploradas na busca dos objetivos referenciados.

1.4 Estrutura do Texto

O texto é composto por seis capítulos. O capítulo um descreve uma pequena introdução, o tema selecionado, a motivação para o tema escolhido e os objetivos propostos para o desenvolvimento deste trabalho.

No capítulo dois é apresentado o escopo da pesquisa realizada, envolvendo conceitos sobre computação ubíqua, descoberta de recursos, tecnologias para processamento semântico. Também são descritas as ferramentas para construção e processamento de ontologias. Neste capítulo é apresentado o *middleware* EXEHDA.

O capítulo três relaciona alguns mecanismos de descoberta de recursos, sendo classificados por utilizar ou não recursos de semântica na descrição e/ou consulta por seus recursos.

No capítulo quatro é apresentado o EXEHDA-SD, bem como as escolhas para sua modelagem, componentes da arquitetura de software, funcionamento do processador semântico, formato de representação de consultas, preferências do usuário e a forma de ativação do serviço.

O capítulo cinco apresenta os estudos de caso utilizados para validar o EXEHDA-SD. Os estudos de caso foram organizados em três cenários.

No capítulo seis é feito o fechamento do trabalho com algumas considerações finais. Também é realizado um comparativo dos trabalhos de descoberta de recursos diretamente relacionados ao EXEHDA-SD, são apresentadas as contribuições da pesquisa realizada, as publicações realizadas e os trabalhos futuros.

2 ESCOPO DE PESQUISA

Este capítulo apresenta um resumo dos estudos que fundamentam a modelagem do EXEHDA-SD. É apresentado o *middleware* EXEHDA (*Execution Environment for Highly Distributed Applications*) (YAMIN, 2004) (AUGUSTIN; YAMIN; SILVA, 2008), responsável pelo ambiente ubíquo do trabalho. Conceitos sobre descoberta de recursos são trabalhados, descrevendo os passos fundamentais no processo de descoberta, bem como características que os mecanismos devem considerar no ambiente ubíquo. É feito um estudo sobre as tecnologias de processamento semântico, com ênfase na manipulação de ontologias. São apresentadas algumas técnicas para o desenvolvimento, processamento e armazenamento de ontologias. Por fim, a API Jena é estudada para a integração com a linguagem de programação Java, pois o EXEHDA-SD está sendo prototipado nesta linguagem, assim como o *middleware* EXEHDA.

2.1 Computação Ubíqua: Principais Conceitos

Este capítulo sintetiza o estudo desenvolvido a respeito de computação ubíqua e relaciona alguns projetos relevantes na área.

A evolução dos Sistemas de Informação Distribuídos, com possibilidades de acesso a redes sem fio, através de diversas tecnologias, juntamente com a Computação Móvel em crescente avanço tecnológico e a disseminação de pequenos dispositivos com acesso à Internet possibilitam o acesso de Sistemas de Informação a qualquer hora em praticamente qualquer lugar. Mark Weiser (WEISER, 1991) definiu o termo Computação Ubíqua pela primeira vez, através de seu artigo "*The Computer for the 21st Century*". Weiser teorizou que no futuro os usuários utilizariam recursos de computação sem perceber ou necessitar de conhecimentos técnicos. Ele também descreveu uma proliferação de dispositivos de diferentes tamanhos indo desde os dispositivos portáteis até grandes dispositivos compartilhados. Essa proliferação de dispositivos realmente aconteceu com os dispositivos utilizados normalmente e o desenvolvimento e produção da infraestrutura necessária para suportar uma computação presente em qualquer lugar, a qualquer momento.

O termo computação pervasiva muitas vezes é utilizado como sinônimo à computação ubíqua. A principal diferença entre os dois conceitos é que a computação pervasiva é uma visão *bottom-up* que emergiu da exploração em larga escala de serviços de computação, enquanto computação ubíqua é uma abordagem *top-down* onde os serviços seriam utilizados de maneira transparente e integrados ao ambiente (ROBINSON; VOGT; WAGALLA, 2004).

A computação ubíqua contempla quatro princípios fundamentais (HANSMANN

et al., 2003):

1. **Descentralização:** esse modelo distribui as responsabilidades entre vários dispositivos, os quais assumem e executam certas tarefas e funções, simples e específicas para a finalidade que o dispositivo foi criado. Para isso uma rede dinâmica de relações é formada entre os dispositivos e entre dispositivos e servidores do ambiente, caracterizando um sistema distribuído. Para computação pervasiva, estes diversos dispositivos devem interagir de maneira dinâmica e autônoma, mantendo os dados sempre sincronizados em tempo de execução. Essa interação deve ocorrer entre os mais diferentes tipos de dispositivos, os quais possuem poder computacional diferentes;
2. **Diversificação:** nesse novo paradigma, ao invés de se ter um único computador que seja capaz de executar todas as tarefas, tem-se a diversificação de dispositivos que são responsáveis por determinada tarefa;
3. **Conectividade:** tem-se a visão da conectividade sem fronteiras, em que dispositivos e as aplicações que executam neles movem-se juntamente com o usuário, de forma transparente, entre diversas redes heterogêneas, tais como as redes sem fio de longa distância e redes de média e curta distância. Para que se atinja a conectividade e interoperabilidade desejada é preciso basear as aplicações em padrões comuns, levando ao desafio da especificação de padrões abertos;
4. **Simplicidade:** os dispositivos devem ser especializados, o que os tornam menos aptos a um uso geral, porém bem mais simples de serem usados em seu propósito específico. Devem ser úteis, convenientes e simples de serem usados, de modo que não seja necessária a leitura de um complexo manual para que possam ser utilizados.

A computação “invisível” promovida pela computação ubíqua é realizada através de interfaces de comunicação diferentes do tradicional teclado e mouse. As aplicações precisam ser sensíveis ao contexto e adaptarem seu comportamento através da informação capturada do ambiente.

A utilização de interfaces naturais possibilita uma maior capacidade nas comunicações entre pessoas e computadores. O objetivo dessas interfaces naturais é suportar formas comuns de expressão humana. Esforços anteriores se focaram em interfaces de reconhecimento de voz e escrita, mas estas interfaces ainda não lidam de forma adequada com erros que ocorrem naturalmente com estes sistemas. Além disso essas interfaces são muito difíceis de serem implementadas. A computação ubíqua inspira o desenvolvimento de aplicações que não utilizam o *desktop*. Implícito a isto está a consideração que a interação física entre humano e computadores será bem diferente do *desktop* atual com teclado, mouse, monitor, e será mais parecida com a maneira como os humanos interagem com o mundo físico. Interfaces que suportem formas de computação humanas mais naturais (fala, escrita e gestos) estão começando a substituir os dispositivos mais tradicionais. Estas interfaces se sobressaem por causa da sua facilidade de aprendizado e de uso. Além disso elas podem ser usadas por pessoas com deficiência física, para quem o tradicional mouse e teclado são menos acessíveis.

As aplicações para a computação ubíqua precisam ser sensíveis ao contexto, adaptando o seu comportamento baseando-se na informação adquirida do ambiente físico e

computacional. Nos últimos anos tivemos muitos avanços na área de localização e reconhecimento de identidade, mas ainda existem numerosos desafios na criação de representações de contexto reutilizáveis, e de reconhecimento de atividades. Duas demonstrações de computação ubíqua foram produzidas no laboratório de pesquisa da Olivetti e do Xerox Parctab, ambas demonstrações com aplicativos sensíveis à localização. Esses dispositivos forneciam a localização do usuário e proviam serviços interessantes como mapas, "sigame" automático e etc. Apesar da conexão entre dispositivos computacionais e o mundo físico não ser nova, esta simples aplicação sensível à localização é talvez a primeira demonstração ligando uma atividade humana implícita com serviços computacionais. As aplicações mais abrangentes são a navegação baseada em sistema GPS para carros e dispositivos portáteis que variam o conteúdo mostrado dando ao usuário a localização física dentro de uma área. Outra parte importante é o reconhecimento de objetos pessoais. Antigamente, sistemas se focavam no reconhecimento de algum tipo de código de barras ou etiqueta de identificação enquanto os trabalhos recentes incluem o uso de reconhecimento de imagem. Apesar de terem sido demonstrados vários sistemas que reconhecem a identidade da pessoa e sua localização eles ainda são difíceis de serem implementados.

2.2 Middleware EXEHDA

O EXEHDA está em desenvolvimento em um consórcio de pesquisa formado por universidades do RS. O projeto ISAM – Infraestrutura de Suporte às Aplicações Móveis (ISAM, 2009) foi concebido para execução de aplicações móveis distribuídas com comportamento adaptativo e considerando a computação pervasiva. A execução das aplicações no ISAM é gerenciada pelo *middleware* EXEHDA (YAMIN, 2004) (AUGUSTIN; YAMIN; SILVA, 2008).

Neste sentido, como aspecto central da sua contribuição, o EXEHDA propõe uma solução integrada para suporte à Computação Ubíqua, implementada na forma de um *middleware* que visa a criar e gerenciar um ambiente pervasivo, bem como promover a execução, sob este ambiente, das aplicações que expressam a semântica *sigame*. Estas aplicações são, por natureza, distribuídas, móveis e adaptativas ao contexto em que seu processamento ocorre, estando disponíveis a partir de qualquer lugar, todo o tempo.



Figura 2.1: Middleware EXEHDA (YAMIN, 2004)

O EXEHDA (YAMIN, 2004) é o *middleware* que disponibiliza a abstração do am-

biente pervasivo deste trabalho. O EXEHDA, figura 2.1, é um componente da arquitetura ISAM. A arquitetura ISAM foi modelada baseada nos conceitos do modelo Holoparadigma, remodelado para o ambiente pervasivo.

As principais funcionalidades do EXEHDA são:

- gerenciar aspectos funcionais e não funcionais da execução das aplicações;
- suportar adaptações dinâmicas na execução das aplicações;
- fornecer mecanismos para construir, gerenciar e disseminar informações de contexto;
- usar as informações de contexto em seus mecanismos de tomada de decisão;
- decidir, juntamente com as aplicações, a respeito de ações de adaptação;
- oferecer aos usuários um comportamento que expresse a semântica *siga-me* das aplicações pervasivas em que a aplicação segue o usuário em sua movimentação pelo ambiente ubíquo.

Os recursos da infraestrutura física, representados pela figura 2.2, são mapeados para três abstrações básicas as quais são utilizadas na composição do ambiente ubíquo:

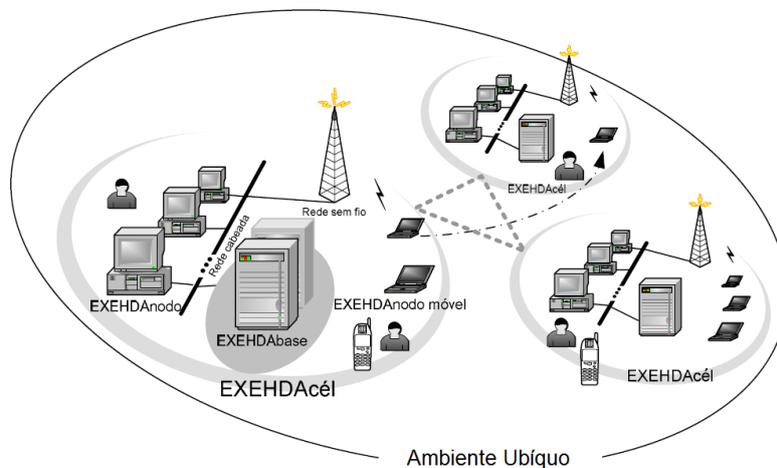


Figura 2.2: Ambiente Ubíquo (YAMIN, 2004)

- EXEHDAcél: denota a área de atuação de uma EXEHDAbase, e é composta por esta e por EXEHDAnodes;
- EXEHDAbase: é o ponto de contato para os EXEHDAnodes. Uma EXEHDAbase é responsável por todos os serviços básicos de uma célula de execução e, embora constitua uma referência lógica única, seus serviços, sobretudo por aspectos de escalabilidade, poderão estar distribuídos entre vários equipamentos;
- EXEHDAnode: são os equipamentos de processamento disponíveis no ISAMpe, sendo responsáveis pela execução das aplicações. Um subtipo dessa abstração é o

EXEHDAmob-node. Esses são os nós do sistema com elevada portabilidade, tipicamente dotados de interface de rede para operação sem-fio e, neste caso, integram a célula à qual seu ponto-de-acesso está subordinado. São funcionalmente análogos aos EXEHDAnodes, mas tipicamente são recursos com uma capacidade mais restrita (por exemplo, PDAs).

O núcleo mínimo do EXEHDA, o qual é instalado em todo EXEHDAnode que for integrado ao ISAMpe, é composto por dois componentes:

- *ProfileManager*: responsável por interpretar as informações contidas nos perfis de execução, tornando esses dados disponíveis para outros serviços em tempo de execução;
- *ServiceManager*: realiza a ativação de serviços em um nó, baseado nas informações providas pelo ProfileManager.

Os serviços do EXEHDA estão organizados em quatro grandes subsistemas: execução distribuída, adaptação, comunicação e acesso pervasivo. O processo de descoberta de recursos previsto pelo *middleware* EXEHDA está distribuído entre vários serviços do subsistema de execução distribuída.

O subsistema de Execução Distribuída é composto pelos seguintes serviços:

- *Executor*: realiza o disparo de aplicações, criação e migração de seus objetos;
- *Cell Information Base (CIB)*: implementa a base de informações da célula, mantendo os dados estruturais da EXEHDACell, tais como, informações sobre os recursos que a compõe, informação de vizinhança e atributos que descrevem as aplicações em execução;
- *OXManager*: a abstração OX - Objeto eXehda -, provida pelo *middleware* às aplicações, consiste em uma instância de objeto, criada por intermédio do serviço Executor, à qual pode ser associada meta-informação em tempo de execução;
- *Discoverer*: é responsável pela localização de recursos especializados no ISAMpe a partir de especificações abstratas dos mesmos. As especificações caracterizam o recurso a ser descoberto por meio de atributos e seus respectivos valores;
- *ResourceBroker*: faz o controle da alocação de recursos às aplicações;
- *Gateway*: faz a intermediação das comunicações entre os nós externos a uma célula e os recursos internos a ela, podendo alternar a visibilidade dos recursos de uma célula quando vistos de fora dela;
- *StdStreams*: provê o suporte ao redirecionamento dos streams padrões de entrada, saída e erro. Sua funcionalidade se dá numa perspectiva por aplicação, sem a necessidade de modificação no código da mesma;
- *Logger*: esta funcionalidade é frequentemente empregada para registro de operações importantes e/ou críticas realizadas, facilitando a identificação de situações de intrusão, ou de uso indevido do sistema;

- *Dynamic Configurator - DC*: realiza a configuração do perfil de execução do *middleware*.

O Subsistema de reconhecimento de contexto e adaptação é composto pelos seguintes serviços:

- *Collector*: responsável pela extração da informação bruta que dará origem aos elementos de contexto;
- *Deflector*: disponibiliza a abstração de canais *multicast* para uso na disseminação das informações monitoradas;
- *ContextManager*: responsável pelo refinamento da informação bruta produzida pela monitoração para produção de informações abstratas referentes aos elementos de contexto;
- *AdaptEngine*: controla as adaptações de cunho funcional e provê facilidades para definição e gerência de comportamentos adaptativos por parte das aplicações;
- *Scheduler*: serviço central na gerência das adaptações de cunho não-funcional, ou seja, que não implicam alteração de código.

O Subsistema de comunicação é composto pelos seguintes serviços:

- *Dispatcher*: disponibiliza um modelo de comunicação através de troca de mensagens ponto-a-ponto com garantia de entrega e ordenamento das mensagens;
- *WORB*: modelo de comunicação baseado em invocações remotas de método, similar ao RMI, porém sem exigir a manutenção da conexão durante toda a execução da chamada remota;
- *CCManager*: disponibiliza desacoplamento temporal e espacial através de um mecanismo baseado na abstração espaço de tuplas.

O Subsistema de acesso pervasivo é composto pelos seguintes serviços:

- *BDA*: o serviço BDA (Base de Dados pervasiva das Aplicações) contempla métodos para a recuperação do código integral de uma aplicação ou de componentes específicos e suas dependências;
- *AVU*: o AVU (Ambiente Virtual do Usuário) é responsável pela manutenção do acesso pervasivo ao ambiente virtual, da forma mais eficiente possível;
- *SessionManager*: gerencia a sessão de trabalho do usuário, sendo definida pelo conjunto de aplicações correntemente em execução para aquele usuário. A informação que descreve o estado da sessão de trabalho é armazenada no AVU, estando portando disponível de forma pervasiva;
- *Gatekeeper*: responsável por intermediar acessos entre as entidades externas à plataforma ISAM e os serviços do middleware de execução, conduzindo os procedimentos de autenticação necessários.

O processo de gerenciamento de cada célula é autônomo em relação às outras células, e cada célula é responsável por gerenciar e prover acesso aos componentes computacionais locais, os quais podem ser dados, código, dispositivos, serviços ou outros recursos. Cada célula tem associada uma *Cell Information Base* (CIB), que mantém controle de toda a informação estática e dinâmica originada internamente à célula. Além disso, cada célula tem um conjunto dinâmico de outras células conhecidas no ISAMpe, o qual compõe sua vizinhança.

2.3 Descoberta de Recursos: Fundamentos e Requisitos

Esta seção apresenta sucintamente os objetivos dos mecanismos de descoberta, os conceitos envolvidos e as características arquiteturais que devem ser perseguidas pelos mecanismos de descoberta.

2.3.1 Objetivos Gerais

Com cada vez mais dispositivos movendo-se dentro de um ambiente ubíquo, um grande número de dispositivos podem estar presentes em um dado ambiente a qualquer momento. A conectividade global remove o senso de localização do mundo físico. Um dispositivo localizado do outro lado do mundo pode ser acessado da mesma forma se ele estivesse localizado na sala ao lado, por estas razões o número de dispositivos que o cliente pode acessar é muito grande, e cabe ao serviço de descoberta selecionar os recursos viáveis para serem utilizados pelo cliente.

Atualmente não existe um padrão na terminologia utilizada pelos mecanismos de descoberta para referenciar os recursos. Segundo McGrath (MCGRATH, 2000), dispositivos e serviços são recursos equivalentes. Um serviço é uma interface para uma aplicação ou dispositivo pelo qual o cliente pode acessar a aplicação ou dispositivo. A localização e interação com serviços em um ambiente ubíquo é de responsabilidade do serviço de descoberta. Segundo (THOMPSON, 2006), o serviço de descoberta pode ser dividido em tarefas de descrição, disseminação, seleção e interação. Cada componente tem um conjunto único de responsabilidades para o realizar o serviço de descoberta.

Conforme (MARIN-PERIANU; HARTEL; SCHOLTEN, 2005) os protocolos de descoberta de recursos possuem os seguintes objetivos:

- descoberta: habilidade de encontrar um provedor de serviço na rede de acordo com os critérios descritos pelo cliente. Para concretizar este requisito os protocolos necessitam utilizar uma linguagem de descrição para facilitar o processo de descoberta em que as requisições de serviços (recursos) também são expressas utilizando essa linguagem. Os protocolos devem também armazenar as informações dos recursos. O armazenamento pode ser centralizado, distribuído ou híbrido. A procura pelos recursos deve ser endereçada a um diretório ou disseminada na rede;
- transparência: a rede precisa organizar e disponibilizar informação sobre seu conteúdo sem intervenção humana. Para isso os protocolos precisam manter atualizadas as descrições dos recursos, alterando quando houver alterações nos recursos. A consistência deve ser mantida verificando a disponibilidade dos serviços.

Na computação ubíqua o mecanismo de descoberta deve atender aos seguintes requisitos (SCHAEFFER, 2005): utilização de informações do contexto de execução, uti-

lização de estratégias para manutenção automática de consistência, expressividade na descrição de recursos e critérios de pesquisa, possibilidade de interoperabilidade com outras estratégias de descoberta, suporte à descoberta de recursos em larga escala e utilização de preferências por usuário.

2.3.2 Principais Requisitos e Funcionalidades

As características arquiteturais dos mecanismos de descoberta de recursos desta seção foram organizados em dez etapas, conforme o trabalho de (ZHU; MUTKA; NI, 2005).

1. Nomeação de Recursos e Atributos: quando um cliente solicita um recurso é necessário especificar o nome do recurso e atributos. O mecanismo de descoberta precisa possuir um formato para os nomes de recursos e atributos (*template-based*). Alguns mecanismos oferecem um conjunto pré-definido de atributos e nomes de recursos frequentemente utilizados (*predefined*).

Descrição de recurso é uma abstração das características e facilidades inerentes a um recurso. Essa abstração torna-se necessária em diferentes etapas do processo de descoberta quando:

- um recurso, na arquitetura centralizada, é registrado pelo seu provedor junto ao diretório;
- um provedor, na arquitetura distribuída, divulga na rede, através de mensagens de anúncios, a disponibilidade dos serviços que oferece;
- o serviço é requisitado por outros dispositivos ou mesmo por outros recursos.

Gonzalez-Castillo et al. em (GONZALEZ-CASTILLO; TRASTOUR; BARTOLINI, 2001) identificou os seguintes requisitos que devem ser atendidos pela linguagem para expressar as descrições dos serviços:

- Alto grau de flexibilidade e expressividade: é a capacidade da linguagem permitir que a publicidade seja composta de vários graus de complexidade e completude. Dependendo das propriedades de um serviço que precisa ser descrito, algumas propriedades podem ser expressas com simples pares atributo/valor, outros podem necessitar de mais estruturas. Os provedores de serviços podem descrever determinados aspectos do serviço a um grande nível de detalhamento, mas podem não especificar certas propriedades porque não são conhecidas, não aplicáveis ou porque precisam ser negociadas mais tarde;
- Suporte a relacionamentos entre conceitos: para o *matchmaking* realizar *matches* complexos, baseado em relacionamentos, a linguagem precisa permitir o relacionamento entre conceitos;
- Suporte a tipos de dados: atributos como quantidades, datas e preços podem fazer parte da descrição do serviço. Por isso a linguagem de descrição precisa suportar tipos de dados para facilitar a expressividade e o emparelhamento (*matching*) de atributos no serviço de descrições;

- Expressar restrições e limitações: serviços e requisições precisam ser descritos com definições conceituais de instâncias aceitáveis ao invés de uma simples instância do serviço. A linguagem de descrição deve facilitar a descrição entre limites sobre certos parâmetros;
 - Nível de concordância semântico: para cada *matchmaker* entender e comparar diferentes descrições de serviços, eles precisam compartilhar a mesma semântica. Para isso é necessário o uso de ontologias comuns na descrição dos serviços, permitindo interoperabilidade entre diferentes provedores de serviços e clientes.
2. Método de Comunicação Inicial: os clientes, recursos e diretórios precisam de um método de comunicação inicial. Isto pode ser feito através de mensagens *unicast*, mas é necessário o conhecimento prévio do endereço destino. Outra solução é através de mensagens UDP *multicast*. Clientes, recursos e diretórios recebem poucas mensagens UDP *multicast* para encontrar endereços *unicast* dinamicamente e mudar a comunicação para *unicast*. E, por último, o método de comunicação através de mensagens *broadcast*. *Broadcast* tem as mesmas vantagens de *multicast*, mas geralmente é limitada a uma rede.
3. Descoberta e Registro: clientes, recursos e diretórios podem utilizar dois métodos para descobrir e registrar informação: baseado em anúncio (*announcement-based*) e baseado em consulta (*query-based*). Na técnica baseada em anúncio as partes interessadas escutam um canal. Neste método, um cliente pode aprender que o recurso existe e o diretório pode registrar a informação do recurso. No método baseado em consulta o cliente recebe uma resposta imediata à consulta e não precisa processar anúncios não relacionados.

O mecanismo de descoberta pode consultar o diretório sobre um determinado recurso ou acessar o recurso desejado diretamente.

Basicamente, existem duas formas para se consultar informações sobre os serviços disponíveis na rede: a descoberta passiva, onde os provedores anunciam periodicamente os seus serviços para toda a rede, e a descoberta ativa, onde o cliente envia mensagens de descoberta para provedores ou diretórios com o intuito de obter informações sobre um serviço específico ou sobre todos os serviços disponíveis. Essas abordagens também são conhecidas, na literatura, como proativa e reativa, respectivamente, em uma alusão à classificação utilizada com os protocolos de roteamento planos para redes sem fio ad hoc de saltos múltiplos. Grande parte dos protocolos de descoberta implementa ambas as abordagens.

Na descoberta de recursos baseada em *query* deve-se prestar atenção aos seguintes fatores:

- *Query Language and Advertising Language*: linguagem com suporte à formulação de consultas, bem como descrição de serviços e recursos é crucial para o processo de descoberta. Isto é importante para avaliar a expressividade da linguagem e sua facilidade em formular consultas e publicações utilizadas durante o processo de descoberta. Atenção especial deve ser dada para o suporte semântico que a linguagem fornece para expressar diferentes aspectos de publicação e consultas;

- *Scalability*: é importante analisar como um sistema reage quando há um crescimento em uma ou mais de suas dimensões. A escalabilidade do mecanismo de descoberta é afetado pela escalabilidade de subcomponentes como processadores e dispositivos de armazenamento;
- *Reasoning support*: um mecanismo de descoberta automático pode ser melhorado significadamente se o processamento das descrições da máquina são melhorados. Estas descrições são checadas se elas são equivalentes umas às outras. Novo conhecimento pode ser inferido baseado em fatos existentes e este conhecimento pode ser adicionado durante o processo de descoberta.

Clientes procuram por um determinado recurso preenchendo valores nos campos do serviço de descrição. Alguns protocolos utilizam pares de atributos-valores hierárquicos, como (BALAZINSKA; BALAKRISHNAN; KARGER, 2002), muitos descrevem seus recursos em XML, como (SCHAEFFER, 2005). Alguns utilizam ontologias para uma melhor classificação e expressividade, tal como, (TANGMUNARUNKIT; DECKER; KESSELMAN, 2003).

Consultas de usuários compostas de pares de atributos e valores requerem que o conteúdo da comparação satisfaça todos os pares presentes na requisição (MARINPERIANU; HARTEL; SCHOLTEN, 2005). Se um atributo não é especificado, geralmente considera-se qualquer valor. Alguns protocolos possibilitam a seleção com vários critérios na qual resulta uma coleção de nodos que satisfazem mais de uma condição.

4. Infraestrutura do Serviço de Descoberta: a infraestrutura dos mecanismos de descoberta são baseadas em diretório ou em modelos sem diretório. Em modelos baseados em diretório são mantidas as informações sobre os recursos e processadas as consultas e os anúncios. Modelos não baseados em diretório são ideais para pequenos ambientes, com poucos recursos.

Uma estratégia de descoberta de recursos costuma apresentar uma arquitetura com dois ou três componentes. No primeiro caso, figura 2.3, são empregados apenas Resource Components (RCs) e User Components (UCs). No segundo caso, figura 2.4, acrescenta-se a uma terceira estrutura chamada Directory Component (DC).

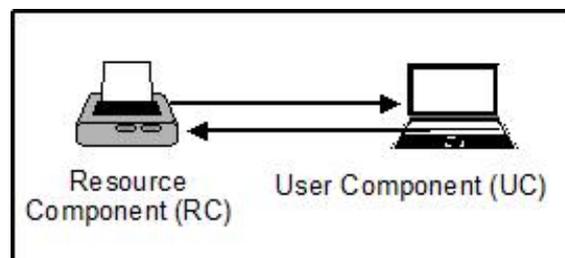


Figura 2.3: Descoberta de Recursos: Arquitetura de 2 Componentes

O diretório (DC) é uma entidade que gerencia as informações dos serviços disponíveis na rede. Os mecanismos de descoberta podem adotar uma abordagem centralizada, baseada em diretórios os quais são responsáveis pelo processamento de anúncios e consultas em nome de todos os dispositivos da rede, ou distribuída

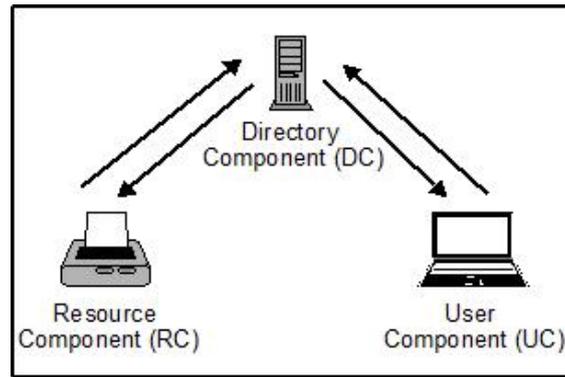


Figura 2.4: Descoberta de Recursos: Arquitetura de 3 Componentes

em que cada dispositivo é responsável por manter as informações sobre os serviços que disponibiliza ou conhece.

As arquiteturas centralizadas podem utilizar um ou mais diretórios para tratar o registro das informações de serviços e as solicitações dos clientes, subdividindo-se em estruturas hierárquicas e planas. No modelo hierárquico, os diretórios estão organizados em uma estrutura em árvore, similar à estrutura adotada pelo DNS. No modelo plano, os diretórios trocam informações entre si sobre os serviços que cada um gerencia.

A vantagem dessa técnica é que o gerenciamento dos recursos distribuídos é centralizada, o controle de autoridade e qualidade é facilmente gerenciada e a descrição do serviço é consistente no estilo e apresentação.

Como desvantagem está a pouca escalabilidade, pois há um limite do número de recursos que podem ser gerenciados e registrados no registro global. Pode ocorrer também um alto tempo de resposta com acessos paralelos ao registro central.

As arquiteturas distribuídas baseiam-se na interação direta entre clientes e provedores de serviços, não existindo a sobrecarga da administração do diretório, como ocorre nas arquiteturas centralizadas. Nas arquiteturas distribuídas, as entidades produzem um maior volume de tráfego de mensagens de controle e o processamento das requisições é transferido dos diretórios para os clientes do sistema. Alguns protocolos oferecem suporte tanto à arquitetura centralizada quanto à distribuída.

(MARIN-PERIANU; HARTEL; SCHOLTEN, 2005) define uma arquitetura híbrida da arquitetura centralizada e distribuída. Cada VO (Virtual Organization) tem seu próprio registro local e toda organização é gerenciada por um registro global centralizado.

Esta técnica tem o benefício de compartilhar o gerenciamento e a administração dos registros locais e globais. As VOs são responsáveis pelo gerenciamento de seus próprios recursos, mas o registro global é de responsabilidade de gerenciamento de alto nível. Quando um serviço está ausente no registro local dentro de uma VO, a requisição é enviada para o registro global requisitando onde encontrar este serviço. Ele então pode contactar diretamente a VO responsável.

Esta técnica escala bem até o limite da capacidade de armazenagem do registro global.

Os recursos são administrados duas vezes, nos registros locais de cada VO e também no registro global. Se o registro global falhar, não será possível comunicar entre as VOs. Entretanto, um registro global replicado é necessário para garantir um sistema totalmente funcional.

5. Serviço de Informações de Estado: muitos mecanismos de descoberta de recursos mantêm o estado dos recursos como *soft state*. Um anúncio de recurso especifica o tempo de vida do mesmo. Antes do recurso expirar, um cliente ou diretório pode renovar o tempo de registro. Alternativamente, clientes e diretórios podem manter o estado do recurso com um *hard state*. *Hard state* necessita de poucos anúncios, mas requer que clientes e diretórios periodicamente chequem o recurso para certificar que sua informação está atualizada.

6. Escopo da Descoberta: entende-se por “Escopo da Descoberta” o conjunto de serviços que podem ser descobertos por um cliente. O conceito de escopo possibilita que os serviços disponíveis em uma rede possam ser observados sob diferentes perspectivas.

A definição do escopo da descoberta pode ser feita com base na topologia da rede, nas permissões de acesso embutidas nos diferentes usuários de um domínio administrativo e nas informações de contexto.

Em protocolos de descoberta distribuídos, o escopo é geralmente definido em função da topologia da rede e o alcance do mecanismo de descoberta é determinado, em parte, pelas regras de roteamento.

Nos protocolos de descoberta centralizados, baseados na estrutura de diretórios, o escopo pode ser expandido interconectando-se diretórios em diferentes domínios administrativos com relações de confiança previamente estabelecidas. O escopo minimiza computação desnecessária nos clientes, recursos e diretórios. Escopos baseiam-se em topologias de rede, papéis de usuários e informações de contexto.

7. Seleção de Recursos: através do escopo é limitado o número de recursos que satisfazem a requisição do cliente. O resultado pode conter uma lista de recursos. Quando isto acontece o mecanismo de descoberta pode oferecer seleção manual ou automática de recursos.

8. Invocação de Recursos: após a seleção do recurso, um cliente invoca-o. Invocação envolve o endereço de rede do recurso, mecanismo de comunicação básico e operações específicas para um domínio de aplicação.

9. Utilização do Recurso: em protocolos que utilizam o mecanismo *lease-based* o cliente e o recurso negociam um período de utilização o qual o cliente pode cancelar ou renovar. No mecanismo *explicitly release* o cliente precisa explicitamente solicitar o recurso, uma vez feito isto, sua utilização está garantida.

10. Apurar o Estado do Recurso: um cliente pode obter o estado do recurso sondando-o, *polling*, periodicamente, ou através do *service event notification*. Neste método, clientes registram com um serviço, e o serviço notifica-os quando alguma coisa de interesse acontece.

2.3.3 Gerenciamento da Informação

O método de consulta utilizado em um protocolo de descoberta de serviços pode ser entendido como um processo de busca atrelado a um casamento (*matching*) entre as requisições de descoberta (demandas) e as descrições de serviços locais e remotos (ofertas), estas últimas divulgadas através de anúncios ou registradas em diretórios. O algoritmo de *matching* utilizado pelo mecanismo de consulta representa uma função que aceita, como entrada, a demanda e um conjunto de descrições de ofertas, provendo, como resultado, o subconjunto das ofertas que satisfazem a demanda especificada.

Em (DECKER; WILLIAMSON; SYCARA, 1996), é destacada a diferença entre os termos *matchmaking* e *brokering*. *Matchmaking* é um processo que permite que um agente com certo objetivo possa se comunicar diretamente com outros agentes para satisfazer a este objetivo, envolvendo três diferentes agentes:

- Consumidor: um agente com um objetivo que deve ser atingido por algum outro agente;
- *Matchmaker*: um agente que sabe como se comunicar com outros agentes e sabe de suas capacidades;
- Provedor de Recursos: um *matchmaking* agente que disponibiliza suas capacidades para serem utilizadas por outros agentes.

O processo de *brokering* é formado por um agente com determinado objetivo que pode ser alcançado por outro agente. O processo de *brokering* também envolve três agentes diferentes:

- Consumidor: um agente com um objetivo que deve ser atingido por algum outro agente;
- *Broker*: um agente que sabe como se comunicar com outros agentes e sabe de suas capacidades. É responsável por selecionar os agentes que melhor atendam à solicitação de um serviço;
- Provedor de Recursos: um agente que disponibiliza suas capacidades para serem utilizadas por outros agentes.

No *matchmaking* a decisão de qual agente irá prover o serviço é tomada pelo cliente tendo a responsabilidade de negociar com o provedor de recursos diretamente para a realização do serviço, já no *brokering* esta seleção é feita pelo *broker*.

O algoritmo de *matching* pode empregar diferentes funções de comparação, conforme representado na figura 2.5: uma função simples, baseada na comparação de atributos (influenciada pela descrição baseada em pares atributo-valor), uma função semântica (influenciada pela uso de linguagens de descrição) ou uma função dependente de linguagem de programação.

Algoritmos de *matching* baseados na comparação sintática de atributos podem ser implementados através da comparação de um único atributo, geralmente o identificador ou o tipo do serviço, ou de vários atributos. Algoritmos de *matching* baseados na função de comparação semântica beneficiam-se dos mecanismos de consulta embutidos nas linguagens de descrição, provenientes da utilização de padrões relacionados ao XML os

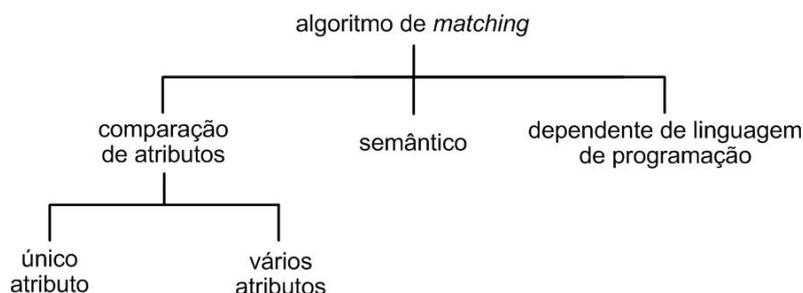


Figura 2.5: Algoritmos de Matching para Consulta por Recursos

quais oferecem mecanismos para se representar a forma como os recursos se relacionam, incluindo propriedades como domínio e cardinalidade, e restrições de integridade, o que garante consultas mais poderosas. A representação semântica dos serviços consiste em um conjunto de informações que abrangem, mas não se limitam, a descrição de suas capacidades, funcionalidades, portabilidade e requisitos do sistema – como largura de banda, sistema operacional e processador. Uma função de *matching* semântico introduz a possibilidade de se obter resultados aproximados em resposta a uma requisição de serviço. Nesse caso, dependendo dos requisitos definidos na consulta, o resultado da função de *matching* pode ser satisfatório mesmo se uma, ou mais características do serviço não sejam satisfeitas integralmente. Por exemplo, se uma requisição especifica um serviço que possa ser executado em sistemas baseados em um processador de uma determinada família (por exemplo, Intel Pentium), e uma instância desse serviço, compatível com sistemas baseados nessa família for descoberta, um resultado aproximado é encontrado. Por fim, existem abordagens que atrelam a função de comparação à linguagem de programação utilizada. Este é o caso de Jini que apresenta um forte acoplamento à linguagem de programação Java. Como resultado do processamento da requisição de descoberta pelo algoritmo de *matching*, são obtidas informações sobre o serviço requisitado as quais permitirão a sua invocação e utilização. Essas informações são encaminhadas ao dispositivo que originou a requisição, encapsuladas em uma mensagem de resposta à solicitação do serviço.

2.3.4 Suporte à Mobilidade

O suporte à mobilidade implica que a informação sobre os serviços disponíveis na rede, quer seja armazenada nos diretórios (centralizada) ou em cada dispositivo da rede (distribuída), deva ser atualizada em função das modificações na topologia da rede, provocadas pela mobilidade dos dispositivos (LIMA, 2007). Se, em um dado grupo, um dispositivo armazena informações sobre os serviços dos demais dispositivos, espera-se que ele mantenha informações corretas, na medida do possível. Se o dispositivo altera a sua posição em relação ao grupo, ou se algum membro do grupo se desloca, as informações de serviços devem ser atualizadas o mais rapidamente possível. Somente dessa forma, pode-se esperar, com uma certa probabilidade, que um protocolo de descoberta consiga descobrir serviços em tempo hábil. Caso as informações mantidas por um dispositivo sobre os serviços da rede sejam obsoletas, há uma grande chance desse dispositivo responder a uma requisição de serviço com informações desatualizadas. Ao receber a resposta à sua solicitação, o cliente tentará acessar o serviço e, só então, irá detectar a sua indisponibilidade, pois o provedor do serviço pode ter se deslocado ou se tornado

inalcançável. Existem dois métodos principais para solucionar esse problema: proativos e reativos. No método proativo, os dispositivos mantêm uma visão atualizada das informações sobre os serviços disponíveis na rede com a troca periódica de mensagens de anúncio contendo dados mais recentes sobre o serviço. No método reativo, a informação é atualizada em razão da ocorrência de eventos na rede, como, por exemplo, a indisponibilidade de uma rota para um provedor ou a detecção de mudanças de estado informadas pelo próprio serviço.

2.3.5 Descoberta com Semântica

Um dos problemas em mecanismos de descoberta baseados em palavras-chave é que eles não conseguem capturar completamente a semântica da consulta do usuário porque eles não consideram os relacionamentos entre as palavras.

Um mecanismo de descoberta de recursos que utiliza tecnologias de web semântica, em especial ontologias, pode facilmente ser estendido, adicionando-se um vocabulário e regras de inferência para incluir novos conceitos sobre recursos e aplicações.

No trabalho de (SHARMA A BAWA, 2007) é feito um comparativo de vários critérios, confrontando mecanismos de descoberta baseados em sintaxe com mecanismos de descoberta baseados em semântica, conforme tabela 2.1.

Tabela 2.1: Comparativo de Descoberta de Recursos Baseado em Sintaxe e Semântica

Critério	Sintaxe	Ontologias
Descrição do Recurso e Requisição	Simétrico	Assimétrico
Manutenção e Compartilhamento da Ontologia	Difícil de manter e compartilhar	Fácil de manter e compartilhar
Preferências de Matching	Muito limitado, baseado em sintaxe	Facilmente adicionado nos recursos e requisições
Checagem da Integridade	Sem checagem de integridade	Feito na base do conhecimento do domínio
Expressividade	Menos Expressivo	Muito expressivo, a requisição pode ser modelada para aplicações específicas do domínio
Flexibilidade e Extensibilidade	Menos flexível e extensível	Novos conceitos e restrições podem ser facilmente adicionadas dentro da Ontologia a qualquer momento

2.4 Revisando Tecnologias de Processamento Semântico

Considerando o interesse central deste trabalho de explorar modelos semânticos para descoberta de recursos na computação ubíqua, o estudo da Web Semântica e de suas diferentes tecnologias se mostram um aspecto central para qualificação das soluções a serem buscadas.

Esta seção sintetiza o estudo feito envolvendo tecnologias de Web Semântica para a construção e processamento de ontologias.

A Web Semântica descrita por Tim Berners-Lee (BERNERS-LEE; HENDLER; LASSILA, 2001), é uma extensão da Web atual, na qual a informação tem um significado bem definido, permitindo pessoas e computadores trabalharem em cooperação. Na Web atual a informação é processada pelos computadores a nível sintático, no futuro da Web Semântica será possível os computadores processarem e raciocinarem as informações no nível semântico. A Web Semântica faz uso de várias tecnologias para possibilitar esta automatização semântica. O W3C (*World Wide Web Consortium*) define e mantém uma arquitetura em camadas para Web Semântica, conforme a figura 2.6.

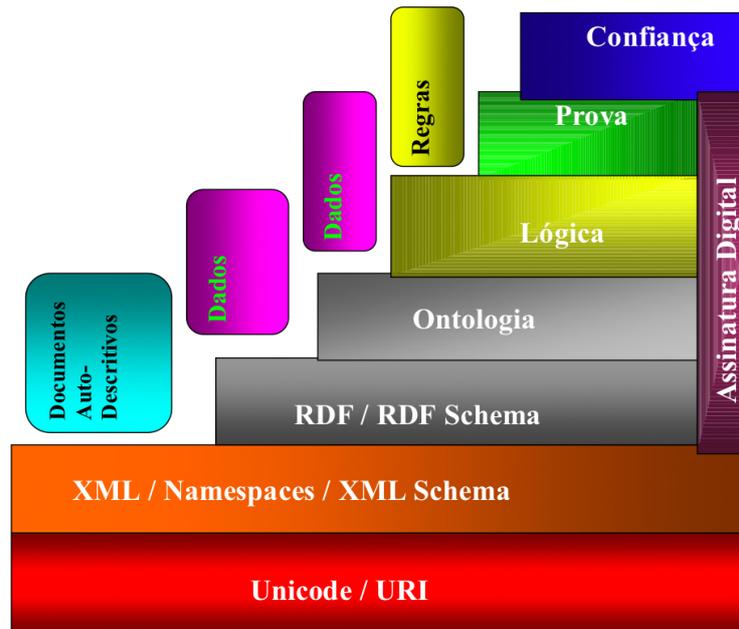


Figura 2.6: Camadas da Web Semântica Proposta pela W3C

A camada denominada **Unicode / URI** fornece a interoperabilidade em relação à codificação de caracteres e ao endereçamento e nomeação de recursos da Web Semântica.

A camada denominada de **XML / Namespace / XML Schema** fornece a interoperabilidade em relação à sintaxe de descrição de recursos da Web Semântica.

A camada denominada **RDF / RDF Schema** fornece um *framework* para representar informação (metadados) sobre recursos.

A camada denominada de **Ontologia** fornece suporte para a evolução de vocabulários e para processar e integrar a informação existente sem problemas de indefinição ou conflito de terminologia. A linguagem RDF-Schema permite a construção de ontologias com expressividade e inferência limitadas, pois fornece um conjunto básico de elementos para a modelagem, e poucos desses elementos podem ser utilizados para inferência. A *Web Ontology Language* (OWL) estende o vocabulário da *RDF Schema* para a inclusão de elementos com maior poder com relação à expressividade e inferência.

A camada denominada **Lógica** fornece suporte para a descrição de regras para expressar relações sobre os conceitos de uma ontologia as quais não podem ser expressas com a linguagem de ontologia utilizada.

As camadas denominadas de **Prova e Confiança** fornecem o suporte para a execução das regras, além de avaliar a correção e a confiabilidade dessa execução. Essas

camadas ainda estão em desenvolvimento e dependem da maturidade das camadas inferiores.

2.4.1 Lógica de Descrição

A Lógica de Descrição, ou *Description Logic* (DL), é uma evolução dos formalismos de representação do conhecimento baseados em objeto ao qual corresponde um subconjunto estruturado da lógica de primeira ordem. Em termos gerais as lógicas de descrições são formalismos para representar conhecimento e raciocinar sobre ele. A OWL é baseada em lógica de descrição, havendo uma correspondência entre as linguagens XML para expressar ontologias e uma lógica de descrição.

Em DL, o conhecimento é representado por conceitos (predicados ou classes) e papéis (relacionamentos binários). Conceitos e papéis podem ser construídos utilizando construtores disponibilizados pela linguagem. Uma base de conhecimento DL consiste de uma TBox (*Terminological Knowledge*) e uma ABox (*Assertional Knowledge*). TBox contém definições de conceitos e axiomas (definem como conceitos e papéis estão relacionados). ABox contém o conhecimento extensional que especifica indivíduos do domínio. Ele é a instanciação da estrutura de conceitos.

Na Figura 2.7 são ilustrados os construtores da lógica descritiva ALC (*Attribute Language with Complement*) (operador de conjunção/interseção, disjunção/união, negação, e os quantificadores “para todo” e “existe”).



Figura 2.7: Construtores da Lógica de Descrição

A utilização de DL como método de representação do conhecimento possibilita a utilização de sistemas de raciocínio. Sistemas de raciocínio são sistemas que têm como objetivo processar conhecimento representado explicitamente e encontrar informações implícitas nestas informações, através de mecanismos específicos.

2.4.2 Ontologias

Ontologia é um modelo de dados que representa um conjunto de conceitos compartilhados (GRUBER, 1993). Ontologias são utilizadas para compartilhar informações de um domínio, composto de um vocabulário bem definido e com um entendimento comum e não ambíguo dos termos e conceitos utilizados pelas aplicações. Os elementos que formam uma ontologia são definidos pela tripla: Classes, Atributos e Relacionamentos. Um grande benefício no uso de ontologias é a possibilidade de descrever os relacionamentos entre os objetos. O conjunto destes relacionamentos é chamado de semântica. Este modelo semântico facilita o entendimento dos significados dos objetos pelas máquinas.

Uma ontologia pode ser definida como um método de representar itens de conhecimento através da definição dos relacionamentos e tipos de conceitos dentro de um domínio específico de conhecimento e a implementação desses relacionamentos em software (JEPSEN, 2009).

As ontologias não apresentam sempre a mesma estrutura, mas existem características e componentes básicos comuns presentes em grande parte delas. Mesmo apresentando

propriedades distintas, é possível identificar tipos bem definidos.

Os componentes básicos de uma ontologia são classes (organizadas em uma taxonomia), relações (representam o tipo de interação entre os conceitos de um domínio), axiomas (usados para modelar sentenças sempre verdadeiras) e instâncias (utilizadas para representar elementos específicos, ou seja, os próprios dados) (Gruber, 1996; Noy & Guinness, 2001).

Algumas das propostas definem tipos de ontologias relacionando-as à sua função (MIZOGUCHI; VANWELKENHUYSEN; IKEDA, 1995), ao grau de formalismo de seu vocabulário (Uschold & Gruninger, 1996), à sua aplicação (Jasper & Uschold, 1999) e à estrutura e conteúdo da conceitualização (Van-Heijst, Schreiber & Wielinga, 1997), (Haav & Lubi, 2001).

Em (GUARINO, 1998), propõe uma classificação de ontologias sobre três aspectos:

- Pelo nível de detalhes:
 - Ontologias de referência (off-line)
 - Ontologias compartilháveis (on-line)
- Pelo nível de dependência de uma tarefa ou ponto de vista:
 - Ontologias de alto nível
 - Ontologias de domínio
 - Ontologias de tarefas
 - Ontologias de aplicações
- Ontologias de representação.

2.5 Construção e Processamento de Ontologias

Gruber (GRUBER, 1993) apresenta quatro componentes que devem ser definidos para especificação de uma ontologia: classes, relações, funções e axiomas. Estes componentes são definidos da seguinte forma:

- Conceitos/Classes: ontologias são organizadas em taxonomias que podem ser interpretadas como sendo a identificação e a classificação dos termos presentes na ontologia;
- Relações: representam um tipo de interação entre conceitos e o domínio. Exemplos de relações binárias: “subclasse de” e “conectado a”;
- Funções: consistem de um caso especial de relações, em que o n-ésimo elemento do relacionamento é único para os n-1 elementos precedentes. Exemplo de funções: “Mãe-de”, associando um ou vários filhos à sua mãe;
- Axiomas: são sentenças formais que possuem o resultado lógico sempre verdadeiro.

Alguns outros termos importantes definidos no trabalho de (LOPES, 2005):

- Taxonomia: consiste de um conjunto de termos ou conceitos que organizados em uma hierarquia formam uma ontologia;
- Slots/Papéis/Propriedades (*Slots/Roles/Properties*): representam as várias características e atributos de um conceito;
- *Facets*: descrevem restrições nos *slots*;
- Instâncias: representam elementos.

2.5.1 XML

XML (*eXtensive Markup Language*) é uma linguagem de marcação recomendada pela W3C. Em meados da década de 1990, o World Wide Web Consortium (W3C) começou a trabalhar em uma linguagem de marcação que combinasse a flexibilidade da SGML (Linguagem Padronizada de Marcação Genérica) com a simplicidade da HTML. O XML é um formato para criação de documentos com dados organizados de forma hierárquica.

Uma característica do XML é sua extensibilidade: é possível escrever suas próprias *tags* para descrever o conteúdo de um específico tipo de texto, por exemplo, bem como definir schemas que descrevem a estrutura de um tipo particular de documento XML.

No XML, são especificados em schemas quais *tags* podem ser usadas e onde elas podem ocorrer. Assim, diz-se que todo documento XML que segue essas especificações está conforme o determinado *schema*. Além disso, XML não inclui instruções de formatação/visualização. Assim, mantendo os dados separados das instruções de apresentação, podem-se expor os mesmos dados de diferentes maneiras.

2.5.2 RDF/RDF-Schema

O *Resource Description Framework* (RDF) é um padrão recomendado pelo W3C para descrever recursos da Web, desde fevereiro de 2004. Ele foi projetado para ser lido e entendido por computadores, é escrito XML e não foi construído para ser visualizado pelas pessoas.

RDF (KLYNE; CARROLL, 2004) identifica coisas usando identificadores Web (URIs), e descreve recursos com propriedades e valores das propriedades.

Um modelo RDF possui os seguintes objetos:

- Recursos: é qualquer coisa que tem uma URI, por exemplo, "http://www.w3schools.com/rdf"
- Propriedades: é um Recurso que tem um nome, por exemplo "homepage"
- Literais: é o valor da propriedade, por exemplo "http://www.w3schools.com"
- Declaração: é a declaração de um recurso mais as propriedades desse recurso e o valor dessas propriedades.

A combinação de um Recurso, uma Propriedade e um Valor de Propriedade formam uma Declaração, também conhecido como sujeito, predicado e objeto.

Para a visualização da representação do RDF, podem-se utilizar grafos, a figura 2.8 mostra como é feita essa representação.



Figura 2.8: Representação RDF em Grafo

O RDF-Schema ou RDF-S estende as capacidades do RDF adicionando alguns tipos de restrições, permitindo validar valores de propriedades. Desta forma, o RDF-S adiciona um vocabulário para descrever classes e objetos, as suas propriedades e o tipo de dados dessas propriedades. As propriedades mais utilizadas pela linguagem são: *rdf:subClassOf* que representa um relacionamento hierárquico e a propriedade *rdf:type* que indica as classes das quais um recurso é instância.

É possível a criação de ontologias simples em RDF-Schema, devido sua expressividade limitada a hierarquias de subclasses e propriedades com definições de domínio e faixa de valores para estas. Ontologias mais complexas não são possíveis.

2.5.3 OWL

Web Ontology Language (OWL) é construída no topo da RDF, utilizada para processamento de informações da Web, para ser interpretada por computadores, e não para ser lida por pessoas. OWL (BECHHOFER et al., 2004) é escrita em XML, e é um padrão recomendado pelo W3C.

A OWL deriva das linguagens OIL (*Ontology Inference Layer*) (FENSEL; HORROCKS; VAN HARMELEN, 2000) e DAML (*DARPA Agent Markup Language*). A OIL foi a primeira linguagem direcionada à Web Semântica, sendo unida à linguagem DAML+OIL.

A linguagem OWL possui as características do RDF e um vocabulário maior que a DAML+OIL, oferecendo mais recursos. A linguagem OWL foi construída para ser utilizada por aplicações que necessitam realizar o processamento do significado das informações antes de apresentá-las aos usuários. Seu vocabulário permite a descrição de classes e propriedades, relacionamentos entre as classes, cardinalidade, igualdade, tipos e características de propriedades, entre outras funcionalidades.

Os recursos que a linguagem OWL oferece são divididos em três sub-linguagens:

- **OWL Lite:** fornece uma hierarquia de classificação e funcionalidades de restrições simples. Por exemplo: a OWL Lite suporta cardinalidade, mas só permite os valores 0 e 1. Ela se torna mais fácil de ser implementada em uma ferramenta e faz com que a transição de outros modelos de vocabulários e taxonomias para OWL seja mais rápida.
- **OWL DL:** fornece uma maior expressividade e, ao mesmo tempo, seus sistemas mantêm a completude (garantia que todas as conclusões serão executadas) e decidibilidade (todos os cálculos terminarão em tempo finito) do sistema. A OWL DL inclui todos os artefatos da linguagem OWL, mas impõem restrições quanto à sua utilização.
- **OWL Full:** fornece a máxima expressividade e liberdade sintática do RDF. A OWL Full permite que uma ontologia aumente o significado do vocabulário (RDF ou

OWL) predefinido. Não é esperado que nenhum software suporte todas as características da OWL Full.

OWL Full pode ser vista como uma extensão da RDF, enquanto OWL Lite e OWL DL são extensões de uma visão delimitada de RDF. Portanto, todo documento OWL é um documento RDF e todo documento RDF é um documento OWL Full. Mas somente alguns documentos RDF serão válidos em OWL Lite ou OWL DL.

OWL faz parte da "Visão de Web Semântica" (BERNERS-LEE; HENDLER; LASSILA, 2001), pois a informação disponibilizada na Web possui significado exato, pode ser processada por computadores, e computadores podem integrar a informação da web.

OWL difere de RDF por conter uma linguagem mais completa com maior capacidade de interpretação. OWL contém um grande vocabulário e sintaxe mais completa que RDF.

2.5.4 Protégé

Protégé (BIOMEDICAL INFORMATICS RESEARCH, 2009) é uma plataforma de código aberto, livre que disponibiliza um conjunto de ferramentas para construir modelos e aplicações baseadas em conhecimento com ontologias. Protégé implementa um conjunto de estruturas e ações que suportam criação, visualização e manipulação de ontologias em vários formatos de representação.

A plataforma Protégé suporta duas principais formas de modelar ontologias:

- Protégé-Frames editor: habilita usuários a construir e instanciar ontologias que são baseadas em frames. Neste modelo, uma ontologia consiste de um conjunto de classes organizadas em uma hierarquia para representar um domínio de conceitos, um conjunto de *slots* associados às classes descrevem suas propriedades e relacionamentos, em um conjunto de instâncias destas classes;
- Protégé-OWL editor: habilita usuários a construir ontologias, em particular em OWL. Uma ontologia OWL pode incluir descrições de classes, propriedades e suas instâncias.

2.5.5 SPARQL

SPARQL (SEABORNE, 2008) é uma linguagem de consulta e protocolo de acesso a dados em RDF, recomendada pela W3C. Seu nome é um acrônimo recursivo que significa *SPARQL Protocol and RDF Query Language*.

A especificação SPARQL funciona com outras tecnologias de web semântica da W3C. Entre elas, está a RDF, para representar dados; a RDF-Schema; a *Web Ontology Language* (OWL), para construir vocabulários; e a *Gleaning Resource Descriptions Dialects of Languages* (GRDDL), para extração automática de dados semânticos de documentos.

Outros padrões, como o WSDL (*Web Service Definition Language*), também podem ser usados pela SPARQL.

O ARQ (ARQ - A SPARQL PROCESSOR FOR JENA, 2010) é um “motor de consultas” (*query engine*) que fornece suporte à linguagem SPARQL para o framework Jena (DICKINSON, 2008).

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. 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 sequência 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 opcional, visto que a consulta é realizada sobre a ontologia ou modelo no qual se está trabalhando.

Apesar do SPARQL ter sido projetado apenas para ser uma linguagem de consulta e recuperação de dados, já existem alguns trabalhos como em (SEABORNE; MANJUNATH, 2008) que estendem SPARQL tornando-o uma linguagem também de atualização de dados.

2.5.6 API Jena

Jena (DICKINSON, 2008) é um Framework Java, de código aberto, desenvolvido por Brian McBride da *HP Labs Semantic Web Programme*, para desenvolvimento de aplicações com suporte à Web Semântica. A API Jena suporta as linguagens RDF, RDFS, OWL e SPARQL, incluindo uma máquina de inferência baseada em regras.

O Framework Jena inclui:

- API para RDF;
- leitura e escrita RDF em RDF/XML, N3 e N-Triples;
- API para OWL;
- armazenamento em memória ou de forma persistente;
- máquina de pesquisa SPARQL.

A API Jena permite criar e manipular grafos RDF, representados pelos recursos, propriedades e literais, formando tuplas que darão origem aos objetos criados pelo java. Esse conjunto de objetos é usualmente chamado de *model*. O *Model* é o conjunto de

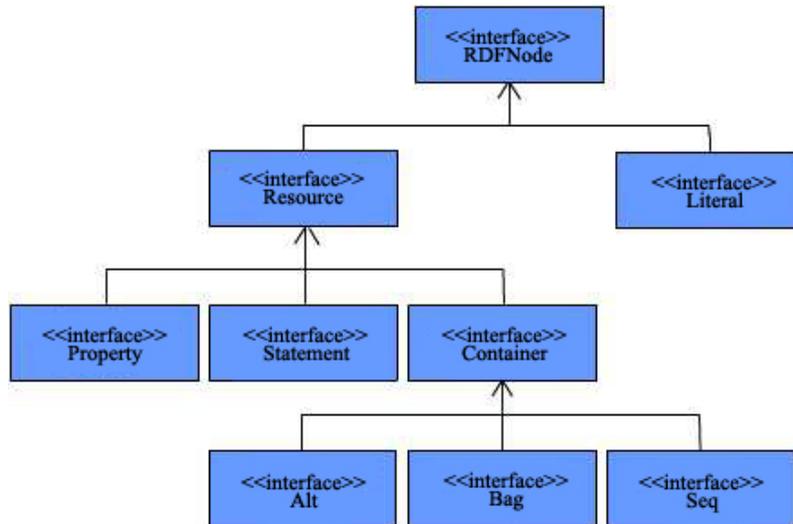


Figura 2.9: Hierarquia de Interfaces da API Jena (VERZULLI, 2001)

declarações que forma o grafo completo. Essa relação é demonstrada por (VERZULLI, 2001) na figura 2.9.

A arquitetura base da API Jena, figura 2.10, é composta por três camadas:

- *Ontology Model*: contém todas as classes necessárias para trabalhar com ontologias descritas em OWL, DAML, OIL ou RDFS. Neste módulo a classe mais relevante é a *OntModel* que representa um modelo ontológico. Essa classe, no entanto, é uma interface;
- *Graph Interface Reasoner*: permite fazer inferências sobre modelos OWL. O uso das inferências sobre modelos semânticos é permitir obter informação adicional (inferida) sobre as ontologias;
- *Graph Interface Base RDF*: OWL definida sobre RDFs. O Jena usa a API de RDF para manipular as ontologias.

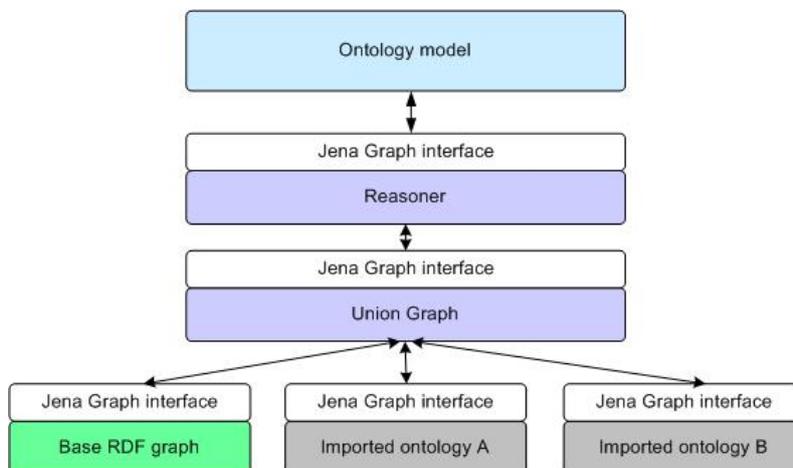


Figura 2.10: Camadas da API Jena (DICKINSON, 2008)

As interfaces para programação são descritas da seguinte maneira:

- **RDFNode**: interface que possui a função de fazer a ligação de todos os outros elementos do RDF, a fim de formar as triplas para a criação dos elementos;
- **Resource**: interface que representa objetos que possuam um URI;
- **Literal**: interface que representa valores usados como objetos em triplas (sujeito, predicado, objeto). A interface **Literal** possui métodos para converter valores para vários tipos de dados em Java, como **String**, **int** e **double**;
- **Property**: interface que representa as propriedades em uma tripla;
- **Statement**: interface que representa uma tripla;
- **Container**: interface que representa um conjunto de objetos (**Alt**, **Bag**, **Seq**) em uma tripla.

2.5.7 Raciocinadores

Os *reasoners* (raciocinadores) são mecanismos computacionais criados para realizar inferências lógicas a partir de um conjunto de fatos ou axiomas, norteiam-se pelas regras pré-definidas na ontologia, descobrindo inconsistências, dependências escondidas e eventuais redundâncias. Através dos *reasoners* é possível que novos conhecimentos, além dos contidos explicitamente nas ontologias, sejam agregados e apresentados de forma transparente.

A API Jena fornece alguns motores de inferência e possibilita a criação de novos motores quando necessário, ou a possibilidade de estender os já existentes. Abaixo é exibida uma breve descrição dos *reasoners* disponibilizados pelo Jena (REYNOLDS, 2009):

- *Transitive Reasoner*: disponibiliza suporte para armazenar e percorrer classes e propriedades ligadas. Implementa apenas as propriedades transitivas e simétricas de `rdfs:subPropertyOf` e de `rdfs:subClassOf`;
- *RDFS Rule Reasoner*: implementa um subconjunto configurável das implicações RDFS;
- *OWL, OWL Mini, OWL Micro Reasoners*: implementação incompleta da linguagem OWL-Lite;
- *DAML Micro Reasoner*: usada internamente para viabilizar o uso da API legada de DAML, fornece uma capacidade mínima de inferência;
- *Generic Rule Reasoner*: raciocinador baseado em regras que suporta a criação de regras definidas pelo usuário. Suporta encadeamento para frente (*forward chaining*), encadeamento para trás (*tabled backward chaining*) e estratégias de execução híbridas.

O *Generic Rule Reasoner* (GRR) é o mais independente dos motores de inferência da API Jena, por isso terá uma descrição mais detalhada. Ele foi idealizado tanto para implementar o reasoner RDFS quanto o OWL, mas também possibilita que o programador importe as regras dos outros *reasoners* existentes. Dessa maneira ele se torna o reasoner

mais abrangente do Jena. Uma regra é definida como uma instância da classe *Rule* que contém uma lista de premissas e uma lista de conclusões sobre as mesmas, opcionalmente a regra definida possui um nome e um sentido. Uma premissa ou uma conclusão pode ser uma tripla, uma tripla estendida ou uma chamada a procedimento primitivo (chamado *builtin*). O Jena (REYNOLDS, 2009) também disponibiliza um *parser* para checar a legalidade das regras definidas seguindo a sintaxe original, mas também permite que um outro *parser* seja definido pelo usuário para se obter um melhor diagnóstico dos erros encontrados, já que o *parser* disponibilizado pelo Jena não se demonstra muito eficiente nesse diagnóstico.

Como mencionado anteriormente, o GRR possui suporte a três tipos de mecanismos de ativação de regras:

- *Forward Chaining Engine* (Para frente): no momento em que o modelo de inferência recebe a primeira consulta, todos os dados relevantes do modelo são enviados para o mecanismo de regras. Quando uma regra causa a criação de triplas extras, novas regras podem ser disparadas. Nesse momento, se as regras não forem bem definidas, pode acontecer um loop infinito. Cada vez que são criadas ou removidas triplas do modelo pelos próprios métodos da API, as regras podem ser ativadas. A inferência acaba quando as regras pararem de ser ativadas. O algoritmo utilizado por este motor de inferência trabalha de forma incremental;
- *Backward Chaining Engine* (Para trás): no modo para trás o *reasoner* usa uma estratégia de execução parecida com o mecanismo do Prolog. No momento em que o modelo de inferência recebe uma consulta, ele a transforma em um objetivo, e o motor de inferência aplica as regras de modo a tentar atingir esse objetivo, unindo as triplas armazenadas com as regras de *backward chaining*. Nesse caso, o motor de inferência não trabalha incrementalmente, isto é, sempre que os dados originais forem alterados, todo o processamento realizado é perdido;
- *Hybrid* (híbrido): utiliza os dois mecanismos acima de forma conjunta. O mecanismo para frente executa primeiro e guarda um conjunto de deduções. Caso uma regra para frente crie novas regras para trás, ela vai instanciá-la de acordo com as variáveis guardadas nas deduções e depois irá passar as regras instanciadas para o mecanismo LP (*logic programming*) para trás. Todas as consultas são resolvidas posteriormente pelo LP *engine* usando a mistura dos dados brutos e das deduções que vieram do mecanismo para frente. O LP *engine* tem seu funcionamento semelhante à linguagem Prolog.

2.6 Considerações Sobre o Capítulo

Este capítulo apresentou conceitos sobre computação ubíqua, a arquitetura e os subsistemas do *middleware* EXEHDA, fundamentos e requisitos no processo de descoberta de recursos, e algumas tecnologias necessárias para a criação e manipulação de ontologias. Essas tecnologias possibilitam um entendimento semântico entre máquinas, estendendo a capacidade de reconhecer conceitos e termos que não estão explicitamente definidos. No próximo capítulo são apresentados trabalhos relacionados sobre descoberta de recursos.

3 TRABALHOS EM DESCOBERTA DE RECURSOS

Neste capítulo estão relacionados alguns mecanismos de descoberta de recursos e a forma que estes mecanismos utilizam para armazenar e pesquisar pelos recursos descobertos. Esses mecanismos foram agrupados de acordo com a forma que utilizam para a realização de consultas por recursos, ou seja, baseados em sintaxe ou semântica.

3.1 Baseados em Sintaxe

Esta seção apresenta mecanismos de descoberta de recursos que realizam o *matching*, ou seja, a consulta por recursos utilizando a comparação exata de atributos. Alguns utilizam operadores lógicos para expandir um pouco os resultados desejados. Esses mecanismos utilizam uma sintaxe bem definida, mas não utilizam nenhum tipo de semântica na descrição e consulta por recursos.

3.1.1 INS/TWINE

O INS/Twine (BALAZINSKA; BALAKRISHNAN; KARGER, 2002) é um serviço para descoberta de recursos de forma escalável no qual entidades chamadas *resolvers* colaboram como *peers* para distribuir as informações relacionadas aos recursos e responder consultas, conforme figura 3.1.

O INS/Twine não possui expressividade para especificação de consultas. Seu esquema de representação de consultas é definido por um documento XML, mas ele realiza apenas combinações exatas de atributos e valores.

3.1.2 UPnP

O *Universal Plug-and-Play* (UPnP FORUM, 2008) possui uma arquitetura para conexão de dispositivos através de uma rede *peer-to-peer*. Através do UPnP o dispositivo pode juntar-se a uma rede de forma dinâmica, obter um endereço IP, anunciar e descobrir recursos. Utiliza o protocolo SSDP, *Simple Service Discovery Protocol*, para descoberta de recursos. O dispositivo envia uma mensagem de anúncio (*ssdp:alive*) *multicast* dos seus serviços para os possíveis clientes do serviço. O SSDP envia mensagem de busca (*ssdp:discover*) *multicast* quando um novo dispositivo é adicionado à rede, todos os dispositivos que escutarem a mensagem devem respondê-la através de *unicast* para quem enviou a consulta. O UPnP utiliza XML para descrever as características do dispositivo.

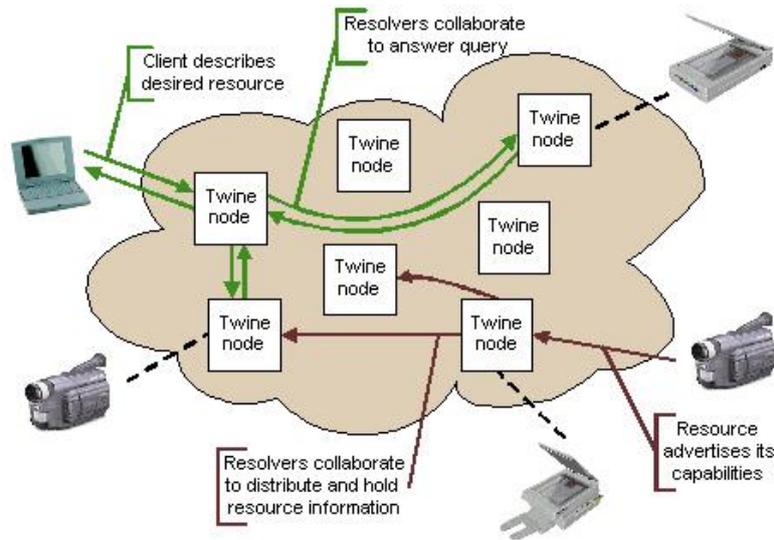


Figura 3.1: Arquitetura INS/Twine (BALAZINSKA; BALAKRISHNAN; KARGER, 2002)

A mensagem de anúncio contém a URL de um XML que permite a descrição precisa das características do dispositivo UPnP.

3.1.3 Allia

O Allia é um *framework* para descoberta de recursos em ambientes *ad-hoc* desenvolvido por pesquisadores da University of Maryland Baltimore County (RATSIMOR et al., 2002). Ele é baseado em agentes e governado por políticas, utilizando *caching* em uma rede *peer-to-peer*.

Cada dispositivo no ambiente é autossuficiente. Entretanto, utiliza recursos/serviços da vizinhança, sempre que estiverem disponíveis. Alterações na topologia de rede são automaticamente refletidas nas alianças formadas. Um nodo não precisa registrar ou remover seu registro nas alianças dos vizinhos, quando muda sua localização.

Não é especificado nenhum mecanismo para descrições durante o processo de descoberta de recursos. Recursos podem seguir a especificação Jini e serem descritos e acessados através de *proxies* Jini.

3.1.4 Jini

O Jini é uma arquitetura para descoberta de recursos baseada em Java (JINI, 2009), e desenvolvida pela Sun Microsystems. Utiliza mensagens *multicast* para localizar *Lookup Service* e registra os recursos no *Lookup Service* através do procedimento *Join*. Quando o cliente não encontra o *Lookup Service* ele faz a requisição diretamente ao recurso (*Peer-lookup*), figuras 3.2 e 3.3.

Dispositivos e aplicações se registram com a rede Jini utilizando um processo chamado Descoberta e Adesão (*Discovery and Join*). Para se unir a uma rede Jini, um dispositivo ou aplicação se registra na Tabela de Consulta (*Lookup Table*) de um Serviço de Consulta (*Lookup Service - LS*) que é a base de dados para todos os serviços na rede (similar ao DA no SLP). Além de ponteiros para os serviços, a Tabela de Consulta de Jini

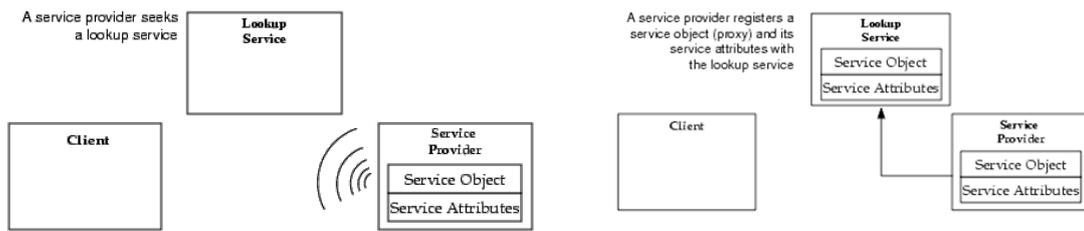


Figura 3.2: Jini - Descoberta e Join (JINI, 2009)

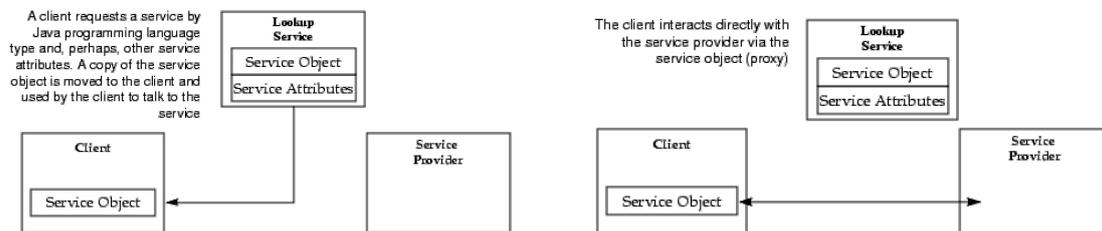


Figura 3.3: Jini - Lookup e Cliente (JINI, 2009)

pode armazenar trechos de código Java associados a esses serviços. Isso significa que serviços podem armazenar *drivers* para os dispositivos, uma interface e outros programas que ajudem o usuário a ter acesso aos serviços. Quando um cliente quer utilizar o serviço, o código objeto é carregado da Tabela de Consulta para a máquina virtual do cliente. Enquanto uma consulta de serviço em SLP devolve uma URL de Serviço, o código objeto Jini oferece acesso direto ao serviço usando uma interface conhecida pelo cliente. Essa mobilidade de código substitui a necessidade de pré-instalar *drivers* para o cliente.

A requisição e a seleção devem ser implementadas como objetos Java, seguindo as interfaces Jini específicas. Quando o serviço é localizado, Jini fornece um *stub* Java RMI para acessá-lo. O protocolo Jini é baseado em licenças (*leases*), dessa forma todos os anúncios e registros são considerados válidos apenas por um período de tempo específico e relativamente curto. Clientes e serviços que são executados por muito tempo devem renovar suas licenças periodicamente, assim entidades que falham são removidas automaticamente de todos os serviços de busca quando a licença expira. Além disso, a renovação periódica de licenças pelas entidades é capaz de reconstruir o estado global no caso do travamento de um LS. Jini não suporta operação sem diretório, e não pode interoperar com nenhum outro protocolo ou linguagem. Uma vez que o protocolo depende do uso de stubs Java, um dispositivo deve implementar uma máquina virtual Java (JVM) ou usar um *proxy*. Esses requisitos fazem com que Jini não seja ideal para a utilização em pequenos dispositivos.

A expressividade na representação de propriedades de recursos é limitada devido a ser baseada no processo de filtragem nos valores de atributos de uma interface Java

3.1.5 SLP

O SLP (*Service Location Protocol*) é um protocolo desenvolvido dentro do Internet Engineering Task Force (IETF), e é uma especificação independente de linguagem. O protocolo define três tipos de agentes e as mensagens de requisição e resposta trocadas entre eles (GROUP, 1999). SLP é descentralizado, leve, escalável e extensível para ser utilizado dentro de uma organização. Ele permite, mas não obriga uma administração centralizada. Sua arquitetura é constituída de três componentes principais: Agentes do Usuário (UA) que executam a descoberta do serviço em nome do cliente; Agentes de Serviço (SA) que anunciam a localização e características dos serviços, em nome dos próprios serviços e Agentes de Diretório (DA) que coletam endereços de serviços e informações recebidas dos SAs em suas bases de dados e respondem às requisições de serviços dos UAs. Quando um novo serviço se conecta à rede, o SA se comunica com o DA para anunciar sua existência (registro de serviço). Quando um usuário precisa de um certo serviço, o UA consulta os serviços disponíveis na rede a partir do DA (solicitação de serviço). Depois de receber o endereço e as características do serviço desejado, o usuário pode finalmente utilizar o serviço. Antes que o cliente (UA ou SA) possa se comunicar com o DA, ele deve descobrir a existência do DA. Há três métodos diferentes para a descoberta do DA: estático, ativo e passivo. Na descoberta estática, os agentes do SLP obtêm o endereço do DA através de DHCP (*Dynamic Host Configuration Protocol*). Servidores DHCP distribuem os endereços dos DAs para as aplicações que desejarem. Na descoberta ativa, UAs e SAs enviam solicitações de serviço para o endereço de grupo *multicast* (239.255.255.253). Um DA ouvindo neste endereço vai receber essas solicitações em algum momento e respondê-las diretamente ao agente solicitante. No caso de descoberta passiva, DAs periodicamente enviam anúncios *multicast* para seus serviços. UAs e SAs ficam sabendo o endereço do DA a partir desses anúncios e podem se comunicar diretamente com ele.

A linguagem de consulta provida pelo SLP é mais poderosa que a usada por Jini e UPnP, porém é baseada em uma sintaxe própria (baseada em string e não em XML), o que dificulta uma possível interoperabilidade com outras especificações.

3.1.6 Globus Toolkit

O Globus Toolkit é um software livre utilizado para construir sistemas e aplicações em grade (TOOLKIT, 2009). Ele está sendo desenvolvido pela Globus Alliance. O Globus possui um serviço chamado MDS (*Monitoring & Discovery System*) responsável por descobrir os recursos disponíveis e seu status. Adota o padrão LDAP para representação e procura (*query*) de recursos. Sua estrutura é descentralizada e permite escalabilidade. O MDS possui uma estrutura hierárquica que consiste de três componentes principais, conforme figura 3.4:

- *Information Providers (IPs)*: coletam os dados do recurso e se comunicam com o GRIS (sensores);
- *Grid Resource Information Service (GRIS)*: executa no recurso e atua como um gateway de informações para esse recurso;
- *Grid Index Information Service (GIIS)*: provê um diretório agregado (indexador) de dados que facilita a descoberta e monitoração.

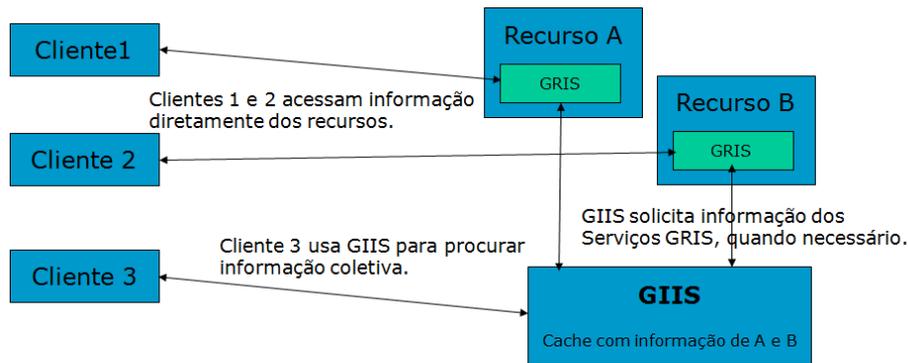


Figura 3.4: Globus Toolkit (TOOLKIT, 2009)

3.1.7 PerDis

O PerDis, *Pervasive Discovery Service* (SCHAEFFER, 2005), é um serviço proposto para o EXEHDA responsável pela descoberta de recursos que interage com os serviços de reconhecimento e adaptação de contexto do *middleware*. Esta interação incorpora ao mecanismo de descoberta o tratamento de informações relacionadas ao contexto dos usuários e recursos. A arquitetura do PerDis, figura 3.5, atende aos requisitos necessários a uma estratégia para descoberta de recursos na computação pervasiva. Esses requisitos abordam os seguintes aspectos:

- utilização de informações do contexto de execução;
- utilização de estratégias para manutenção automática de consistência;
- expressividade (parcial) na descrição de recursos e critérios de pesquisa;
- possibilidade de interoperabilidade com outras estratégias de descoberta;
- suporte à descoberta de recursos em larga-escala;
- utilização de preferências por usuário.

Os componentes da arquitetura do PerDis são os seguintes:

- *Resource Component (RC)*: responsável por informar as características relacionadas a um determinado recurso que permitam a um usuário optar ou não por sua utilização. É composto pelos módulos *Resource descriptor*, responsável por manter as propriedades descritivas do recurso e *Lease renewer* que realiza o gerenciamento da renovação do *lease* utilizado para indicar periodicamente a disponibilidade do recurso;
- *User Component (UC)*: habilita o usuário a realizar o processo de descoberta, interagindo com os demais componentes da arquitetura. Seus módulos são: *Query builder* - interface para montagem de consultas a recursos e *Result Browser* - permite ao usuário inspecionar os recursos retornados em uma consulta;

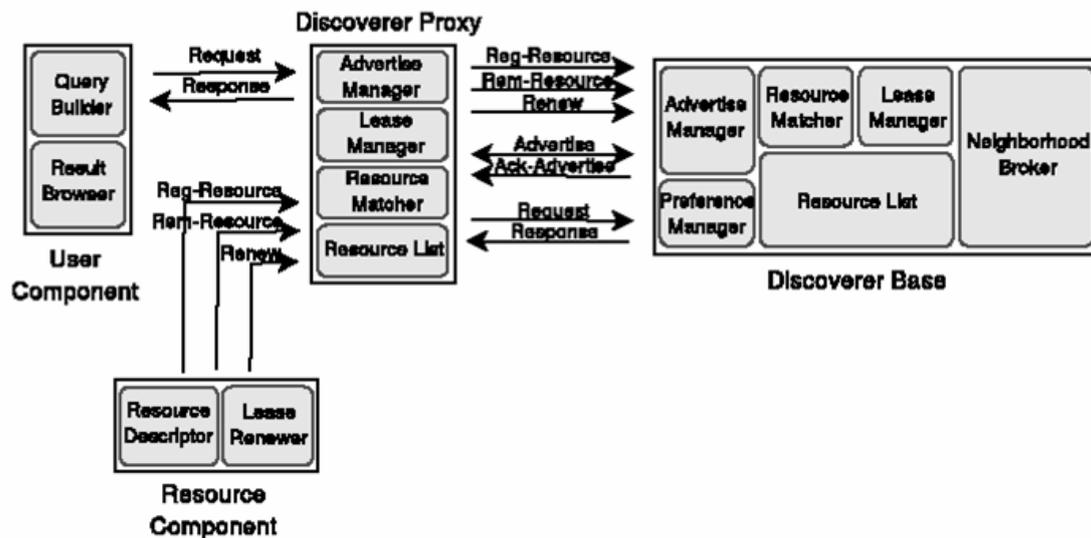


Figura 3.5: Arquitetura PerDis (SCHAEFFER, 2005)

- *Discoverer Base (DB) e Discoverer Proxy (DP)*: Atuam como catálogo de recursos, abrangendo os recursos disponibilizados em uma célula de execução. Os principais módulos comuns a essas duas entidades são:
 - *Advertise manager*: realiza anúncio *multicast* da instância do serviço de descoberta no momento em que este for disparado, a fim de que a localização e a comunicação entre as instâncias desses componentes possam ser feitas de forma automática;
 - *Resource List*: mantém a lista de recursos gerenciados pela instância do serviço de descoberta;
 - *Lease Manager*: responsável pelo gerenciamento do controle de *lease*, realizando o descarte de recursos que não tenham sido renovados a um determinado período de tempo;
 - *Resource Matcher*: realiza a comparação de critérios de pesquisa.

O *discoverer base* ainda possui alguns módulos adicionais não presentes em *discoverer Proxy*:

- *Preference manager*: aplica preferências de usuários;
- *Neighborhood broker*: usado na comunicação *peer-to-peer* entre células que compõem o ambiente pervasivo.

3.1.8 Condor

O Projeto Condor (THAIN; TANNENBAUM; LIVNY, 2005) nasceu na Universidade de Wisconsin, com objetivo de criar um ambiente capaz de prover grande poder computacional a médio e longo prazo. Para isso, utiliza-se de recursos computacionais ociosos de equipamentos não dedicados, apesar de ser possível utilizá-lo em *clusters* dedicados.

O Condor introduziu o conceito de *High-Throughput Computing*, ou seja, Computação de Alta Vazão.

Em muitos projetos de pesquisa ou de engenharia é comum a necessidade de execução de programas que necessitem de semanas ou meses de execução para serem concluídos. Para esse tipo de aplicação que o Condor foi desenvolvido.

O funcionamento do Condor baseia-se na execução de *jobs*. O usuário submete um *job* ao sistema. Condor, por sua vez, encontra os recursos disponíveis para a execução. O sistema Condor monitora continuamente a execução dos *jobs*. Quando uma máquina executando um *job* torna-se indisponível (por exemplo, pelo retorno do usuário), Condor pode realizar um *checkpoint* do *job* e migrá-lo para uma outra máquina que esteja disponível. Dessa forma, a execução na outra máquina poderá prosseguir do ponto em que havia parado.

O usuário Condor tem a ilusão de que as tarefas remotas são executadas localmente. De forma a tornar possível o acesso à maior quantidade possível de máquinas, Condor não assume que as máquinas para a execução remota montam os mesmos sistemas de arquivos da máquina do usuário. Para solucionar esta questão, Condor realiza o redirecionamento das chamadas de sistema da máquina que está executando o *job* para a máquina base do usuário. Condor provê uma biblioteca de redirecionamento para esta função.

Organização Básica:

- Sistema organizado em aglomerados;
- Um aglomerado, tipicamente, compreende as máquinas em uma rede local;
- O aglomerado é composto por três tipos de nós, conforme a figura 3.6, (fornecedor de recursos, consumidor de recursos e gerenciador do aglomerado).

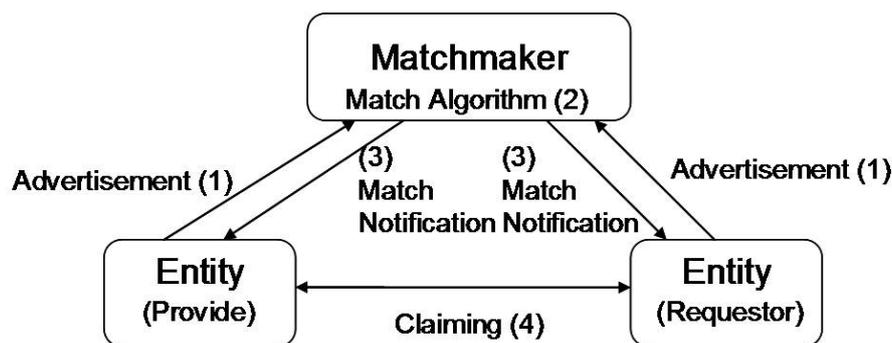


Figura 3.6: Arquitetura Condor (THAIN; TANNENBAUM; LIVNY, 2005)

As informações são disseminadas através da linguagem ClassAd e as entradas são do tipo atributo e valor.

3.1.9 Salutation

Salutation é formado por um consórcio aberto de indústrias, chamado Consórcio Salutation, é responsável pelo desenvolvimento de sua arquitetura que define um modelo abstrato com três componentes: Cliente, Servidor e Salutation Manager (SLM). O Salutation Manager gerencia toda a comunicação e faz a ponte através de diferentes meios

de comunicação. Serviços registram suas capacidades com um SLM, e clientes consultam o SLM quando necessitam de um serviço. Depois de descobrir um serviço desejado, clientes podem solicitar a utilização do serviço através do SLM. Salutation define seu protocolo baseado no SunRPC. O modelo pode ser implementado com ou sem um serviço de diretório (SLM) separado, e os diretórios podem ser organizados como uma hierarquia ou algum outro grafo de diretórios cooperativos. Quando implementado sem um diretório, clientes e serviços podem localizar uns aos outros diretamente, através de *broadcast* local. Essa configuração sem diretório permite ao Salutation trabalhar corretamente em uma rede sem administração, como por exemplo uma rede residencial ou automotiva. Salutation é uma protocolo bem estabelecido, com algumas implementações comerciais para Windows (95/98 and NT) e previsões de implementações adicionais para Palm OS e Windows CE planejadas para um futuro próximo. Exceto pelo SunRPC, Salutation é composto por tecnologias neutras. Além disso, Salutation foi projetado para ser compatível com tecnologias sem fio, e já existem adaptações de Salutation para IrDA e Bluetooth.

3.2 Baseados em Semântica

Esta seção apresenta mecanismos de descoberta de recursos que realizam *matching*, ou seja, a consulta por recursos utilizando mecanismos semânticos. Através da semântica é possível obter resultados mais expressivos, pois o mecanismo pode obter entendimento de conceitos que não estavam declarados explicitamente.

3.2.1 OMM (Ontology-Based MatchMaker)

Ontology-Based MatchMaker (TANGMUNARUNKIT; DECKER; KESSELMAN, 2003) é um serviço de *matchmaking* em ambiente de grade. Este projeto utiliza tecnologias de Web Semântica para solucionar o problema de *matching* de recursos no ambiente distribuído. Foi desenvolvido um protótipo de um seletor de recursos baseado em ontologias que utiliza, além de ontologias, bases de conhecimento e regras para solucionar o *matching* de recursos no ambiente distribuído.

No OMM são criadas ontologias separadas para a descrição de recursos e suas requisições, o que permite ao cliente do recurso não conhecer como o recurso foi descrito exatamente, sendo realizado o *match* semântico através dos termos definidos nestas ontologias.

Principais características do OMM:

- descrição assimétrica de recursos e requisições: a descrição do recurso e a requisição são modelados separadamente, mas é utilizado o *match* semântico para a associação entre eles;
- compartilhamento e manutenção de ontologias;
- restrições bilaterais através de utilização de políticas de uso;
- habilidade em especificar preferências de *matching*;
- verificação de integridade através de regras de restrições;
- expressividade na descrição das requisições;

- flexibilidade e extensibilidade.

A arquitetura do OMM é formada por três módulos:

- *Matchmaking Module*: consultas são realizadas através da linguagem TRIPLE e o sistema de inferência escrito em java e C, baseado em Jini. Este módulo realiza *matchmaking* e retorna o resultado como um objeto java contendo a lista de recursos que satisfazem a consulta;
- *Request Handling*: um serviço persistente é utilizado para suportar vários clientes simultaneamente. Clientes submetem requisições em RDF e recebem uma lista de recursos que satisfazem a consulta. Usuários podem expressar suas preferências para recursos selecionados utilizando uma função de *ranking*. Clientes podem indicar o número de recursos retornados que irão ser ordenados baseados em seus valores de *ranking*;
- *Resource Discovery*: é o módulo chave do OMM, responsável por coletar dinamicamente informação dos recursos de várias fontes, transformando a informação para a ontologia e atualizando a base de conhecimento.

3.2.2 A Grid Service Discovery Matchmaker Based on Ontology Description

No artigo de (LUDWIG; SANTEN, 2002) é proposto um *framework* para o *matching* na descoberta de serviços para um ambiente de grade, baseado em ontologias.

A justificativa apresentada no trabalho para utilização do *matching* semântico é o fato de que a maioria dos mecanismos de descoberta realizam o processo de descoberta através de comparação baseada em *string* e inteiros.

A ontologia é utilizada para que o provedor dos serviços e o usuário deste serviço possam compartilhar um entendimento comum das capacidades do serviço. Esta ontologia foi desenvolvida com a linguagem DAML-S.

O *framework* desenvolvido é baseado no *matchmaker* LARKS. O mecanismo de *matching* é composto de três estágios de filtragem que são: contexto, sintático e semântico aos quais a ontologia fornece a base do conhecimento. A diferença em relação ao LARKS é o filtro semântico.

Três componentes são necessários para o SD *matching*: o provedor de recursos, o solicitador de serviço e o serviço *matchmaker*. O provedor de serviços registra a descrição de seus serviços no serviço *matchmaker*. O solicitador de serviços requisita um serviço e envia uma requisição ao serviço *matchmaker*. O serviço *matchmaker* retorna os resultados ao solicitador de serviço. O solicitador de serviço decide qual serviço utilizar de acordo com a necessidade do cliente.

O *matchmaker* processa a requisição seguindo os seguintes passos:

- comparando a requisição com todos os anúncios registrados no banco de dados;
- decidindo qual provedor de serviço tem o melhor *matching* com relação à requisição, dependendo do algoritmo e da ontologia definida;
- provendo as informações ao requisitante.

São utilizados filtros em três estágios:

- *Matching* de Contexto: seleciona os anúncios no banco de dados auxiliar tal que possam ser comparados à requisição em contexto igual ou similar;
- *Matching* Sintático: compara a requisição com os anúncios selecionados no estágio anterior em três etapas que são: a comparação de perfis, *matching* de similaridade e *matching* de assinatura. O serviço de ontologia provê o modelo de serviço e o fundamento do serviço para a comparação de perfis. Os outros estágios focam na entrada e saída de restrições e declarações na especificação;
- *Matching* Semântico: checa se as restrições de entrada e saída dos pares de requisição e os anúncios se emparelham logicamente.

3.2.3 Ontologias Aplicadas à Descrição de Recursos em Ambientes Grid

No trabalho de (PERNAS; DANTAS, 2004), foi desenvolvida uma ontologia com o objetivo de facilitar a busca e seleção de recursos. Desta forma, é possível descrever os recursos disponíveis em um ambiente de grade de forma sintática e semântica através de um vocabulário comum ao domínio. No desenvolvimento desta ontologia, foi definido primeiramente o domínio da aplicação, em seguida este domínio foi transformado em um vocabulário e, finalmente, foram criados os axiomas que guardam todas as regras a serem respeitadas durante a inclusão de novo vocabulário na ontologia. A ontologia foi desenvolvida utilizando a linguagem OWL. Também foi desenvolvido um serviço de grade de modo a externalizar a ontologia aos usuários do ambiente de grade.

Na arquitetura proposta, figura 3.7, a ontologia localiza-se praticamente em uma camada à parte do ambiente de grade, e as requisições vindas dos usuários devem primeiramente realizar uma consulta à ontologia que, por sua vez, utiliza os metadados e as visões semânticas para obter maior informação a respeito dos recursos.

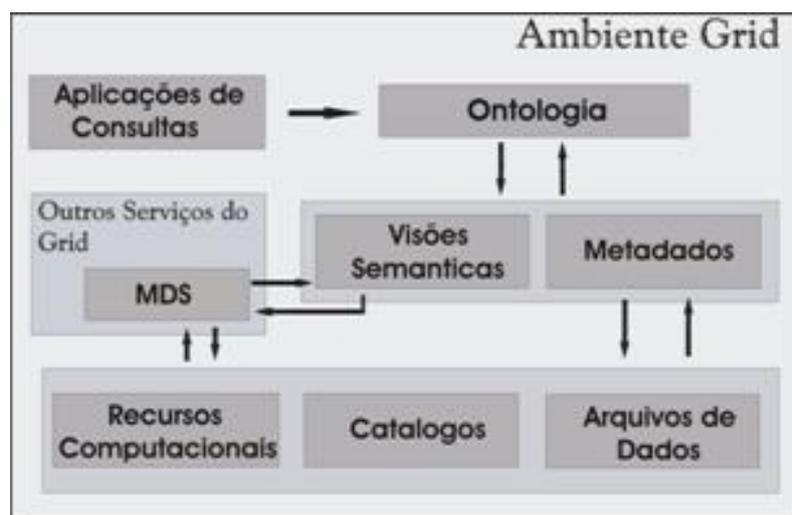


Figura 3.7: Arquitetura de Pernas e Dantas (PERNAS; DANTAS, 2004)

A ontologia proposta foi documentada com os seguintes componentes:

- Dicionário de Dados: agrega todas as classes e instâncias de classe da ontologia, juntamente com suas significações;
- Árvore de Classificação de Conceitos: agrupa todas as classes e subclasses da ontologia;
- Tabelas de Atributos de Classes e de Instâncias: apresentam, respectivamente, para cada instância e para cada classe da ontologia, todos os seus atributos;
- Tabelas de Instâncias: apresentam a descrição, atributos e valores de cada instância da ontologia;
- Árvores de Classificação de Atributos: mostram os atributos inferidos através da existência de outros atributos de hierarquia superior.

3.2.4 Serviço Baseado em Semântica para Descoberta de Recursos em Grade Computacional

O trabalho de (ALLEMAND, 2006) propõe um mecanismo para descoberta de recursos em grade que utiliza tecnologias para converter as descrições dos recursos coletados em XML para um *template* ontológico escrito em OWL. Sua arquitetura é composta por quatro camadas:

- *Fabric*: está diretamente ligada aos recursos computacionais e de rede;
- *Middleware*: nesta camada é provido o acesso aos recursos remotos, sendo que neste trabalho foi utilizado o *Globus Toolkit* e como provedor de informação o *Ganglia*. Os dados sobre os recursos da grade capturados pelo provedor de informações *Ganglia* são armazenados no repositório no formato XML;
- *Conhecimento*: provê serviço de descoberta semântica de recursos armazenados no repositório de recursos. Através da API *Jena* são transferidas as informações dos recursos em XML para o repositório semântico, na linguagem de ontologia OWL. O raciocinador utilizado é o *Pellet*;
- *Aplicação*: localizada no topo da arquitetura, provê o uso dos recursos do ambiente de grade.

Para a edição do *template* ontológico e tratamento dos diversos tipos de recursos computacionais no ambiente de grade foi utilizado o *Protégé-OWL*. O Raciocinador, segundo componente da camada de *Conhecimento*, interage com o *Repositório Semântico* para selecionar os recursos de forma apropriada. Neste trabalho foi utilizado o *middleware* de grade *Globus Toolkit 4 (GT4)* e seu serviço *Monitoring and Discovery System (MDS4)*, juntamente com o *Ganglia*, para coletar automaticamente as informações sobre os recursos da grade computacional. Como motor de inferência empregou-se a ferramenta *Pellet-OWL* por meio da realização de um estudo de caso com uso do protótipo.

A linguagem de consulta de inferência utilizada é a *RDQL* (SEABORNE, 2004).

3.2.5 DReggie

Esse trabalho desenvolvido por (CHAKRABORTY et al., 2001), é um aperfeiçoamento do serviço de descoberta Jini. Ele aperfeiçoa a técnica de descrição de serviços e utiliza raciocinadores no mecanismo de *matching*. Em DReggie os serviços são descritos utilizando a *DARPA Agent Markup Language* (DAML). A ontologia em DAML foi criada para descrever serviços com suas propriedades e atributos. Os serviços de anúncio e requisição utilizam a mesma ontologia. O serviço de *lookup* do Jini é aprimorado com um raciocinador baseado em Prolog que é capaz de *matching* complexos baseados em descrições DAML.

DReggie fornece todas as funções e métodos definidos no core do Jini, além de algumas funcionalidades extras. Modificações e adições a esse pacote inclui:

- adição de uma classe que aceita, como entrada bem formada, uma descrição DAML como argumento;
- adição de uma classe que implementa as funções necessárias para comparar duas instâncias DAML;
- modificação no registro de classes para permitir o DReggie determinar o mecanismo de *matching* apropriado;
- modificação da classe de pesquisa para permitir DReggie encontrar informações sobre o pacote *parser* apropriado.

Processo de descoberta com a utilização do raciocinador, considerando “Servidor” o dispositivo que contém informação sobre os serviços e “Cliente” o dispositivo que solicita o serviço de descoberta:

- servidor inicializa o código, carrega a base de conhecimento e inicializa o motor de raciocínio;
- cliente estabiliza uma conexão ao servidor;
- cliente envia uma mensagem de “requisição de descoberta semântica” ao servidor. A mensagem contém várias informações e a consulta em DAML;
- servidor recebe a mensagem, analisa-a e determina se é uma “requisição de descoberta semântica”;
- no servidor, a função para processar a procura semântica é chamada;
- a função de consulta chama duas funções Prolog para manipular a consulta. A primeira função é usada para converter dados DAML em triplas. A segunda função chama o predicado Prolog no motor de raciocínio para realizar a procura semântica.

Esse mecanismo foi desenvolvido especialmente para aplicações de *m-commerce*, ou seja, aplicações que envolvem a transferência de propriedade ou direitos de utilização de bens e serviços que é iniciado e/ou concluído usando um dispositivo de comunicação móvel.

3.3 Considerações Sobre o Capítulo

Este capítulo apresentou alguns mecanismos de descoberta de recursos, com foco na forma com que realizam a descrição e consulta de recursos.

Os mecanismos foram analisados, verificando-se as seguintes características:

- Descrição: linguagem utilizada para descrever os recursos;
- Consulta: como é realizado o *matching*. Se é utilizada uma linguagem específica ou apenas o casamento de pares (Atributo/Valor);
- Abrangência: qual a abrangência de descoberta do mecanismo;
- Arquitetura: a organização arquitetural da descoberta foi classificada em:
 1. Diretório Centralizado Plano;
 2. Diretório Centralizado Hierárquico;
 3. Diretório Descentralizado (Distribuído e/ou Replicado).
- Preferências: se o mecanismo utiliza algum tipo de preferência do usuário para a realização da consulta ou processamento de resultados.

Tabela 3.1: Comparativo dos Trabalhos Relacionados

	Modelos	Descrição	Consulta	Abrangência	Arquitetura	Pref.
1	Allemand	OWL	RDQL	Grade	3	-
2	Allia	-	-	Ad-hoc	3	Sim
3	Condor	ClassAd	A/V	LAN	-	-
4	DReggie	DAML	Prolog	<i>m-commerce</i>	1	Sim
5	GLOBUS	LDAP	LDAP	Grade	3	-
6	Ludwig, Santen	DAML-S	LARKS	Grade	3	-
7	INS/TWINE	XML	-	LAN	1,2	-
8	Jini	LP A/V	LP	LAN	1,2	Sim
9	OMM	TRIPLE	LP	Grade	3	Sim
10	PerDis	XML	A/V	Ubíquo	3	Sim
11	Pernas, Dantas	OWL	-	Grade	3	-
12	Salutation	<i>templates</i> A/V	A/V	LAN	1,2,3	-
13	SLP	STRING	A/V	LAN	1,3	Sim
14	UPnP	XML	A/V	LAN	3	-

*1-Diretório Centralizado Plano, 2-Centralizado Hierárquico, 3-Descentralizado

Dos 14 trabalhos representados na tabela 6.1 apenas os modelos 1, 4, 6, 9 e 11 utilizam uma linguagem com maior expressividade na descrição dos recursos. Destes, apenas os trabalhos 1 e 11 utilizam a linguagem OWL que possibilita uma maior expressividade na descrição dos recursos.

A grande maioria dos mecanismos utilizam comparações de Atributo/Valor no processo de *matching*. O trabalho 1 utiliza a linguagem de consulta RDQL. A linguagem RDQL é antecessora da SPARQL. A SPARQL tem todas características da RDQL com as seguintes funcionalidades agregadas:

- possibilidade de adicionar informações opcionais nos resultados da consulta;
- disjunção dos padrões de grafos;
- testes mais expressivos (suporte a data/hora, por exemplo)
- nomeação de grafos;
- ordenação.

Os mecanismos 2, 3, 4, 7, 8, 12, 13 e 14 foram projetados para descoberta em pequenas redes. Os trabalhos 1, 9 e 11 foram desenvolvidos para computação em grade tradicional, já o mecanismo de descoberta 10 foi projeto para ambientes de computação ubíqua promovido pelo *middleware* EXEHDA.

A maioria possui uma arquitetura descentralizada, sendo que alguns possibilitam o seu uso também em arquiteturas centralizadas.

Seis trabalhos consideram as preferências do usuário.

O próximo capítulo apresenta o EXEHDA-SD, descrevendo as características do mecanismo, os componentes de sua arquitetura, o seu processador semântico e a forma como os recursos são descritos e consultados. Também são apresentados os componentes para gerenciamento de preferências do usuário e ativação do mecanismo.

4 EXEHDA-SD: CONCEPÇÃO E MODELAGEM

O mecanismo de descoberta de recursos proposto almeja atender as particularidades decorrentes do ambiente ubíquo. Neste sentido, o EXEHDA-SD está sendo modelado para que localize recursos disponíveis no ambiente, com o objetivo de suprimir as demandas do cliente de forma transparente e automática, levando em conta aspectos como: heterogeneidade, escalabilidade, dinamicidade e preferências do cliente. Este capítulo apresenta as definições do projeto com base nos fundamentos teóricos e sua arquitetura de software, tendo como eixo central o detalhamento do processamento semântico utilizado no mecanismo.

4.1 Características Envolvidas no Projeto do EXEHDA-SD

Ambientes computacionais ubíquos integram diversos dispositivos conectados através de redes cabeadas ou não. Dispositivos móveis compactos como *smartphones* e PDAs geralmente possuem um poder computacional reduzido, ao contrário de servidores e *workstations* que seu *hardware* possui uma grande capacidade de processamento. A computação ubíqua é composta de dispositivos heterogêneos e dinâmicos que podem entrar e sair do ambiente a qualquer momento. Devido à grande abrangência dos ambientes ubíquos os recursos que satisfazem a necessidade do usuário ou da aplicação podem estar fisicamente afastados do local que originou a solicitação. O mecanismo de descoberta de recursos tem a finalidade de gerenciar a relação entre os consumidores e provedores de recursos. Este gerenciamento tem por objetivo localizar recursos heterogêneos e dispersos que estejam disponíveis, com pouca ou nenhuma intervenção do usuário neste processo.

O mecanismo de descoberta de recursos é fundamental para a descoberta e registro dos recursos do ambiente. Eles estão conectados por diversas tecnologias de rede e de vários tipos, como LANs, WANs e MANETs. Sua organização pode utilizar técnicas centralizadas, não centralizadas ou híbridas.

A escalabilidade do mecanismo de descoberta deve ser considerada, pois o ambiente ubíquo pode ter uma dimensão muito grande, envolvendo instituições, cidades, estados e até países. Por isso o modelo a ser proposto levará em consideração a escalabilidade, permitindo a interoperabilidade do mecanismo de descoberta entre as células do ambiente ubíquo, promovido pelo EXEHDA.

O mecanismo de descoberta de recursos proposto está sendo modelado para atuar

no Subsistema de Execução Distribuída do *middleware* EXEHDA. A figura 4.1 destaca o Subsistema do qual o serviço EXEHDA-SD fará parte.

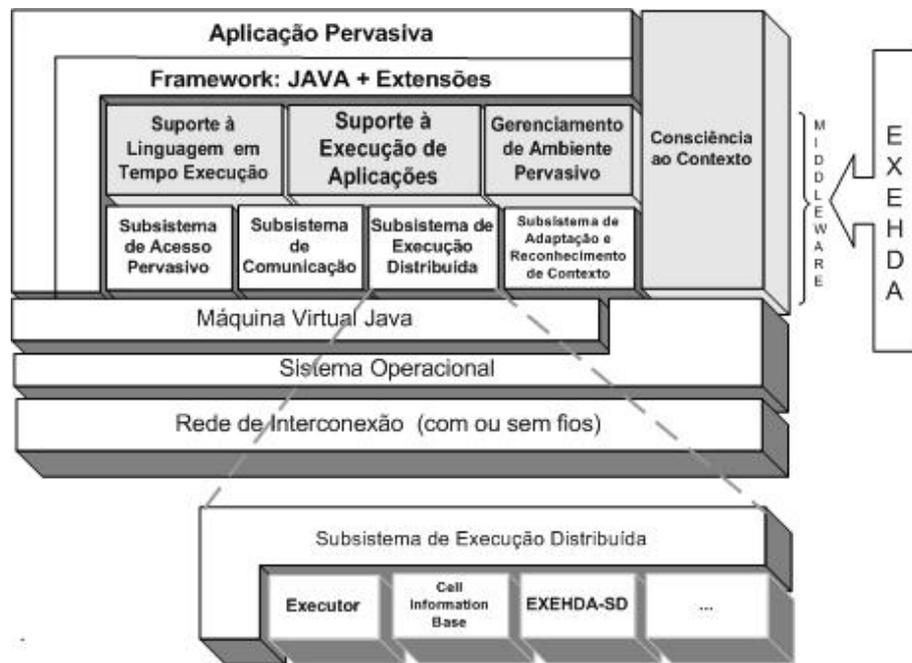


Figura 4.1: Subsistema do EXEHDA Diretamente Relacionado ao EXEHDA-SD

Um dos diferenciais desta proposta é a expressividade na descrição dos recursos. A descrição dos recursos será feita utilizando tecnologias de Web Semântica, promovendo uma melhor organização sintática e semântica na representação e consulta. Também será possível a criação de regras para restringir consultas incompatíveis, como exemplo, uma solicitação de um cliente por um determinado sistema operacional com sistema de arquivos incompatível com o sistema operacional solicitado.

O uso de ontologias na representação dos recursos tornará a consulta por recursos mais precisa, pois não haverá ambiguidade de conceitos no domínio da ontologia.

Em ambiente ubíquo envolvendo várias organizações, os recursos são gerenciados por mais de uma pessoa. Este procedimento pode acarretar problemas, pois um mesmo recurso pode ser nomeado de várias maneiras.

A utilização de *matching* sintático realiza apenas a comparação de atributos idênticos, descartando recursos que, apesar de serem idênticos ao solicitado, foram descritos de forma diferente ao passado pelo usuário/cliente.

No *matching* semântico a consulta é feita verificando instâncias disponíveis da classe do recurso solicitado. Caso não haja nenhuma instância exata disponível no momento da classe do recurso solicitado, há a possibilidade de apresentar como resposta à consulta recursos semelhantes existentes em uma subclasse ou numa superclasse. A figura 4.2 exemplifica a organização do conceito chamado “Sistema Operacional” como superclasse dos conceitos “Windows”, “Unix” e “MacOS”. O conceito “Unix” possui uma subclasse chamada “Linux”. Quando o mecanismo de descoberta procura por recursos que possuem sistema operacional “Unix” o processador semântico irá consultar se existem instâncias na classe Unix e através do raciocinador buscar os sistemas operacionais que são do tipo “Unix” nas subclasses. Este processo possui várias vantagens comparado aos mecanismos que não utilizam semântica em seu processo de busca por

recursos. A instanciação dos recursos nas classes da ontologia não precisa estar descrita de forma exata, por exemplo instâncias de Debian, debian, deBian ou debbian se forem instanciadas dentro da classe “Linux” serão localizadas independentemente da forma que foram descritas.

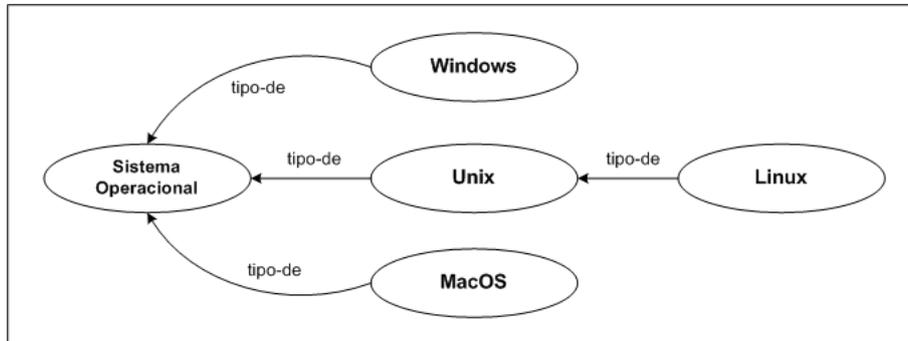


Figura 4.2: Definição de Classe, Subclasse e Superclasse

Para concepção do mecanismo de descoberta de recursos para o *middleware* EXEHDA (YAMIN, 2004) com suporte semântico entende-se que se faz necessário atender as seguintes características:

- a descoberta de recursos deverá considerar a estrutura celular promovida pelo EXEHDA;
- recurso é o nome dado para qualquer nodo, dispositivo, aplicação ou serviço presente no ambiente;
- todo recurso disponível no *middleware* EXEHDA deverá estar associado a um nodo;
- um nodo é representado por um equipamento com poder de processamento, mesmo que reduzido, com interface de rede e capaz de executar o núcleo mínimo do EXEHDA;
- os recursos serão referenciados através do ID da célula e o ID do nodo;
- todos recursos existentes na célula deverão ser previamente cadastrados pelo administrador do ambiente;
- o contexto gerenciado pelo *middleware* EXEHDA baseia-se nas políticas dos componentes da aplicação;
- o cliente pode ser identificado como um usuário ou uma aplicação;
- para atender a escalabilidade sem sobrecarregar a rede pretende-se utilizar redes *super peer*, já previstas no modelo de descoberta PerDis (SCHAEFFER, 2005);
- o catálogo de recursos de cada célula estará localizado no EXEHDABase da respectiva célula;
- a ontologia e a máquina de inferência também estarão localizadas no EXEHDABase da célula;

- haverá um gerenciador de recursos para cadastrar todos os recursos de cada célula;
- os recursos ativos e disponíveis estarão instanciados num repositório dinâmico;
- todo recurso possuirá um *lease* associado, indicando seu tempo de vida;
- os recursos deverão enviar mensagens ao diretório de cada célula para renovar seu *lease*, caso contrário será removido do repositório dinâmico;
- o cliente poderá especificar escopo da consulta (próprio nodo, célula local e/ou células vizinhas);
- o usuário poderá também especificar preferências, desde que não conflitem com as políticas da aplicação;
- a consulta por recursos será realizada primeiramente na célula em que se encontra o cliente/requisitante. Será feita uma consulta na ontologia local por recursos idênticos ou semelhantes;
- o processo de pesquisa por recursos levará em consideração os perfis de acesso vinculados aos usuários e recursos.

A invocação e alocação de recursos é realizada pelo serviço *Resource Broker* do *middleware* EXEHDA, juntamente com o serviço *Gateway* quando o recurso encontra-se em célula diferente do cliente. O mecanismo proposto poderá notificar o cliente quando o recurso desejado estiver disponível. A notificação será enviada quando o recurso anunciar sua presença ao diretório.

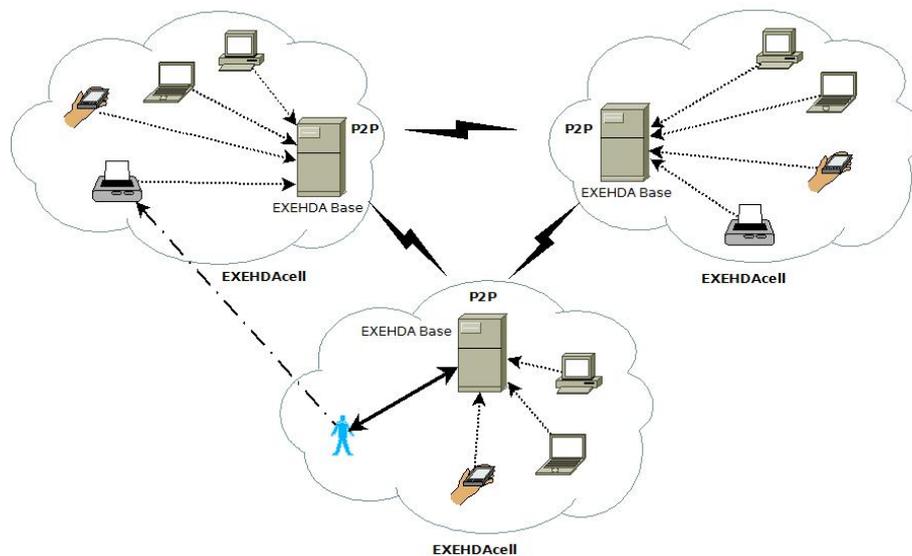


Figura 4.3: EXEHDA-SD: Ambiente Celular de Descoberta

A figura 4.3 caracteriza a área de atuação do mecanismo de descoberta de recursos proposto. A figura demonstra um cliente realizando uma consulta por uma determinada impressora ao diretório (EXEHDA Base). O diretório encontrou uma instância de impressora em outra célula e retorna o resultado ao cliente. Com os dados do recurso solicitado

o cliente acessa diretamente a impressora solicitada. A comunicação entre as células é realizada através dos EXEHDA Base de cada célula, formando uma topologia *super peer*. Em cada célula, todos os recursos são cadastrados no EXEHDA Base local.

4.2 Modelagem da Arquitetura de Software

A arquitetura proposta na figura 4.4 está organizada em três componentes distintos: (CD) Componente Diretório, (CR) Componente Recurso e (CC) Componente cliente.

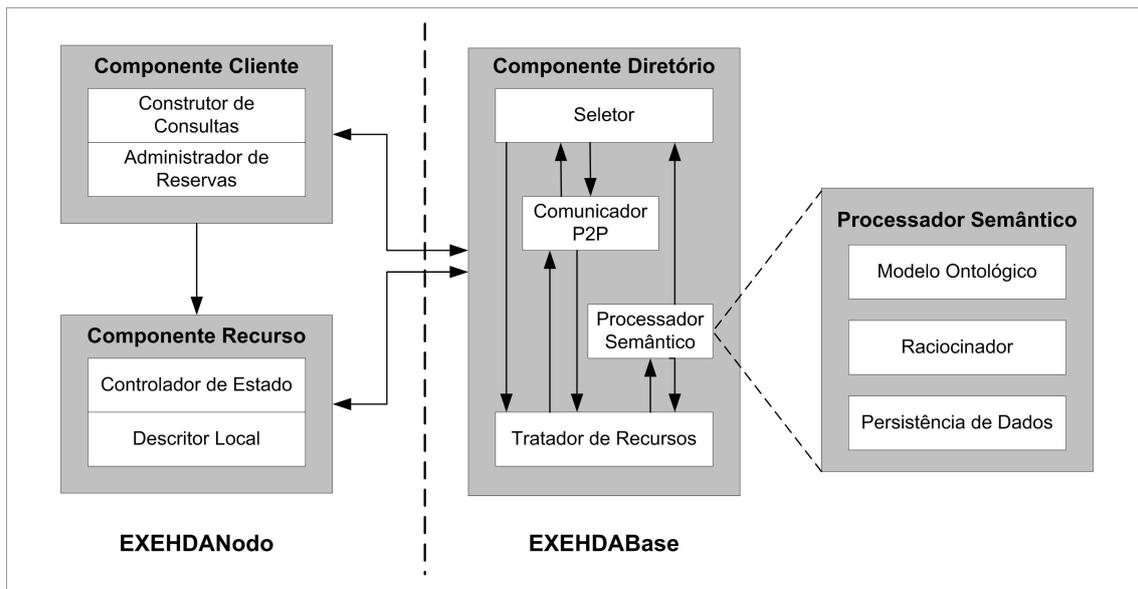


Figura 4.4: EXEHDA-SD: Arquitetura do Modelo Proposto

4.2.1 CC - Componente Cliente

O CC é responsável pela especificação dos recursos desejados, através de critérios que definem características pertinentes no processamento da consulta, e ainda, dependendo da natureza da consulta são utilizadas as preferências do cliente. O cliente é formado por componentes de software de uma aplicação instanciada no modelo ontológico do ambiente ubíquo. Na condição de clientes, estes componentes são os responsáveis pelo disparo da consulta por recursos.

O CC é composto dos módulos Construtor de Consultas e Administrador de Reservas.

4.2.1.1 Construtor de Consultas

O Construtor de Consultas é responsável por interpretar os recursos necessários para atender uma aplicação e gerar o arquivo XML com a pesquisa desejada. A aplicação é formada por vários componentes de software e cada componente contém a descrição dos recursos necessários para sua execução. O arquivo XML é enviado ao CD para geração da consulta em formato SPARQL. Este módulo também tem por objetivo receber as respostas das consultas realizadas.

4.2.1.2 Administrador de Reservas

O Administrador de Reservas é um módulo responsável por notificar o cliente quando o diretório verificar que o recurso desejado tornou-se disponível. Quando o CD recebe a mensagem do CR informando sua presença no ambiente ele verifica se o recurso que se tornou disponível está sendo aguardado por um cliente, caso afirmativo, o CD envia uma mensagem ao Administrador de Reservas do CC.

4.2.2 CR - Componente Recurso

O Componente Recurso (CR) é responsável por notificar o estado atual recurso. Isto é feito por troca de mensagens com o CD, dentro de um intervalo de tempo. O CR possui uma descrição em linguagem OWL dos seus recursos para facilitar seu deslocamento para células diferentes da célula origem. Quando o recurso ingressa pela primeira vez em uma célula, o CR envia o arquivo OWL com a descrição dos seus recursos para o CD da célula que está sendo visitada. O CD adiciona as informações descritas em OWL na ontologia local. O CR é composto pelos módulos Controlador de Estado e Descritor local.

4.2.2.1 Controlador de Estado

O Controlador de Estado tem por finalidade anunciar a disponibilidade do recurso no ambiente ubíquo. O anúncio é feito enviando uma mensagem ao Tratador de Recursos do Diretório. A técnica a ser utilizada pelo mecanismo para controlar o estado dos recursos será *soft state* para evitar que clientes e diretórios tenham que verificar o estado de cada recurso do ambiente. Como o ambiente ubíquo pode conter centenas ou até milhares de recursos, a melhor alternativa para esta situação é o *soft state*, ou seja, os recursos se “anunciarão” para CD. As mensagens de anúncio ocorrerão após o CD confirmar o registro do recurso no Diretório da célula.

4.2.2.2 Descritor Local

O Descritor Local tem por objetivo armazenar e manter atualizado o arquivo OWL com as descrições do recurso localmente. Este arquivo é gerado pela interface de cadastro de recursos, localizada no Diretório, no momento em que o recurso é instanciado na ontologia, ou são alterados atributos. Este módulo irá receber um novo arquivo de descrições em OWL sempre que o recurso sofrer alterações no descritor, localizado no Diretório. A cada ativação do recurso na célula, o Descritor Local enviará o arquivo de descrições em OWL para o Tratador de Recursos do Diretório.

4.2.3 CD - Componente Diretório

O Componente Diretório é formado por quatro módulos: Tratador de Recursos, Processador Semântico, Comunicador P2P e Seletor.

4.2.3.1 Tratador de Recursos

O Tratador de Recursos tem as seguintes funcionalidades:

- realizar a manutenção dos recursos no diretório (adicionar, remover e editar);

- disponibilizar uma interface de consulta aos usuários para que possam verificar se os recursos desejados estão disponíveis no momento;
- comunicar o Processador Semântico sobre mudança de estado dos recursos para que seja atualizada a ontologia;
- manter o CRA (Controle de Recursos Ativos);
- manter o CNR (Controle de Notificação de Recursos Disponíveis);
- receber a consulta dos CC (Componente Cliente) em formato XML;
- receber e registrar recursos anunciados pelos CR (Componentes Recursos) no repositório de recursos ativos. Estes recursos deverão estar cadastrados e autorizados previamente pelo administrador do ambiente.

Quando um CC (Componente Cliente) solicita que seja notificado sobre a disponibilidade de algum recurso, primeiramente é realizada a consulta sobre todos os recursos “Ativos” da célula, caso a consulta não retorne resultados que atendam à requisição, a consulta será refeita fazendo uso dos recursos “Inativos”, mas existentes na ontologia. Caso a consulta retorne recursos que satisfaçam a requisição, o Tratador de Recursos notificará o CC quando o CR do recurso desejado renovar o *lease*, alterando o estado do recurso para “Ativo”. No estado “Ativo” o recurso estará disponível para ser alocado pelo *Resource Broker* do *middleware* EXEHDA.

O CRA (Controle de Recursos Ativos) é um repositório mantido pelo Tratador de Recursos. Neste repositório são armazenados o *Nodo_ID* e o *lease*. O objetivo do CRA é gerenciar os recursos que entram e saem do ambiente celular. Quando um recurso entra no ambiente o CR envia uma mensagem para o Tratador de Recursos e este adiciona o *Nodo_ID* onde está localizado o recurso e o *lease* padrão definido pelo mecanismo. O *lease* é um intervalo de tempo que é gerenciado pelo Tratador de Recursos. O CR precisa renovar o *lease* do recurso localizado no CD periodicamente, caso contrário o Tratador de Recursos irá remover o recurso do CRA.

O CNR (Controle de Notificação de Recursos Disponíveis) também é mantido pelo Tratador de Recursos. Quando uma pesquisa é recebida pelo Tratador de Recursos, e no seu perfil está declarada a solicitação de notificação de recursos disponíveis, o Tratador de Recursos irá monitorar a presença do recurso desejado no ambiente.

Quando o CR anuncia um recurso ao Tratador de Recursos, este verifica se o recurso está no CRA e renova seu *lease* ou adiciona o recurso com o *lease* padrão caso não esteja presente do CRA. Após, o Tratador de Recursos consulta o CNR para verificar se o recurso não está sendo esperado por algum cliente. Se afirmativo, o cliente será notificado que o recurso está disponível.

Este módulo também tem por objetivo controlar o agendamento da disponibilidade do recurso. Ao ativar o serviço de descoberta o mecanismo, por padrão, compartilha todos os recursos localizados no nodo. Através da interface representada na figura 4.5 é possível selecionar o nodo que se deseja tornar indisponível no ambiente ubíquo.

Após selecionado o nodo é possível definir os intervalos de data e hora em que o nodo selecionado estará indisponível, conforme figura 4.6.

Pode-se tomar como exemplo um recurso que deverá estar indisponível ao ambiente nos dias da semana: segundas-feiras, quartas-feiras e sextas-feiras e no seguinte horário: 15h às 21h.

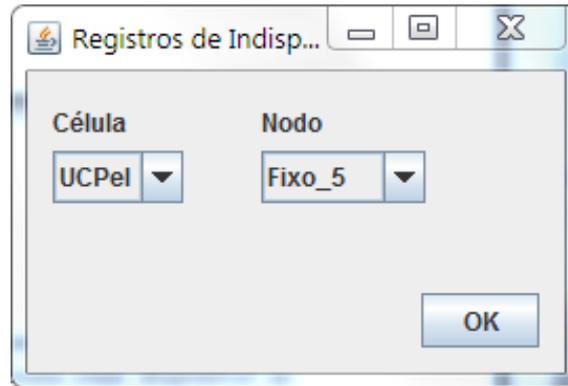


Figura 4.5: EXEHDA-SD: Indisponibilidades - Seleção de Nodo

Data Inicial	Hora Inicial	Data Final	Hora Final
08/06/2010	13:55	08/06/2010	18:45
08/06/2010	13:55	08/06/2010	18:45
08/06/2010	13:55	08/06/2010	18:45
08/06/2010	13:55	08/06/2010	18:45

Figura 4.6: EXEHDA-SD: Indisponibilidades - Intervalos de Tempo

4.2.3.2 Processador Semântico

O Processador Semântico é implementado em Java com API Jena e é formado por três componentes: Modelo Ontológico, Raciocinador e Persistência de Dados.

A escolha da API Jena foi definida pelo G3PD devido sua integração com a linguagem Java, utilizada pelo *middleware* EXEHDA. A possibilidade de persistência de dados com banco de dados livres e a utilização de raciocinadores embutidos na própria API também foram considerados.

O Modelo Ontológico é responsável pela manutenção da ontologia do mecanismo de descoberta. Esta ontologia é desenvolvida na linguagem OWL.

O raciocinador é responsável por aplicar regras e processar as consultas realizadas em SPARQL. O raciocinador irá procurar por recursos idênticos e semelhantes ao solicitado.

O módulo de Persistência de Dados é responsável pelo armazenamento das triplas da ontologia no banco de dados.

O Processador Semântico tem as seguintes funcionalidades:

- instanciar os recursos na ontologia. Os recursos são cadastrados através da interface disponibilizada pelo Tratador de Recursos;
- converter o formato da consulta enviada pelo Tratador de Recursos para a linguagem SPARQL;
- processar as consultas em SPARQL na ontologia local, instanciada no banco de dados PostgreSQL;

- inferir novos conceitos através da interpretação de regras;
- enviar a consulta ao *Comunicador P2P*, quando o escopo envolver células vizinhas.

4.2.3.3 Comunicador P2P

O Comunicador P2P é o módulo responsável pela comunicação com as células vizinhas. A comunicação entre as células será realizada utilizando tecnologias P2P entre os EXEHDABase de cada célula. Será utilizada uma variação do P2P puro, chamada *super peer*. Neste modelo a comunicação cliente/servidor ocorrerá apenas entre os diretórios localizados nos EXEHDABase de cada célula. Os CC e CR acessarão apenas o diretório da célula local. O Comunicador P2P é responsável por repassar a pesquisa para as células vizinhas de acordo com o número de saltos definidos pelo CC.

A figura 4.7 exemplifica o fluxo de informação entre dois Comunicadores P2P de células distintas.

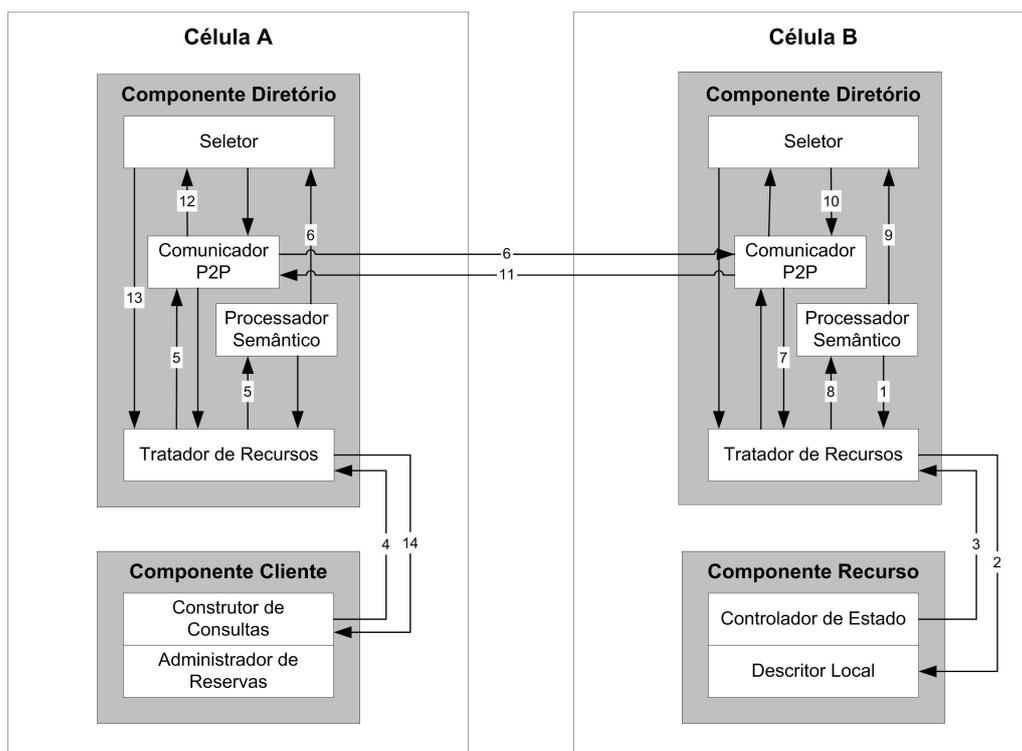


Figura 4.7: EXEHDA-SD: Comunicador P2P

O fluxo de informação entre os Comunicadores P2P representado na figura 4.7 está organizado em duas fases:

- **Configuração celular:** instanciação e presença de recursos no ambiente celular (Etapas de 1 a 3);
- **Pesquisa por recursos:** nesta fase é demonstrado o fluxo de informação a partir da consulta de recursos entre células, originada por um cliente (Etapas 4 a 11).

As fases descritas acima estão organizadas em 11 etapas:

1. Administrador do ambiente instancia um novo recurso no modelo ontológico e o Processador Semântico envia o arquivo no formato OWL com as descrições do recurso cadastrado ao Tratador de Recursos;
2. Tratador de Recursos envia arquivo OWL para o módulo Descritor Local do CR que contém os recursos instanciados no CD;
3. Controlador de estado do CR envia uma mensagem ao Tratador de Recursos para anunciar sua presença no ambiente, tornado-se disponível;
4. Módulo Construtor de Consultas gera um arquivo XML contendo as características do recurso desejado pelo cliente e envia para o Tratador de Recursos, no CD de sua célula;
5. Tratador realiza a leitura do arquivo XML e identifica que o escopo de pesquisa envolve várias células e repassa o XML para o Comunicador P2P. O Tratador de recursos interpreta a consulta especificada em XML, gerando uma nova consulta na linguagem SPARQL e repassa ao Processador Semântico juntamente com regras de inferência especificadas no arquivo XML, se existirem;
6. Processador Semântico processa a consulta e as regras de inferência no modelo ontológico da célula local e repassa os resultados da consulta para o Seletor. O Comunicador P2P envia o arquivo XML para células vizinhas utilizando tecnologia *super peer*. Também é verificado se o escopo de pesquisa envolve células estáticas definidas pelo administrador do ambiente;
7. Comunicador P2P da célula B recebe o arquivo XML e a mensagem do P2P cliente contendo o ID da Base que originou a consulta. O arquivo XML é repassado para o Tratador de Recursos;
8. Tratador de Recursos interpreta o arquivo XML e gera a consulta na linguagem SPARQL e repassa ao Processador Semântico com regras de inferência, se especificadas;
9. O Processador Semântico processa a consulta no modelo ontológico e repassa os resultados ao Seletor;
10. O Seletor aplica as preferências do usuário, se especificadas, e entrega os resultados da célula ao Comunicador P2P;
11. O Comunicador P2P decrementa a profundidade de pesquisa informada na mensagem recebida pelo P2P origem e repassa os resultados da pesquisa ao Comunicador P2P que solicitou a pesquisa.

4.2.3.4 Seletor

Quando uma pesquisa por recursos retornar mais de um recurso, o módulo Seletor fará a classificação e ordenará os recursos posicionando os que melhor satisfazem a requisição no topo da lista e os que menos satisfazem ficarão ao final.

O Seletor é responsável por receber e organizar as respostas das consultas realizadas em outras células, recebidas pelo módulo P2P. O processamento do resultado da consulta envolvendo várias células colocará no topo da lista os recursos que estiverem:

- localizados na própria célula que originou a consulta;
- nas células vizinhas estáticas;
- em células com menor profundidade de pesquisa.

O Seletor aplicará as preferências do cliente no resultado da pesquisa, se for definido do arquivo XML do Componente Cliente.

O recurso estará indisponível quando não estiver ativo, ou seja o módulo Controlador de Estado do CR não enviou uma mensagem para renovar o *lease* do recurso. Quando o *lease* do recurso não é renovado o CD altera o estado para indisponível. Neste estado o recurso não é considerado nas consultas realizadas pelos CC. O estado disponível ocorre quando o *lease* do recurso não é zero, ou seja o módulo Controlador de Estado do CR tem renovado o *lease* do recurso.

4.3 Processador Semântico

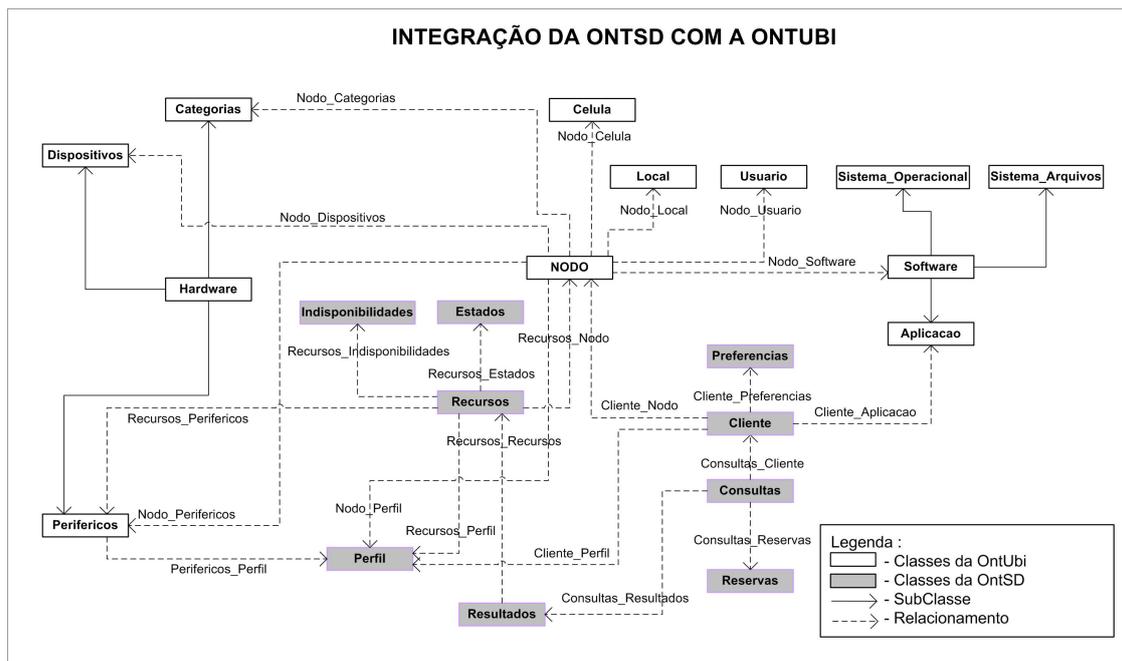
O Processador Semântico é composto por ferramentas que agregam semântica na representação e consulta por recursos. Dentre as tecnologias utilizadas pelo processador estão as linguagens Java e OWL, API Jena, Banco de Dados PostgreSQL e linguagem de consulta SPARQL integrada à API Jena através do “motor de consultas” ARQ.

O modelo ontológico utilizado pelo Processador Semântico é composto por duas ontologias:

- OntUbi: ontologia caracterizada pela figura 4.8, desenvolvida no Protégé, pelo grupo G3PD da Universidade Católica de Pelotas, para ser utilizada pelos mecanismos de consciência e adaptação de contexto e descoberta de recursos. Esta ontologia está em constante atualização e continuará a ser ampliada e detalhada no decorrer do trabalho;
- OntSD: a figura 4.9 apresenta a OntSD, também desenvolvida no Protégé para atender exclusivamente o EXEHDA-SD. Esta ontologia integra-se à OntUbi através de vários relacionamentos entre classes. Na figura estão representados os recursos do ambiente ubíquo, os quais são instanciados nas classes “Perifericos” e “Nodo” da OntUbi. O Cliente representa uma aplicação existente no ambiente, ativada por um usuário. O cliente é representado pelas classes “Aplicacao” e “Usuario” da OntUbi. A aplicação para entrar em execução no ambiente ubíquo necessita de recursos mínimos. Estes recursos estão definidos nos componentes de software da aplicação.

O EXEHDA-SD utiliza ontologias para representação dos recursos no ambiente ubíquo e pesquisa dos recursos instanciados. O Processador Semântico é responsável pela instanciação de novos recursos e processamento de regras de inferência, possibilitando ao mecanismo de descoberta o reconhecimento de conceitos implícitos na ontologia. Ele é formado por três componentes:

- Modelo Ontológico: é a representação do domínio da computação ubíqua promovida pelo *middleware* EXEHDA. Este modelo foi construído através da ferramenta Protégé (BIOMEDICAL INFORMATICS RESEARCH, 2009) que facilitou a criação de classes, relacionamentos e atributos. As ontologias utilizadas pelo mecanismo foram construídas na linguagem OWL;



- **Raciocinador:** é o mecanismo responsável pelo processamento das regras de inferência e também do processamento de subconceitos. O raciocinador utilizado no mecanismo é da própria API Jena;
- **Persistência de Dados:** através da Persistência de Dados é possível armazenar as triplas (sujeito, predicado e objeto) que formam a ontologia em banco de dados. O mecanismo de Persistência de Dados utilizado é disponibilizado pela API Jena e o banco de dados utilizado é o PostgreSQL.

Os recursos disponíveis no ambiente ubíquo promovido pelo EXEHDA são caracterizados por nodos, que podem ser móveis, fixos ou base. Estes nodos são formados por componentes de *hardware* e *software*. Desta forma podemos considerar um nodo qualquer equipamento que tenha condições de executar o núcleo mínimo do EXEHDA. Os demais recursos espalhados pelo ambiente ubíquo devem estar obrigatoriamente conectados a um nodo, por exemplo, impressoras, *scanners*, *webcam*, etc.

Toda célula no ambiente ubíquo possui uma identificação única, assim como os nodos cadastrados na célula. Toda célula possui uma única Base. Esta Base pode ser formada por várias máquinas. A localização de células, nodos e bases é feita pelo ID delas.

Essas premissas foram definidas pelo Grupo de Pesquisa em Processamento Paralelo e Distribuído (G3PD) da Universidade Católica de Pelotas levando em conta as características do *middleware* EXEHDA.

4.3.1 Definições da OntUbi

Na ontologia OntUbi estão descritos todos os recursos do ambiente. As consultas realizadas pelo Processador Semântico serão realizadas a partir da classe “Nodo” ou da

classe “Periféricos”, levando em consideração todos os relacionamentos definidos nestas classes.

Abaixo estão descritas as classes, subclasses, relacionamentos e atributos contidos na OntUbi que estão diretamente relacionadas na descrição dos recursos do ambiente ubíquo:

- Celula: contém informações sobre as células do EXEHDA, inclusive nodos e EXEHDA Base;
 - Relacionamentos:
 - * Celula_local: relacionamento atribuído que busca instâncias da Classe Local;
 - * Celula_Nodo: relacionamento que busca instâncias da Classe Nodo;
 - * Celula_Rede: atribuído à classe Celula e busca instâncias da classe Rede;
 - * Celula_Usuario: atribuído à classe Celula e busca instâncias da classe Usuario;
 - Atributos:
 - * Celula_ID: identificação do nome da célula;
- Nodo: entidade principal do ambiente, representa um dispositivo com capacidade computacional;
 - Superclasse: raiz;
 - Subclasses: Fixo, Movel e Base;
 - Relacionamentos:
 - * Nodo_Celula: relacionamento inverso de Celula_Nodo. É atribuído à classe Nodo e busca instâncias na classe Celula;
 - * Nodo_Categoria: atribuído à classe Nodo e busca instâncias da classe Categorias;
 - * Nodo_Dispositivos: atribuído à classe Nodo e busca instâncias da classe Dispositivos;
 - * Nodo_Local: atribuído à classe Nodo e busca instâncias da classe Local;
 - * Nodo_Periféricos: atribuído à classe Nodo e busca instâncias da classe Periféricos;
 - * Nodo_Software: atribuído à classe Nodo e busca instâncias da classe Software;
 - * Nodo_Usuario: atribuído à classe Nodo e busca instâncias da classe Usuario;
 - * Nodo_Estados: atribuído à classe Nodo e busca instâncias da classe Estados;
 - * Nodo_Perfil: atribuído à classe Nodo e busca instâncias da classe Perfil;
- Hardware: qualquer tipo de hardware utilizado por Nodos ou Periféricos que poderão ser utilizados no ambiente. Esta classe possui as seguintes subclasses:
 - Superclasse: raiz

- Subclasses:
 - * Categorias: define os tipos de nodos do ambiente;
 - * Dispositivos: é organizada em subclasses que contêm componentes que formam o hardware dos nodos e dispositivos;
 - * Perifericos: contêm os periféricos do ambiente.
- Atributos:
 - * Hardware_Fabricante: descrição do fabricante;
 - * Hardware_Modelo: descrição do modelo.
- Dispositivos: componentes de hardware que formam nodos e periféricos;
 - Superclasse: Hardware;
 - Subclasses: Armazenamento, Audio, Bateria, Entrada, Memoria, Portas, Processador, Rede, Tela;
 - Atributos: Processador_Cores, Processador_Unidade, Processador_Vel, Armazenamento_Capacidade, Armazenamento_Unidade, Armazenamento_Vel, Memoria_Capacidade, Memoria_Padiao, Memoria_Unidade;
- Perifericos: periféricos disponíveis no ambiente;
 - Superclasse: Hardware;
 - Subclasses: Atuador, Impressora, Scanner e Sensor;
 - Relacionamentos: Perifericos_Local, Perifericos_Nodo, Perifericos_Perfil.
- Impressora: categorias de impressoras;
 - Superclasse: Perifericos;
 - Subclasses: Jato_Tinta, Laser e Matricial;
 - Atributos: Impressora_cor, Impressora_Velocidade.
- Software: qualquer software que pode ser utilizado no ambiente;
 - Superclasse: raiz;
 - Subclasses: Aplicacoes, SISARQ e SO.
- SO: tipos de sistemas operacionais;
 - Superclasse: Software;
 - Subclasses: Unix e Windows.
- SISARQ: tipos de sistemas de arquivos;
 - Superclasse: Software;
 - Subclasses: EXT2, EXT3, EXT4, FAT16, FAT32 e NTFS.
- Local: todos os locais onde pode ser usado o modelo;
- Usuario: todos os usuários que podem usar o ambiente;
- Perfil: papéis possíveis dos usuários no *middleware* EXEHDA.

4.3.2 Definições da OntSD

A ontologia OntSD contém as classes, relacionamentos, atributos e restrições para contemplar as funcionalidades propostas pela arquitetura de software do EXEHDA-SD. As principais funcionalidades obtidas pela OntSD são:

- consultas baseadas no perfil do cliente (considera-se cliente um usuário ou aplicação);
- notificação de recursos disponíveis, quando o cliente solicitar um determinado recurso com notificação de disponibilidade, o cliente será notificado assim que o recurso estiver disponível;
- controle de estado;
- utilização das preferências do cliente no processo de busca;
- agendamento da indisponibilidade de recursos.

A figura 4.10 contém as classes, os atributos e os relacionamentos, os quais são descritos a seguir:

- Consultas: são instanciadas todas as consultas realizadas, originadas do arquivo XML;
 - Consultas_ID (int): identificação da consulta;
 - Consultas_ID_Cliente (string): ID do Cliente que originou a Consulta;
 - Consultas_Escopo (int): Escopo da Pesquisa (1 - próprio nodo, 2 - célula local, 3 - células vizinhas estáticas, 4 - células vizinhas dinâmicas + células vizinhas estáticas);
 - Consultas_Desc (string): Consulta em linguagem SPARQL, interpretada do arquivo XML;
 - Consultas_Preferencias (boolean): Utiliza ou não preferências do cliente;
 - Consultas_Notif (boolean): Notifica ou não o cliente sobre recurso disponível.
- Cliente: instâncias de clientes, ou seja, aplicações executadas por usuários que necessitam de determinados recursos;
 - Cliente_ID (int): identificação do cliente;
- Preferencias: preferências utilizadas na consulta, definidas pelo cliente;
 - Preferencias_ID (int): identificação da preferencia;
 - Preferencias_C1 (string): conceito1;
 - Preferencias_C1V (int): valor do conceito1 (1-Qualquer, 2-Sempre ou 3-Nunca);
 - Preferencias_C2(string): conceito2;
 - Preferencias_C2V (int): valor do conceito2 (1-Qualquer, 2-Sempre ou 3-Nunca);

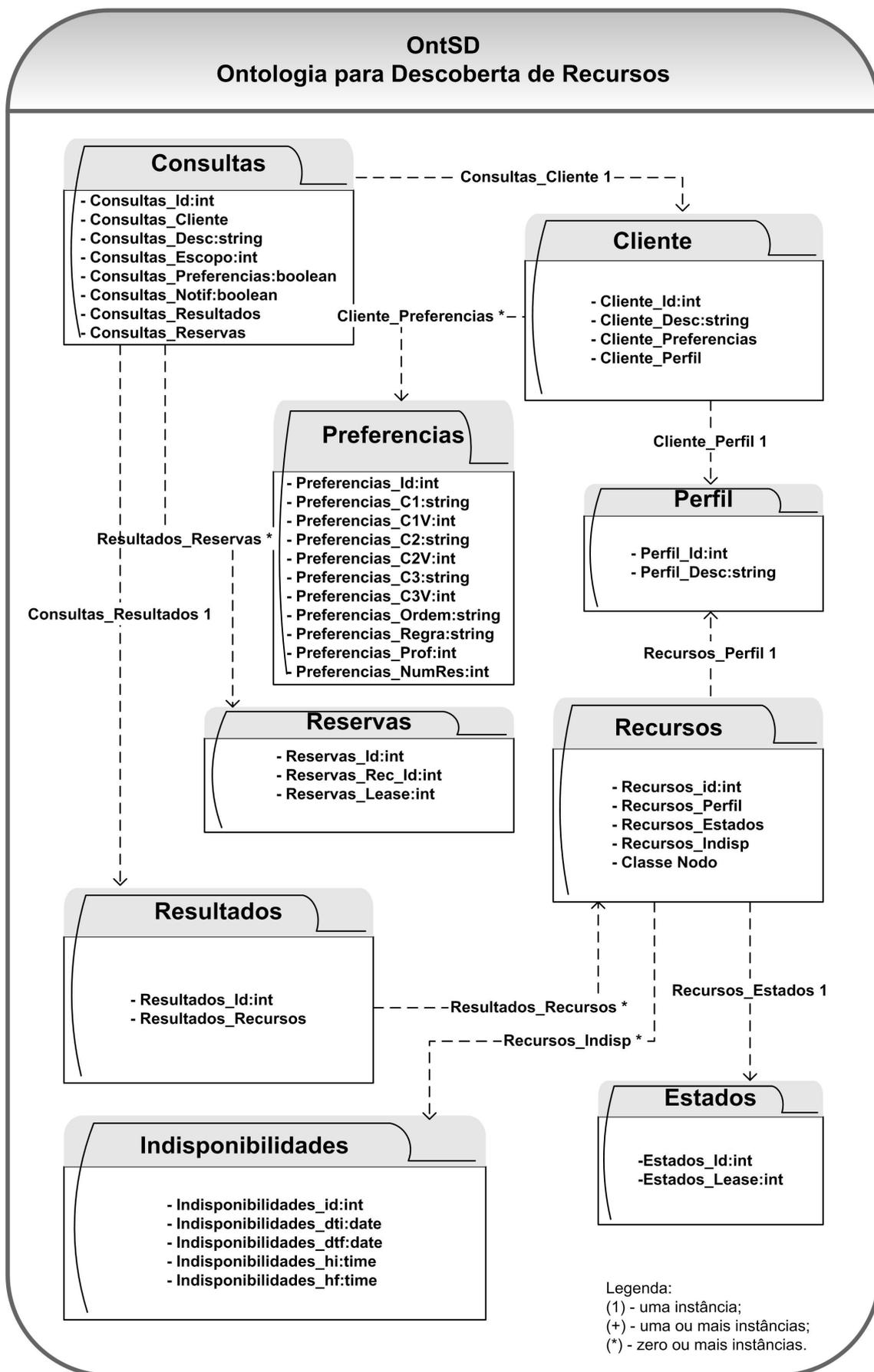


Figura 4.10: EXEHDA-SD: Classes da OntSD

- Preferencias_C3(string): conceito3;
 - Preferencias_C3V (int): valor do conceito3 (1-Qualquer, 2-Sempre ou 3-Nunca);
 - Preferencias_Ordem (string): classe ou variável que deve ser considerada na ordenação dos resultados;
 - Preferencias_Regra (string): regras de inferência definidas pelo cliente;
 - Preferencias_Prof (int): profundidade da pesquisa, quando utilizar células vizinhas dinâmicas;
 - Preferencias_NumRes (int): número de respostas desejadas, caso não seja informado o padrão é sempre 1.
- Perfil: instâncias de perfis de clientes disponíveis no ambiente. Serve para controlar o acesso de recursos;
 - Perfil_ID (int): identificação do perfil;
 - Perfil_Desc (string): descrição do perfil;
 - Reservas: contém instâncias de reservas de recursos indisponíveis que clientes desejam ser notificados quando estes tornarem-se disponíveis;
 - Reservas_ID (int): identificação da reserva;
 - Reservas_Lease (int): tempo de validade da reserva, definido em horas;
 - Resultados: instâncias de resultados obtidos do processamento das consultas;
 - Resultados_ID (int): identificação do resultado.
 - Indisponibilidades: esta classe armazena as indisponibilidades agendadas para determinado recurso. O agendamento é realizado pelo administrador do ambiente. A informação armazenada nesta classe refere-se ao período em que o recurso, mesmo que ativo, não estará com seus recursos disponíveis;
 - Indisponibilidades_ID (int): identificação da indisponibilidade;
 - Indisponibilidades_dti (date): data inicial da indisponibilidade;
 - Indisponibilidades_hi (time): hora inicial da indisponibilidade;
 - Indisponibilidades_dtf (date): data final da indisponibilidade;
 - Indisponibilidades_hf (time): hora final da indisponibilidade.
 - Estados: contém instâncias para controlar a disponibilidade de todos os recursos gerenciáveis pelo ambiente e o *lease*, tempo de vida associado ao recurso;
 - Estados_ID (int): identificação do estado;
 - Estados_Lease (int): tempo de vida do recurso.
 - Recursos: classe dinâmica que contém instâncias de todos os recursos ativos e disponíveis na célula. Além dos atributos e relacionamentos especificados contém também todos atributos e relacionamentos da classe Nodo da OntUbi. A classe Nodo da OntUbi contém instâncias de recursos estáticos, podendo estar presente ou não no ambiente.

- Recursos_ID (int): identificação do recurso;
- Classe Nodo: referência que esta classe contém todas as classes, atributos e relacionamentos da classe “Nodo” da OntUbi.

A seguir a descrição dos Relacionamentos entre as classes na OntSD:

- Consultas_Cliente 1: instância do cliente que deseja realizar a consulta, uma instância;
- Cliente_Preferencias *: instância contendo as preferências do cliente, zero ou mais instâncias;
- Cliente_Perfil 1: instância vinculando um perfil ao cliente, uma instância;
- Consultas_Resultados +: instâncias contendo os resultados da consulta. O resultado pode ser vazio ou executar várias vezes a mesma consulta, uma ou mais instâncias;
- Consultas_Reservas *: quando um resultado for vazio para a classe de recursos dinâmicos, mas não vazio para classe de recursos estáticos e o cliente deseja notificação de disponibilidade é gerada uma instância na Classe Reservas, zero ou mais instâncias;
- Resultados_Recurros *: um resultado pode retornar zero ou mais recursos, zero ou mais instâncias;
- Recursos_Perfil 1: cada recurso tem um perfil associado a ele, uma instância;
- Recursos_Estados 1: todo recurso ativo e disponível tem uma instância na classe estado para controlar sua presença no ambiente, uma instância;
- Recursos_Indisp *: o recurso pode estar ativo, mas com seus recursos indisponíveis, zero ou mais instâncias.

4.3.3 Consulta sem Raciocinador

Através de experimentos com a API Jena foi possível verificar o aumento de resultados numa consulta por recursos na ontologia proposta. Na ontologia foram instanciados alguns recursos através da ferramenta Protégé. Os seguintes recursos foram instanciados nas classes:

- Intel: Celeron, Pentium e Athlon;
- Unix: Solaris;
- Linux: Debian, Ubuntu e Kurumin;
- Windows: Windows_Vista_Premium, Windows_98 e Windows_XP;
- Nodo: Nodo_1, Nodo_9 e Nodo_6.

No primeiro experimento foi realizada a consulta por instâncias de sistemas operacionais UNIX sem o uso de raciocinadores. Esta consulta foi realizada através da linguagem SPARQL.

Listagem 4.1: SPARQL: Consulta 1

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ont: <http://localhost/Ontologia.owl#>
SELECT *
  WHERE
    {?recurso rdf:type ont:Unix}

```

Esta consulta foi realizada sem auxílio do raciocinador da API Jena e apresentou a seguinte resposta:

Instâncias de Unix sem Raciocinador:

```

=====
<http://localhost/Ontologia.owl#Solaris>

```

A consulta retornou apenas o Sistema Operacional Solaris, como sistema operacional Unix disponível.

Observando a ontologia representada na figura 4.8 é possível verificar que existem instâncias de Linux disponíveis. O Linux é uma subclasse de Unix. Como o raciocinador não estava ativo, o resultado não apresentou os conceitos similares ao solicitado.

4.3.4 Consulta com Raciocinador

No segundo teste foi ativado o raciocinador e realizada a mesma consulta, apresentando o seguinte resultado:

Instâncias de Unix com Raciocinador:

```

=====
<http://localhost/Ontologia.owl#Solaris>
<http://localhost/Ontologia.owl#Kurumin>
<http://localhost/Ontologia.owl#Debian>
<http://localhost/Ontologia.owl#Ubuntu>

```

O raciocinador apresentou também os sistemas operacionais compatíveis com Unix.

A segunda consulta realiza uma procura por nodos com Sistema Operacional Unix e Processadores Intel. Sendo que Unix e Intel são classes que contêm subclasses.

Listagem 4.2: SPARQL: Consulta 2

<pre> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX ont: <http://localhost/Ontologia.owl#> SELECT * WHERE {?nodo ont:SistOper ?SO . ?SO rdf:type ont:Unix . ?nodo ont:Proces ?Processador . ?Processador rdf:type ont:Intel } </pre>

O resultado obtido com a segunda consulta foi o seguinte:

```

<http://localhost/Ontologia.owl#Nodo_1> | <http://localhost/Ontologia.owl#Debian>
| <http://localhost/Ontologia.owl#Pentium>

```

O Nodo_1 possui Sistema Operacional Debian e Processador Pentium, conforme pode ser verificado na ontologia representada pela figura 4.8.

4.3.5 Verificação da Ontologia

Através da API Jena também foi possível verificar inconsistências nos dados da ontologia. O código em java utilizado foi o seguinte:

Listagem 4.3: JENA: Verificação da Ontologia

```
public static void Validar(String Ontology) {
    OntModel model = ModelFactory.createOntologyModel(OntModelSpec.
        OWL_MEM);
    model.read(Ontology);
    Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
    reasoner = reasoner.bindSchema(model);
    InfModel owlInfModel = ModelFactory.createInfModel(reasoner,
        model);
    ValidityReport vrp1 = owlInfModel.validate();
    if (vrp1.isValid()) {
        System.out.println("Ontologia OK");
    } else {
        System.out.println("Ontologia com problemas...");
        for (Iterator i = vrp1.getReports(); i.hasNext(); ) {
            System.out.println(" - " + i.next());
        }
    }
}
```

Para verificação da ontologia foi alterado o valor da “Veloc_Proc” que está definido como “float” de 3.0 para a string “tres”.

Listagem 4.4: OWL: Representação de Nodo

```
<Nodo rdf:ID="Nodo_1">
  <URL rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >atlas.ucpel.tche.br </URL>
  <Num_Proc rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
  >1</Num_Proc>
  <Nodo_Ativo rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
  >true </Nodo_Ativo>
  <Veloc_Proc rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
  >3.0</Veloc_Proc>
  <SistemaArquivos rdf:datatype="http://www.w3.org/2001/XMLSchema#
  string"
  >ext3 </SistemaArquivos>
  <Memoria rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
  >4000</Memoria>
  <SistOper rdf:resource="#Debian"/>
  <Proces rdf:resource="#Pentium"/>
</Nodo>
```

O resultado apresentado foi a inconsistência da ontologia gerada: Ontologia com problemas... - Error ("range check"): "Incorrectly typed literal due to range (prop, value)"
 Culprit = http://localhost/Ontologia.owl#Nodo_1
 Implicated node: http://localhost/Ontologia.owl#Veloc_Proc
 Implicated node: 'tres' http://www.w3.org/2001/XMLSchema#float

4.3.6 Persistência de Dados

Através do suporte à persistência de dados da API Jena foi possível armazenar a ontologia no banco de dados MySQL, conforme o código java abaixo:

Listagem 4.5: JENA: Persistência de Dados

```
public static void Importar_Ont(String Ontology) {
    String url = "jdbc:mysql://localhost/jena";
    String driver = "com.mysql.jdbc.Driver";
    DBConnection conn = new DBConnection ( url, "root", "mysql", "
        mysql" );
    ModelRDB model;
    if( !conn.containsModel("DR")){
        System.out.println("Ontologia está sendo armazenada...");
        model = ModelRDB.createModel(conn,"DR");
    }
    else {
        System.out.println("Ontologia já existe no BD, abrindo...")
        ;
        model = ModelRDB.open(conn,"DR");
    }
    model.begin();
    model.read(Ontology);
    model.commit();
}
```

A figura 4.11 mostra as triplas que formam a ontologia, armazenadas no banco de dados MySQL.

Subj	Prop	Obj	GraphID
Uv: http://localhost/Ontologia.owl	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Ontology	
Uv: http://localhost/Ontologia.owl#Aplicacoes:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	
Uv: http://localhost/Ontologia.owl#Software:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	
Uv: http://localhost/Ontologia.owl#Aplicacoes:	Uv: http://www.w3.org/2000/01/rdf-schema#subClassOf	Uv: http://localhost/Ontologia.owl#Software:	
Uv: http://localhost/Ontologia.owl#Aluno:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	
Uv: http://localhost/Ontologia.owl#Usuarios:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	
Uv: http://localhost/Ontologia.owl#Aluno:	Uv: http://www.w3.org/2000/01/rdf-schema#subClassOf	Uv: http://localhost/Ontologia.owl#Usuarios:	
Uv: http://localhost/Ontologia.owl#memory-description:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	
Uv: http://localhost/Ontologia.owl#hw-description:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	
Uv: http://localhost/Ontologia.owl#memory-description:	Uv: http://www.w3.org/2000/01/rdf-schema#subClassOf	Uv: http://localhost/Ontologia.owl#hw-description:	
Uv: http://localhost/Ontologia.owl#Dispositivos:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	
Uv: http://localhost/Ontologia.owl#Hardware:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	
Uv: http://localhost/Ontologia.owl#Dispositivos:	Uv: http://www.w3.org/2000/01/rdf-schema#subClassOf	Uv: http://localhost/Ontologia.owl#Hardware:	
Uv: http://localhost/Ontologia.owl#Processador:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	
Uv: http://localhost/Ontologia.owl#Processador:	Uv: http://www.w3.org/2000/01/rdf-schema#subClassOf	Uv: http://localhost/Ontologia.owl#Hardware:	
Uv: http://localhost/Ontologia.owl#Celula:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	
Uv: http://localhost/Ontologia.owl#Professor:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	
Uv: http://localhost/Ontologia.owl#Professor:	Uv: http://www.w3.org/2000/01/rdf-schema#subClassOf	Uv: http://localhost/Ontologia.owl#Usuarios:	
Uv: http://localhost/Ontologia.owl#resolution-description:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	
Uv: http://localhost/Ontologia.owl#screen-description:	Uv: http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Uv: http://www.w3.org/2002/07/owl#Class	

Figura 4.11: EXEHDA-SD: Ontologia Armazenada no Banco de Dados

Após armazenar as triplas da ontologia no banco de dados é possível abri-lo e realizar consultas SPARQL diretamente no banco, conforme código Java abaixo:

Listagem 4.6: JENA: Consulta com Banco de Dados

```

public static void Consulta_BD(String Ontology) {
    String url = "jdbc:mysql://localhost/jena";
    String driver = "com.mysql.jdbc.Driver";
    DBConnection conn = new DBConnection ( url , "root" , "mysql" , "
        mysql" );
    ModelRDB model = ModelRDB.open(conn,"DR");
    String ocQuery = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf
        -syntax-ns#>" +
        "PREFIX ont: <http://localhost/Ontologia.owl#>" +
        " +
        "SELECT * " +
        "WHERE " +
        "{?recurso rdf:type ont:Unix} ";
    QueryExecution ocQe = QueryExecutionFactory.create(ocQuery ,
        model);
    System.out.println("Instâncias de Unix:");
    ResultSet ocResults = ocQe.execSelect();
    ResultSetFormatter.out(ocResults);
    ocQe.close();
}

```

4.3.7 Definição de Regras

A utilização de regras permite aumentar o entendimento de termos que estão subentendidos, mas os computadores não conseguem entender. Para exemplificar foi criado um arquivo com três regras definindo os termos *dualcore* (nodos com dois processadores), *quadcore* (nodos com quatro processadores) e *manycore* (nodos com mais de quatro processadores), da seguinte forma:

Listagem 4.7: JENA: Definição de Regras (Ontologia.rules)

```

@prefix ont: <http://localhost/Ontologia.owl#>.
@include <RDFS>.

[dualcore: (?n ?p ont:Nodo) (?n ont:Num_Proc 2) -> (?n rdf:type ont:
    dualcore)]
[quadcore: (?n ?p ont:Nodo) (?n ont:Num_Proc 4) -> (?n rdf:type ont:
    quadcore)]
[manycore: (?n ?p ont:Nodo) (?n ont:Num_Proc ?x) greaterThan(?x,4) ->
    (?n rdf:type ont:manycore)]

```

Para o processamento das regras foi criado um código em Java que adiciona os conceitos deduzidos pelas regras na ontologia e realiza a consulta através de SPARQL pelo novo conceito deduzido.

Listagem 4.8: JENA: Código Java com Regras

```

public static void Aplicar_Regras(String Ontology) {
    OntModel model = ModelFactory.createOntologyModel(OntModelSpec.
        OWL_MEM);
    model.read(Ontology);
    Reasoner reasoner = new GenericRuleReasoner(Rule.rulesFromURL("
        file:///d:/jena/Ontologia.rules"));
    InfModel infModel = ModelFactory.createInfModel(reasoner , model
        );
}

```

```

Model model2 = infModel.getDeductionsModel();
model.add(model2);
reasoner = reasoner.bindSchema(model);
String ocQuery = "PREFIX rdf: <http://www.w3.org/1999/02/22-
    rdf-syntax-ns#>" +
    "PREFIX ont: <http://localhost/Ontologia.owl
    #>" +
    "SELECT * " +
    "WHERE " +
    "{?nodo rdf:type ont:dualcore }" ;

QueryExecution ocQe = QueryExecutionFactory.create(ocQuery,
    model);
ResultSet ocResults = ocQe.execSelect();
ResultSetFormatter.out(ocResults);
ocQe.close();
}

```

O resultado da consulta por nodos *dualcore* é apresentado abaixo:

Listagem 4.9: JENA: Resultado da Consulta

```

run :
-----
| nodo |
=====
| <http://localhost/Ontologia.owl#Nodo_6> |
| <http://localhost/Ontologia.owl#Nodo_2> |
-----
CONSTRUÍDO COM SUCESSO (tempo total: 3 segundos)

```

A ontologia proposta beneficiará o modelo de descoberta de recursos por:

- possuir um domínio bem definido com classes e subclasses que serão utilizadas em todo o domínio do EXEHDA, evitando ambiguidade de conceitos sobre um mesmo recurso;
- possibilitar consultas semânticas;
- inferir novos conceitos sobre os recursos existentes;
- possibilitar a criação de regras.

Para o processamento da ontologia e integração com o mecanismo de descoberta de recursos a ser modelado, possivelmente será utilizado a API Jena. A Jena, como já foi descrita anteriormente, possui suporte a raciocinadores e persistência de dados com suporte a banco de dados MySQL.

4.4 Descrição e Consulta de Recursos

Esta seção descreve o formato em que os recursos são armazenados no Componente Diretório e a maneira em que o Componente Cliente organiza os atributos e valores sobre o recurso desejado para compor a pesquisa.

4.4.1 Representação e Manutenção de Recursos

Todos recursos existentes no ambiente ubíquo disponíveis para serem utilizados são pré-cadastrados pelo administrador do ambiente. Este cadastro é feito através de uma interface disponibilizada pelo Tratador de Recursos localizado no Componente Diretório. Estes recursos são instanciados no modelo ontológico através do Processador Semântico.

Quando um recurso é inserido ou alteradas as descrições dele o Tratador de Recursos envia um arquivo no formato OWL para o nodo proprietário dos recursos para armazenar a cópia local, atualizada, das descrições.

A instanciação de recursos é realizada através da interface de cadastro de recursos. A figura 4.12 apresenta a interface para cadastro de nodos.

Figura 4.12: EXEHDA-SD: Cadastro de Recursos - Nodos

A interface representada pela figura 4.13 possibilita ao administrador do ambiente instanciar periféricos no modelo ontológico.

Desta forma os recursos são representados em linguagem OWL que possibilita uma grande expressividade.

4.4.2 Especificação da Consulta por Recursos

O EXEHDA-SD processa a consulta dos Componentes Cliente através de arquivos XML. O arquivo XML descreve o recurso desejado através de atributos e valores e define o perfil a ser aplicado no processo de descoberta. Também é especificado no arquivo XML o ID do cliente, se serão consideradas as preferências do cliente, se deseja notificação de recurso disponível e escopo da pesquisa.

Exemplo de uma consulta com especificação de critérios:

Figura 4.13: EXEHDA-SD: Cadastro de Recursos - Periféricos

Listagem 4.10: XML: Consulta

```
<?xml version="1.0" encoding="UTF-8"?>
<CONSULTA>
  <ID_CLIENTE>1200</ID_CLIENTE>
  <PERFIL>Administrador</PERFIL>
  <PREF_CLIENTE>S</PREF_CLIENTE>
  <NOTIF_CLIENTE>N</NOTIF_CLIENTE>
  <NOTIF_TEMPO>24</NOTIF_TEMPO>
  <ESCOPO>2</ESCOPO>
  <TIPO>Nodo</TIPO>
  <CRITERIO nome='SO' valor='Unix' op='equ' />
  <CRITERIO nome='Processador' valor='Intel' op='equ' />
</CONSULTA>
```

Na listagem 4.10 é especificada uma consulta em XML contendo as seguintes *tags*:

- <ID_CLIENTE> Especifica o ID do cliente;
- <PERFIL> Indica o perfil do cliente a ser considerado na consulta;
- <PREF_CLIENTE> Utiliza ou não as preferências do cliente, quando existirem;
- <NOTIF_CLIENTE> Se o cliente deseja ser notificado, quando o recurso desejado está disponível. Será notificado apenas quando a consulta não retornar resultados;

- <NOTIF_TEMPO> Define o tempo, em horas, que o serviço de descoberta aguardará pela presença do recurso indisponível. Foi adotado o padrão de 24 horas, podendo ser alterado pelo cliente;
- <ESCOPO> Escopo da Pesquisa: 1-próprio nodo, 2-célula local, 3-células vizinhas estáticas, 4-células vizinhas estáticas e dinâmicas;
- <TIPO> Tipo de recurso desejado, valores possíveis: Nodo, Perifericos ou Slots;
- <CRITERIO> O critério de pesquisa é composto por “nome”, “valor” e op (operador). É possível utilizar vários critérios para refinar a pesquisa. Os operadores possíveis estão relacionados na tabela 4.1.

Tabela 4.1: Lista de Operadores Válidos para Definição de Critérios

Operador	Representação
Igual (=)	equ
Diferente (\neq)	neq
Maior (>)	gre
Maior ou igual (\geq)	geq
Menor (<)	les
Menor ou igual (\leq)	leq

O Tratador de Recursos do Diretório receberá o arquivo XML e converterá em SPARQL para enviar ao Processador Semântico.

Listagem 4.11: SPARQL: Consulta

```
"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#" +
"PREFIX ont: <http://www.owl-ontologies.com/Ontology1251223167.owl#" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#" +
"SELECT ?nodo " +
"WHERE " +
" { ?nodo rdf:type ont:Nodo ." +
" ?nodo ont:Nodo_Status 'Ativo' ." +
" ?nodo ont:Nodo_Base false ." +
" ?nodo ont:Nodo_Celula ?celula ."+
" ?celula ont:Celula_ID 'UCPel' ." +
" ?nodo ont:Nodo_Software ?SO ." +
" ?SO rdf:type ont:Unix ." +
" ?nodo ont:Nodo_Dispositivos ?processador ." +
" ?processador rdf:type ont:Intel }";
```

A consulta representada na listagem 4.11 levou em consideração o perfil “Administrador” especificado no arquivo XML. Esse perfil define que a consulta retornará recursos que o perfil “Administrador” possa utilizar. Serão consideradas as preferências do cliente, instanciadas no modelo ontológico através do módulo Preferências do Usuário. O cliente não será notificado quando o recurso desejado estiver disponível. Esta notificação poderá ocorrer se no momento do *matching* o recurso estiver ativo, mas indisponível. O escopo define a abrangência da consulta. Partindo que a consulta foi originada da célula

chamada “UCPel” a consulta em SPARQL irá pesquisar na ontologia por nodos que possuam instâncias de Sistema Operacional “Unix” e instâncias de Processador “Intel”. O processamento semântico da consulta poderá retornar, além da resposta exata, recursos similares ao solicitado. Essa funcionalidade permite expandir a potencialidade do mecanismo retornando recursos que talvez possam atender a demanda do cliente quando o desejado não estiver disponível.

O Tratador de Recursos possui uma interface para realização de consultas por nodos e periféricos. Esta interface é uma funcionalidade adicional ao mecanismo para permitir ao usuário pesquisar previamente por recursos nas células do EXEHDA antes de ativar uma aplicação.

A interface de consulta EXEHDA-SD apresenta uma barra de menu com as opções: Nodos, Periféricos, Slots, Pref_Nodo, Manutenção, Regras e Preferências. O formulário principal contém os seguintes elementos:

- Perfil:** Menu suspenso com a opção "Todos" selecionada.
- Célula_ID:** Campo de texto para entrada de um ID específico.
- Deduzido:** Campo de texto para entrada de termos deduzidos.
- Local:** Menu suspenso com a opção "Todos" selecionada.
- Filtros de Recursos:**
 - Nodo:** Menu suspenso com "Todos" selecionado.
 - Processador:** Menu suspenso com "Todos" selecionado.
 - Clock:** Campo de texto com o valor "0" e uma unidade de "GHZ".
 - S.O.:** Menu suspenso com "Todos" selecionado.
 - Cores:** Campo de texto com o valor ">=" e "0".
 - Memória:** Campo de texto com o valor ">=" e "0" e uma unidade de "GB".
 - Armazenamento:** Campo de texto com o valor ">=" e "0" e uma unidade de "GB".
 - Sist. arquivos:** Menu suspenso com "Todos" selecionado.
 - Status:** Menu suspenso com "Todos" selecionado.
- Preferência do usuário:** Caixa de seleção desativada.
- Botões:** "Buscar.." e "Fechar".

Figura 4.14: EXEHDA-SD: Consulta por Recursos - Nodos

A pesquisa por nodos, utilizando a interface representada na figura 4.14, é realizada informando os seguintes campos:

- Perfil: define qual o perfil a ser utilizado na pesquisa (administrador, convidado, pesquisador, etc.);
- Célula: deve-se informar o ID da célula que se deseja pesquisar (opcional);
- Deduzido: através deste campo é possível pesquisar por conceitos deduzidos através de regras de inferência. Para realizar pesquisas por este campo é necessário aplicar as regras de inferência no modelo ontológico previamente;
- Nodo: seleciona-se o tipo de nodo (Fixo, Móvel, Base ou Todos);
- Processador. Define-se o tipo de processador desejado (Intel, IBM, AMD ou Todos);
- Clock: através do campo texto informa-se a velocidade desejada do processador, unidade de frequência e operador lógico;
- SO: sistema operacional do Nodo (Unix, Linux, Windows, Todos);

- Cores: número de processadores do nodo é informado no campo texto, assim como o operador lógico;
- Memória: no campo texto é informada a quantidade de memória desejada no nodo, unidade de capacidade e operador lógico a ser utilizado;
- Armazenamento: deve-se informar a quantidade de espaço em disco do nodo, unidade de capacidade e operador lógico;
- Sist.Arquivos: seleciona-se o sistema de arquivos que o nodo deve possuir;
- Status: estado no nodo (Ativo, Inativo ou Todos);
- Preferência do Usuário: deve-se completar as informações sobre preferências na aba “Preferências” e marcar esta opção para considerar as preferências no processo de busca por recursos.

A figura 4.15 apresenta a interface para consulta de periféricos no ambiente ubíquo. Nesta interface é possível selecionar o perfil desejado, célula, tipo de periférico, descrever o modelo desejado e o estado (*status*).

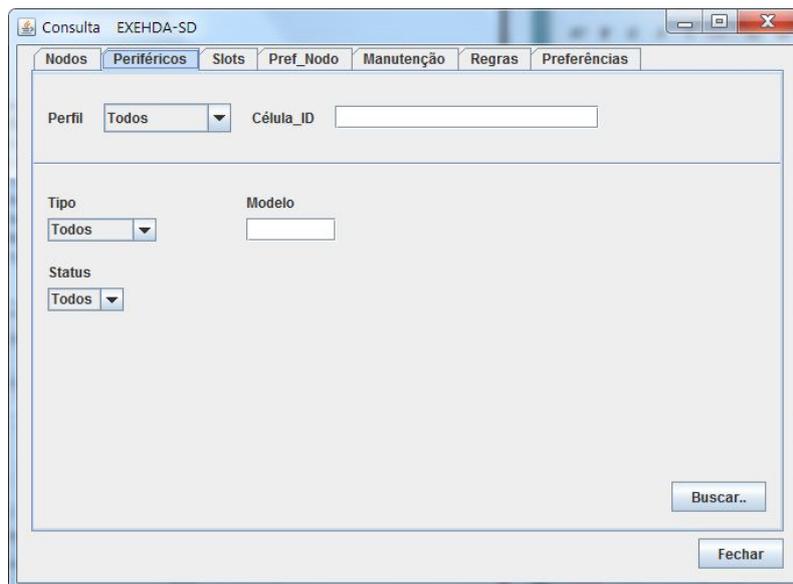


Figura 4.15: EXEHDA-SD: Consulta por Recursos - Periféricos

A interface de consulta disponibilizada pelo Tratador de Recursos também permite especificar regras de inferência para serem utilizadas pelo Processador Semântico.

A figura 4.16 apresenta quatro exemplos de regras para serem interpretadas pelo raciocinador do processador semântico:

- pronto - esta regra possui duas triplas que devem ser verdadeiras para associar o conceito “pronto” a um nodo qualquer. A primeira tripla verifica se um conceito qualquer pertence à classe “Nodo” e a segunda verifica se este mesmo conceito possui o atributo “Nodo_Status” com o valor “Ativo”;

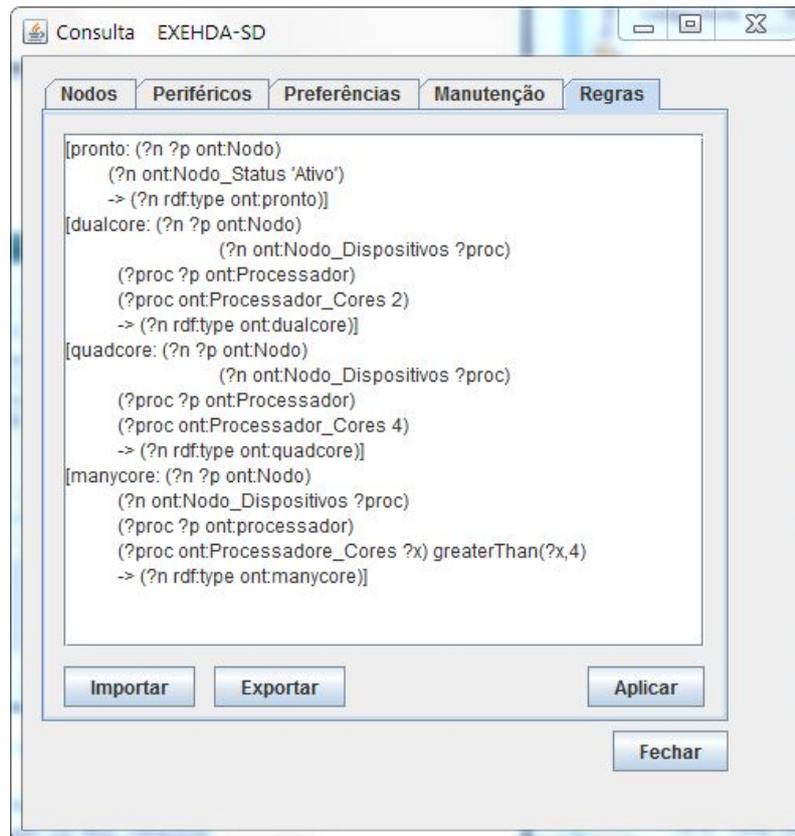


Figura 4.16: EXEHDA-SD: Consulta por Recursos - Regras

- dualcore - associa o conceito “dualcore” a conceitos que pertencem à classe “Nodo”, possuem um relacionamento com a classe “Processador” e o atributo “Processador_Cores” definido na classe “Processador” deve ter valor “2”;
- quadcore - igual à regra dualcore, apenas com o valor do atributo “Processador_Cores” com valor “4”;
- manycore - da mesma forma da regra dualcore, mas o valor do atributo “Processador_Cores” deve ser maior que “4”.

4.5 Preferências de Pesquisa do Usuário

O cliente, representado por uma aplicação ou usuário, pode definir um perfil contendo suas preferências que serão utilizadas no processo de descoberta por recursos. Para a seleção dos critérios de preferência a serem definidos pelo cliente foram considerados os trabalhos de (KORBINIAN FRANK; MITIE, 2007), (SIBERSKI; PAN; THADEN, 2006) e (SILVA, 2007).

O conceito de preferência faz parte do cotidiano de todas as pessoas. Às vezes preferências são expressas em termos de desejos ou objetivos, mas nem sempre elas podem ser totalmente satisfeitas. Essa é a principal característica de preferências: quando não são atendidas totalmente, as pessoas geralmente estão preparadas para alternativas diferentes ou um pouco piores do que a ideal. Isso difere do conceito de restrições nas quais alternativas não são aceitas.

Em termos gerais, existem basicamente duas formas de representar ou modelar preferências: a abordagem numérica ou quantitativa e a abordagem simbólica ou qualitativa.

Na abordagem numérica ou quantitativa, geralmente funções de utilidade calculam o grau ou ranking das alternativas e, com isso, a alternativa de maior valor calculado é escolhida como solução. Esse método fornece um modelo simples para capturar e computar preferências. A abordagem quantitativa também é chamada por alguns autores de ordem total que consiste de um arranjo de objetos ordenados do melhor para o pior. Na literatura podemos encontrar outros termos associados à ordem total, como: ordem completa, ordem simples, ordem linear, entre outros.

Já o enfoque simbólico ou qualitativo representa preferências em termos de ordem parcial e se assemelha mais como realmente as pessoas expressam preferências. Não existem funções de utilidade ou probabilidades associadas às alternativas. Somente se expressa que A é mais importante que B, ou “prefiro A a B”, não mencionando preferência sobre todo o conjunto.

As preferências do usuário são organizadas em duas categorias:

- *Service Discovery Preferences* onde o usuário define suas restrições (Sempre, Nunca, Apenas, etc.). Por exemplo o usuário deseja encontrar impressoras no ambiente ubíquo, mas não deseja impressoras do tipo matricial.
- *Service Selection Preferences* onde o usuário personaliza o resultado da consulta. Isto é feito pelo componente “Seletor” do CD. Exemplos de personalização são: nodos com maior quantidade de memória, com mais espaço em disco, com processador mais veloz, etc.

O EXEHDA-SD possui uma interface, conforme figura 4.17, para o cliente definir suas preferências de pesquisa. O Seletor irá fazer uso das preferências para criar uma relação dos recursos que melhor satisfazem a consulta.

Preferências definidas pelo cliente:

- Tipos de Recursos Preferenciais: o cliente poderá selecionar alguns critérios que serão levados em conta na pesquisa. Estes critérios podem ocorrer sempre ou nunca, conforme desejo do cliente. Os conceitos são referências a classes ou subclasses do modelo ontológico;
- Ordenação de preferência (maior velocidade, maior HD ou maior memória);
- Número de resultados que devem ser retornados: 1 ou n;
- Profundidade da Pesquisa (*hops*): pode ser definido o número de saltos que serão considerados, quando o escopo da pesquisa envolver células vizinhas dinâmicas, ou seja, conexões P2P;
- Regras de inferência definidas pelo cliente para serem utilizadas no *matching*.

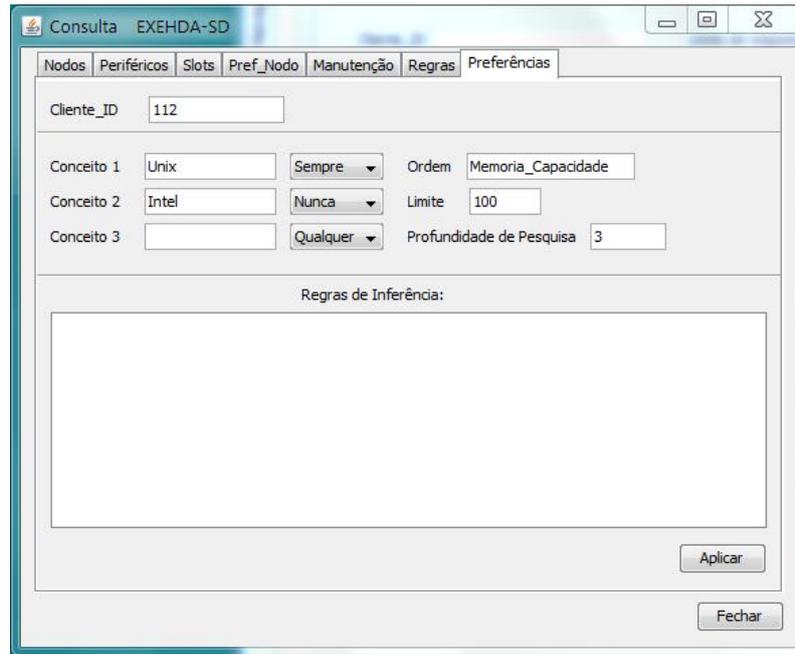


Figura 4.17: EXEHDA-SD: Preferências do usuário

4.6 Ativação do EXEHDA-SD

O perfil de execução do EXEHDA é especificado através de um arquivo XML, conforme a listagem 4.12 contendo os serviços que serão ativados. O serviço EXEHDA-SD será ativado através deste perfil em cada nodo do ambiente. Desta forma será possível definir os parâmetros para descoberta de recursos para o nodo.

Listagem 4.12: XML: Profile de Ativação CC e CR

```
<EXEHDA>
  <PROFILE NAME="DESKTOP-NODE">

    <SERVICE name="cib" loadPolicy="boot">
      <PROP name="impl" value="org.isam.exehda.services.cib.
        CibImplClient"/>
      <PROP name="contactAddress" value="//143.54.7.137/cib"/>
    </SERVICE>

    <SERVICE name="sdcr" loadPolicy="boot">
      <PROP name="impl" value="org.isam.exehda.services.sd.sdrc"/>
      <PROP name="rep" value="/exehdasd"/>
      <PROP name="port" value="3020"/>
      <PROP name="time" value="60"/>
    </SERVICE>

    <SERVICE name="sdcc" loadPolicy="demand">
      <PROP name="impl" value="org.isam.exehda.services.sd.sdcc"/>
      <PROP name="port" value="3021"/>
    </SERVICE>

    . . .

  </PROFILE>
```

</EXEHDA>

O XML apresentado na listagem 4.12 utiliza o formato adotado pelo *middleware* EXEHDA (YAMIN, 2004). Esta listagem contém os dois serviços do EXEHDA-SD que devem ser executados nos nodos. O “sdcr” será executado na inicialização do nodo. As propriedades definidas para este serviço especificam a classe java do serviço, o local onde serão armazenadas as descrições em OWL dos recursos do nodo, a porta de acesso ao CD, e o tempo que será utilizado para renovar o *lease* no CD. Este tempo está definido em segundos.

O serviço “sdcc” ativa o Componente Cliente no nodo. As propriedades descritas no arquivo especificam a classe java e a porta a ser utilizada na conexão com o Componente

O CR poderá ser ativado neste perfil para notificar o CD da presença do nodo na célula, e os recursos agregados ao nodo.

O CC também deve ser ativado para permitir ao nodo processar as consultas para satisfazer as políticas dos componentes das aplicações. Se o CC não estiver ativo no perfil, não será possível realizar as consultas desta forma, apenas pela interface adicional localizada no CD.

4.7 Considerações Sobre o Capítulo

Este capítulo apresentou as características consideradas para modelagem do EXEHDA-SD, bem como sua arquitetura de software. O processador semântico e seus componentes foram descritos em detalhes. A forma como os recursos são descritos e consultados pelo mecanismo foi apresentada em seção específica, assim como, a definição das preferências do usuário e a forma de ativação do mecanismo. No próximo capítulo são apresentados estudos de caso que visam avaliar o EXEHDA-SD.

5 EXEHDA-SD: ESTUDOS DE CASO

Este capítulo apresenta três estudos de caso empregados para avaliar as funcionalidades do EXEHDA-SD. Os estudos de caso foram direcionados ao cenário da empresa EMBRAPA Clima Temperado, localizada em Pelotas, RS. Os estudos de caso envolvem a descoberta de recursos, promovida pelo EXEHDA-SD, considerando a localização do recurso desejado, disponibilidade, perfil e preferências do pesquisador. O processo de descoberta realizado nos cenários utiliza-se de raciocinador para processar os relacionamentos definidos no modelo ontológico.

A escolha da empresa EMBRAPA Clima Temperado deve-se ao fato dela fazer parte do projeto plenUS (*plentiful of Ubiquitous Systems*). O plenUS tem por missão disponibilizar sistemas ubíquos, que explorem relações pró-ativas entre usuários, softwares e equipamentos, com intuito de promover soluções computacionais que contribuam de forma sinérgica no atendimento das atividades fins da Embrapa Clima Temperado.

O EXEHDA-SD irá integrar a arquitetura de software do *middleware* USE, utilizado pelo plenUS.

5.1 Embrapa Clima Temperado

A Embrapa Clima Temperado, Unidade descentralizada da Embrapa, Empresa Brasileira de Pesquisa Agropecuária, vinculada ao Ministério da Agricultura, Pecuária e Abastecimento (MAPA), está localizada em Pelotas, Rio Grande do Sul, Brasil. A Embrapa Clima Temperado é uma Unidade de Pesquisa Eco-regional e desenvolve atividades que buscam viabilizar soluções de pesquisa, desenvolvimento e inovação (PD&I) para a sustentabilidade da agricultura na região de clima temperado, em benefício da sociedade. A empresa pretende ser referência internacional na geração de conhecimentos, tecnologias e inovação para o desenvolvimento sustentável da agricultura de clima temperado. Para atender esses propósitos, a Embrapa Clima Temperado atua em sua Sede, localizada na BR 392 - km 78 e em duas Estações Experimentais, Cascata (EEC), localizada no distrito de Cascata, em Pelotas e Terras Baixas (ETB), localizada no município de Capão do Leão. A Unidade apresenta uma estrutura qualificada de 25 laboratórios. Nos 476.000 km² da área de abrangência da Unidade, nos estados do RS, SC e centro-sul do PR, desenvolvem-se pesquisas nos principais sistemas de produção regionais com ênfase especial às cadeias produtivas dos grãos, frutas, leite e hortaliças e suas interações com o ambiente e o mercado, na busca de sustentabilidade. Ações transversais qualificam os trabalhos de pesquisa como os zoneamentos agroclimáticos e de solos, otimização do uso e qualidade da água e novos formatos tecnológicos, como os sistemas agroecológicos de

produção. Novas oportunidades como agro-energia (biodiesel e etanol) e o estudo de novas rotas de insumos, bem como outros temas e ações de desenvolvimento, fazem parte da matriz de PD&I da Unidade. Os valores que balizam as práticas e os comportamentos da Embrapa e de seus integrantes e representam as doutrinas essenciais e duradouras da empresa, são: excelência em pesquisa e gestão; responsabilidade sócio-ambiental; ética; respeito à diversidade e à pluralidade; comprometimento e cooperação.

Para a execução dos cenários propostos a estrutura física da Embrapa Clima Temperado foi organizada em células, da seguinte forma:

- EMBRAPA_SEDE1 e EMBRAPA_SEDE2 - localizada na BR 392 - km 78;
- EMBRAPA_EEC - Estação Experimental localizada na Cascata;
- EMBRAPA_ETB - Estação Experimental de Terras Baixas, localizada no Capão do Leão.

Estas definições de célula foram instanciadas no modelo ontológico, através do Tratador de Recursos. A figura 5.1 apresenta a organização celular definida para o estudo de caso da empresa EMBRAPA.

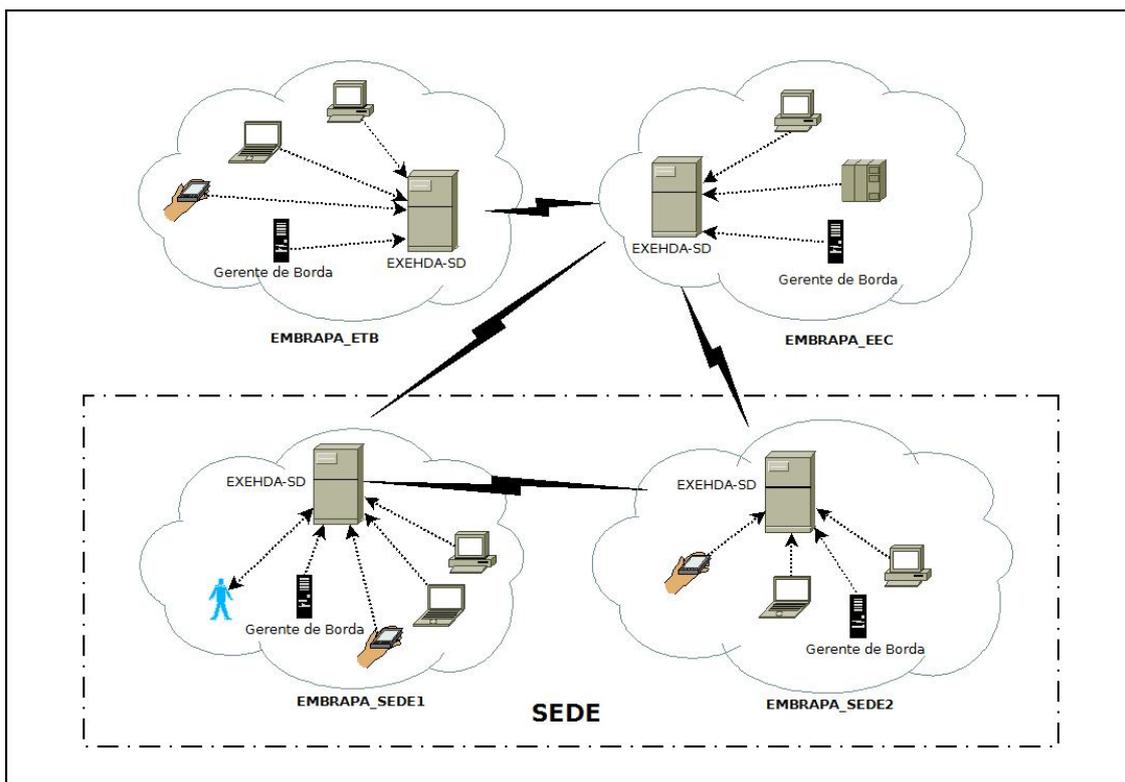


Figura 5.1: CENÁRIO: Organização Celular

5.1.1 Casa de Vegetação

A casa de vegetação é um ambiente monitorado e controlado para realização de experimentos pelos pesquisadores lotados na instituição local e das demais EMBRAPAs do país. Este ambiente é formado por uma estrutura coberta e abrigada artificialmente,

para diferentes tipos de plantas e cultivos, protegendo-os contra os agentes meteorológicos externos; mas também como um meio controlado no qual o crescimento da planta depende de fatores como a água, a iluminação, os fertilizantes, o oxigênio e o dióxido de carbono.

As casas de vegetação possuem sistemas de controle baseados na combinação de hardware e software, devendo ser robusto e integrado ao processo da casa de vegetação considerando-se tanto as estratégias de controle como os sensores adequados ao processo e parâmetros a serem mensurados e sistemas de aquisição e processamento de dados, em tempo real (TERUEL, 2010).

Os sensores compõem um elemento importante nos sistemas de controle, visto que fornecem dados que serão usados para definir a atuação no processo de uma casa de vegetação. Os sensores, bem como os atuadores são conectados aos gerenciadores de borda, cuja finalidade é receber os dados dos sensores e ativar os atuadores.

A seleção dos sensores deve ser criteriosa atendendo às exigências e condições do processo, às características estáticas (como *range*, precisão, exatidão, repetitividade e sensibilidade), e às características dinâmicas, como o tempo de resposta devendo, também, adequar-se aos fenômenos e características dos processos e instalações.

Os parâmetros e processos que devem ser controlados em casa de vegetação são: temperatura, ventilação, umidade relativa do ar, iluminação, concentração de CO₂, carga térmica interna, irrigação, tratamentos químicos, aplicação de nutrientes e o abastecimento de água (NATIONAL GREENHOUSE MANUFACTURERS ASSOCIATION. ENVIRONMENTAL, 2010).

Nos cenários propostos será considerado que as casas de vegetação envolvidas possuam pelo menos os sensores de temperatura e umidade e um atuador para controle da umidade.

Será considerado como recurso a ser descoberto os sensores espalhados entre as casas de vegetação da mesma célula e entre células diferentes.

O espaço disponível para utilização pelo pesquisador dentro da casa de vegetação é chamado de *slot*. Cada casa possui um número limitado de *slots* que são considerados recursos a serem gerenciados pelo EXEHDA-SD.

As casas de vegetação estão representadas por canteiros compostos de sensores e atuadores controlados e administrados por um ou mais gerente de borda. Este gerente tem por finalidade receber a informação dos sensores e ativar os atuadores. Com o (CR) Componente Recurso do EXEHDA-SD ativado no gerente de borda, ele anuncia o estado dos sensores e atuadores conectados a ele. Através das mensagens de anúncio o Tratador de Recursos do (CD) Componente Diretório manterá um repositório dinâmico dos recursos ativos no ambiente.

5.1.2 Tipos de Recursos Envolvidos

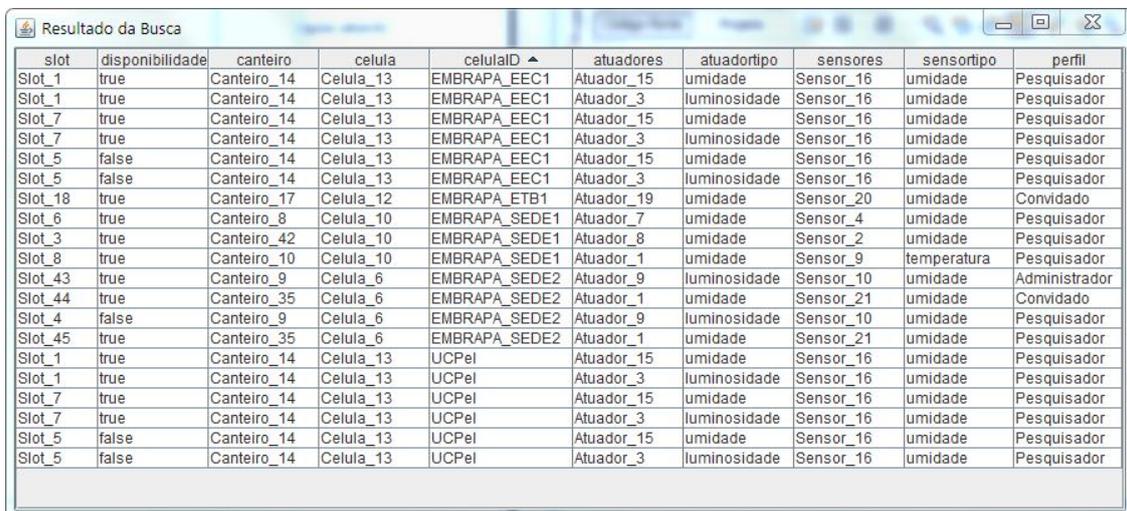
Além dos recursos envolvidos nas casas de vegetação também serão gerenciados os recursos computacionais utilizados na estrutura administrativa da EMBRAPA, tais como: servidores, *desktops*, *notebooks*, *netbooks*, impressoras, *scanners*, etc. Os recursos são instanciados no modelo ontológico através do administrador do ambiente com a interface disponibilizada pelo Tratador de Recursos.

5.1.3 Ambiente de Teste

Para avaliação do EXEHDA-SD foi construído um ambiente de teste composto por:

- Nodo com 2 cores, 2GB RAM, 2GHz, 160GB HD;
- NetBeans IDE 6.5.1 - Pacote para programação em Java com componentes “Swing” para geração de interfaces;
- JDK 1.6 - Linguagem Java;
- Conector Postgresql JDBC4 8.4-701 - Conector do banco de dados Postgresql com a linguagem Java;
- API Jena 2.6.2 - Interface de programação para manipulação de ontologias através da linguagem Java;
- PostgreSQL 8.4 - banco de dados;
- pgAdmin 1.10.2 - interface para manipulação do banco de dados PostgreSQL;
- Protégé 3.4.1 - editor de ontologias.

Para a avaliação do EXEHDA-SD foram instanciados no modelo ontológico os seguintes slots, canteiros, atuadores e sensores espalhados pelas células da EMBRAPA e uma célula fictícia UCPel:



slot	disponibilidade	canteiro	celula	celulaID	atuadores	atuadortipo	sensores	sensortipo	perfil
Slot_1	true	Canteiro_14	Celula_13	EMBRAPA_EEC1	Atuador_15	umidade	Sensor_16	umidade	Pesquisador
Slot_1	true	Canteiro_14	Celula_13	EMBRAPA_EEC1	Atuador_3	luminosidade	Sensor_16	umidade	Pesquisador
Slot_7	true	Canteiro_14	Celula_13	EMBRAPA_EEC1	Atuador_15	umidade	Sensor_16	umidade	Pesquisador
Slot_7	true	Canteiro_14	Celula_13	EMBRAPA_EEC1	Atuador_3	luminosidade	Sensor_16	umidade	Pesquisador
Slot_5	false	Canteiro_14	Celula_13	EMBRAPA_EEC1	Atuador_15	umidade	Sensor_16	umidade	Pesquisador
Slot_5	false	Canteiro_14	Celula_13	EMBRAPA_EEC1	Atuador_3	luminosidade	Sensor_16	umidade	Pesquisador
Slot_18	true	Canteiro_17	Celula_12	EMBRAPA_ETB1	Atuador_19	umidade	Sensor_20	umidade	Convidado
Slot_6	true	Canteiro_8	Celula_10	EMBRAPA_SEDE1	Atuador_7	umidade	Sensor_4	umidade	Pesquisador
Slot_3	true	Canteiro_42	Celula_10	EMBRAPA_SEDE1	Atuador_8	umidade	Sensor_2	umidade	Pesquisador
Slot_8	true	Canteiro_10	Celula_10	EMBRAPA_SEDE1	Atuador_1	umidade	Sensor_9	temperatura	Pesquisador
Slot_43	true	Canteiro_9	Celula_6	EMBRAPA_SEDE2	Atuador_9	luminosidade	Sensor_10	umidade	Administrador
Slot_44	true	Canteiro_35	Celula_6	EMBRAPA_SEDE2	Atuador_1	umidade	Sensor_21	umidade	Convidado
Slot_4	false	Canteiro_9	Celula_6	EMBRAPA_SEDE2	Atuador_9	luminosidade	Sensor_10	umidade	Pesquisador
Slot_45	true	Canteiro_35	Celula_6	EMBRAPA_SEDE2	Atuador_1	umidade	Sensor_21	umidade	Pesquisador
Slot_1	true	Canteiro_14	Celula_13	UCPel	Atuador_15	umidade	Sensor_16	umidade	Pesquisador
Slot_1	true	Canteiro_14	Celula_13	UCPel	Atuador_3	luminosidade	Sensor_16	umidade	Pesquisador
Slot_7	true	Canteiro_14	Celula_13	UCPel	Atuador_15	umidade	Sensor_16	umidade	Pesquisador
Slot_7	true	Canteiro_14	Celula_13	UCPel	Atuador_3	luminosidade	Sensor_16	umidade	Pesquisador
Slot_5	false	Canteiro_14	Celula_13	UCPel	Atuador_15	umidade	Sensor_16	umidade	Pesquisador
Slot_5	false	Canteiro_14	Celula_13	UCPel	Atuador_3	luminosidade	Sensor_16	umidade	Pesquisador

Figura 5.2: CENÁRIO: Slots Instanciados

5.2 Cenário 1 - Pesquisa por Recursos com Regras Inferência Definidas pelo Cliente

Este estudo de caso explora os aspectos escalabilidade, preferências do cliente, perfil de acesso e expressividade na representação e consulta por recursos através do processador semântico.

Neste cenário o pesquisador localizado na célula EMBRAPA_SEDE1 executa uma aplicação, localizada em seu nodo, que tem por objetivo encontrar *slots* livres, em casas de vegetação, que possuam sensor e atuador de umidade.

A listagem 5.1 apresenta a regra de inferência construída pelo cliente para facilitar a consulta por recursos que atendam as características desejadas.

Listagem 5.1: CENÁRIO 1 - Regra do Cliente

```
[pref_cliente1: (? slot ?p ont:Slot)
  (? slot ont:Slot_Canteiro ?canteiro)
  (? canteiro ont:Canteiro_Atuaadores ?atuadores)
  (? atuadores ?p ont:Atuador)
  (? atuadores ont:Atuador_Tipo 'umidade')
  (? canteiro ont:Canteiro_Sensores ?sensores)
  (? sensores ?p ont:Sensor)
  (? sensores ont:Sensor_Tipo 'umidade')
-> (? slot rdf:type ont:rec_pref_cliente1)]
```

A regra cria um novo conceito no modelo ontológico chamado “rec_pref_cliente1”. Este conceito será relacionado a canteiros de vegetação que possuam pelo menos um atuador e um sensor de umidade.

Esta regra definida para o cliente está instanciada no modelo ontológico nas preferências do cliente de ID 12, conforme a figura 5.3.

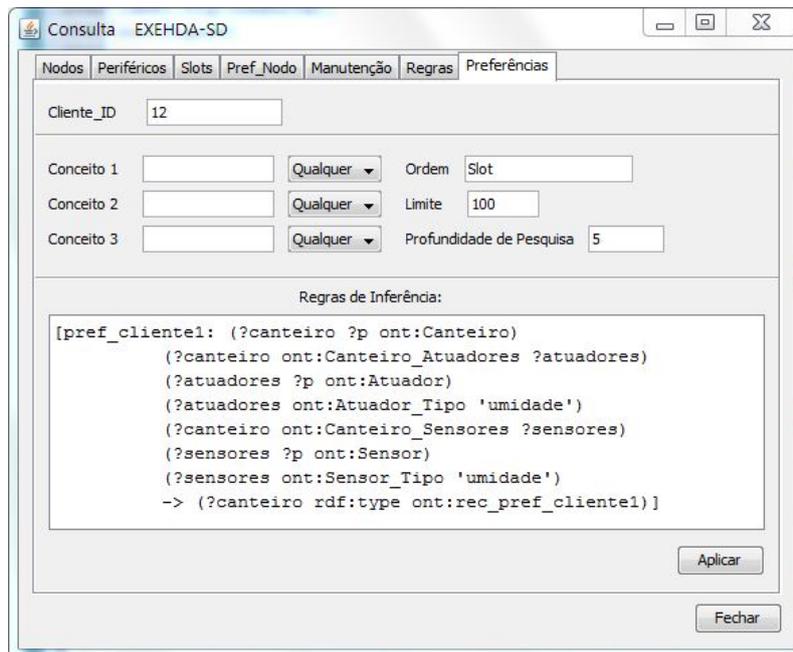


Figura 5.3: CENÁRIO 1: Preferências do Cliente

A figura 5.4 apresenta a interface da aplicação utilizada pelo pesquisador para realizar as consultas. Esta interface permite selecionar o perfil do cliente, a célula que deseja-se pesquisar, o conceito deduzido (criado através de regras de inferência), a situação do *slot* no canteiro de vegetação (livre ou ocupado), local do canteiro, nome do canteiro, tipo de atuador e tipo de sensor.

A especificação da consulta pelo CC em XML é representada pela listagem 5.2.

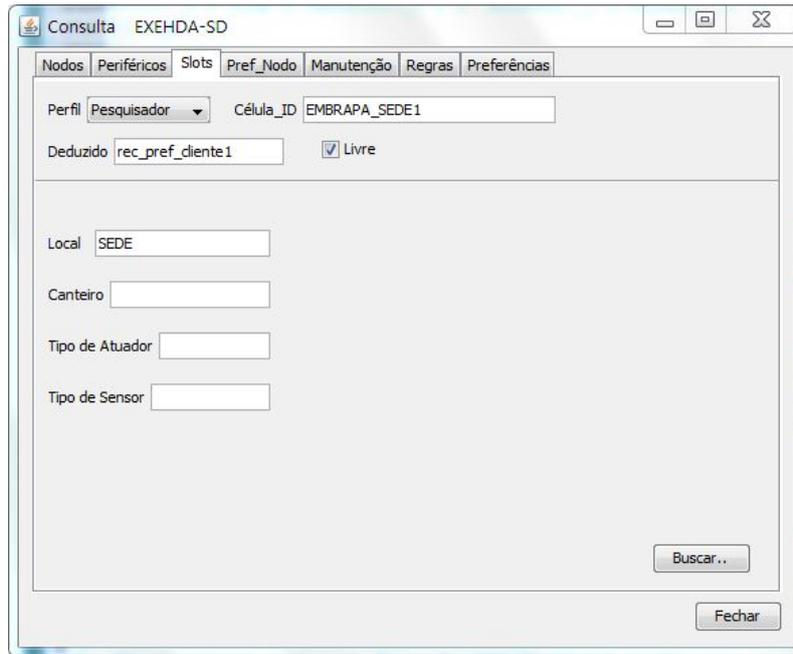


Figura 5.4: CENÁRIO 1: Interface de Consulta por Slots

Listagem 5.2: CENÁRIO 1: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CONSULTA>
  <ID_CLIENTE>12</ID_CLIENTE>
  <PERFIL>Pesquisador</PERFIL>
  <PREF_CLIENTE>S</PREF_CLIENTE>
  <NOTIF_CLIENTE>N</NOTIF_CLIENTE>
  <NOTIF_TEMPO>24</NOTIF_TEMPO>
  <ESCOPO>2</ESCOPO>
  <TIPO>Slot</TIPO>
  <CRITERIO nome="" Canteiro_Instituicao'' valor="" SEDE'' op="" equ
    '' />
  <CRITERIO nome="" Slot_Livre'' valor="" true'' op="" equ'' />
  <CRITERIO nome="" Inferido'' valor="" rec_pref_cliente1'' op=""
    equ'' />
</CONSULTA>
```

Considerando que o pesquisador solicitou uma consulta por slots livres localizados na mesma célula que ele está no momento, utilizando as preferências definidas, inclusive regra de inferência, não deseja ser notificado da futura disponibilidade de recursos.

Ao executar esta consulta através de uma aplicação (12) sendo executada em um nodo localizado na célula EMBRAPA_SEDE1, o CC gera o XML com as informações necessárias para realizar a consulta e envia ao Tratador de Recursos do CD da célula.

O Processador Semântico traduz e processa a consulta de XML para SPARQL, envia para Seletor para aplicar as preferências do usuário e o Tratador de Recursos envia no formato XML o resultado ao CC.

Listagem 5.3: CENÁRIO 1: SPARQL

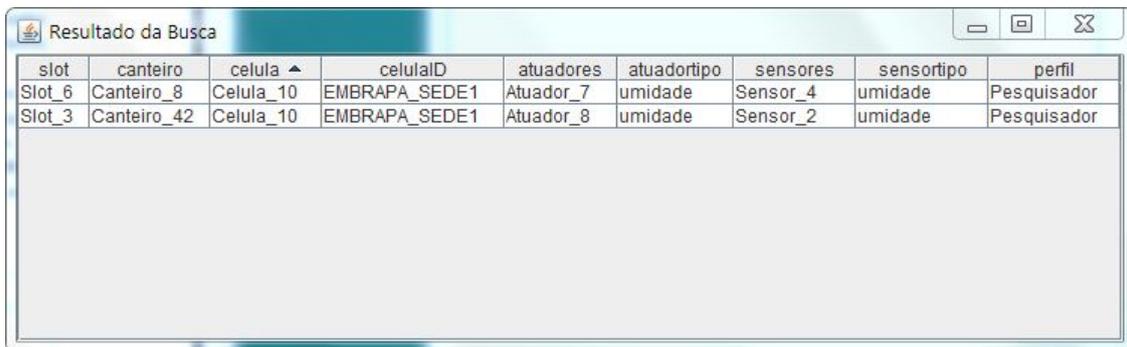
```
"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#" +
"PREFIX ont: <http://www.owl-ontologies.com/Ontology1251223167.owl#" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#" +
```

```

"SELECT ?nodo " +
"WHERE " +
"{ ?slot rdf:type ont:Slot ." +
" ?slot ont:Slot_Canteiro ?canteiro ." +
" ?canteiro ont:Canteiro_Celula ?celula ." +
" ?celula rdf:type ont: Celula ." +
" ?celula ont: Celula_ID 'EMBRAPA_SEDE1' ." +
" ?slot ont:Slot_Livre true ." +
" ?slot ont:Canteiro_Instituicao ont:SEDE ." +
" ?slot rdf:type ont:rec_pref_cliente1 ." +
" ?slot ont:Slot_Perfil 'Pesquisador' }" ;

```

O resultado obtido pelo cliente é o seguinte:

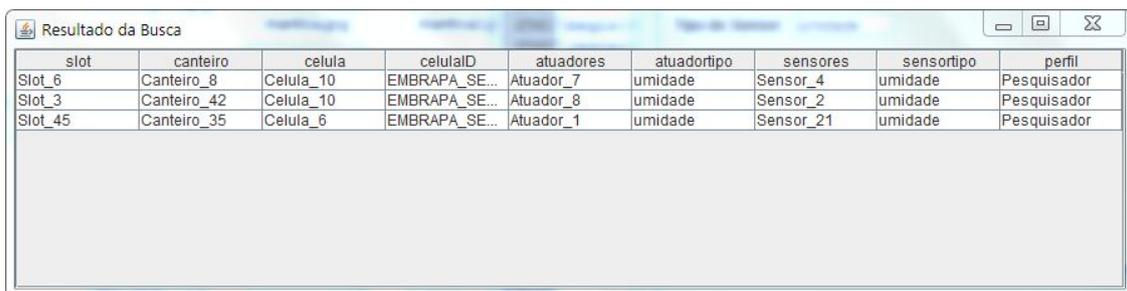


slot	canteiro	celula	celulaID	atuadores	atuadortipo	sensores	sensortipo	perfil
Slot_6	Canteiro_8	Celula_10	EMBRAPA_SEDE1	Atuador_7	umidade	Sensor_4	umidade	Pesquisador
Slot_3	Canteiro_42	Celula_10	EMBRAPA_SEDE1	Atuador_8	umidade	Sensor_2	umidade	Pesquisador

Figura 5.5: CENÁRIO 1: Resultado da Consulta (1)

Os slots 3 e 6 possuem sensores e atuadores de umidade e estão localizados na mesma célula em que o cliente estava no momento da solicitação da consulta.

Alterando o escopo da consulta para utilizar células vizinhas estáticas e dinâmicas (4), será considerada a profundidade de pesquisa definida pelo cliente. Como foi definido o valor “5”, os Comunicadores P2P irão replicar a consulta aos Comunicadores P2P de células vizinhas, até o quinto nível. O Resultado da consulta é apresentado na figura 5.6.



slot	canteiro	celula	celulaID	atuadores	atuadortipo	sensores	sensortipo	perfil
Slot_6	Canteiro_8	Celula_10	EMBRAPA_SE...	Atuador_7	umidade	Sensor_4	umidade	Pesquisador
Slot_3	Canteiro_42	Celula_10	EMBRAPA_SE...	Atuador_8	umidade	Sensor_2	umidade	Pesquisador
Slot_45	Canteiro_35	Celula_6	EMBRAPA_SE...	Atuador_1	umidade	Sensor_21	umidade	Pesquisador

Figura 5.6: CENÁRIO 1: Resultado da Consulta (2)

A mesma consulta realizada alterando sem definir o local dos *slots*. Os Comunicadores P2P enviarão as consultas para serem processadas em outras células e retornam o resultado à célula que originou a consulta. Desta forma o resultado retornará uma quantidade maior de *slots* disponíveis em células vizinhas, conforme a figura 5.7.

O EXEHDA-SD possibilita localizar recursos dispersos entre as células do ambiente ubíquo de forma escalável. Desta maneira o ambiente pode expandir-se de forma

slot	canteiro	celula	celulaID	atuadores	atuadortipo	sensores	sensortipo	perfil
Slot_6	Canteiro_8	Celula_10	EMBRAPA_SEDE1	Atuador_7	umidade	Sensor_4	umidade	Pesquisador
Slot_3	Canteiro_42	Celula_10	EMBRAPA_SEDE1	Atuador_8	umidade	Sensor_2	umidade	Pesquisador
Slot_1	Canteiro_14	Celula_13	UCPel	Atuador_15	umidade	Sensor_16	umidade	Pesquisador
Slot_1	Canteiro_14	Celula_13	EMBRAPA_EEC1	Atuador_15	umidade	Sensor_16	umidade	Pesquisador
Slot_7	Canteiro_14	Celula_13	UCPel	Atuador_15	umidade	Sensor_16	umidade	Pesquisador
Slot_7	Canteiro_14	Celula_13	EMBRAPA_EEC1	Atuador_15	umidade	Sensor_16	umidade	Pesquisador
Slot_45	Canteiro_35	Celula_6	EMBRAPA_SEDE2	Atuador_1	umidade	Sensor_21	umidade	Pesquisador

Figura 5.7: CENÁRIO 1: Resultado da Consulta (3)

dinâmica sem afetar o desempenho do mecanismo de descoberta apresentado neste cenário.

5.3 Cenário 2 - Descoberta e Seleção de Nodos Através de Raciocinadores

Pesquisador deseja rodar uma aplicação que tem como requisito nodos com SO Unix com no mínimo 2 cores, 4GB RAM, 500GB HD, sistema de arquivos EXT4, localizados na sede da Embrapa para processamento dos dados obtidos de uma pesquisa. Caso a pesquisa não retorne nenhum nodo disponível, deseja ser informado da disponibilidade, quando houver.

Para a avaliação deste cenário foram instanciados os seguintes nodos no modelo ontológico:

nodo	cellID	processador	cores	vel	memo	disco	sisarq	so	Local	perfil
Nodo_18	EMBRAPA_SEDE1	Intel_17	6	3	4	320000	EXT4_6	Debian	SEDE	Pesquisador
Nodo_12	EMBRAPA_SEDE2	Intel_21	4	2	4	500000	EXT3_1	Solaris	SEDE	Administrador
Nodo_13	EMBRAPA_EEC1	Intel_21	4	2	4	80000	EXT2_7	AIX	EEC	Convidado
Nodo_14	EMBRAPA_SEDE1	Intel_21	4	2	4	320000	NTFS_11	Debian	SEDE	Administrador
Nodo_7	EMBRAPA_EEC1	Intel_21	4	2	4	500000	EXT3_1	Debian	EEC	Pesquisador
Nodo_9	EMBRAPA_ETB1	Intel_21	4	2	4	320000	NTFS_11	Windows_Vista_Premium	ETB	Pesquisador
Nodo_15	EMBRAPA_SEDE2	Intel_16	2	2	4	160000	EXT3_1	Solaris	SEDE	Pesquisador
Nodo_3	EMBRAPA_SEDE2	Intel_16	2	2	8	200	EXT4_6	Ubuntu	SEDE	Pesquisador
Nodo_1	EMBRAPA_SEDE1	Intel_12	1	2	4	500000	EXT3_1	Solaris	SEDE	Pesquisador

Figura 5.8: CENÁRIO 2: Nodos Instanciados

Listagem 5.4: CENÁRIO 2: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CONSULTA>
  <ID_CLIENTE>11</ID_CLIENTE>
  <PERFIL>Pesquisador</PERFIL>
  <PREF_CLIENTE>S</PREF_CLIENTE>
  <NOTIF_CLIENTE>S</NOTIF_CLIENTE>
  <NOTIF_TEMPO>24</NOTIF_TEMPO>
  <ESCOPO>4</ESCOPO>
  <TIPO>Nodo</TIPO>
  <CRITERIO nome='SO' valor='Unix' op='equ' />
  <CRITERIO nome='cores' valor='2' op='gequ' />

```

```

<CRITERIO nome=' ' Armazenamento_Capacidade' ' valor=' ' 500' ' op=' '
gequ' ' />
<CRITERIO nome=' ' Memoria_Capacidade' ' valor=' ' 4' ' op=' ' gequ' ' />
<CRITERIO nome=' ' Campus' ' valor=' ' SEDE' ' op=' ' equ' ' />
</CONSULTA>

```

Na listagem 5.4 é especificada a consulta em XML.

O Tratador de Recursos, localizado no Componente Diretório, recebe o XML e gera a consulta na linguagem SPARQL para ser executada pelo Processador Semântico.

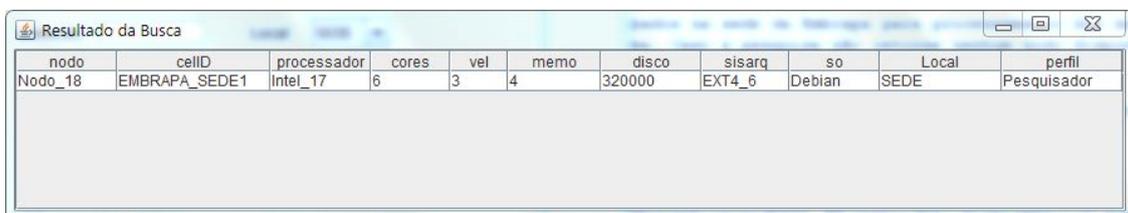
Listagem 5.5: CENÁRIO 2: SPARQL

```

"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
"PREFIX ont: <http://www.owl-ontologies.com/Ontology1251223167.owl#>" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
"SELECT ?nodo " +
"WHERE " +
" { ?nodo rdf:type ont:Nodo ." +
" ?nodo ont:Nodo_Celula ?celula ." +
" ?nodo ont:Nodo_Perfil 'Pesquisador' ." +
" ?nodo ont:Nodo_Software ?SO ." +
" ?SO rdf:type ont:Unix ." +
" ?nodo ont:Nodo_Dispositivos ?processador ." +
" ?processador ont:Processador_Cores ?cores ." +
" ?nodo ont:Nodo_Dispositivos ?memoria ." +
" ?memoria rdf:type ont:Memoria ." +
" ?memoria ont:Memoria_Capacidade ?memo ." +
" ?nodo ont:Nodo_Dispositivos ?armazenamento ." +
" ?armazenamento rdf:type ont:Armazenamento ." +
" ?armazenamento ont:Armazenamento_Capacidade ?disco ." +
" ?nodo ont:Nodo_Local ?local ." +
" ?local rdf:type ont:Campus ." +
" ?local ont:Campus 'SEDE' ." +
" FILTER (?cores >= 2) ." +
" FILTER (?memo >= 4) ." +
" FILTER (?disco >= 500) ." +
" }";

```

O resultado da consulta, considerando o escopo (entre células vizinhas estáticas e dinâmicas), o perfil “Pesquisador”, Preferências do Cliente, Notificação de disponibilidade de recurso é apresentado na figura 5.9.



nodo	cellID	processador	cores	vel	memo	disco	sisarg	so	Local	perfil
Nodo_18	EMBRAPA_SEDE1	Intel_17	6	3	4	320000	EXT4_6	Debian	SEDE	Pesquisador

Figura 5.9: CENÁRIO 2: Resultado da Consulta

Apenas o Nodo 18 atendeu os requisitos da pesquisa solicitada pelo cliente. O EXEHDA-SD retornou uma resposta devido à utilização do raciocinador utilizado na API Jena que inferiu que sistemas operacionais Linux (Debian, Ubuntu, etc.) são um subtipo de Unix.

5.4 Cenário 3 - Exemplo do Fluxo de Dados da Arquitetura do EXEHDA-SD

Este cenário apresenta o fluxo de dados entre os componentes da arquitetura de software do EXEHDA-SD. O fluxo de dados será demonstrado, considerando o cenário 1, a partir da ativação de um gerente de borda, com sensores de temperatura e umidade. Para compreensão da arquitetura serão demonstrados os fluxos de dados que ocorrem durante o anúncio de recursos no ambiente e durante o processo de *matching*.

Ao ser ativado um Gerente de borda, localizado no nodo 33, previamente instanciado pelo administrador do ambiente, é enviada uma mensagem ao CD para anunciar sua presença no ambiente. Este gerente contém 2 sensores ativos. Ao ativar o nodo pela primeira vez é enviado um arquivo com as descrições dos recursos vinculados ao nodo. Estas descrições, através do Processador Semântico, são instanciadas na classe “Recursos” e é criada uma instância na classe “Estados” para controlar o estado ativo (*lease*) do recurso. O recurso deverá renovar seu *lease* através de mensagens enviadas ao Tratador de Recursos localizado no Diretório. O Tratador de Recursos irá decrementar o tempo definido, em segundos, no *lease*. Se o recurso não renovar seu tempo, e este chegar a zero, o Tratador de Recursos irá remover a instância do recurso da classe “Recursos” e a instância da classe “Estados”, tornando o recurso inativo.

Se este Gerente de Borda que está localizado na célula EMBRAPA_SEDE1 for deslocado para a célula EMBRAPA_EEC1 o Descritor Local, localizado no CR enviará o arquivo com as descrições dos recursos do nodo, em OWL, para o CD da nova célula. O Tratador de Recursos da célula EMBRAPA_EEC1 encaminhará as descrições ao Processador Semântico que instanciará no modelo ontológico. Após instanciar o recurso, o Tratador de Recursos controlará o estado através do *lease*.

O Tratador de Recursos da célula EMBRAPA_EEC1 enviará uma mensagem ao Descritor Local, localizado no nodo, para notificar a confirmação do recebimento do arquivo de descrições. A partir deste momento o Controlador de Estado do nodo enviará mensagens ao Diretório para renovar o seu *lease*, tornando o recurso ativo e disponível.

A figura 5.10 apresenta o fluxo de dados envolvendo a entrada de um recurso no ambiente.

As seguintes etapas envolvem o fluxo de dados da figura 5.10:

1. O CR do nodo que contém recursos disponíveis envia ao CD uma mensagem em XML contendo a descrição dos seus recursos em OWL;
2. O Tratador de Recursos do CD recebe o arquivo XML do CR e envia as descrições dos recursos em OWL ao Processador Semântico;
3. O Processador Semântico instancia os recursos no modelo ontológico e notifica o Tratador de Recursos se a operação foi bem sucedida;
4. O Tratador de Recursos notifica o Descritor Local do CR sobre o recebimento das descrições;
5. O Controlador de Estado do CR renova o seu *lease* alocado no CD. Isto é feito após o recebimento da confirmação do instanciamento dos recursos no Diretório.

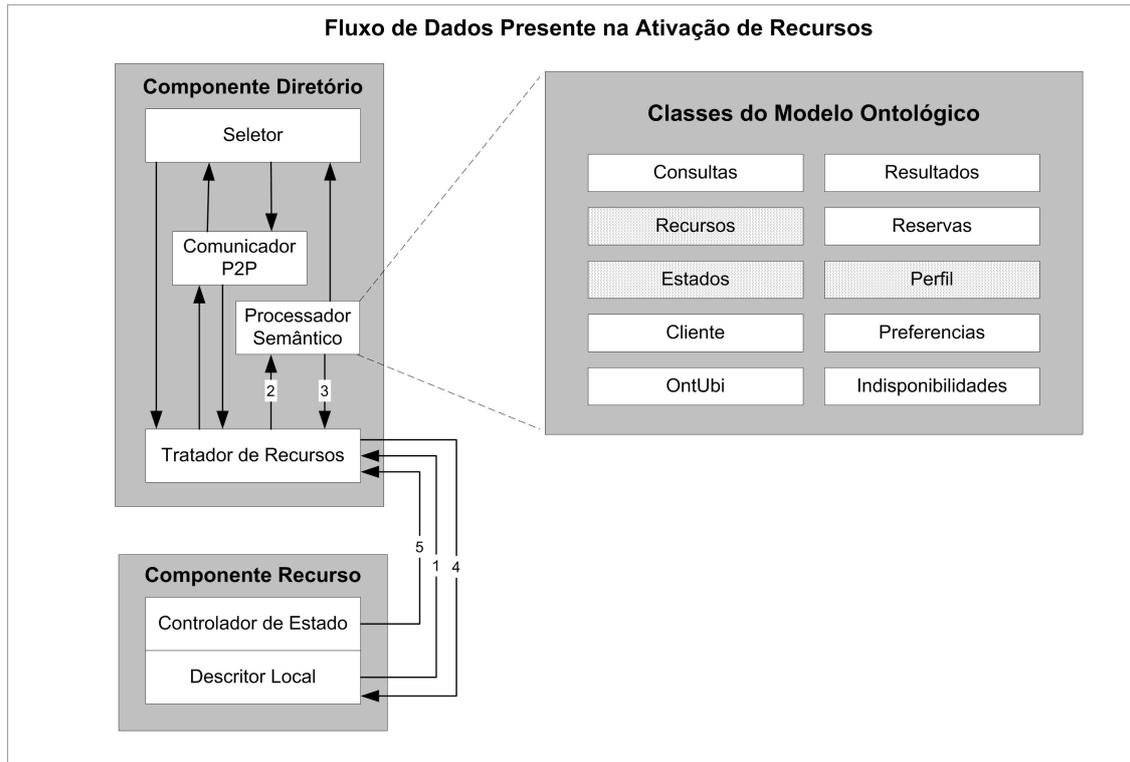


Figura 5.10: CENÁRIO 3: Anúncio de Recursos

Quando um CC envia o XML com a especificação da consulta para o CD, o Tratador de Recursos irá criar uma instância na classe “Consultas” e uma instância na classe “Reservas”, caso tenha sido especificado pelo cliente a notificação de recurso disponível. O Tratador de Recursos irá repassar a consulta ao Processador Semântico que realizará o processamento da consulta na Classe “Recursos”, o Processador Semântico considerará apenas recursos que possuam o mesmo perfil do cliente. O resultado da consulta será instanciado na classe “Resultados”. O Tratador de Recursos enviará o resultado da consulta ao Construtor de Consultas, localizado no Componente Cliente. O fluxo de dados da consulta é demonstrado na figura 5.11.

As seguintes etapas envolvem o fluxo de dados da figura 5.11:

1. O CC envia um arquivo XML com a especificação da consulta ao CD;
2. O Tratador de Recursos interpreta a consulta para SPARQL e repassa uma cópia do XML ao Comunicador P2P (quando o escopo envolver células vizinhas);
3. O Processador Semântico e o Comunicador P2P enviam os resultados obtidos ao Seletor;
4. O Seletor aplica as preferências do cliente e envia o resultado final do *matching* ao Tratador de Recursos;
5. O Tratador de Recursos envia o resultado da pesquisa em XML ao Construtor de Consultas do CC;

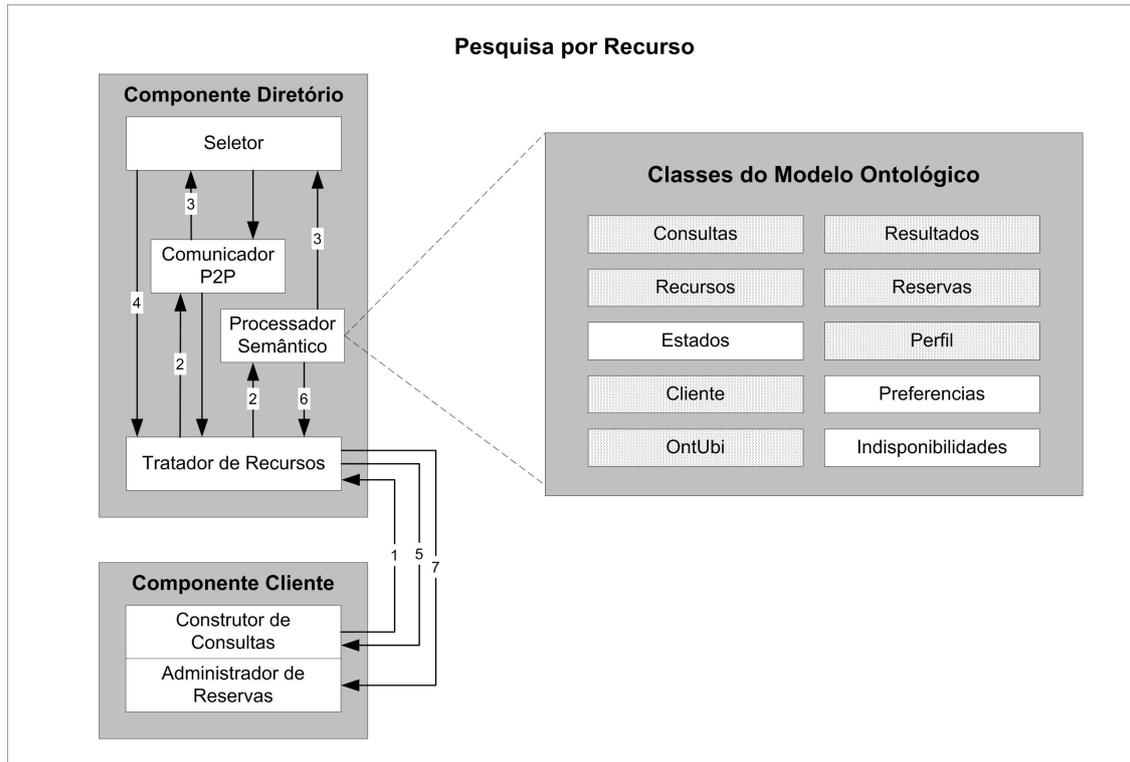


Figura 5.11: CENÁRIO 3: Consulta por Recursos

6. Caso o CC tenha definido no arquivo XML que deseja notificação de recurso disponível, o Processador Semântico realizará o *matching* na classe dinâmica de “Recursos” e se não encontrar recursos que atendam a consulta, repetirá a consulta na classe de recursos estática. Caso encontre o recurso desejado nesta classe criará uma instância na classe “Reservas” que é verificada cada vez que um novo recurso anuncia sua presença no ambiente;
7. No momento em que o recurso desejado torna-se ativo e disponível o Tratador de Recursos envia uma mensagem ao Administrador de Reservas do CC notificando a disponibilidade do recurso desejado.

Caso tenha sido definido pelo cliente que deseja ser notificado da disponibilidade de um recurso, o Tratador de Recursos encaminhará a consulta ao Processador Semântico, e este realizará a consulta na classe “Recursos” (contendo apenas recursos ativos). Caso o resultado da consulta não retorne recursos válidos, a mesma consulta será realizada no repositório de recursos estáticos, localizado na classe “Nodo”. Se nesta segunda consulta encontrar um recurso válido, será instanciado na classe “Reservas” o ID do recurso encontrado, que existe no ambiente, mas no momento está inativo. O Tratador de Recursos irá decrementar o *lease* definido nesta classe, sendo que o padrão adotado é de 24 horas, podendo ser alterado pelo cliente. Se dentro do período válido da reserva, o Recurso for instanciado na classe “Recursos”, o cliente será notificado da presença do recurso no ambiente através de uma mensagem enviada ao Administrador de Reservas, localizado no Componente Cliente. Caso o *lease* definido na classe “Reservas” expirar, o Tratador de Recursos irá remover a instância da classe.

O cliente responsável pela solicitação dos recursos que serão consultados pelo

mecanismo de descoberta pode possuir preferências instanciadas no modelo ontológico. Estas preferências são utilizadas durante o processamento da consulta e após através do módulo Seletor. As preferências do cliente são instanciadas apenas na célula em que foram cadastradas, portanto quando uma pesquisa envolve células vizinhas as preferências não serão consideradas no processamento da consulta, mas serão utilizadas pelo módulo Seletor, localizado na célula origem da consulta, que aplicará as preferências do cliente no conjunto de resultados obtidos.

O administrador do ambiente pode tornar um recurso ativo e disponível, ou seja, instanciado na classe “Recursos” indisponível. Isto é feito através de uma interface que permite selecionar o recurso e definir o período que o recurso estará indisponível. Este procedimento fará com que o recurso, mesmo que ativo, não seja considerado nas respostas das consultas.

As preferências do cliente e o controle da disponibilidade dos recursos são instanciadas no modelo ontológico através das interfaces disponibilizadas pelo CD.

5.5 Considerações Sobre o Capítulo

Neste capítulo foram apresentados três cenários para avaliar o EXEHDA-SD. Estes cenários foram construídos para demonstrar a expressividade na representação e consulta potencializada pelo mecanismo, bem como utilização de regras de inferência e raciocinadores. Os estudos de caso foram modelados para atender as demandas da empresa EMBRAPA relativas à descoberta e localização de recursos em casas de vegetação. O próximo capítulo apresenta um resumo do estudo realizado para a modelagem do EXEHDA-SD, bem como um comparativo do EXEHDA-SD com os trabalhos relacionados, as principais contribuições da pesquisa realizada, as publicações realizadas e os trabalhos futuros.

6 CONSIDERAÇÕES FINAIS

A modelagem do serviço de descoberta de recursos apresentado neste trabalho levou em consideração a revisão bibliográfica dos conceitos envolvidos em Computação Ubíqua, enfatizando a importância dos mecanismos de descoberta de recursos nesses ambientes. Durante o estudo foram relacionados alguns projetos que realizam a abstração do ambiente ubíquo, procurando atingir a transparência e a onipresença contidas na visão de Mark Weiser. O EXEHDA-SD foi modelado para atender as demandas do EXEHDA, quanto à descoberta de recursos, qualificando trabalhos anteriores de descoberta de recursos que realizam o *matching* sintático, apenas.

Para delinear as características essenciais de um mecanismo de descoberta de recursos com suporte à processamento semântico foi realizado um estudo das características básicas, levando em conta a arquitetura, escopo da descoberta, descrição dos recursos, gerenciamento da informação, requisição e anúncio de recursos, armazenamento das informações, métodos de seleção e invocação e suporte à mobilidade. Foi observada a importância da escalabilidade dos mecanismos de descoberta, devido ao contexto ubíquo.

A escolha da linguagem de descrição e consulta de recursos pode tornar o mecanismo de descoberta com maior ou menor expressividade. Muitas linguagens realizam o *matching* através de pares de atributos, comparando igualdades entre valores, limitando o número de respostas da consulta solicitada. Um mecanismo com maior expressividade utiliza linguagens de descrição e consulta com suporte semântico, descrevendo relacionamentos entre conceitos. Isto torna o entendimento de conceitos que estão implícitos inteligíveis pelas máquinas.

As tecnologias para processamento semântico também foram estudadas a fim de compreender a criação e a utilização de ontologias. As ontologias podem ser criadas através de diversas linguagens, entre elas, a OWL é a mais completa. A linguagem OWL contém os relacionamentos entre sujeito, predicado e objeto. Estes relacionamentos tornam o entendimento de conceitos bem mais precisos e organizados dentro do domínio da ontologia.

O processamento da ontologia pode ser realizado por várias ferramentas, este trabalho selecionou a API Jena. Através da API Jena é possível processar as ontologias, consultar classes, subclasses, superclasses e instâncias. Também é possível armazenar a ontologia em banco de dados através da persistência de dados, disponível na API.

A possibilidade de utilizar raciocinadores expande os benefícios das ontologias. Com os raciocinadores é possível inferir conhecimento da ontologia, seja através de regras restritivas, ou através de novos conhecimentos inferidos. A consulta de instâncias da ontologia pode ser feita através da API Jena com a utilização da linguagem SPARQL.

Com a revisão das tecnologias de processamento semântico notou-se a importância do uso de ontologias e suas tecnologias para a realização do *matching* semântico desejado no mecanismo de descoberta.

Neste trabalho foram estudados vários mecanismos de descoberta de recursos. Muitos deles não atendem a heterogeneidade, dinamicidade e escalabilidade necessárias do ambiente ubíquo. A maioria dos mecanismos estudados realizam o *matching* através de pares de atributos, alguns poucos possibilitam a utilização de operadores lógicos, mas mesmo assim possuem uma baixa expressividade na representação e consulta de recursos.

Alguns trabalhos de descoberta de recursos com a utilização de semântica foram apontados aqui, mas a proposta destes mecanismos está direcionada à computação em grade. Alguns implementam o modelo ontológico em DAML, ou realizam consultas através da linguagem TRIPLE ou RQL.

O mecanismo de descoberta de recursos proposto neste trabalho foi projetado para utilizar ontologias escritas em OWL, realizar consultas em SPARQL, processar ontologias com API Jena, inferir conhecimento com raciocinadores da API Jena e utilizar persistência de dados em PostgreSQL.

Durante a fase de testes do processador semântico verificou-se uma demora muito grande no processamento do modelo ontológico ao utilizar a persistência de dados com o banco de dados MySQL. Com a troca do banco de dados para PostgreSQL e seu conector Java, o tempo para processar a mesma consulta foi em torno de dez vezes mais rápido. Por este motivo, optou-se pela troca do banco de dados no decorrer da modelagem, concepção do EXEHDA-SD.

O EXEHDA-SD expande a capacidade de descrever os recursos do ambiente ubíquo promovido pelo *middleware* EXEHDA e maximiza os resultados das consultas através de uma linguagem com grande expressividade. A utilização do modelo ontológico composto pelas ontologias OntUbi e OntSD garantem ao serviço de descoberta uma organização dos conceitos envolvidos na representação do recursos e a possibilidade de inferir sobre novos conceitos através de regras de inferência que podem ser aplicadas no modelo ontológico de forma dinâmica. O EXEHDA-SD foi modelado para ser escalável através de comunicadores P2P entre células, considera as preferências do cliente, utiliza perfis de acesso aos recursos, possibilita o agendamento da disponibilidade dos recursos, notifica o cliente quando um desejado recurso torna-se disponível e controla o estado dos recursos, levando em conta a dinamicidade em que os recursos entram e saem do ambiente.

6.1 Discussão dos Trabalhos Relacionados ao EXEHDA-SD

Dos trabalhos relacionados discutidos foram selecionados para uma comparação com o EXEHDA-SD os que apresentam maior grau de similitude. A discussão completa dos trabalhos relacionados é feita no capítulo três.

No comparativo apresentado na tabela 6.1, verifica-se que o trabalho mais próximo ao EXEHDA-SD é o PerDis (SCHAEFFER, 2005). O PerDis foi modelado para descoberta de recursos em ambientes pervasivos/ubíquos, pois permite a localização em larga escala e considera a dinamicidade em que os recursos entram e saem do ambiente, assim como o EXEHDA-SD utiliza redes *super-peer* para localização de recursos entre células. O PerDis considera as preferências do usuário através do AVU (Ambiente Virtual

Tabela 6.1: Comparativo dos Trabalhos Relacionados com o EXEHDA-SD

Modelos	Descrição	Consulta	Abrangência	Arquitetura	Preferências
Allemand	OWL	RDQL	Grade	3*	-
DReggie	DAML	Prolog	<i>m-commerce</i>	1*	Sim
Ludwig, Santen	DAML-S	LARKS	Grade	3*	-
OMM	TRIPLE	LP	Grade	3*	Sim
PerDis	XML	A/V	Ubíquo	1+3*	Sim
Pernas, Dantas	OWL	-	Grade	3*	-
EXEHDA-SD	OWL	SPARQL	Ubíquo	1+3*	Sim

*1-Diretório Centralizado Plano, 2-Centralizado Hierárquico, 3-Descentralizado

do Usuário) do *middleware* EXEHDA, estas preferências são aplicadas sobre o resultado final da consulta. Estas preferências de usuário especificadas pelo modelo PerDis possui forte dependência do *middleware*, dificultando a utilização do modelo em outros *middlewares*. O EXEHDA-SD possui um módulo próprio dentro do Componente Diretório para gerenciar e aplicar as preferências do usuário, as quais são aplicadas durante e após o processo de *matching*. Um grande diferencial entre o EXEHDA-SD e o PerDis está na representação e consulta por recursos, pois no EXEHDA-SD isto é feito através do Processador Semântico, responsável pela instanciação de recursos no modelo ontológico definido na linguagem OWL e consulta através de linguagem SPARQL, disponibilizada através da API Jena. No PerDis a consulta é realizada através de comparação de operadores lógicos e comparação de atributos e valores, permitindo apenas *matching* sintático, não possui motores de inferência para dedução de recursos semelhantes.

O trabalho de (ALLEMAND, 2006) utiliza ontologias para representação e consulta por recursos, mas foi projetado para *grids*. O controle de recursos ativos no ambiente é feito pelo software Ganglia que gera um repositório de recursos em XML. O mecanismo lê o repositório XML e converte para um repositório semântico para utilizar raciocinadores. Ele não considera as preferências do usuário.

No trabalho de (PERNAS; DANTAS, 2004) é apresentado um modelo para descrição de recursos em ambientes *Grid*. Neste modelo é utilizada a linguagem OWL para descrição, mas depende da ferramenta Protégé para instanciação dos recursos. Não possui uma interface própria para gerenciamento de recursos, como a interface desenvolvida para o EXEHDA-SD. A manipulação dos recursos é feita através das APIs disponibilizadas pela ferramenta Protégé.

A arquitetura de software proposta no modelo de (LUDWIG; SANTEN, 2002) contém um módulo responsável pela descoberta de recursos no ambiente de computação em *Grid*. Este módulo recebe recursos descritos em vários formatos, como XML, LDAP, etc. e atualiza na base de conhecimento ontológico do módulo OMM *MatchMaker*.

Em DReggie, (CHAKRABORTY et al., 2001), foi concebido um modelo para atuar com aplicações de comércio eletrônico em dispositivos móveis. A proposta do DReggie é expandir a capacidade de *matching* utilizada pelo Jini e outros mecanismos de descoberta de recursos. Para isto é utilizada uma linguagem de descrição de recursos expressiva, chamada DAML. DReggie utiliza um raciocinador Prolog para realizar *matching* na base de conhecimento. O EXEHDA-SD utiliza raciocinadores disponíveis na

API Jena e foi modelado para atuar tanto com dispositivos fixos e móveis espalhados pelo ambiente ubíquo.

O EXEHDA-SD também contempla as seguintes características que não foram identificadas nos trabalhos relacionados:

- perfil definido nos recursos e clientes para realizar o *matching*, o resultado da consulta somente retornará recursos ativos que estejam de acordo com o perfil do cliente;
- quando o cliente solicitar notificação de recurso disponível, o mecanismo realiza o *matching* no repositório de recursos ativos, caso não retorne resultados, repete no repositório de recursos estáticos. Se obtiver resposta na consulta armazena o ID do recurso na classe Reservas da OntSD. Quando o recurso desejado tornar-se disponível o mecanismo notificará o cliente de sua existência no ambiente ubíquo;
- todos os recursos disponíveis nos nodos ativos do ambiente ubíquo são compartilhados. Através do agendamento de indisponibilidades do EXEHDA-SD é possível definir períodos de tempo em que o recurso não estará acessível no ambiente.

6.2 Principais Contribuições da Pesquisa Realizada

- participação na criação da OntUbi, ontologia que representa o ambiente ubíquo gerenciado pelo *middleware* EXEHDA;
- adaptação da OntUbi para melhor atender o processo de descoberta de recursos. A modelagem da OntSD e a criação de novas classes e atributos na OntUbi contribuíram para atender as demandas da arquitetura de software do EXEHDA-SD;
- possibilitar uma alta expressividade na descrição e consulta por recursos através da linguagem OWL;
- possibilitar inferências lógicas através de regras definidas pelo cliente e/ou administrador do ambiente. Através das regras é possível ao mecanismo de descoberta inferir novos conhecimentos, expandindo assim a capacidade de atender as requisições por recursos;
- consultas consideram os perfis do cliente e recursos envolvidos. O modelo ontológico foi concebido prevendo que todo recurso possui um perfil de execução, bem como todo cliente está associado a um ou mais perfis;
- preferências do cliente são consideradas no processo de consulta por recursos. As preferências definidas pelo cliente poderão ser consideradas durante o processo de *matching* e após através do módulo “Seletor”;
- possibilidade de agendar períodos em que os recursos não estarão disponíveis ao ambiente. Por padrão, todos os recursos que entram no ambiente ubíquo serão compartilhados. Neste sentido, foi definida uma interface para instanciar no modelo ontológico períodos de indisponibilidade para os recursos. Estes períodos são definidos pelo administrador do ambiente;

- localização de recursos baseada no escopo definido (no próprio nodo, célula local, células vizinhas estáticas e células vizinhas estáticas mais células vizinhas dinâmicas);
- definição do formato de comunicação entre Componente Cliente, Componente Diretório e Componente Recurso;
- implementação e testes do processador semântico do EXEHDA-SD;
- implementação da interface para gerenciar recursos no ambiente;
- implementação da interface para avaliar o processador semântico.

6.3 Publicações Realizadas

- WSCAD 2010: DILLI, Renato, YAMIN, Adenauer, WARKEN, Nelsi, RODRIGUES, Sérgio, VENECIAN, Luthiano. Uma Arquitetura com Suporte Semântico para Descoberta de Recursos na UBICOMP. XI Simpósio em Sistemas Computacionais, 2010, Petrópolis, RJ.
- CLCAR 2010: DILLI, Renato, YAMIN, Adenauer, PALAZZO, Luiz. Descoberta de Recursos com Suporte Semântico na Computação em Grade. Conferencia Latino Americana de Computación de Alto Rendimiento, 2010, Gramado, RS.
- CLEI 2010: DILLI, Renato, YAMIN, Adenauer, PALAZZO, Luiz. Um Modelo de Matching Semântico Direcionado à Descoberta de Recursos na UBICOMP. XXXVI Conferência Latino-americana de Informática, 2010, Asunção, Paraguai.
- SBCUP 2010: DILLI, Renato, SANTIN, Mateus, YAMIN, Adenauer, PALAZZO, Luiz. Explorando Processamento Semântico na Descoberta de Recursos para UBICOMP. II Simpósio de Computação Ubíqua e Pervasiva, 2010, Belo Horizonte, MG.
- ERAD 2010: DILLI, Renato, PALAZZO, Luiz A. M., YAMIN, Adenauer C. EXEHDA-SD: Um Mecanismo para Descoberta de Recursos com Suporte Semântico para UBICOMP. 10a Escola Regional de Alto Desempenho, 2010, Passo Fundo, RS. ERAD 2010 - Fórum de Pós Graduação da 10a Escola Regional de Alto Desempenho, 2010.
- CIC-UCPel 2009: DILLI, Renato, YAMIN, Adenauer C. Uma Contribuição para Descoberta de Recursos na UBICOMP. Congresso de Iniciação Científica. Universidade Católica de Pelotas. Pelotas, RS. Mostra de Pós-Graduação, 2009.
- CIC-UCPel 2009: PEREIRA, Eduardo, YAMIN, Adenauer C., DILLI, Renato. Explorando a Localização de Recursos na Computação Ubíqua. Congresso de Iniciação Científica. Universidade Católica de Pelotas. Pelotas, RS. Iniciação Científica, 2009.
- ERAD 2009: DILLI, Renato, PALAZZO, Luiz A. M., YAMIN, Adenauer C. Avaliando o Uso de Web Semântica na Descoberta de Recursos da Computação Ubíqua.

9a Escola Regional de Alto Desempenho, 2009, Caxias do Sul, RS. ERAD 2009 - Fórum de Pós Graduação da 9a Escola Regional de Alto Desempenho, 2009.

- CIC-UCPel 2008: DILLI, Renato, PALAZZO, Luiz A. M., YAMIN, Adenauer C. Ontologias na Descoberta de Recursos da Computação Pervasiva. Congresso de Iniciação Científica. Universidade Católica de Pelotas. Pelotas, RS. Mostra de Pós-Graduação, 2008.
- WIKI de trabalho: DILLI, Renato. Produção realizada no curso de mestrado em ciência da computação do PPGINF/UCPel. Disponível em <http://olaria.ucpel.tche.br/dilli>. 2010.

6.4 Trabalhos Futuros

O EXEHDA-SD foi modelado para atender a descoberta de recursos no ambiente celular promovido pelo *middleware* EXEHDA. Muitas funcionalidades foram modeladas e algumas implementadas. Para concepção e expansão do modelo, destacam-se os seguintes aspectos a considerar:

- criar um módulo no Componente Diretório, da arquitetura de software, para a interoperabilidade de ontologias. Desta forma será possível que cada célula possua um modelo ontológico diferenciado, mas equivalente;
- separar os demais componentes da arquitetura do Processador Semântico para otimizar o custo computacional. Desta maneira apenas o *matching* seria processado no modelo ontológico, os demais componentes utilizariam Banco de Dados tradicionais;
- agregar novos algoritmos no processo de ranking utilizado pelo módulo Seletor;
- desenvolver uma interface para facilitar a construção de regras de inferência;
- atualizar e expandir o modelo ontológico, visando a atender a instanciação de novos recursos;
- revisar as restrições, principalmente cardinalidades nas classes do modelo ontológico;
- concluir a implementação dos componentes da arquitetura de software;
- integrar o EXEHDA-SD ao *middleware* USE do ambiente ubíquo PlenUS em desenvolvimento pelo G3PD.

REFERÊNCIAS

ALLEMAND, J. N. C. **Serviço Baseado em Semântica para Descoberta de Recursos em Grade Computacional**. 2006. 120p. Dissertação de Mestrado — Departamento de Ciência da Computação, UnB, Brasília, DF.

ARQ - A SPARQL Processor for Jena. Disponível em: <<http://jena.sourceforge.net/ARQ/>>. Acesso em janeiro de 2010.

AUGUSTIN, I.; YAMIN, A.; SILVA, L. Building a Smart Environment at Large-scale with a Pervasive Grid Middleware. In: GRID COMPUTING RESEARCH PROGRESS. 1 ED. NEW YORK, 2008. **Anais...** Nova Science Publishers, 2008. v.1, p.323–344.

BALAZINSKA, M.; BALAKRISHNAN, H.; KARGER, D. INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery. **Lecture Notes in Computer Science**, [S.l.], v.2414, p.195–??, 2002.

BECHHOFFER, S.; HARMELEN, F. van; HENDLER, J.; HORROCKS, I.; MCGUINNESS, D. L.; PATEL-SCHNEIDER, P. F.; STEIN, L. A. **OWL Web Ontology Language Reference**. Disponível em: <<http://www.w3.org/TR/owl-ref/>>. Acesso em junho de 2007.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web. **Scientific American**, [S.l.], v.5, n.284, p.34–43, May 2001.

BIOMEDICAL INFORMATICS RESEARCH, S. C. for. **Protégé**. Disponível em: <<http://protege.stanford.edu/>>. Acesso em maio de 2009.

CHAKRABORTY, D.; PERICH, F.; AVANCHA, S.; JOSHI, A. **DReggie**: Semantic Service Discovery for M-Commerce Applications.

COSTA, C. A. da; YAMIN, A. C.; GEYER, C. F. R. Toward a General Software Infrastructure for Ubiquitous Computing. **IEEE Pervasive Computing**, [S.l.], v.7, n.1, p.64–73, 2008.

DECKER, K.; WILLIAMSON, M.; SYCARA, K. **Matchmaking and Brokering**.

DICKINSON, I. **Jena API - A Semantic Web Framework for Java**. Disponível em: <<http://jena.sourceforge.net/ontology/>>. Acesso em maio de 2009.

FENSEL, D.; HORROCKS, I.; VAN HARMELEN, F. OIL in a Nutshell. In: EUROPEAN WORKSHOP ON KNOWLEDGE ACQUISITION, MODELING AND MANAGEMENT, 12., 2000. **Anais...**, 2000.

GONZALEZ-CASTILLO, J.; TRASTOUR, D.; BARTOLINI, C. **Description Logics for Matchmaking of Services**. [S.l.]: Hewlett Packard Laboratories, 2001. (HPL-2001-265).

GROUP, N. W. **SLPv2**. Disponível em: <http://www.ietf.org/rfc/rfc2608.txt>. Acesso em maio de 2009.

GRUBER, T. A Translation Approach to Portable Ontology Specifications. **Knowledge Acquisition**, [S.l.], p.199–220, 1993.

GUARINO, N. Formal ontology and information systems. In: GUARINO, N. (Ed.). **Formal Ontology in Information Systems**. Amsterdam: IOS Press, 1998. p.3–18.

HANSMANN, U.; MERK, L.; NICKLOUS, M. S.; STOBER, T. **Pervasive Computing: The Mobile World**. 2nd.ed. Berlin, Germany: Springer-Verlag, 2003.

ISAM. **Infra-estrutura de Suporte às Aplicações Móveis**. Disponível em: <http://www.inf.ufrgs.br/isam/index.html>. Acesso em junho de 2009.

JEPSEN, T. C. Just What Is an Ontology, Anyway? **IT Professional**, Los Alamitos, CA, USA, v.11, p.22–27, 2009.

JINI. **Jini**. Disponível em: <http://www.jini.org>. Acesso em maio de 2009.

KLYNE, G.; CARROLL, J. **Resource Description Framework (RDF): Concepts and Abstract Syntax**. Disponível em: <http://www.w3.org/TR/rdf-concepts/>. Acesso em janeiro de 2008.

KORBINIAN FRANK, V. S.; MITIE, J. Personalizable Service Discovery in Pervasive Systems. , [S.l.], 2007.

LIMA, L. **Um Protocolo para Descoberta e Seleção de Recursos em Grades Móveis Ad hoc**. 2007. 213p. Tese (Doutorado em Informática) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ.

LOPES, J. G. **Matching Semântico de Recursos Computacionais em Ambientes de Grade com Múltiplas Ontologias**. 2005. 116p. Dissertação de Mestrado — Departamento de Ciência da Computação, UnB, Brasília, DF.

LOPES, J.; PILLA, M.; YAMIN, A. EXEHDA: a Middleware for Complex Heterogeneous and Distributed Applications. , [S.l.], 2007.

LUDWIG, S. A.; SANTEN, P. V. **A Grid Service Discovery Matchmaker Based On Ontology**.

MARIN-PERIANU, R.; HARTEL, P. H.; SCHOLTEN, J. **A Classification of Service Discovery Protocols**. [S.l.]: Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, 2005. Technical report. (TR-CTIT-05-25).

MCGRATH, R. E. **Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing**. [S.l.: s.n.], 2000.

MIZOGUCHI, R.; VANWELKENHUYSEN, J.; IKEDA, M. Task ontology for reuse of problem solving knowledge. **IOS Press**, [S.l.], p.p. 46–59, 1995.

NATIONAL Greenhouse Manufacturers Association. Environmental. Disponível em: <<http://www.ngma.com/standardpdf/Environmental.pdf>>. Acesso em junho de 2010.

PERNAS, A. M.; DANTAS, M. A. R. Ontologias Aplicadas a Descrição de Recursos em Ambientes Grid. **INFOCOMP Journal of Computer Science**, [S.l.], v.3, n.2, p.26–31, 2004.

RATSIMOR, O.; CHAKRABORTY, D.; JOSHI, A.; FININ, T. Allia: alliance-based service discovery for ad-hoc environments. In: **SECOND ACM INTERNATIONAL WORKSHOP ON MOBILE COMMERCE (WMC-02)**, 2002, New York. **Proceedings...** ACM Press, 2002. p.1–9.

REYNOLDS, D. **Jena 2 Inference support**. Disponível em: <<http://jena.sourceforge.net/inference/index.html>>. Acesso em maio de 2009.

ROBINSON, P.; VOGT, H.; WAGEALLA, W. **Privacy, Security and Trust within the Context of Pervasive Computing (The Kluwer International Series in Engineering and Computer Science)**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2004.

ROBINSON, R. Resource Discovery in Pervasive Computing Environments. , [S.l.], 2006.

SCHAEFFER, A. E. PerDis: um Serviço para Descoberta de Recursos no ISAM Pervasive Environment. **Universidade Federal do Rio Grande do Sul**, [S.l.], 2005.

SEABORNE, A. **RDQL - A Query Language for RDF**. Disponível em: <<http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/>>. Acesso em junho de 2009.

SEABORNE, A. **SPARQL - A Query Language for RDF**. Disponível em: <<http://http://www.w3.org/TR/rdf-sparql-query/>>. Acesso em junho de 2007.

SEABORNE, A.; MANJUNATH, G. **SPARQL/Update - A language for updating RDF graphs**. Disponível em: <<http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/>>. Acesso em janeiro de 2010.

SHARMA A BAWA, S. Resource Discovery using Ontology Approach in Grid. **COIT-2007**, [S.l.], p.63–67, Março 2007.

SIBERSKI, W.; PAN, J. Z.; THADEN, U. Querying the Semantic Web with Preferences. In: **INTERNATIONAL SEMANTIC WEB CONFERENCE**, 2006. **Anais...** Springer, 2006. p.612–624. (Lecture Notes in Computer Science, v.4273).

SILVA, L. **OWLPREF: Uma Representação Declarativa de Preferências Qualitativas para a Web Semântica**. 2007. 126p. Mestrado em Ciência da Computação — UNIFOR, Fortaleza, CE.

SOLDATOS, J.; PANDIS, I.; STAMATIS, K.; POLYMENAKOS, L.; CROWLEY, J. L. Agent based middleware infrastructure for autonomous context-aware ubiquitous computing services. **Journal of Computer Communications**, [S.l.], 2006.

TANGMUNARUNKIT, H.; DECKER, S.; KESSELMAN, C. Ontology-Based Resource Matching in the Grid - The Grid Meets the Semantic Web. In: INTERNATIONAL SEMANTIC WEB CONFERENCE, 2003. **Anais...** Springer, 2003. p.706–721. (Lecture Notes in Computer Science, v.2870).

TERUEL, B. J. Controle automatizado de casas de vegetação: Variáveis climáticas e ferti-gação. **Revista Brasileira de Engenharia Agrícola e Ambiental**, [S.l.], v.14, n.3, p.237–245, 2010.

THAIN, D.; TANNENBAUM, T.; LIVNY, M. Distributed computing in practice: the Condor experience. **Concurrency and Computation: Practice and Experience**, [S.l.], v.17, n.2-4, p.323–356, Feb. 2005.

THOMPSON, M. S. **Service Discovery in Pervasive Computing Environments**. 2006. 135p. Tese (Doctor of Philosophy in Computer Engineering) — Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

TOOLKIT, G. **Globus Toolkit**. Disponível em: <http://www.globus.org>. Acesso em maio de 2009.

UPNP FORUM, M. of the. **UPnP™ Device Architecture 1.1**. Disponível em: <http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture.pdf>. Acesso em maio de 2009.

VERZULLI. **Using the Jena API to Process RDF**. Disponível em: <http://www.xml.com/pub/a/2001/05/23/jena.html>. Acesso em maio de 2009.

WEISER, M. The Computer for the 21st Century. **Scientific American**, [S.l.], v.3, n.265, p.94–104, Setembro 1991.

YAMIN, A. **Arquitetura para um Ambiente de Grade Computacional Direcionado às Aplicações Distribuídas, Móveis e Conscientes do Contexto da Computação Pervasiva**. 2004. 195p. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre, RS.

ZHU, F.; MUTKA, M.; NI, L. Service Discovery in Pervasive Computing Environments. **IEEE Pervasive Computing**, [S.l.], v.4, n.4, p.81–90, 2005.