UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ALBERTO EGON SCHAEFFER FILHO

PerDiS: um Serviço para Descoberta de Recursos no ISAM Pervasive Environment

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

Prof. Dr. Cláudio Fernando Resin Geyer
Orientador

Porto Alegre, setembro de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Schaeffer Filho, Alberto Egon

PerDiS: um Serviço para Descoberta de Recursos no ISAM Pervasive Environment / Alberto Egon Schaeffer Filho. – Porto Alegre: PPGC da UFRGS, 2005.

103 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientador: Cláudio Fernando Resin Geyer.

1. Descoberta de recursos. 2. Serviço. 3. Computação *pervasiva*. 4. Requisitos. 5. Middleware. 6. Escalabilidade. I. Geyer, Cláudio Fernando Resin. II. Título.

AGRADECIMENTOS

Inicialmente, gostaria de fazer um agradecimento especial aos professores, funcionários e colegas do Instituto de Informática, que de uma forma ou de outra contribuiram para o meu desenvolvimento acadêmico e pessoal.

Agradeço ao professor Cláudio Geyer pela excelente orientação que recebi ao longo da minha pesquisa.

Agradeço ao Adenauer Yamin, Iara Augustin, Luciano Cavalheiro e Maurício Moraes pelas incontáveis contribuições, estendendo também o agradecimento aos demais integrantes do nosso grupo de pesquisa.

Agradeço a minha mãe, Adélia Debarba Juchem, por todo o apoio e incentivo dado.

Finalmente, gostaria de agradecer ao meu pai, Alberto Egon Schaeffer, pelo exemplo de dedicação, responsabilidade e caráter (em memória).

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	Application Program Interface
AVA	Ambiente Virtual da Aplicação
AVU	Ambiente Virtual do Usuário
BDA	Base de Dados pervasiva das Aplicações
CIB	Cell Information Base
DB	Discoverer Base
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DP	Discoverer Proxy
EXEHDA	EXecution Environment for High Distributed Applications
GPS	Global Positioning System
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
ISAM	Infra-estrutura de Suporte às Aplicações Móveis
JLS	Jini Lookup Service
JVM	Java Virtual Machine
LC	Lookup Component
P2P	Peer-to-Peer
PDA	Personal Digital Assistent
PerDiS	Pervasive Discovery Service
RC	Resource Component
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SLP	Service Location Protocol
SOAP	Simple Object Access Protocol
SSDP	Simple Service Discovery Protocol

UC	User Component
UPnP	Universal Plug and Play
URL	Universal Resource Locator
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSLT	eXtensible Stylesheet Language Transformation

LISTA DE FIGURAS

LISTA DE QUADROS

RESUMO

Estratégias para descoberta de recursos permitem a localização automática de dispositivos e serviços em rede, e seu estudo é motivado pelo elevado enriquecimento computacional dos ambientes com os quais interage-se. Essa situação se deve principalmente à popularização de dispositivos pessoais móveis e de infra-estruturas de comunicação baseadas em redes sem-fio. Associado à rede fixa, esse ambiente computacional proporciona um novo paradigma conhecido como computação *pervasiva*.

No escopo de estudo da computação *pervasiva*, o Grupo de Processamento Paralelo e Distribuído da Universidade Federal do Rio Grande do Sul desenvolve o projeto ISAM. Este engloba frentes de pesquisa que tratam tanto da programação de aplicações *pervasivas* como também do suporte à execução dessas. Esse suporte é provido pelo *middleware* EXEHDA, o qual disponibiliza um conjunto de serviços que podem ser utilizados por essas aplicações ou por outros serviços do ambiente de execução. Essa dissertação aborda especificamente o *Pervasive Discovery Service* (*PerDiS*), o qual atua como um mecanismo para descoberta de recursos no ambiente *pervasivo* proporcionado pelo ISAM. A concepção do *PerDiS* baseou-se na identificação dos principais requisitos de uma solução para descoberta de recursos apropriada para utilização em um cenário de computação *pervasiva*.

Resumidamente, os requisitos identificados nessa pesquisa e considerados pelo *PerDiS* tratam de questões relacionadas aos seguintes aspectos: *a)* utilização de informações do contexto de execução, *b)* utilização de estratégias para manutenção automática da consistência, *c)* expressividade na descrição de recursos e critérios de pesquisa, *d)* possibilidade de interoperabilidade com outras estratégias de descoberta, *e)* suporte à descoberta de recursos em larga-escala, e *f)* utilização de preferências por usuário.

A arquitetura *PerDiS* para descoberta de recursos utiliza em sua concepção outros serviços disponibilizados pelo ambiente de execução do ISAM para atingir seus objetivos, e ao mesmo tempo provê um serviço que também pode ser utilizado por esses. O modelo proposto é validado através da implementação de um protótipo, integrado à plataforma ISAM. Os resultados obtidos mostram que o *PerDiS* é apropriado para utilização em ambientes *pervasivos*, mesmo considerando os desafios impostos por esse paradigma.

Palavras-chave: Descoberta de recursos, Serviço, Computação *pervasiva*, Requisitos, Middleware, Escalabilidade.

***PerDiS:* a Resource Discovery Service in the ISAM Pervasive Environment**

ABSTRACT

Resource discovery strategies enable automatic location of devices or services on a network, and the investigaton of such strategies is required due to the high computational enhancement of the environments which we interact with. This situation arises from the popularization of personal mobile devices and wireless-based communication infrastructures. Associated with the fixed network, this computational environment provides a new paradigm known as pervasive computing.

In the scope of study of pervasive computing, the Parallel and Distributed Processing Group of the Federal University of Rio Grande do Sul has developed the ISAM project. It encompasses reasearch activities that deal with pervasive applications programming, as well as their execution support. This support is provided by the EXEHDA middleware, which offers a set of services to applications or other services. This dissertation specifically addresses the *Pervasive Discovery Service* (*PerDiS*), which acts as a resource discovery mechanism in the ISAM pervasive environment. The *PerDiS* conception was based on the identification of the main requirements for a resource discovery solution suitable for use in a pervasive computing scenario.

Briefly, the requirements identified in this research and considered by *PerDiS* deal with issues related to the following aspects: *a)* use of execution context information, *b)* use of automatic strategies for consistency maintenance, *c)* expressiveness in resource and search criteria description, *d)* possibility of interoperability with other discovery approaches, *e)* large-scale resource discovery support, and *f)* use of user preferences.

The *PerDiS* resource discovery architecture uses other services of the ISAM execution environment for achieving its goals, and at the same time provides a service which can be also used by them. The proposed model is validated through the implementation of a prototype, integrated with the ISAM platform. The obtained results have shown that *PerDiS* is suitable for use in a pervasive environment, even considering the challenges imposed by this paradigm.

Palavras-chave: Resource discovery, Service, Pervasive computing, Requirements, Middleware, Scalability.

1 INTRODUÇÃO

Este capítulo tem como objetivo fazer uma abordagem inicial ao trabalho desenvolvido. Inicialmente, são caracterizados o problema e o escopo de pesquisa no qual o trabalho está inserido. Após, é feita uma descrição das principais contribuições do autor e, por fim, a estrutura do texto é apresentada.

1.1 O problema

A pesquisa de estratégias para descoberta de recursos motiva-se no crescente enriquecimento computacional dos ambientes com os quais interagimos. Esse aumento da complexidade deve-se principalmente ao surgimento de avançados dispositivos pessoais móveis e da popularização de infra-estruturas de comunicação sem-fio (FRIDAY; DAVIES; CATTERALL, 2001) que, integradas à infra-estrutura da rede fixa, definem um novo ambiente computacional, conhecido como computação *pervasiva*. Em um ambiente *pervasivo*, uma ampla variedade de recursos pode estar em uso por muitas pessoas, a todo momento (CHALMERS; DULAY; SLOMAN, 2004a). Adicionalmente a recursos computacionais típicos (poder de processamento e armazenamento, por exemplo), dispositivos sem-fio disponibilizam novas funcionalidades, como câmeras digitais, microfones e receptores GPS, entre outros recursos (MCKNIGHT, 2004). Nesse cenário, mecanismos para descoberta de recursos permitem a localização automática de dispositivos ou serviços disponibilizados em rede.

O projeto ISAM (2004) objetiva explorar alternativas para a modelagem e execução de aplicações *pervasivas*, que em sua visão são aplicações móveis, distribuídas e adaptativas ao contexto por natureza. Sua arquitetura de *software* disponibiliza ferramentas para a programação dessas aplicações e gerencia o ambiente *pervasivo*, chamado ISAMpe, através de um *middleware* adaptativo baseado em serviços (YAMIN et al., 2003). Dentre os serviços que compõem o *middleware*, destaca-se o serviço de descoberta de recursos, tema do presente trabalho, destinado a descobrir automaticamente os dispositivos e serviços que integram o ISAMpe.

Nesta dissertação é apresentado o serviço *PerDiS* (*Pervasive Discovery Service*) para descoberta de recursos. A concepção do *PerDiS* baseou-se na identificação dos principais requisitos de uma solução para descoberta de recursos apropriada para utilização em um cenário de computação *pervasiva*. Em relação a essas questões, enquanto algumas já são tratadas pelas estratégias de descoberta existentes atualmente, outras são abordadas apenas parcialmente ou até mesmo não consideradas por tais protocolos. A proliferação de pesquisas na área de descoberta de recursos confirma que as soluções comerciais existentes atualmente não tratam suficien-

temente bem as necessidades impostas pelos dispositivos móveis (BELLAVISTA; CORRADI; STEFANELLI, 2002).

Os requisitos identificados nessa pesquisa e considerados pelo *PerDiS* tratam dos seguintes aspectos:

- *Utilização de informações do contexto de execução:* devido à imprevisibilidade das condições de execução, a necessidade de adaptar o comportamento do serviço de descoberta em relação a determinadas alterações no contexto deve ser considerada;

- *Utilização de estratégias para manutenção automática da consistência:* em um ambiente altamente dinâmico, a manutenção da consistência do catálogo de recursos é necessária para permitir identificar quais recursos estão realmente disponíveis na rede, e quais estão desconectados ou temporariamente inacessíveis;

- *Expressividade na descrição de recursos e critérios de pesquisa:* conforme a quantidade e variedade dos recursos disponíveis aumenta, maior é a necessidade de um mecanismo que permita expressar claramente as características de um recurso;

- *Possibilidade de interoperabilidade com outras estratégias de descoberta:* a utilização de mecanismos que habilitem uma futura interoperabilidade com outras soluções de descoberta deve ser considerada, uma vez que dificilmente haverá uma solução única, apropriada para todas as situações e ambientes;

- *Suporte à descoberta de recursos em larga-escala:* em um ambiente de computação *pervasiva*, especialmente no ambiente *pervasivo* provido pela infraestrutura ISAM, o suporte à descoberta de recursos geograficamente dispersos é uma característica fundamental;

- *Utilização de preferências por usuário:* finalmente, a personalização do serviço de descoberta por usuário deve ser considerada, a fim de permitir uma diminuição da interação do usuário com o serviço, e adaptar o funcionamento do *PerDiS* de acordo com suas preferências pessoais.

A principal motivação desse trabalho é a carência de um mecanismo para descoberta de recursos apropriado para um cenário de computação *pervasiva*, considerando os novos requisitos e desafios impostos por esse ambiente computacional. Uma etapa fundamental no desenvolvimento desse trabalho foi a pesquisa e a identificação de tais necessidades.

A arquitetura *PerDiS* para descoberta de recursos utiliza em sua concepção outros serviços disponibilizados pelo ambiente de execução do ISAM para atingir seus objetivos, e ao mesmo tempo provê um serviço que também pode ser utilizado por esses. O modelo proposto foi validado através da implementação de um protótipo, integrado à plataforma ISAM.

1.2 Objetivos

Os objetivos que guiaram o desenvolvimento desse trabalho são descritos a seguir.

1.2.1 Objetivo geral

O objetivo geral desse trabalho é desenvolver uma solução para descoberta de recursos apropriada para utilização em ambientes *pervasivos*.

1.2.2 Objetivos específicos

- Pesquisar a bibliografia existente relacionada ao tema descoberta de recursos;
- Identificar os requisitos necessários a uma solução para descoberta de recursos na computação *pervasiva*;
- Modelar e implementar uma solução para descoberta de recursos como um serviço da plataforma ISAM;
- Avaliar o funcionamento do serviço implementado, integrado ao *middleware* do ISAM.

1.3 Escopo de pesquisa do trabalho

O desenvolvimento desse trabalho ocorreu no escopo de pesquisa do projeto ISAM (*Infra-estrutura de Suporte às Aplicações Móveis*), em desenvolvimento pelo Grupo de Processamento Paralelo e Distribuído da Universidade Federal do Rio Grande do Sul. O principal objetivo do projeto é explorar alternativas que facilitem a tarefa de projetar aplicações distribuídas para ambientes *pervasivos*. O ISAM provê uma plataforma integrada, da linguagem ao ambiente de execução, para construir e executar esse tipo de aplicação.

O ISAM deu o suporte necessário para a pesquisa, provendo uma plataforma de computação *pervasiva* que disponibilizou a infra-estrutura para o desenvolvimento desse trabalho. A instanciação do *PerDiS* como um dos serviços do *middleware* do ISAM permitiu sua interação com outros serviços já existentes no *middleware*, aumentando as potencialidades do *PerDiS* e ao mesmo tempo reduzindo sua complexidade.

1.4 Contribuição do autor

Como contribuição principal do autor destaca-se a modelagem e a implementação de uma solução para descoberta de recursos que considera os requisitos impostos pela computação *pervasiva*, no escopo do projeto ISAM. A concepção da solução foi fundamentada na identificação desses requisitos, a qual constituiu uma etapa preliminar dos esforços compreendidos por essa pesquisa.

1.5 Estrutura da dissertação

O restante desse texto está organizado da seguinte forma:

- o capítulo 2 descreve brevemente o projeto ISAM, escopo de pesquisa desse trabalho, e sua relação com a computação *pervasiva*;
- o capítulo 3 apresenta uma breve introdução ao tema *descoberta de recursos*, descrevendo algumas definições e princípios básicos relacionados ao assunto, e

uma compilação dos principais requisitos identificados como necessários para uma solução de descoberta de recursos na computação *pervasiva*, que atenda às exigências da arquitetura ISAM;

- o capítulo 4 descreve o modelo *PerDiS* para descoberta de recursos, sua arquitetura e como cada um de seus principais componentes aborda os requisitos identificados;
- o capítulo 5 apresenta os principais aspectos relacionados à implementação do *PerDiS*, um estudo de caso em um cenário real e os resultados obtidos através de testes e medições com o protótipo;
- o capítulo 6 caracteriza alguns trabalhos relacionados e faz uma comparação de funcionalidades desses com o *PerDiS*;
- finalmente, o capítulo 7 apresenta as conclusões dessa pesquisa e algumas sugestões de atividades a serem desenvolvidas como trabalhos futuros.

2 A COMPUTAÇÃO PERVASIVA E O PROJETO ISAM

Este capítulo tem por objetivo apresentar as principais características e conceitos relacionados à computação *pervasiva*, bem como algumas das questões de pesquisa encontradas nessa área. Em seguida, o projeto ISAM será introduzido, sendo apresentada sua relação com a computação *pervasiva*.

2.1 Caracterizando a computação pervasiva

Conceitualmente, a computação *pervasiva* é caracterizada pela existência de ambientes saturados de capacidades tecnológicas que, integradas de tal forma ao cotidiano de seus usuários, acabam por tornarem-se indistinguíveis (WEISER, 1991). Um dos aspectos fundamentais encontrados na computação *pervasiva* corresponde à noção de *Smart Space*, o qual seria uma área bem definida cuja infra-estrutura computacional estaria embutida e integrada com a própria construção de forma a possibilitar a interação entre os dois ambientes (computacional e real) (SATYANARAYANAN, 2001). Destacam-se também como características da computação *pervasiva* a importância da pró-atividade por parte do ambiente e a capacidade de combinar diferentes formas de conhecimento sobre o sistema, de modo a auxiliar ou tomar decisões que beneficiem o usuário.

2.2 Áreas relacionadas

A computação *pervasiva* corresponde ao terceiro passo em uma linha de evolução que se iniciou na metade dos anos 70 com os sistemas distribuídos e foi seguida com o surgimento da computação móvel no início dos anos 90 (SATYANARAYANAN, 2001). Alguns dos problemas encontrados na computação *pervasiva* já foram identificados e estudados previamente ao longo dessa evolução. Em alguns casos, as soluções existentes podem ser aplicadas diretamente, mas em outros, soluções novas devem ser desenvolvidas a fim de tratar aspectos característicos da computação *pervasiva*. Além disso, ainda existem novos problemas introduzidos por essa área de pesquisa, os quais não foram estudados ou considerados anteriormente. A seguir, as etapas da evolução que precedem a computação *pervasiva* serão brevemente descritas.

2.2.1 Sistemas distribuídos

A área de pesquisa correspondente aos sistemas distribuídos posiciona-se na intersecção entre computadores pessoais e redes locais (LANs). Essa pesquisa, que iniciou-se na metade dos anos 70 e se estendeu até o início dos anos 90, criou um *framework* conceitual e uma base algorítmica de grande valor em todos os trabalhos envolvendo dois ou mais computadores conectados por uma rede. Muitas das questões que são fundamentais para a computação *pervasiva* foram abordadas pela pesquisa em sistemas distribuídos e são bastante discutidas em livros técnicos específicos (TANENBAUM, 2002; LYNCH, 1996; MULLENDER, 1993; COULOURIS; DOLLIMORE; KINDBERG, 2001). Pode-se destacar entre essas questões:

- *Comunicação remota:* aborda questões como protocolos em camadas, invocação remota de métodos (RMI) (HENRY, 2000) e chamada remota de procedimentos (RPC) (BIRRELL; NELSON, 1984), e utilização de *timeouts*;
- *Tolerância a falhas:* tratamento de transações atômicas, distribuídas ou aninhadas, protocolo de *commit* de duas fases (GRAY; REUTER, 1993), e questões relacionadas à alta disponibilidade, como controle de réplicas e recuperação de falhas (JALOTE, 1994; ANDERSON; LEE, 1981);
- *Acesso a informações remotas:* aspectos como manutenção de *caches*, sistemas de arquivos distribuídos (SATYANARAYANAN, 1989) e bancos de dados distribuídos (OZSU; VALDURIEZ, 1998);
- *Segurança:* aborda aspectos como autenticação mútua baseada em encriptação e privacidade (POPEK; KLINE, 1979; NEEDHAM; SCHROEDER, 1978).

2.2.2 Computação móvel

A computação móvel iniciou-se com o surgimento de computadores *laptops* e das redes sem-fio no início dos anos 90, e trouxe uma série de novos desafios aos pesquisadores que tentavam construir um sistema distribuído com clientes móveis. Embora muitos dos princípios de projeto de um sistema distribuído continuassem a ser aplicados, os ambiente móveis introduziram algumas limitações que forçaram o desenvolvimento de técnicas especializadas. Essas limitações podem ser resumidas por *variação imprevisível na qualidade da rede*, *menor confiança e robustez dos elementos móveis*, *limitações dos recursos locais (peso/tamanho)* e *consumo de energia* (SATYANARAYANAN, 1996).

A computação móvel ainda é um campo em evolução, porém livros técnicos recentes já abordam parcialmente o conhecimento envolvido nesse tipo de pesquisa (B'FAR, 2004; MALLICK, 2003; UMAR, 2004; MAHGOUB; ILYAS, 2004). As principais áreas de pesquisa encontradas na computação móvel são as seguintes:

- *Redes móveis:* aborda aspectos como Mobile IP (BHAGWAT; PERKINS; TRIPATHI, 1996), protocolos *ad hoc* (ROYER; TOH, 1999) e técnicas para melhoramento do desempenho do protocolo TCP em redes sem-fio (BAKRE; BADRINATH, 1995);
- *Acesso móvel a informações:* operações desconectadas (HOLLIDAY; AGRAWAL; ABBADI, 2002; KISTLER; SATYANARAYANAN, 1992), acesso a arquivos usando largura de banda adaptativa (MUMMERT; EBLING; SATYANARAYANAN, 1995) e controle seletivo de consistência de dados (TERRY et al., 1995);

- *Adaptação de aplicações:* utilização de *proxies* e gerenciamento adaptativo de recursos (AUGUSTIN et al., 2002; YAMIN et al., 2002; NOBLE et al., 1997);

- *Técnicas para redução de consumo de energia:* adaptação conforme a disponibilidade de energia (FLINN; SATYANARAYANAN, 2004, 1999), na velocidade do processador (HSU; KREMER, 2003; WEISER et al., 1994) e no gerenciamento de memória (LI et al., 2004; LEBECK et al., 2000);

- *Sensibilidade à localização:* trata questões como sensoriamento de localização e comportamento consciente de localização (ABRAHAM; DOLEV; MALKHI, 2004; KAASINEN, 2003).

2.3 Questões de pesquisa consideradas pela computação pervasiva

Ao passo que os ambientes *pervasivos* tornam-se uma realidade, é necessário pensar como eles serão programados. A programação em rede requisita a constante renovação dos modelos de computação, como o cliente-servidor, transações, objetos distribuídos, *web services*, operações desconectadas e grades computacionais. Assim como ocorreu com esses modelos, é necessário o desenvolvimento de *middlewares* para suportar a programação de ambientes pervasivos, escondendo a complexidade do sistema e apresentando uma visão mais programática para os desenvolvedores (HELAL, 2005).

Como mencionado anteriormente, a visão proposta pela computação *pervasiva* apresenta uma interação natural e transparente entre o usuário e a infra-estrutura computacional que permeia o ambiente. Como a mobilidade é uma característica intrínseca do usuário, esta deve ser suportada pela computação *pervasiva*. Dessa forma, essa área de pesquisa engloba os estudos considerados na computação móvel, porém vai muito além, incorporando quatro novas questões de pesquisa (SATYANARAYANAN, 2001).

2.3.1 Uso efetivo de *Smart Spaces*

O termo *Smart Space* é um conceito chave na computação *pervasiva*, e corresponde a uma área onde a infra-estrutura computacional permeia a própria construção. Exemplos de *Smart Spaces* poderiam ser uma sala de reuniões, um corredor ou até mesmo um jardim.

Uma definição semelhante ao conceito de *Smart Spaces* corresponde à noção de *Active Spaces* (ROMAN, 2003; ROMAN et al., 2002; BRESLER; AL-MUHTADI; CAMPBELL, 2004). Um *Active Space* seria um espaço físico (uma região geográfica com fronteiras limitadas e bem definidas) coordenado por uma infra-estrutura de *software* baseada em contextos, o que aumenta a habilidade de usuários móveis de interagir com seus ambientes (físico e digital) e configurá-los de forma transparente.

A fusão entre o mundo computacional e o mundo real permite o sensoriamento e o controle de um sobre o outro. Essa influência pode acontecer nos dois sentidos, como por exemplo: *(a)* o ajuste automático da temperatura e iluminação de uma sala baseada no perfil dos ocupantes, ou *(b)* o comportamento diferenciado do *software* no computador de um usuário, dependendo de sua localização corrente.

2.3.2 Invisibilidade

A visão original de computação *pervasiva* proposta por Mark Weiser no início dos anos 90 (WEISER, 1991) (chamada de computação ubíqua) corresponde a um completo desaparecimento da tecnologia do ponto de vista do usuário. Na prática, porém, uma aproximação desse ideal seria assegurar uma distração mínima do usuário em função dos elementos computacionais. Mesmo que na maior parte do tempo o ambiente *pervasivo* atenda às expectativas do usuário de forma transparente, em alguns casos uma interação explícita com o ambiente pode ser necessária para evitar alguma situação desagradável.

2.3.3 Escalabilidade localizada

Junto com o aumento da sofisticação dos *Smart Spaces*, ocorre também uma maior interação entre os espaços computacionais pessoais dos usuários e seus arredores. Essa questão traz sérias implicações relacionadas à largura de banda, energia e distração, as quais são agravadas pela presença de um grande número de usuários. Nesse sentido, escalabilidade torna-se um problema crítico na computação *pervasiva*. No ambiente proporcionado pela computação *pervasiva*, a localização das interações entre o usuário e o meio tem um papel fundamental, e a densidade das interações deve ser reduzida no momento em que algum usuário move-se para fora das proximidades. De outra forma, o usuário e o sistema computacional seriam sobrecarregados por interações distantes que são de pouca relevância. Embora um usuário móvel distante de sua casa ainda gere algumas interações de longa distância, a preponderância de suas interações será local.

2.3.4 Mascaramento de condições desiguais

Por último, o quarto motivo de pesquisa imposto pela computação *pervasiva* trata do desenvolvimento de técnicas para o mascaramento de condições desiguais dos ambientes. A taxa de penetração da tecnologia da computação *pervasiva* na infra-estrutura não será uniforme e irá variar consideravelmente dependendo de muitos fatores não técnicos, como de estrutura organizacional, econômicos e modelos de negócio. Durante esse período, existirá uma grande diferença na "esperteza" dos ambientes. Esse fator pode influenciar negativamente o objetivo da computação *pervasiva* de ser invisível para os usuários.

Roussos e colegas (ROUSSOS; MARSH; MAGLAVERA, 2005) argumentam que a curto prazo os *smart-phones* serão o principal dispositivo de interação de usuários com o ambiente *pervasivo*. Uma forma de reduzir a variação de sofisticação observada por um usuário é fazer com que seu espaço de computação pessoal compense as deficiências de alguns ambientes. Um exemplo trivial seria um sistema que suportasse operação desconectada a fim de mascarar a falta de cobertura de redes sem-fio em alguns ambientes. A completa invisibilidade pode ser impossível, mas reduzir a variabilidade é uma alternativa plausível.

2.4 O projeto ISAM

O projeto ISAM (*Infra-estrutura de Suporte às Aplicações Móveis*), em desenvolvimento no Grupo de Processamento Paralelo e Distribuído da Universidade Federal do Rio Grande do Sul, provê uma plataforma comum que torna possível

a execução de aplicações através de uma variedade de equipamentos, além de sua distribuição e instalação automática. O principal objetivo do projeto é explorar alternativas que facilitem a tarefa de projetar aplicações distribuídas para ambientes *pervasivos*. O ISAM provê uma plataforma integrada, da linguagem ao ambiente de execução, para construir e executar esse tipo de aplicação. Para o ISAM, aplicações para computação *pervasiva* são distribuídas, móveis e conscientes de contexto por natureza, agregando, dessa forma, questões de pesquisa pertinentes a diversas áreas de estudo. Seu *middleware* é direcionado ao suporte do gerenciamento de recursos em redes heterogêneas, mobilidade lógica e física, adaptação dinâmica e execução de aplicações distribuídas baseadas em componentes.

Sob a perspectiva do ISAM, as condições do contexto são monitoradas de uma forma pró-ativa e o suporte de execução deve permitir que o *middleware* e a aplicação utilizem essas informações no gerenciamento de seus aspectos funcionais e não-funcionais. O ISAM possui como uma necessidade fundamental o emprego de uma estratégia para descoberta de recursos que permita identificar e utilizar os recursos computacionais que compõem o ambiente de execução. Nesse sentido, uma das motivações para o desenvolvimento dessa pesquisa é fornecer uma contribuição ao projeto, modelando e desenvolvendo uma solução para descoberta de recursos no escopo de projeto ISAM.

2.4.1 A arquitetura ISAM

A arquitetura de *software* ISAM (AUGUSTIN et al., 2002), apresentada na figura 2.1, é uma solução integrada para o desenvolvimento e execução de aplicações *pervasivas* de propósitos gerais. A representação da consciência do contexto nesta figura como um módulo virtual tem por objetivo ressaltar sua importância na arquitetura, bem como caracterizar sua presença na concepção de todos os outros componentes. A arquitetura do ISAM é organizada logicamente em três camadas, as quais são descritas a seguir.

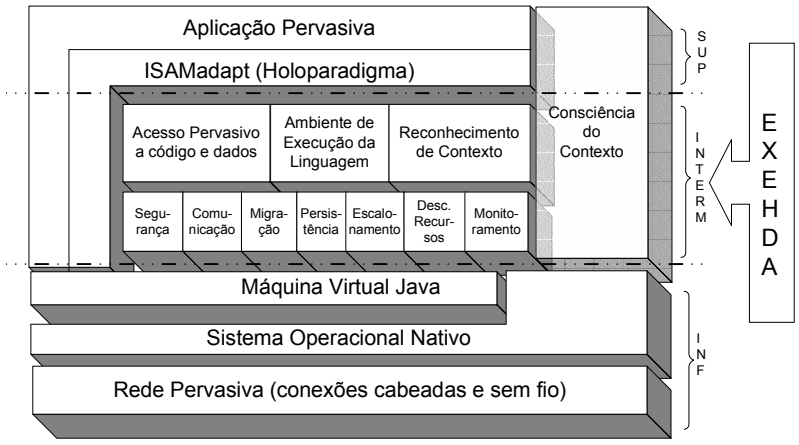

Figura 2.1: Arquitetura ISAM

Na *camada de aplicação (sup)* está a linguagem de programação (ISAMadapt), a qual disponibiliza abstrações para programação de aplicações distribuídas, móveis e conscientes do contexto, direcionadas à computação *pervasiva* (AUGUSTIN, 2003; AUGUSTIN et al., 2002; AUGUSTIN; YAMIN; GEYER, 2002). A linguagem ISAMadapt emprega alguns conceitos derivados do Holoparadigma (BARBOSA,

2002). O Holoparadigma definiu o conceito de entes, entidades semelhantes a agentes móveis que permitem a expressão explícita da mobilidade lógica e implícita da mobilidade física, e se comunicam através da leitura e escrita em um *blackboard*. A organização lógica dos entes de uma aplicação é mapeada para uma árvore de entes (SCHAEFFER FILHO, 2002). Ao conceito de entes foram adicionadas pelo ISAMadapt abstrações para a expressão de um comportamento *pervasivo*: contexto, adaptadores, comandos e políticas de adaptação.

A *camada intermediária (interm)* corresponde ao *middleware* EXEHDA, e nela estão os mecanismos de suporte à execução da aplicação *pervasiva* e às estratégias de adaptação (YAMIN, 2004; YAMIN et al., 2003, 2002; SILVA, 2003; REAL, 2004). Esta camada é formada por dois níveis.

O primeiro nível é composto por três módulos de serviço à aplicação:

- *Acesso Pervasivo a Dados e Código:* compreende os componentes que disponibilizam o ambiente *pervasivo*, incluindo *Ambiente Virtual do Usuário (AVU)*, *Ambiente Virtual da Aplicação (AVA)* e *Base de Dados pervasiva das Aplicações (BDA)*. O *Ambiente Virtual do Usuário* é composto pelos elementos que integram a interface de interação do usuário com o sistema. Este módulo é responsável por implementar o suporte para que a aplicação que o usuário está executando em uma localização possa ser instanciada e continuada em outra localização sem descontinuidade, permitindo o estilo de aplicações siga-me (*follow-me*) num ambiente *pervasivo*. O desafio da adaptabilidade para implementar o AVU é dar suporte aos usuários em diferentes localizações, com diferentes sistemas de interação, que demandam diferentes sistemas de apresentação, dentro dos limites da mobilidade. Por sua vez, entende-se como AVA o conjunto de atributos que identificam uma execução específica de uma aplicação, enquanto a BDA constitui o repositório de códigos das aplicações em geral;

- *Ambiente de Execução da Linguagem:* é o encarregado pelo gerenciamento da aplicação durante seu tempo de vida;

- *Serviço de Reconhecimento de Contexto:* é responsável por informar o estado dos elementos de contexto de interesse da aplicação e do próprio ambiente de execução.

No segundo nível da camada intermediária, estão os serviços básicos do EXEHDA, os quais fornecem as funcionalidades necessárias para o primeiro nível e cobrem vários aspectos, tais como: migração (mecanismos para deslocar um componente de *software* de uma localização física para outra), persistência (mecanismo para aumentar a disponibilidade e o desempenho do acesso aos dados), descoberta de recursos (para dar suporte ao movimento dos dispositivos móveis e dos componentes entre diferentes células), comunicação (com possibilidade de ser anônima e assíncrona), escalonamento (permite decidir o melhor nó para criar os componentes da aplicação) e monitoramento (sensores que fornecem informações sobre o ambiente de execução e as aplicações).

A *camada inferior (inf)* da arquitetura é composta pelos sistemas e linguagens nativas que integram o meio físico de execução. Por questões de portabilidade, nesta camada a plataforma base de implementação é a Máquina Virtual Java. A arquitetura supõe a existência de uma rede global com suporte à operação sem-fio,

interligada a outra cabeada que disponibilize uma infra-estrutura de equipamentos e serviços em escala global (rede *pervasiva*).

Como visto, a arquitetura ISAM está organizada em camadas lógicas, com níveis diferenciados de abstração, e está direcionada para a busca da manutenibilidade da qualidade de serviços oferecida ao usuário móvel através do conceito de adaptação. Nesta, o sistema se adapta para fornecer a qualidade nos serviços prestados, enquanto que a aplicação se adapta para atender à expectativa do usuário móvel, mantendo a funcionalidade da aplicação.

2.4.2 O ISAMpe

O ambiente de computação provido pelo ISAM é chamado ISAMpe (ISAM *pervasive environment*, figura 2.2), o qual possui uma organização celular. Este tem como premissa integrar os cenários *(i)* da computação em grade, *(ii)* da computação móvel e *(iii)* da computação sensível ao contexto.

O meio físico sobre o qual o ISAMpe é definido constitui-se por uma rede infra-estruturada, cuja composição final pode ser alterada pela agregação dinâmica de nós móveis. Este meio é caracterizado como um sistema distribuído híbrido (YAMIN et al., 2003).

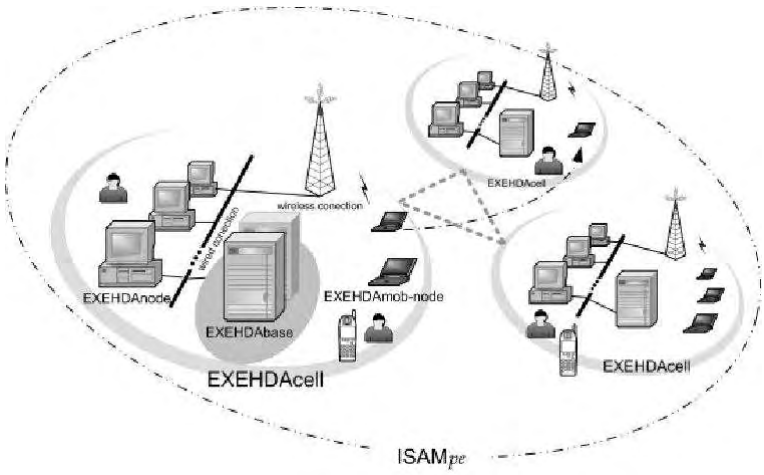

Figura 2.2: ISAM *pervasive environment*

Os recursos da infra-estrutura física são mapeados para três abstrações básicas, as quais são utilizadas na composição do ISAMpe:

- *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 a qual seu ponto-de-acesso está subordinado. São funcionalmente análogos aos EXEHDAnodes, mas tipicamente são recursos com uma capacidade mais restrita (por exemplo, PDAs);

- *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;

- *EXEHDAcell*: denota a área de atuação de uma EXEHDAbase, e é composta por esta e por EXEHDAnodes. Os principais aspectos considerados na definição da abrangência de uma célula são: o escopo institucional, a proximidade geográfica e o custo de comunicação.

Essa organização celular assemelha-se ao conceito de *virtual organization* encontrado na computação em grade (FOSTER; KESSELMAN; TUECKE, 2001), o que conduz a um procedimento de gerência dos equipamentos análogo ao praticado nesses ambientes (GEYER et al., 2002; YAMIN et al., 2003). 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.

2.4.3 O ambiente de execução EXEHDA

Nessa seção será sumarizado o EXEHDA (YAMIN, 2004), um *middleware* adaptativo e orientado a serviços destinado a dar suporte à execução de aplicações *pervasivas*. Devido a questões como a imprevisibilidade encontrada nos ambientes *pervasivos*, a consciência do contexto é um aspecto chave do *middleware*. O EXEHDA emprega diversas estratégias em seus serviços para permitir a adaptação ao estado corrente do contexto de execução, como carga sob demanda de serviços adaptativos, e descoberta e configuração dinâmica. Além disso, o EXEHDA também é responsável pelo gerenciamento dos recursos físicos que compõem o ISAMpe.

2.4.3.1 Arquitetura baseada em serviços

O EXEHDA é baseado em serviços que apresentam duas perspectivas: (*i*) criação e gerenciamento do ambiente *pervasivo*, provendo serviços para controlar o meio físico onde o processamento está sendo realizado; e (*ii*) suporte à execução de aplicações *pervasivas*, provendo serviços e abstrações necessárias para realizar a adaptação da aplicação ao longo de sua execução e deslocamento do usuário.

O *middleware* provê suporte para adaptação da aplicação, e ao mesmo tempo é auto-adaptativo. Isso permite ao sistema ajustar-se em decorrência de mudanças relacionadas aos recursos disponíveis, realocando e reescalonando dinamicamente os recursos para a aplicação. Nesse cenário, há uma preocupação especial na disponibilização de mecanismos para a construção e manipulação de informações de contexto, necessárias para permitir esse comportamento. Assim, as informações de contexto podem ser usadas nos procedimentos de decisão, colaborativamente com a aplicação.

Nessa abordagem, um núcleo mínimo do *middleware* tem sua funcionalidade estendida por serviços plugáveis (YAMIN et al., 2003). O carregamento de serviços é feito sob demanda e estes são adaptativos ao contexto de execução. Assim, pode-se usar implementações de um serviço que são melhor ajustadas para cada dispositivo e, ao mesmo tempo, reduzir o consumo de recursos carregando somente serviços

que serão efetivamente usados. Essa funcionalidade é customizada individualmente para cada nó através de um perfil de execução que define o conjunto de serviços que deverá ser ativado no nó, associando a cada serviço uma implementação (selecionada entre as alternativas disponíveis para cada serviço).

O núcleo mínimo do *middleware* é formado 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, baseada nas informações providas pelo *Profile Manager*. O código dos serviços é carregado sob demanda de um repositório de serviços, o qual pode ser local ou remoto, dependendo da capacidade de armazenamento do dispositivo e da natureza dos serviços a serem carregados.

2.4.3.2 Subsistemas do EXEHDA

Os demais serviços do EXEHDA estão organizados em quatro grandes subsistemas: execução distribuída, adaptação, comunicação e acesso pervasivo (figura 2.3). Esses subsistemas são descritos a seguir.

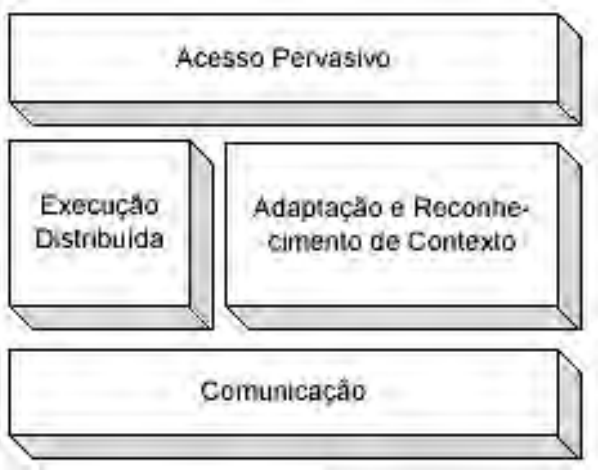

Figura 2.3: Subsistemas do EXEHDA

Subsistema de Execução Distribuída

O Subsistema de Execução Distribuída é responsável pelo suporte ao processamento distribuído no EXEHDA. No intuito de promover uma execução efetivamente *pervasiva*, este subsistema interage com outros subsistemas do EXEHDA. Em específico, interage com o Subsistema de Reconhecimento de Contexto e Adaptação, de forma a prover um comportamento distribuído e adaptativo às aplicações ISAM.

Dentre os serviços de suporte à execução distribuída providos por esse subsistema, pode-se destacar os seguintes: ***a)*** *Executor*, o qual acumula as funções de disparo de aplicações, e de criação e migração de seus objetos; ***b)*** *Cell Information Base (CIB)*, que implementa a base de informações da célula, mantendo os dados estruturais da EXEHDAcell, tais como, informações sobre os recursos que a compõem, informação de vizinhança, e atributos que descrevem as aplicações em

execução; ***c)*** *Discoverer*, responsável pela localização de recursos no ISAMpe a partir de especificações abstratas dos mesmos (o projeto e desenvolvimento desse serviço é o foco do presente trabalho); ***d)*** *ResourceBroker*, realiza o controle da alocação de recursos às aplicações; e ***e)*** *Gateway*, que faz a intermediação das comunicações entre os nós externos a uma célula e os recursos internos a ela, podendo alterar a visibilidade dos recursos de uma célula quando vistos de fora dela (da sua ação integrada com o *ResourceBroker* decorre o controle de acesso aos recursos de uma EXEHDAcell).

Subsistema de Reconhecimento de Contexto e Adaptação

O suporte à adaptação no EXEHDA está associado à operação do Subsistema de Reconhecimento de Contexto e Adaptação. Os serviços que integram esse subsistema tratam desde a extração da informação bruta a respeito dos recursos que compõem o ISAMpe, passando pela identificação e composição em alto nível dos elementos de contexto, até o disparo das ações de adaptação em reação a modificações no estado de tais elementos de contexto.

A seção 4.3.1 apresenta de uma forma mais detalhada cada um dos serviços desse subsistema. Brevemente, esses serviços são: ***a)*** *Collector*, responsável pela extração da informação diretamente dos recursos envolvidos, aglutinando dados oriundos de vários componentes monitores; ***b)*** *Deflector*, disponibiliza a abstração de canais de *multicast* na disseminação das informações monitoradas; ***c)*** *ContextManager*, responsável pelo refinamento da informação bruta produzida pela monitoração para produção de informações abstratas referentes aos elementos de contexto; ***d)*** *AdaptEngine*, provê facilidades para definição e gerência de comportamentos adaptativos por parte das aplicações (adaptações de cunho funcional), liberando o programador de gerenciar os aspectos de mais baixo nível envolvidos na manipulação dos elementos de contexto junto ao *ContextManager*; e ***e)*** *Scheduler*, gerencia as adaptações que não implicam alteração de código (cunho não-funcional), empregando a informação de monitoração para orientar operações de mapeamento de objetos nos recursos disponíveis no ISAMpe.

Subsistema de Comunicação

O Subsistema de Comunicação do EXEHDA disponibiliza mecanismos que contemplam modelos com níveis diferenciados de abstração para as comunicações entre componentes distribuídos. Dispositivos móveis e redes sem-fio proporcionam um ambiente onde a interação contínua entre os componentes da aplicação distribuída não pode ser assegurada, o que demanda estratégias diferentes em situações específicas.

Os serviços desse subsistema são os seguintes: ***a)*** *Dispatcher*, disponibiliza o modelo de comunicação através da troca de mensagens ponto-a-ponto com garantia de entrega e ordenamento através de canais de comunicação; ***b)*** *WORB*, oferece um modelo de comunicação baseado em invocações remotas de método, similar ao RMI, simplificando a construção de serviços distribuídos; e ***c)*** *CCManager*, disponibiliza um mecanismo baseado na abstração de espaço de tuplas, o qual prescinde da coexistência temporal de emissor e receptor.

Subsistema de Acesso Pervasivo

A computação *pervasiva* tem como premissa permitir o acesso a dados e código em qualquer lugar, todo o tempo, e usando qualquer dispositivo. O Subsistema de Acesso Pervasivo do *middleware* provê serviços para dar suporte à essa visão.

Os serviços que compõem este subsistema no EXEHDA são: ***a)*** *Base de Dados pervasiva das Aplicações (BDA)*, permite a recuperação do código integral de uma aplicação ou de componentes específicos e suas dependências, e provê métodos para a gerência das aplicações instaladas; ***b)*** *Ambiente Virtual do Usuário (AVU)*, realiza a manutenção do acesso *pervasivo* ao ambiente computacional do usuário, o qual engloba as aplicações em execução, as informações de personalização das aplicações definidas pelo usuário, o conjunto de aplicações instaladas (i.e. passíveis de serem disparadas por aquele usuário) e arquivos privados; ***c)*** *SessionManager*, gerencia a sessão de trabalho do usuário, sendo definida pelo conjunto de aplicações correntemente em execução; e ***d)*** *Gatekeeper*, intermedia acessos entre as entidades externas à plataforma ISAM e os serviços do *middleware* de execução, conduzindo os procedimentos de autenticação necessários.

3 CONCEITOS BÁSICOS E REQUISITOS DE UMA ESTRATÉGIA PARA DESCOBERTA DE RECURSOS

Nesse capítulo, inicialmente serão apresentados conceitos gerais relacionados à descoberta de recursos, a terminologia utilizada na área e suas principais características. Após, será descrito um conjunto de requisitos identificados como necessários a uma estratégia para descoberta de recursos na computação *pervasiva*. O levantamento dessas necessidades é utilizado no decorrer do texto para guiar a modelagem do serviço proposto para descoberta de recursos.

3.1 Visão geral

A pesquisa em estratégias para descoberta de recursos tem como uma de suas principais motivações o fato de que os ambientes computacionais futuros irão consistir de um amplo conjunto de dispositivos, aplicações e serviços baseados em rede. Essa tendência deve-se, entre outros fatores, ao surgimento de dispositivos pessoais móveis avançados (PDAs e telefones celulares de 3ª geração) e de novas infra-estruturas de comunicação ubíquas baseadas em redes sem-fio (FRIDAY; DAVIES; CATTERALL, 2001). Adicionalmente a recursos computacionais típicos, como poder de processamento, espaço de armazenamento e aplicações, dispositivos sem-fio empregam câmeras digitais, microfones e receptores GPS, entre outros recursos (MCKNIGHT, 2004). Bettstetter e Renner (2000) ainda destacam a crescente popularização de novos serviços, como aqueles destinados ao acesso a informações na Internet, música sob demanda e serviços que utilizem a infra-estrutura computacional de uma rede.

Recursos computacionais distribuídos serão responsáveis por disponibilizar uma série de serviços a usuários ou a outros recursos, e serão exigidas formas mais ricas de comunicação e interação para que esses possam trabalhar em conjunto, com o mínimo de interferência humana. Nesse cenário, torna-se evidente a necessidade de um mecanismo que facilite a administração desses recursos pois, ao passo que usuários necessitem desempenhar essas funções administrativas, é inevitável que erros sejam cometidos (PERKINS; HARJONO, 1996).

Idealmente, usuários deveriam obter acesso aos recursos de forma automática, sem a necessidade de reconfigurar seus sistemas. Em um ambiente para descoberta de recursos, estes devem se anunciar, fornecendo detalhes sobre suas capacidades e informações que permitam a sua utilização (endereço IP, por exemplo) por entidades remotas. Usuários ou recursos devem poder localizar um dado recurso pelas

informações fornecidas ao mecanismo de descoberta e fazer uma seleção inteligente caso múltiplos resultados satisfaçam seus critérios de pesquisa.

Nesse cenário, recursos anunciam-se usando serviços de diretório de forma que clientes possam localizá-los. Um aspecto importante e que deve ser considerado se refere ao fato de que as abstrações necessárias para dar suporte a esse comportamento não são muito diferentes daquelas encontradas nas relações de negócios que ocorrem no nosso dia-a-dia (VINOSKI, 2003). Quando se necessita de um prestador de serviços (como um encanador, por exemplo), pode-se encontrá-lo através de indicações de outras pessoas ou por meio de uma pesquisa na lista telefônica. Dessa forma, pode-se obter uma melhor compreensão de como sistemas de descoberta funcionam comparando-os com situações do cotidiano.

Em ambientes de computação *pervasiva*, numerosos elementos computacionais e sensores devem interagir para alcançar a funcionalidade e inteligência necessárias; nesses ambientes, estratégias de descoberta podem permitir que o espaço *pervasivo* mude e evolua dinamicamente, sem maiores reengenharias do sistema (HELAL, 2002). Harbird e colegas (HARBIRD; HAILES; MASCOLO, 2004) ainda destacam que serviços de descoberta e gerenciamento de recursos irão desempenhar um papel fundamental num cenário de implantação ubíqua sustentável.

3.2 Terminologia utilizada

Ainda existe uma certa carência de padronização em relação à terminologia usada na área de descoberta de recursos. A seguir, são reproduzidas algumas definições empregadas por McGrath (2000) para caracterizar alguns conceitos dessa área de pesquisa, as quais serão utilizadas posteriormente.

3.2.1 Serviços, dispositivos e recursos

Em relação à entidade a ser descoberta, existem três conceitos encontrados na literatura: serviço (*service*), dispositivo (*device*) e recurso (*resource*). Nesse contexto, o termo serviço é utilizado para designar qualquer tipo de funcionalidade em rede que possa estar disponível para acesso como, por exemplo, um servidor de e-mails ou um banco de dados. Por outro lado, o termo dispositivo é utilizado como forma de representar equipamentos conectados a uma rede, como computadores convencionais, computadores de mão ou PDAs, impressoras, telefones celulares e câmeras digitais, por exemplo. É importante salientar que um determinado dispositivo pode ser representado na rede por um ou mais serviços. Como exemplo, tome-se uma impressora, que pode implementar diversos serviços como impressão, envio de *fax*, além de outras funcionalidades, tudo em um único dispositivo. Por último, tanto serviços como dispositivos podem ser referenciados de forma indistinta como recursos.

3.2.2 Procura e descoberta

Quanto ao modo de localização de um recurso, existem duas alternativas: procura (*lookup*) e descoberta (*discovery*). O termo procura é utilizado para referenciar o processo de localizar um recurso específico junto a um repositório de dados conhecido, sempre iniciado por um solicitador externo. Além disso, essa estratégia exige a existência de algum diretório ou outro agente para responder à requisição. Como exemplo de mecanismos de procura pode-se citar o DNS (MOCKAPETRIS; DUNLAP, 1988) e LDAP (CLUET; KAPITSKAIA; SRIVASTAVA, 1999). Por outro

lado, o termo descoberta é usado para designar um processo mais dinâmico, no qual recursos se anunciam para outros na rede e se descobrem de forma automática, conforme são disponibilizados no meio. As informações sobre os recursos (ou a localização de um repositório dessas informações) são difundidas e atualizadas dinamicamente através de anúncios e revalidações de estado dos recursos conectados à rede.

3.3 Características básicas de uma solução para descoberta de recursos

Nessa seção, serão descritas algumas das características tradicionalmente encontradas em soluções para descoberta de recursos, as quais podem ser utilizadas para apresentar de uma forma geral um modelo básico do funcionamento desses sistemas.

3.3.1 Organização dos componentes

Uma estratégia para descoberta de recursos costuma apresentar uma arquitetura composta de dois ou três tipos de componentes. No primeiro caso, são empregados apenas *Resource Components* (RCs) e *User Components* (UCs). No segundo caso, acrescenta-se a esses uma terceira estrutura, chamada *Directory Component* (DC). A arquitetura de dois componentes (figura 3.1) também é chamada de descentralizada (ou *peer-to-peer*) uma vez que nesse modelo ocorre a interação direta entre usuários e recursos (respectivamente através de UCs e RCs), geralmente por meio da troca de mensagens do tipo *multicast*. Na arquitetura de três componentes (figura 3.2), o DC tem o papel de intermediar a comunicação entre usuários e recursos, mantendo uma espécie de catálogo dos recursos disponíveis em um dado momento.

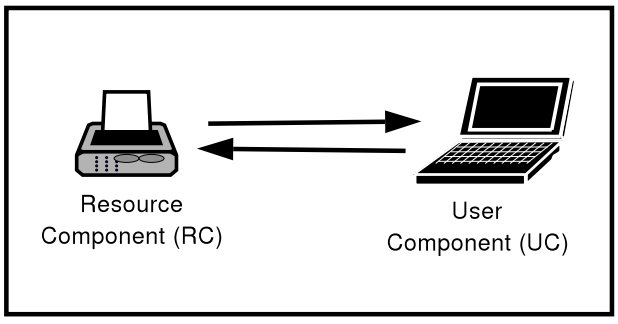

Figura 3.1: Arquitetura de dois componentes para descoberta de recursos

3.3.2 Principais tipos de mensagem

Quanto às mensagens empregadas nos protocolos de descoberta, pode-se identificar três tipos principais: anúncio (*advertise*), requisição (*request*) e resposta (*response*). A primeira é utilizada em estratégias conhecidas como *ativas*, onde os recursos são responsáveis por se anunciarem ao serem disponibilizados na rede. As outras duas são utilizadas em estratégias *passivas*, onde os recursos aguardam o recebimento de uma requisição para só então enviar uma mensagem de resposta indicando a sua presença.

Mensagens de anúncio podem ser enviadas a partir de um RC diretamente para um ou mais UCs (arquitetura de dois componentes), ou então para um DC (ar-

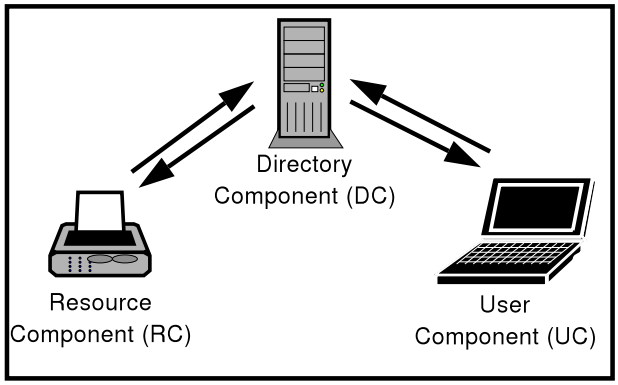

Figura 3.2: Arquitetura de três componentes para descoberta de recursos

quitetura de três componentes). Nesse caso, um DC ainda pode utilizar a mesma estratégia de envio de mensagens de anúncio para que UCs saibam de sua presença. De forma semelhante, mensagens de requisição podem ser enviadas a partir de um UC diretamente para RCs (arquitetura de dois componentes), ou então para um DC (arquitetura de três componentes).

Além disso, pode-se combinar estratégias, como ocorre com o Jini (ARNOLD, 1999), onde os recursos têm um comportamento ativo em um momento inicial e depois de um determinado período passam a apresentar um comportamento passivo.

3.3.3 Descritores e consulta de recursos

A definição dos atributos que informam as características de um recurso é realizada através da criação de uma estrutura denominada *descritor de recurso*. Todas as propriedades que possam ser úteis ao usuário do recurso, ou servirem como critério de escolha, devem ser informadas no descritor. Através de um processo de consulta, o interessado em realizar a descoberta deve especificar seus critérios de pesquisa, identificando propriedades que serão posteriormente verificadas junto a cada descritor que possa satisfazer tais critérios através de um processo chamado *matching* de atributos.

3.3.4 Descoberta interativa x não-interativa

Uma consulta que retorne mais de um resultado que satisfaça os critérios de pesquisa especificados pode permitir ao usuário navegar entre os recursos descobertos para selecionar aquele que lhe for mais conveniente. Esse comportamento caracteriza um mecanismo de descoberta interativo. Através de uma inspeção nos atributos dos serviços recuperados pela consulta o usuário poderia selecionar um serviço para suprir necessidades transientes de uma aplicação ou então configurar seu sistema para que use determinado serviço como opção padrão (GUTTMANN et al., 1999).

A descoberta não-interativa, por outro lado, é aquela onde apenas um recurso é retornado pelo processo de descoberta, mesmo que um número maior de recursos satisfaça os critérios de pesquisa. Nesse caso, a seleção do recurso é deixada a cargo do mecanismo de descoberta.

3.4 Identificação dos requisitos de uma estratégia para descoberta de recursos na computação *pervasiva*

Atualmente existem diversos projetos em andamento ligados à área de descoberta de recursos, tanto acadêmicos como comerciais, desenvolvidos por empresas ou consórcios. Apesar de existir uma certa semelhança operacional entre as estratégias desenvolvidas, estas são implementadas de forma diversa e possuem características próprias. Uma das razões para a existência dessa heterogeneidade refere-se ao fato de que essas tecnologias foram construídas com diferentes domínios de aplicação em mente; enquanto soluções como o SLP (BETTSTETTER; RENNER, 2000) são destinadas ao uso em empresas de grande porte, o UPnP (MICROSOFT CORPORATION, 2000) é destinado a ambientes computacionais domésticos ou pequenos escritórios (FRIDAY; DAVIES; CATTERALL, 2001).

No entanto, a computação *pervasiva* impõe ao desenvolvimento de uma estratégia de descoberta de recursos uma série de novos requisitos e desafios a serem considerados na administração desses. A seguir, serão discutidas algumas dessas questões, as quais guiaram a concepção e a modelagem do serviço de descoberta proposto nesse trabalho. A motivação da pesquisa relacionada ao serviço *PerDiS* decorre do não tratamento dessas necessidades em sua totalidade pelos serviços atuais de descoberta de recursos.

3.4.1 Linguagem expressiva para descrição de recursos

Uma importante motivação dessa pesquisa é o desenvolvimento de uma estratégia apropriada para descrever as características de um recurso e a especificação de construções que permitam a combinação de critérios na realização de consultas. À medida que a quantidade e a variedade de recursos disponíveis para acesso ampliarem-se, maior será a importância de um mecanismo de filtragem adequado (RAKOTONIRAINY; GROVES, 2002).

Uma das principais deficiências em relação às estratégias existentes para descoberta de recursos é a falta de linguagens expressivas e de representações que permitam descrever adequadamente as características de um recurso (CHAKRABORTY; CHEN, 2000). Esse aspecto refere-se tanto à definição das características do recurso a ser disponibilizado, como também à forma como os critérios de pesquisa são especificados no processo de descoberta.

Torna-se evidente, em um ambiente altamente heterogêneo e saturado de capacidades computacionais como o proposto pela computação *pervasiva*, a importância de um mecanismo de consulta não apenas baseado em um simples *matching* de atributos mas também que considere a possibilidade de consultas com critérios opcionais, multi-valorados, combinados e que permitam faixas de valores. Tal mecanismo deve ser flexível e extensível de forma que novas funcionalidades possam ser adicionadas à linguagem de consulta sem que a mesma perca sua generalidade. Além disso, existe a necessidade de que o mesmo possa ser aplicado a cenários variados, com diferentes classes de recursos a serem descobertos, de forma que a solução a ser desenvolvida seja de propósito geral.

3.4.2 Escalabilidade da solução

Um problema encontrado nas estratégias para descoberta existentes é a baixa escalabilidade dessas soluções. Nesse texto, o termo "escalabilidade da solução" se

refere a algum tipo de suporte à descoberta de recursos dispersos geograficamente. A maioria dos protocolos dá suporte a algum tipo de organização hierárquica, a qual pode ser geralmente usada para aumentar a confiabilidade através de redundância e aumentar a escalabilidade através da distribuição. Porém, apesar de algumas afirmações de *marketing*, nenhum desses protocolos poderia ser empregado com eficiência em toda a Internet (MCGRATH, 2000). Helal (HELAL, 2002) afirma que, em sua forma atual, a maioria dos padrões para descoberta de recursos não satisfaz as necessidades de mobilidade e outros requisitos, sendo portanto incerto seu uso em ambientes *pervasivos*.

As soluções atuais para descoberta foram projetadas para uso em redes locais, não sendo adequadas para uso em ambientes de larga-escala como a Internet. A utilização de IP *multicast* em Jini (ARNOLD, 1999), por exemplo, limita o processo de descoberta ao escopo de uma LAN, não havendo um suporte por parte do ambiente que permita a descoberta de recursos dispersos geograficamente.

3.4.3 Utilização de preferências de usuários

Estratégias para descoberta de recursos geralmente não possibilitam a utilização de preferências previamente armazenadas para cada usuário na realização do processo de descoberta. Um aspecto inadequado à *pervasividade* encontrado nos serviços de descoberta atuais refere-se ao fato desses se concentrarem apenas no "presente", não levando em consideração o elemento temporal relacionado à interação com os recursos. Um mecanismo que permitisse a consultas futuras se beneficiarem de informações já armazenadas para refinar os resultados a serem retornados ao usuário pode ser bastante valioso em determinadas situações e ambientes proporcionados pelo surgimento da computação *pervasiva* (FRIDAY; DAVIES; CATTERALL, 2001) como, por exemplo, na previsão de algum comportamento futuro.

3.4.4 Possibilidade de interoperabilidade

Devido à diversidade de soluções para descoberta de recursos existentes e em desenvolvimento, uma das principais questões que devem ser consideradas é a possibilidade de interoperabilidade entre essas estratégias. Em um futuro próximo, é bastante improvável que irá surgir um único *middleware* apropriado para diferentes dispositivos e propósitos (RAATIKAINEN; CHRISTENSEN; NAKAJIMA, 2002).

A implementação de diversos protocolos equivalentes ou a utilização de *proxies* que permitam a comunicação entre diferentes soluções podem ser estratégias inadequadas para uso em dispositivos portáteis e com limitadas capacidades computacionais. Nessas condições, deve-se priorizar o emprego de soluções mais leves e que possam ser providas mais facilmente por pequenos dispositivos.

3.4.5 Estratégias de *self-healing*

O emprego de estratégias de *self-healing* em sistemas para descoberta de recursos permite atenuar, detectar e realizar a recuperação de falhas que possam ocorrer durante o funcionamento. Num cenário de computação *pervasiva*, um grande número de recursos computacionais pode estar envolvido no processo de descoberta, direta ou indiretamente, e a possibilidade de que alguns componentes possam sofrer desconexões ou tornarem-se indisponíveis em momentos críticos deve ser considerada. Uma estratégia de *self-healing* permite ao mecanismo de descoberta recuperar-se de

falhas ou inconsistências no ambiente de forma autônoma. Dabrowski e Mills (2002) enumeram diversas técnicas que, empregadas isoladamente ou combinadas, podem ser usadas para aumentar a robustez do mecanismo de descoberta.

A primeira delas refere-se à *arquitetura e topologia* do sistema de descoberta, que pode ser composta de dois ou três componentes. Enquanto na arquitetura de dois componentes todas as solicitações dos usuários são encaminhadas diretamente aos recursos através de mensagens em *multicast*, a utilização da arquitetura de três componentes permite a descoberta de recursos através de consultas feitas a um diretório de recursos. Nesse caso, torna-se possível a existência de múltiplas cópias do diretório de recursos, a fim de minimizar os efeitos de uma possível falha.

A segunda estratégia de *self-healing* é chamada de *manutenção da consistência*, e é usada para obter-se atualizações de recursos já descobertos. Existem duas alternativas: *polling* e notificação. Na primeira delas, o usuário deve ficar constantemente verificando as condições de um recurso previamente descoberto, seja diretamente junto ao recurso ou então através do componente intermediário que oferece o diretório de recursos (arquitetura de três componentes). Por outro lado, o uso de notificações permite que o próprio recurso envie eventos sempre que seu estado for alterado.

A terceira estratégia corresponde à *detecção de defeitos* em sistemas de descoberta, a qual realiza a monitoração do recebimento de mensagens periódicas especificamente enviadas para informar sobre a disponibilidade de um recurso. O não-recebimento dessas mensagens pode indicar uma falha no recurso ou um particionamento na rede de dados.

Por último, sistemas de descoberta podem empregar técnicas de *recuperação*, sendo que as principais são a manutenção de um *soft-state* e persistência no nível da aplicação. A primeira delas utiliza o envio de mensagens periódicas para transportar informações sobre o estado atual do recurso. O recebimento de uma mensagem atualiza seu estado na *cache*, enquanto que o não-recebimento descarta o estado previamente armazenado. Dessa forma, o estado dos recursos disponíveis em um dado momento é mantido de forma consistente. Por outro lado, a persistência no nível da aplicação confia no emprego de políticas de recuperação implementadas pela própria aplicação para realizar o processo de recuperação, o qual é iniciado pela geração de uma exceção ao detectar-se a ocorrência de um defeito.

3.4.6 Utilização de informações de contexto

Os ambientes computacionais *pervasivos* são tipicamente saturados de recursos computacionais heterogêneos e em constante alteração. O sucesso das aplicações destinadas à execução em ambientes móveis está ligado à necessidade de adaptação dessas e dos dados que elas apresentam às limitações do contexto corrente (CHALMERS; DULAY; SLOMAN, 2004b; CHEN; KOTZ, 2003). Essa preocupação deve ser estendida aos serviços que dão suporte à execução dessas aplicações (serviço de descoberta de recursos, por exemplo), de forma que as funcionalidades providas pelo ambiente de execução permaneçam operacionais na ocorrência de mudanças no contexto de execução.

Mark Weiser apresentou sua visão de um cenário ubíquo, povoado de dispositivos e sensores devidamente integrados à infra-estrutura física do ambiente, ao ponto de tornar a tecnologia indistinguível do meio (WEISER, 1991). Para tornar possível essa visão, é necessário transformar as máquinas isoladas e sem consciência de con-

texto de hoje em recursos computacionais inteligentes, programáveis e conscientes de contexto (RANGANATHAN; AL-MUHTADI; CAMPBELL, 2004).

4 PERDIS: UM MODELO PARA DESCOBERTA DE RECURSOS NO *ISAM PERVASIVE ENVIRONMENT*

Este capítulo apresenta o modelo *PerDiS* (*Pervasive Discovery Service*) para descoberta de recursos. Este considera, em sua concepção, os requisitos identificados como necessários a um mecanismo para descoberta de recursos na computação *pervasiva*. O modelo proposto permite que recursos se registrem em serviços de diretórios e que usuários consultem esses diretórios em busca de recursos que satisfaçam determinados requisitos de pesquisa.

4.1 Descoberta de recursos como um serviço do *middleware* EXEHDA

O desenvolvimento de um mecanismo de descoberta como parte do *middleware* do ISAM habilita a sua interação com outros serviços que compõem o ambiente de execução. Isso permite que funcionalidades providas por outros serviços sejam agregadas ao mecanismo de descoberta de forma a incrementar suas potencialidades.

Por exemplo, serviços específicos do *Subsistema de Reconhecimento de Contexto* do *middleware* são responsáveis pela manutenção das informações referentes ao estado do contexto, as quais guiam muitas das operações do *middleware* e também o comportamento adaptativo da aplicação. A interação com esses serviços permite incorporar ao mecanismo de descoberta o tratamento de informações relacionadas ao contexto dos usuários e recursos.

Além disso, serviços genéricos fornecidos pelo *middleware*, como a *CIB* (*Cell Information Base*), assumem um papel bastante específico quando agregado à arquitetura de descoberta de recursos. A *CIB* tem como função prover uma base para armazenamento de informações de natureza diversa no âmbito de uma célula de execução. Associado ao mecanismo de descoberta, a *CIB* assume o papel de armazenadora da lista de recursos (e de suas propriedades) mantida por um diretório de recursos.

Finalmente, a integração futura com os serviços *Gatekeeper* e *ResourceBroker* do *middleware* irá prover o controle de acesso necessário aos recursos de uma EXEHDAcell, através de mecanismos de autenticação e gerenciamento de alocação desses recursos.

Essa interação com outros serviços traz uma série de benefícios não proporcionados por estratégias independentes (*stand-alone*) para descoberta de recursos. A utilização de serviços de propósitos gerais ou específicos providos pelo *middleware*

tende a incrementar as funcionalidades do mecanismo de descoberta, ao mesmo tempo que permite a redução de sua complexidade.

4.2 Arquitetura *PerDiS* para descoberta de recursos

A arquitetura *PerDiS* foi concebida de forma a atender os requisitos identificados na seção 3.4 como necessários a uma estratégia para descoberta de recursos na computação *pervasiva*. Resumidamente, esses requisitos tratam de questões relacionadas aos seguintes aspectos:

- *Utilização de informações do contexto de execução*;
- *Utilização de estratégias para manutenção automática da 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 (geograficamente dispersos)*;
- *Utilização de preferências por usuário.*

A seguir, as principais características da arquitetura *PerDiS* serão apresentadas (componentes, protocolo de anúncio e tipos de mensagens). O restante do capítulo descreverá como o modelo proposto aborda cada um dos requisitos identificados.

4.2.1 Componentes envolvidos no processo de descoberta

A figura 4.1 apresenta os componentes que definem a arquitetura proposta para descoberta de recursos. Nessa organização, recursos são registrados junto ao serviço de descoberta no momento em que são disponibilizados na rede, e usuários consultam o serviço de descoberta para pesquisar recursos que possam satisfazer suas necessidades. A seguir, serão descritos os principais aspectos relacionados a cada um deles.

Cabe ressaltar que essa organização corresponde à arquitetura de descoberta de recursos existente no âmbito de uma célula de execução do ISAM *pervasive environment*. A descoberta de recursos dispersos geograficamente no ISAMpe e a interação entre os componentes de cada célula envolvidos no processo de descoberta são detalhados na seção 4.6.

4.2.1.1 Resource Component (RC)

O *resource component* é a entidade responsável por informar as características relacionadas a um determinado recurso, que permitam a um usuário optar ou não pela sua utilização. Os principais módulos dessa entidade são os seguintes:

- *Resource descriptor:* módulo responsável por manter as propriedades descritivas referentes ao recurso. A forma de representação dessas informações é ilustrada na seção 4.5.1;
- *Lease renewer:* realiza o gerenciamento da renovação do *lease*, utilizado para indicar periodicamente a disponibilidade do recurso, provendo assim um mecanismo de controle de consistência e tolerância a falhas à arquitetura (estratégia descrita na seção 4.4.1).

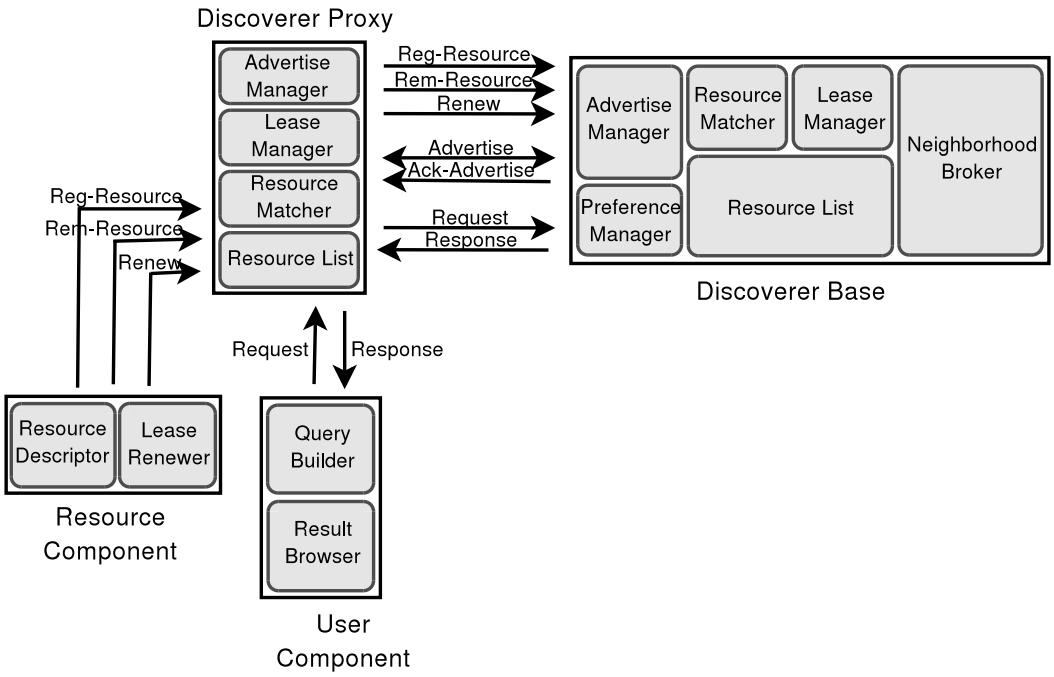

Figura 4.1: Componentes da arquitetura de descoberta

4.2.1.2 User Component (UC)

O papel do *user component* é habilitar o usuário a realizar o processo de descoberta, interagindo com os demais componentes da arquitetura. Seus principais módulos são:

- *Query builder:* interface para montagem de consultas a recursos, permitindo a especificação de condições que devem ser satisfeitas pelos recursos a serem descobertos (mais especificamente, permite a definição de critérios no formato descrito na seção 4.5.2);
- *Result browser:* permite ao usuário inspecionar os recursos retornados em uma consulta.

4.2.1.3 Discoverer Base (DB) e Discoverer Proxy (DP)

O *discoverer base* e o *discoverer proxy* atuam como um catálogo (ou diretório) de recursos, abrangendo, respectivamente, os recursos disponibilizados em uma célula de execução e em um nó específico do ISAMpe. O primeiro deles corresponde a uma instância celular do serviço de descoberta, enquanto o outro corresponde a uma instância local do mesmo serviço. Uma instância DP do serviço de descoberta gerencia os recursos disponibilizados localmente em cada nó do ISAMpe (o próprio *host*, um servidor de banco de dados, um servidor de arquivos, etc.) e comunica-se com a instância DB para que esses recursos sejam catalogados no nível da célula de execução. Recursos e usuários comunicam-se com uma instância *proxy* do serviço de descoberta, a menos que estejam localizados no próprio nó onde se encontra a instância *base* do serviço (nesse caso, a comunicação é realizada diretamente através do DB); essa questão, porém, é totalmente transparente para recursos e usuários. Toda a comunicação entre as instâncias *proxy* e *base* do serviço de descoberta é

realizada de forma transparente também, incluindo a localização automática de instâncias *base* por instâncias *proxy*, no momento em que essas são inicializadas pelo *middleware*. Os principais módulos comuns a essas duas entidades são os seguintes:

- *Advertise manager:* realiza o anúncio em *multicast* da instância do serviço de descoberta (que pode ser um DB ou um DP) 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 no âmbito de uma célula de execução do ISAMpe (descrito na seção 4.2.2);

- *Resource list:* mantém a lista de recursos gerenciados pela instância do serviço de descoberta (apenas recursos locais para um DP, ou todos os recursos da célula no caso de um DB). A funcionalidade desse módulo é proporcionada por um serviço especificamente provido pelo *middleware* EXEHDA, chamado *Cell Information Base (CIB)*;

- *Lease manager:* responsável pelo gerenciamento do controle de *lease*, realizando o descarte de recursos que não tenham sido renovados há um determinado período de tempo (abordado na seção 4.4.1);

- *Resource matcher:* realiza a comparação de critérios de pesquisa (utilizado na seleção dos recursos). Mais especificamente, faz a combinação entre as propriedades descritivas dos recursos registrados e os requisitos especificados pelo usuário referentes aos recursos a serem descobertos (essas duas representações são descritas na seção 4.5).

Além desses, um *discoverer base* ainda apresenta alguns módulos adicionais não existentes em um *discoverer proxy*. Esses são:

- *Preference manager:* aplica preferências pessoais de cada usuário, mantidas pelo serviço *AVU* do *middleware*, aos resultados obtidos pelo processo de descoberta (descrito na seção 4.7);

- *Neighborhood broker:* usado na comunicação *peer-to-peer* entre as células que compõem o ISAM *pervasive environment* de forma a habilitar a descoberta de recursos em larga-escala (abordada na seção 4.6).

Além disso, a utilização de uma instância *proxy* executando localmente junto a usuários e recursos permite concentrar e otimizar certas tarefas comuns a um conjunto de usuários ou recursos. Como exemplo, pode-se citar o gerenciamento de mensagens enviadas em *multicast* para realizar a localização das instâncias de descoberta (seção 4.2.2). Mesmo que exista mais de um recurso por nó de processamento a ser anunciado (o próprio *host* e um servidor de banco de dados, por exemplo), a utilização do *proxy* permite que o protocolo de anúncio seja realizado apenas uma vez (na inicialização do *proxy*). Caso esse componente não estivesse sendo utilizado, cada novo recurso (mesmo que disponibilizado fisicamente junto a um outro recurso) deveria gerenciar explicitamente as mensagens de anúncio a fim de localizar a(s) instância(s) do tipo *base* do serviço de descoberta.

Finalmente, é importante salientar o espectro de atuação do serviço de descoberta. Seu papel é concluído no momento em que apresenta ao usuário, através do

user component, o conjunto de recursos descobertos em uma pesquisa. Esses resultados são descritos em XML, informando os descritores dos recursos encontrados. A forma como o usuário irá acessar os recursos descobertos está fora das atribuições do *PerDiS*. Atualmente, o serviço *RBeing* (FRAINER, 2004) do *middleware* EXEHDA endereça o estudo de mecanismos para acesso aos recursos descobertos pelo *PerDiS*, recebendo como parâmetro o descritor XML de um dado recurso a ser acessado.

4.2.2 Anúncio e comunicação entre instâncias descobridoras

Como mencionado anteriormente, o *advertise manager* é o módulo de um *discoverer proxy/base* responsável por realizar o anúncio em *multicast* da instância descobridora, a fim de que a localização e a comunicação entre esses componentes possam ser feitos de forma automática (considerando o escopo de uma célula de execução no ISAMpe). Nessa organização, é necessário que uma instância *proxy* localize uma ou mais instâncias do tipo *base* ativas na célula de execução, uma vez que as operações sempre são propagadas de um *proxy* para um *base*. Além disso, uma instância *base* também deve localizar outras instâncias do mesmo tipo já inicializadas. Por outro lado, uma instância do tipo *base* não precisa localizar instâncias *proxies*, e tampouco um *proxy* precisa localizar outros *proxies*. Considerando essas condições, foram definidos três tipos de anúncios entre esses componentes:

- *Anúncio de um proxy para bases:* ao ser iniciada, uma instância *proxy* do serviço de descoberta deve descobrir quais são as instâncias do tipo *base* disponíveis em sua célula de execução. Nesse momento, o *proxy* envia uma mensagem de *advertise*, e aguarda um conjunto de mensagens do tipo *ack* enviadas por instâncias *base* (figura 4.2 (a));

- *Anúncio de um base para proxies:* sempre que uma instância *base* do serviço de descoberta é iniciada, esta é responsável por enviar uma mensagem de *advertise* para instâncias *proxies*. Nessa situação, não é necessário que cada *proxy* responda com uma mensagem do tipo *ack*, uma vez que uma instância *base* não precisa conhecer os *proxies*, apenas o contrário é necessário (figura 4.2 (b));

- *Anúncio de um base para outros bases:* além disso, ao ser iniciada, uma instância *base* também deve se anunciar para outras instâncias do tipo *base* potencialmente já em execução na célula em questão do ISAMpe. Nesse caso, a instância anunciante aguarda também o recebimento de mensagens do tipo *ack*, a fim de localizar componentes desse tipo previamente ativados. Essa interação é necessária uma vez que uma instância *base* pode atuar como um *proxy* caso recursos ou usuários a utilizem diretamente; nesse caso, ela precisará conhecer a localização de outras *bases*, a fim de realizar a propagação das operações nela realizadas (figura 4.2 (c)).

Todo o envio de anúncio em *multicast* em que o anunciante aguarda pelo recebimento de respostas de outras instâncias de descoberta tem associado um tempo de espera antes que o protocolo do *PerDiS* prossiga em operação. Esse tempo é denominado *timeout* de *ack*, e é especificado no perfil de configuração do EXEHDA existente em cada nó do ISAMpe.

Após o protocolo de descoberta ter sido executado, toda a comunicação entre as instâncias de descoberta é realizada através do serviço de comunicação *WORB*,

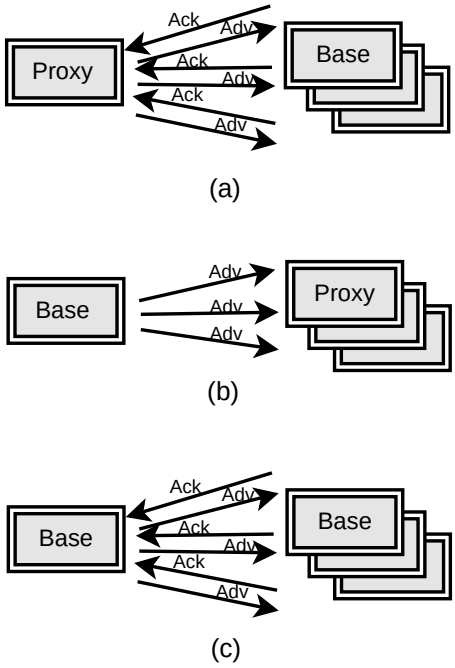

Figura 4.2: Anúncio e comunicação entre *proxies* e *bases*

disponibilizado pelo *middleware* EXEHDA. O serviço *WORB* abstrai aspectos de baixo nível relativos ao tratamento das comunicações em rede. Para tanto, ele oferece um 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 (YAMIN, 2004).

4.2.3 Mensagens do protocolo de descoberta de recursos

O protocolo base para descoberta de recursos é provido pela união de dois conjuntos de mensagens: *mensagens do protocolo de gerenciamento de recursos* e *mensagens do protocolo de pesquisa de recursos.*

4.2.3.1 Mensagens do protocolo de gerenciamento de recursos

As mensagens trocadas entre *resource components* e instâncias do *discoverer* (*proxy* ou *base*) permitem o gerenciamento da lista de recursos disponíveis no ambiente *pervasivo.* As principais mensagens são as seguintes:

- *Advertise:* mensagem enviada em *multicast* anunciando a presença de uma nova instância do serviço de descoberta (*proxy* ou *base*) no ambiente de execução;
- *Ack-Advertise:* enviada por uma instância de descoberta *base* em resposta a uma mensagem de anúncio, identificando seu endereço;
- *Reg-Resource:* mensagem enviada pelo RC com a finalidade de registrar um determinado recurso junto a uma dada instância do serviço de descoberta. Apenas nesse momento o descritor do recurso é enviado. Inicialmente esse registro é realizado na instância de descoberta local (DP) e em seguida é propagado para a(s) instância(s) do tipo *base* da célula de execução;

- *Renew:* utilizada para renovar o tempo de *lease* do recurso junto ao(s) descobridor(es) em que o mesmo estiver registrado. O recebimento dessa mensagem pelo *discoverer* o informa de que a entrada correspondente ao recurso deve ser mantida por um determinado período de tempo. Inicialmente essa renovação é realizada na instância de descoberta local (DP) e em seguida é propagada para a(s) instância(s) do tipo *base* da célula de execução;

- *Remove:* mensagem utilizada para permitir a remoção explícita do registro do recurso nas instâncias de descoberta em que o mesmo estiver presente. Inicialmente essa remoção é realizada na instância de descoberta local (DP) e em seguida é propagada para a(s) instância(s) do tipo *base* da célula de execução.

4.2.3.2 Mensagens do protocolo de pesquisa de recursos

As mensagens trocadas entre *user components* e instâncias do *discoverer* (*proxy* ou *base*) permitem que usuários descubram recursos disponíveis no ambiente *pervasivo*. As principais mensagens são as seguintes:

- *Request:* mensagem enviada pelo usuário a um *discoverer* especificando um conjunto de critérios que corresponde às características do recurso ao qual se deseja obter acesso. Em condições normais uma mensagem de *request* enviada a um DP sempre será propagada a um DB (que mantém o catálogo de recursos de toda a célula de execução), mas como será visto a seguir (seção 4.3), o DP poderá tratar a mensagem localmente em alguns casos;

- *Response:* mensagem enviada por um *discoverer* em resposta a uma mensagem de *request*, informando o conjunto de resultados encontrados que satisfazem os critérios especificados pelo usuário.

4.3 Adaptação a alterações no contexto de usuários e recursos

Nos ambientes proporcionados pelo surgimento da computação móvel e da computação *pervasiva*, onde uma grande diversidade de dispositivos podem ser utilizados em diferentes situações, e na execução de várias tarefas, o contexto corrente irá impactar substancialmente no comportamento apropriado das aplicações (CHALMERS; DULAY; SLOMAN, 2004c).

Um aspecto importante a ser destacado em relação à arquitetura *PerDiS* é a existência de componentes locais (*proxies*), junto a usuários e recursos, para realizar o processo de descoberta. Esse tipo de entidade desempenha um papel fundamental relacionado à adaptação do mecanismo de descoberta a alterações no contexto de usuários e recursos. Através da interação com serviços específicos do *middleware*, destinados à coleta e disseminação de informações de contexto, o componente *proxy* tem condições de receber e tratar adequadamente essas informações, empregando comportamentos adaptativos a alterações de interesse no contexto. Como exemplo, pode-se citar o tratamento da desconexão de usuários ou recursos.

Essa característica do modelo *PerDiS* é proporcionada pela interação com os serviços que formam o *Subsistema de reconhecimento de contexto e adaptação* do *middleware* EXEHDA. A seguir, será apresentada uma visão geral de tais serviços, e como eles interagem com o *PerDiS*.

4.3.1 Serviços para suporte ao reconhecimento de contexto e adaptação do *middleware* EXEHDA

O *Subsistema de Reconhecimento de Contexto e Adaptação* do EXEHDA inclui serviços que tratam desde a extração da informação bruta sobre as características dinâmicas e estáticas dos recursos que compõem o ISAMpe, passando pela identificação em alto nível dos elementos de contexto, até o disparo das ações de adaptação em reação a modificações no estado de tais elementos de contexto. Integram este subsistema os serviços *Collector*, *Deflector*, *ContextManager*, *AdaptEngine* e *Scheduler* (YAMIN, 2004). Particularmente, os serviços *AdaptEngine* e *Scheduler* são responsáveis, respectivamente, pelo controle das adaptações de cunho funcional e não-funcional. Entende-se por adaptação funcional aquela que implica a modificação do código sendo executado. Por sua vez, adaptação não-funcional é aquela que atua sobre a gerência da execução distribuída como, por exemplo, no reposicionamento de objetos de execução entre EXEHDAnodes. A seguir, esses serviços serão brevemente descritos:

- *Collector:* o serviço *Collector* é responsável pela extração da informação bruta (diretamente dos recursos envolvidos) que, posteriormente refinada, dará origem aos elementos de contexto. Para isto, ele aglutina a informação oriunda de vários componentes monitores e a repassa aos consumidores registrados. Um monitor gerencia um conjunto de sensores parametrizáveis. Por outro lado, entre os consumidores da informação extraída pela monitoração estão os serviços *ContextManager* e *Scheduler*. Na arquitetura de monitoração definida para o EXEHDA, cada sensor contribui com a extração de um valor que descreve um aspecto específico, estático ou dinâmico, do recurso sendo monitorado. O conjunto dos sensores existentes em um dado EXEHDAnode, assim como os parâmetros suportados por cada um destes sensores, integra a informação de descrição daquele nó, disponibilizada no serviço *CIB*. O controle da condição de publicação entre o *Collector* e os consumidores registrados é feito por meio do parâmetro *threashold*, que configura o limiar de variação do dado, de forma individualizada para cada sensor, acima do qual uma publicação do novo valor registrado pelo sensor é realizada. Além disso, a extração da informação junto aos monitores não ocorre em momentos arbitrários, mas apenas em instantes discretos, múltiplos de um determinado *quantum* de tempo (fornecido como parâmetros de configuração do serviço *Collector* em cada EXEHDAnode);

- *Deflector:* o objetivo do serviço *Deflector* é disponibilizar a abstração de canais de *multicast* para uso na disseminação das informações monitoradas. A presença do *Deflector* é decorrência da busca de escalabilidade para a arquitetura de monitoração do EXEHDA (MORAES et al., 2005);

- *ContextManager:* o serviço *ContextManager* é responsável pelo refinamento (tratamento) da informação bruta produzida pela monitoração para produção de informações abstratas referentes aos elementos de contexto. A definição de um contexto é feita com o auxílio de uma descrição (XML) de como o dado referente àquele elemento de contexto deve ser produzido a partir da informação proveniente da monitoração, e de todos os estados possíveis para aquele elemento de contexto. Após a definição de um dado contexto, os interessados

no recebimento das informações referentes a este podem registrar-se junto ao *ContextManager*;

- *AdaptEngine:* o *AdaptEngine* é responsável pelas adaptações de cunho funcional. Este serviço provê facilidades para definição e gerência de comportamentos adaptativos por parte das aplicações. Deste modo, libera o programador de gerenciar os aspectos de mais baixo nível envolvidos na definição e liberação dos elementos de contexto junto ao *ContextManager*. *O AdaptEngine* realiza a parametrização do serviço *ContextManager*, a partir das definições providas através do arquivo *context.xml*, para sintetização dos elementos de contexto de interesse de cada aplicação e registra seu interesse em receber notificações de alterações no estado daqueles elementos de contexto. Face à notificação de alteração no estado de algum elemento de contexto, o serviço *AdaptEngine* é responsável pela gerência e notificação dos componentes registrados como sensíveis àquele elemento de contexto;

- *Scheduler:* por último, o *Scheduler* é o serviço central na gerência das adaptações de cunho não-funcional no EXEHDA, isto é, que não implicam alteração de código. Nesse sentido, o *Scheduler* emprega a informação de monitoração, obtida junto ao serviço *Collector*, para orientar operações de mapeamento. Essas operações decorrem de instanciações remotas ou migrações realizadas no ambiente de execução, ou em chamadas de reescalonamento, originadas quando o estado atual de um recurso não satisfizer mais as necessidades de um objeto anteriormente a ele alocado.

4.3.2 *PerDiS:* interação com serviços do *middleware* no processo de adaptação

No modelo *PerDiS*, a funcionalidade de adaptação ao contexto de usuários e recursos é obtida através da interação direta do *discovery proxy* com o serviço *ContextManager*, e indiretamente com os demais serviços do *Subsistema de reconhecimento e adaptação ao contexto*. Nesse caso, o componente *discoverer proxy*, através da implementação da interface *ContextListener* (disponibilizada pelo *ContextManager*), tem condições de receber informações relacionadas a alterações no estado do contexto do ambiente de execução e assim prover um comportamento adaptado a uma determinada situação de interesse (figura 4.3).

Um caso particular de adaptação ao contexto é o tratamento da desconexão de uma instância *proxy*. Nessa situação, um *proxy* deve ter condições de prover uma semântica bem definida às operações de pesquisa e gerenciamento de recursos, mesmo que de uma forma mais restrita. Durante um período em que a conectividade esteja prejudicada, o registro de um recurso deve ser realizado apenas na lista de recursos local, e em um segundo momento, essa informação deve ser propagada para as instâncias *base* do serviço de descoberta. De forma semelhante, a consulta de recursos, normalmente realizada junto à lista de recursos da instância celular (figura 4.4 (a)), deve ser realizada apenas sobre a lista de recursos local caso o usuário que a esteja realizando esteja desconectado (figura 4.4 (b)); quando a conexão for restabelecida, as consultas devem voltar a ser realizadas através da comunicação entre a instância *proxy* e a instância *base* (figura 4.4 (c)). Nessa estratégia é empregado um *timeout* de reconexão informado pelo usuário, de forma que o mesmo possa aguardar

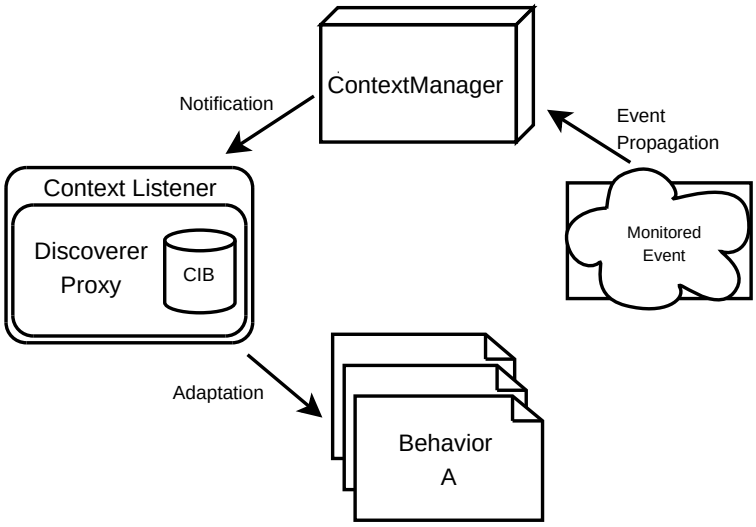

Figura 4.3: Processo de adaptação no modelo *PerDiS*

durante um determinado tempo (antes de realizar a operação local) o restabelecimento da conexão. Caso esse limite de tempo seja atingido, a operação é executada localmente.

Assim, as operações (tanto de gerenciamento como de pesquisa de recursos) podem ser adaptadas de forma que seja provida uma funcionalidade bem definida na ocorrência de problemas de conectividade. Quando os aspectos relacionados à conexão não apresentarem problemas, a instância *proxy* do serviço de descoberta será responsável apenas por realizar a propagação das operações para uma ou mais instâncias do tipo *base*. Nessas condições, um *proxy* não é responsável pelo tratamento das operações localmente, mas sim em colaboração com a(s) instância(s) do tipo *base* do serviço de descoberta.

O tratamento referente à adaptação a uma determinada alteração do contexto deve ser explicitamente previsto e definido pelo componente *proxy*. Inicialmente, um *proxy* deve registrar junto ao serviço *ContextManager* do *middleware* o interesse no recebimento de um ou mais eventos sensoriados pelo *Subsistema de reconhecimento de contexto e adaptação*. Ao ser notificado da ocorrência de alguma alteração de interesse no contexto de execução através do recebimento de um evento, o *proxy* deve implementar o comportamento para tratar essa situação (modificação de *flags* ou alterações de referências para objetos, por exemplo). Caso o *proxy* não reconheça o evento recebido ou não tenha nenhum comportamento alternativo para realizar seu tratamento, este é simplesmente ignorado e nenhuma adaptação é realizada.

A mesma estratégia para o tratamento da adaptação à desconexão pode ser estendida para outros contextos como, por exemplo, memória disponível no dispositivo. Nesse caso, a adaptação poderia referir-se a algum tipo de limitação/redução das funcionalidades do *proxy*, a fim de evitar o consumo excessivo dos recursos em um dispositivo. Trabalhos futuros irão avaliar outras situações que possam beneficiar-se de um comportamento adaptativo do serviço *PerDiS*.

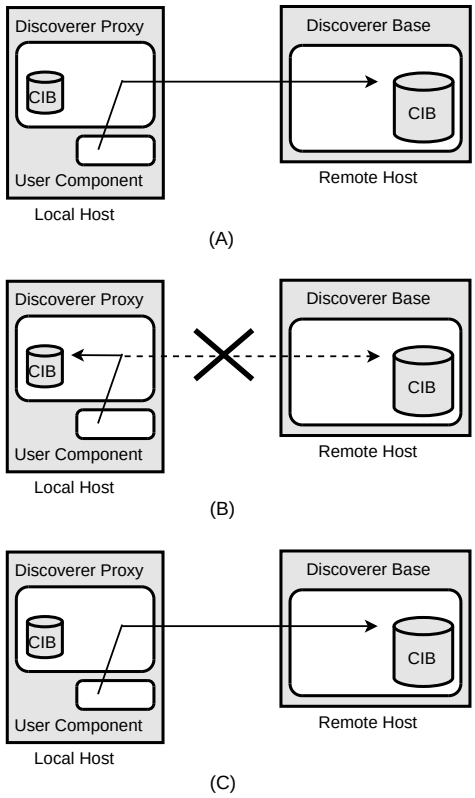

Figura 4.4: Exemplo de adaptação no modelo *PerDiS*

4.4 Manutenção da consistência e redundância de componentes

Uma característica bastante comum num ambiente saturado de dispositivos computacionais (onde muitos destes podem ser móveis) é a possibilidade constante de que estes possam sofrer desconexões ou estar indisponíveis em momentos críticos. Nesse sentido, há a preocupação de prover um mecanismo que possa manter a consistência do catálogo de recursos disponíveis para acesso. O conceito referente à manutenção da consistência empregado pelo *PerDiS* unifica as estratégias para gerenciamento de atualizações relacionadas aos recursos e para detecção de defeitos, apresentadas por Dabrowski e Mills (2002). Dessa forma, informações atualizadas sobre recursos que estejam realmente *online* são mantidas no catálogo de recursos (local ou celular), e informações sobre recursos não disponíveis são descartadas tão logo quanto possível. Além disso, outra necessidade diz respeito à replicação de instâncias de *discoverer bases* dentro de uma célula de execução do ISAMpe, garantindo uma maior disponibilidade do serviço celular de descoberta caso uma ou mais instâncias dessa entidade falhem.

4.4.1 Manutenção da consistência do catálogo de recursos

A manutenção da consistência do catálogo de recursos (mantendo os recursos que estejam realmente disponíveis e descartando os demais) é realizada através do envio periódico de mensagens de renovação de estado pelos *resource components*. Por sua vez, as instâncias do serviço de descoberta (*proxy* e *base*) nas quais um recurso está registrado são responsáveis por realizar o monitoramento dessas mensagens, descar-

tando uma determinada entrada do catálogo de recursos caso esta não tenha sido renovada durante um certo período de tempo. Esse período de tempo é denominado *lease time*, sendo o mesmo conceito empregado pelo Jini (ARNOLD, 1999).

Mais especificamente, o módulo *lease renewer* do *resource component* é responsável por enviar periodicamente mensagens propriamente destinadas a renovar o tempo de *lease* de um determinado recursos. A periodicidade dessas mensagens é especificada no arquivo de descrição das propriedades do recurso. De uma forma bastante simplificada, o *lease renewer* consiste de uma *thread* que fica, a intervalos definidos, enviando uma mensagem de renovação de *lease* ao *discoverer proxy* local, para que este propague a mensagem às instâncias do tipo *base* da célula. Junto a cada mensagem de renovação, é enviado o descritor do recurso e um *flag* que indica se o mesmo foi alterado recentemente. O envio do descritor junto à mensagem de renovação permite que a própria operação faça a reinserção automática do recurso na *CIB* de um *discoverer proxy/base* caso a renovação chegue após o recurso ter sido descartado do catálogo (após o *lease* do recurso em questão ter expirado). O *flag* que indica a alteração no descritor do recurso, por outro lado, permite que a mensagem de renovação atualize automaticamente as propriedades do recurso junto às instâncias de descoberta, no momento em que o *lease* do recurso tiver que ser renovado.

Em contrapartida, o módulo *lease manager* de um *discoverer proxy/base* é responsável pelo monitoramento do recebimento das mensagens de renovação referentes a cada um dos recursos catalogados. É mantido um controle que emite uma notificação ao *discoverer* sempre que um recurso tiver seu *lease* expirado, devido ao fato do mesmo não ter sido renovado em tempo hábil. Nesse caso, é realizado o descarte da entrada referente ao recurso mantido no catálogo. Ao chegar uma mensagem de renovação de *lease* para um determinado recurso, a data de expiração do recurso em questão é atualizada. Caso o recurso já tenha sido descartado da *CIB*, a entrada é reinserida automaticamente com as informações contidas na mensagem. Por fim, se a mensagem indicar que o descritor do recurso foi atualizado recentemente, os dados referentes ao recurso são atualizados junto ao catálogo.

Dessa forma, uma falha no recurso ou mesmo um particionamento de rede que torne o recurso inacessível remotamente implicará a respectiva remoção da entrada referente ao recurso no(s) catálogo(s) de recursos, devido ao não recebimento das mensagens de renovação de *lease*. No momento em que o problema relacionado ao recurso ou à rede for solucionado, as instâncias de descoberta retornarão a receber as notificações de disponibilidade, e a entrada referente ao recurso será reinserida em seus catálogos. Essa estratégia reduz a possibilidade de que usuários descubram recursos não mais disponíveis (ou inacessíveis), mantendo assim a consistência do catálogo de recursos.

4.4.2 Redundância do catálogo de recursos

Como já mencionado, a redundância da instância celular (*discoverer base*) do descobridor de recursos provê uma maior disponibilidade à arquitetura de descoberta, uma vez que, caso ocorra uma falha em um desses componentes, outras réplicas estarão ativas para atender futuras requisições. Essa redundância não exige a implementação de funcionalidades específicas para sua manutenção, e baseia-se em duas características da arquitetura: *(a)* a localização baseada em *multicast* das instâncias de descoberta no escopo de uma célula de execução e *(b)* a utilização do

serviço *CIB* para manutenção da lista de recursos.

No momento em que uma instância *proxy* do serviço de descoberta é ativada, esta é responsável por enviar para um determinado endereço *multicast* mensagens específicas que anunciam essa situação. O mesmo procedimento é realizado quando uma instância *base* é inicializada (seção 4.2.2). Essa abordagem habilita que uma ou mais instâncias do tipo *base* estejam preparadas para tratar mensagens de anúncios, permitindo que recursos posteriormente façam seu registro de forma independente junto a cada um dos DBs ativos na célula de execução.

Além disso, cada instância do serviço de descoberta utiliza o serviço provido pela *CIB* para manter a gerência de sua lista de recursos (com suas respectivas propriedades). A *CIB* provê uma abstração para um armazenamento persistente gerenciado pelo *middleware*, oferecendo uma interface simples para a utilização de um serviço de armazenamento genérico de informações.

Não há nenhuma garantia de que as *CIBs* de duas ou mais instâncias de *discoverer bases* em uma mesma célula tenham uma visão idêntica dos recursos disponíveis na mesma. Caso ocorra um particionamento de rede em uma célula com duas instâncias do tipo *base*, por exemplo, é possível que parte dos recursos só consiga renovar seu estado junto a uma instância, e a outra parte só consiga fazer o mesmo junto à outra instância. Nesse momento, as duas instâncias do tipo *base* terão visões distintas dos recursos disponíveis no ambiente de execução. Porém, quando o problema do particionamento for solucionado, as mensagens de renovação voltarão a ser propagadas entre as instâncias de descoberta (como descrito na seção 4.2.3) e as instâncias replicadas do tipo *base* irão ter novamente uma visão idêntica dos recursos disponíveis.

4.5 Descrição de recursos e critérios de pesquisa

Um mecanismo de descrição é um requisito básico para o compartilhamento de recursos. Antes que recursos ou usuários possam considerar suas necessidades, há a necessidade desses concordarem na maneira a ser utilizada para descrever os recursos (MCKNIGHT, 2004). Como mencionado anteriormente, uma grande deficiência encontrada nos protocolos existentes para descoberta de recursos é a baixa expressividade na forma utilizada para descrever as propriedades que caracterizam um recurso e os critérios de pesquisa utilizados no processo de descoberta. Essa seção descreve a abordagem proposta no presente trabalho para o tratamento desse requisito de grande importância para uma estratégia de descoberta. Essa proposta está fundamentada em três necessidades básicas: *(a)* expressividade da linguagem de descrição, *(b)* facilidade de extensão da mesma através da inclusão de novos operadores e funcionalidades, e *(c)* possibilidade de interoperabilidade com outras formas de descrição e especificação de requisitos.

Para abordar essas questões, propõe-se a utilização da XML (*eXtensible Markup Language*) (BRAY; PAOLI; SPERBERG-MCQUEEN, 1997) na definição dos descritores de recursos e na definição de critérios de consulta. A notação apresentada a seguir permite expressar de forma adequada propriedades e critérios de pesquisa de recursos, habilitando a utilização de diversos operadores sobre os atributos, e a combinação aninhada de múltiplos critérios de pesquisa. A notação proposta pode ser estendida com facilidade, bastando a definição de novas *tags* que expressem a semântica desejada na linguagem. Além disso, a interoperabilidade com outras linguagens

e notações para especificação de recursos (que sejam também baseadas em XML) reside na possibilidade de conversão entre documentos XML de protocolos distintos através da definição de transformações XSL. XSLT (XSL Transformation) (W3C, 1999) é uma linguagem para transformação de documentos XML em outros documentos XML. A XSLT foi projetada para ser usada como parte da XSL, que é uma linguagem de estilos para XML. A XSL inclui um vocabulário XML para especificação de formatação, e define o estilo de um documento XML usando XSLT para descrever como o documento é transformado em outro documento XML.

4.5.1 Especificação de propriedades de recursos

A especificação das propriedades de um recurso é um procedimento fundamental para habilitar o processo de descoberta de recursos pois é através dessas informações que o mecanismo de descoberta irá determinar quais recursos satisfazem os critérios de pesquisa de um usuário, e quais não satisfazem. O *resource component* é responsável por carregar essas informações junto ao *discoverer proxy/base* no momento em que é feito o registro do recurso.

A figura 4.5 apresenta um exemplo de arquivo descritor de recurso, especificando as propriedades de um *host*. Obrigatoriamente, o arquivo de descrição deve informar uma *descrição* para o recurso, o *tipo* do recurso, o intervalo de *lease* especificado para o envio de notificações de disponibilidade e a *url* para acesso ao recurso. Além desses dados, o arquivo permite a especificação de um número não limitado de propriedades relacionadas ao recurso, definidas por um *nome* e um *valor*.

```
<RESOURCE>
    <DESCRIPTION>Another host in the ISAM pervasive environment</DESCRIPTION>
    <TYPE>Host</TYPE>
    <URL>//curly.inf.ufrgs.br/</URL>
    <LEASE>60000</LEASE>
    <PROPERTIES>
        <PROPERTY name="processorType" value="PentiumIII" />
        <PROPERTY name="cpuMhz" value="2400" />
        <PROPERTY name="memoryRamMb" value="512" />
    </PROPERTIES>
</RESOURCE>
```

Figura 4.5: Arquivo exemplo para descrição de um recurso

A abordagem dada à especificação das características de um recurso habilita, além de um elevado grau de expressividade, a possibilidade de conversão para outro formato XML através da definição de um documento de transformação XSLT. Essa característica tende a beneficiar uma futura interoperabilidade entre protocolos.

4.5.2 Especificação de critérios de pesquisa

Essa seção descreve a estratégia definida para a especificação de propriedades de pesquisa de recursos. Esta foi baseada na avaliação de outras soluções já existentes, buscando-se combinar características de expressividade (Globus RSL (CODDINGTON et al., 2003) e SLP (BETTSTETTER; RENNER, 2000)) e possibilidade de interoperabilidade (UPnP (MICROSOFT CORPORATION, 2000)) encontradas nas mesmas.

A figura 4.6 apresenta um exemplo de consulta especificando um conjunto de propriedades referentes a um *host*. A consulta define que deve ser localizado um recurso do *tipo host*, considerando um dado conjunto de critérios. O campo *ttl* está relacionado ao escopo do processo de descoberta, que pode ser realizado sobre uma única célula *(ttl=1)* ou envolver células vizinhas no processo de descoberta. Além disso, uma série de critérios de pesquisa podem ser especificados, definidos por um *nome*, um (ou mais) *valor(es)* e um *operador*. A lista de operadores reconhecidos como válidos é apresentada no quadro 4.1.

```
<QUERY>
    <TTL>1</TTL>
    <TYPE>Host</TYPE>
    <CRITERIA name="processor" value="PentiumIII;PentiumIV" op="mul" />
    <CRITERIA name="cpuMhz" value="1600" op="equ" />
    <CRITERIA name="memoryRamMb" value="256" op="geq" />
</QUERY>
```

Figura 4.6: Exemplo de uma especificação de critérios de pesquisa

Operador	Representação
Igual (=)	equ
Diferente (≠)	neq
Maior (>)	gre
Maior ou igual (≥)	geq
Menor (<)	les
Menor ou igual (≤)	leq
Multi-valorado (|)	mul
Intervalo ([])	ran

Quadro 4.1: Lista de operadores válidos para definição de critérios

Apesar dessa estratégia permitir expressar diferentes relações sobre os valores referentes aos critérios de pesquisa, foi identificada a necessidade de combinar diferentes critérios de pesquisa para satisfazer uma condição maior. O seguinte exemplo ilustra essa situação: *"Localizar um host com um processador Pentium III, e que possua uma CPU de 1600 MHz e 512 MB de memória RAM ou que possua uma CPU de 2400 MHz e 256 MB de memória RAM"*. Situações como essa, envolvendo um agrupamento de dois ou mais conjuntos de condições são tratadas através da definição de ramificações (*branches*) na especificação da consulta. A figura 4.7 ilustra a especificação de uma consulta representando a situação descrita anteriormente. Além disso, ramos podem ser definidos de forma aninhada, permitindo múltiplos níveis de critérios combinados. Um recurso irá satisfazer a especificação de uma consulta se ao menos uma ramificação for satisfeita, adicionalmente aos critérios gerais da consulta.

```
<QUERY>
    <TTL>1</TTL>
    <TYPE>Host</TYPE>
    <CRITERIA name="processor" value="PentiumIII" op="equ" />
    <BRANCHES>
        <BRANCH>
            <CRITERIA name="cpuMhz" value="1600" op="equ" />
            <CRITERIA name="memoryRamMb" value="512" op="equ" />
        </BRANCH>
        <BRANCH>
            <CRITERIA name="cpuMhz" value="2400" op="equ" />
            <CRITERIA name="memoryRamMb" value="256" op="equ" />
        </BRANCH>
    </BRANCHES>
</QUERY>
```

Figura 4.7: Exemplo de uma especificação de critérios de pesquisa contendo ramos

4.6 Descoberta de recursos geograficamente dispersos

Utilizando o conceito de células de execução na composição do ambiente *pervasivo* proposto pelo ISAMpe (como descrito na seção 2.4.2), é apresentada uma arquitetura que possibilita sua utilização na descoberta de recursos geograficamente dispersos. A estratégia em questão habilita uma descoberta orientada a escopos, permitindo a realização do processo de descoberta limitado ao escopo de uma única célula de execução (compreendendo, portanto, apenas os dispositivos e serviços existentes no contexto da célula), ou estendendo o escopo da consulta de forma a incluir as células de execução vizinhas. O escopo do processo de descoberta é especificado como um dos atributos de pesquisa na própria mensagem de *request*. Ao receber essa mensagem, o *discoverer base* verifica se ele será o único responsável pelo processo de descoberta ou se outros *discoverer bases* em células vizinhas serão envolvidos. O *neighborhood broker* é o módulo do *discoverer base* destinado a realizar a comunicação entre DBs residentes em células vizinhas. Em oposição ao modelo centralizado de comunicação existente no contexto de uma célula de execução, a comunicação entre DBs vizinhos na realização do processo de descoberta é uma interação *peer-to-peer* (FOX, 2001).

Esse modelo híbrido de interação na realização da descoberta de recursos torna a solução proposta aplicável em contextos onde os recursos a serem descobertos podem estar dispersos geograficamente.

4.6.1 Redes *peer-to-peer*

Peer-to-peer (ou simplesmente P2P) é uma classe de sistemas ou aplicações auto-organizáveis que utilizam recursos distribuídos - armazenamento, processamento e informação - disponíveis nas fronteiras da Internet (TALIA; TRUNFIO, 2003). Dois nós que mantenham uma conexão aberta, ou aresta, entre si são denominados *vizinhos*. O número de vizinhos que um nó possui é denominado *grau de saída*. As mensagens são roteadas somente entre essas conexões abertas, e se uma mensagem necessita viajar entre dois nós que não sejam vizinhos, ela irá trafegar sobre múltiplas arestas. O comprimento do caminho pelo qual trafega uma mensagem é denominado *hop count* (YANG; GARCIA-MOLINA, 2003).

Em sistemas P2P tradicionais, o nó originador envia sua consulta para todos

os seus vizinhos (através de uma estratégia de *flooding*). Outros protocolos de roteamento, porém, podem enviar uma consulta para um subconjunto específico de vizinhos, de forma a melhorar a eficiência do algoritmo. Além disso, o nó receptor pode propagar a consulta para os seus vizinhos. Geralmente, um campo *time to live* (TTL) é atribuído às mensagens de consulta, o qual especifica o número máximo de *hops* que uma mensagem pode atingir. Ao receber uma consulta, um nó decrementa o TTL da mensagem e, se o mesmo for maior que 0, propaga a mensagem para seus vizinhos. O número de nós que processam a mensagem enviada pelo nó originador é chamado de *alcance* da consulta.

A literatura relacionada a algoritmos e sistemas de pesquisa P2P é bastante extensa. Tsoumakos e Roussopoulos (2003) apresentam uma comparação entre diversas estratégias de pesquisa P2P, avaliando o desempenho de cada uma delas. Pedone e colegas (PEDONE; DUARTE; GOULART, 2002) discutem o problema relacionado a encontrar recursos em sistemas largamente distribuídos, utilizando algoritmos P2P sob uma perspectiva probabilística. Além disso, Mischke e Stiller (2004) descrevem uma metodologia para classificação e projeto de sistemas de pesquisa P2P, e um levantamento das principais tecnologias nessa área.

4.6.2 Aplicando uma abordagem *peer-to-peer* no ISAMpe

Talia e Trunfio (2003) identificam uma série de semelhanças e diferenças entre a tecnologia P2P e a computação em grade, num esforço de colaborar esforços a fim de que a evolução desses conceitos possa ocorrer de forma convergente. Esse estudo propõe a utilização do modelo *Super-Peer* (YANG; GARCIA-MOLINA, 2003) num cenário que integre os dois paradigmas. Considerando os aspectos de distribuição geográfica e organização celular do ISAMpe, os quais lembram a organização espacial de uma grade, o presente trabalho propõe a utilização do mesmo modelo também nesse ambiente.

Um *super-peer* é um nó diferenciado em uma rede *peer-to-peer* que atua como um servidor centralizado para um subconjunto de clientes, e como um igual em uma rede de *super-peers*. Um *super-peer* responde submissões de consultas realizadas por clientes locais, mas também se conecta a outros *super-peers* (como um sistema puramente P2P) para submeter e responder consultas (figura 4.8). Um *super-peer* mantém um índice sobre os dados de seus clientes, de forma que essa informação seja suficiente para responder todas as consultas nele realizadas.

Sistemas P2P como o Napster (NAPSTER WEBSITE, 2004), apesar de permitirem a transferência direta de arquivos entre seus usuários, utilizam um repositório central para indexar os recursos disponíveis na rede. Essa abordagem tende a gerar um gargalo de desempenho e escalabilidade no sistema. Por outro lado, sistemas puramente P2P tendem a ser ineficientes: as primeiras versões do *software* Gnutella (GNUTELLA WEBSITE, 2004), por exemplo, utilizavam a técnica de *flooding* para difundir uma mensagem na rede. Outra fonte de ineficiência nesses sistemas se refere aos gargalos causados pela capacidade muito limitada de alguns *peers*. Todos os *peers* em uma rede Gnutella recebem papéis e responsabilidades idênticas, independentemente de suas capacidades individuais. Um sistema mais eficiente poderia tirar vantagem da considerável heterogeneidade da capacidade dos *peers*. Redes de *super-peers* procuram um balanceamento entre a eficiência de uma pesquisa centralizada, com a autonomia, balanceamento de carga e robustez providas por uma busca distribuída. Uma vez que clientes ficam protegidos de todo o

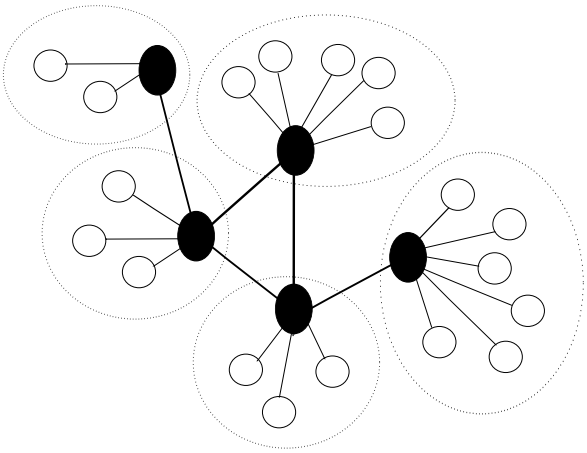

Figura 4.8: Rede de *super-peers* - nós pretos representam *super-peers* e nós brancos representam clientes

processamento de consultas e tráfego, *peers* de menor capacidade podem abrigar clientes, enquanto o núcleo do sistema pode executar de forma eficiente em uma rede de poderosos *super-peers*.

Porém, duas questões importantes não são tratadas diretamente pela abordagem de *super-peers*, e devem ser discutidas:

- *Redundância de super-peers:* o conceito de *super-peers* introduz um ponto único de falhas e um potencial gargalo, mesmo que somente a uma pequena porção do sistema. Yang e Garcia-Molina (2003) apresentam o conceito de *super-peer k-redundant*, onde k nós compartilham a carga de um *super-peer*, formando um único *super-peer* virtual. Nessa organização, cada nó que compõe o *super-peer* virtual (chamado de *parceiro*) é conectado a todos os clientes locais, e possui o índice completo de dados dos clientes. Além disso, as consultas são enviadas a cada um dos parceiros através de uma estratégia de *round-robin*, a fim de realizar o balanceamento de carga entre os nós parceiros. Uma vez que todos os parceiros podem responder consultas, caso um em particular falhe, os outros podem continuar a servir as consultas realizadas por clientes (ou *super-peers* vizinhos);

- *Estratégia para roteamento de mensagens entre super-peers:* a utilização de técnicas de *flooding* na propagação de consultas entre entidades *super-peers* vizinhas pode comprometer a eficiência do sistema. Em redes muito grandes e com várias conexões entre vizinhos (mesmo que considerando-se apenas as ligações entre *super-peers*), contatar todos os nós adjacentes para localizar um recurso é claramente pouco viável. Menascé (2003) propõe o *Probabilistic Search Protocol*, a fim de evitar o *flooding* de mensagens entre os *peers*. Esse protocolo utiliza o parâmetro *broadcast probability (p)* para selecionar um subconjunto de vizinhos para propagar as mensagens de pesquisa de recursos, reduzindo assim a quantidade de mensagens trafegadas na rede.

Esses dois aspectos são considerados na modelagem de uma busca em larga-escala pelo *PerDiS*. Como será visto a seguir, o conceito de *super-peer* é mapeado para um

discoverer base no modelo *PerDiS*, o qual é responsável pelo tratamento de consultas originadas por componentes internos a uma célula de execução, como também pelo tratamento de consultas propagadas por *discoverer bases* residentes em células vizinhas. A redundância de *discoverer bases* é discutida na seção 4.4.2, e habilita a constituição de *super-peers* virtuais, como descrito anteriormente. Já a estratégia de roteamento de consultas entre *discoverer bases* localizados em células vizinhas emprega o conceito de *broadcast probability*, parâmetro que pode ser ajustado a fim de melhorar a eficiência na propagação de consultas.

4.6.3 Interação entre EXEHDAcells

O modelo *PerDiS* considera cada *discoverer base* como um *super-peer* no ISAMpe. Mais especificamente, o módulo *neighborhood broker* de um DB é utilizado para realizar a comunicação com as instâncias *discoverer base* residentes em EXEHDAcells vizinhas, constituindo uma rede de *super-peers*. Um *discoverer base* mantém um catálogo dos recursos gerenciados dentro da própria célula. Dentro de uma célula, o modelo de descoberta é centralizado, e toda a comunicação é realizada ente os *discoverer proxies* e a instância celular do serviço de descoberta (DB).

O *discoverer base* foi concebido como um serviço do *middleware* que executa na EXEHDAbase de uma EXEHDAcell, juntamente com outros serviços. A EXEHDAbase distingue-se de outros EXEHDAnodes pelo seu poder de processamento diferenciado. Esse aspecto é compatível com a premissa de que redes de *super-peers* podem utilizar de uma melhor forma capacidades diferenciadas de nós heterogêneos. Nesse caso, um EXEHDAbase é usado para hospedar um *discoverer base*, o qual atua como um *super-peer* no processo de descoberta realizado pelo *PerDiS*.

4.6.4 O algoritmo de pesquisa *peer-to-peer* empregado pelo *PerDiS*

Essa seção descreve o algoritmo de pesquisa utilizado na descoberta de recursos em larga-escala pelo *PerDiS*. O algoritmo em questão emprega os conceitos de *super-peers* (representados por *discoverer bases* em uma EXEHDAcell) e *broadcast* probabilístico na propagação de consultas a *discoverer bases* localizados em células vizinhas.

Um *discoverer base* descobre as células que estão em sua vizinhança (e conseqüentemente seus *discoverer bases* vizinhos) através de uma consulta à *CIB* local. O modelo atual considera que todas as células vizinhas têm igual probabilidade de receber a propagação da requisição de consulta. Dessa forma, pode-se dizer que a escolha de quais vizinhos irão propagar a consulta é um processo de seleção puramente randômico. Trabalhos futuros, porém, irão aprofundar essa questão, permitindo que sejam associados graus de preferência aos vizinhos de uma célula a fim de realizar um processo de seleção mais inteligente. Esses graus de preferência podem estar relacionados, por exemplo, aos custos de comunicação com uma célula, aos custos monetários para utilização dos equipamentos em uma célula, à carga computacional corrente dos equipamentos em uma célula, ou a uma combinação desses parâmetros.

É considerado que cada *discoverer base db* possui uma vizinhança $V(db)$ definida como o conjunto de *discoverer bases* encontrados nas células adjacentes à célula onde *db* está localizado. Um *discoverer base* mantém também uma lista com as requisições processadas recentemente, de forma a evitar o processamento cíclico das mesmas. Por último, o campo TTL contido em uma mensagem de requisição desempenha um papel fundamental na especificação do escopo do processo de descoberta definido

pelo usuário do serviço *PerDiS*, o qual pode abranger apenas a célula local *(ttl=1)*, envolver as células diretamente vizinhas *(ttl=2)*, envolver as células vizinhas das vizinhas *(ttl=3)*, e assim por diante. A figura 4.9 apresenta o algoritmo implementado e executado por uma instância *discoverer base*.

```
requestResource (request) {

    if (request.ID in recentRequests) {
        results = null                          /* se requisição já foi processada, retorna */
    } else {
        results = match(request)               /* realiza processamento sobre catálogo local */
        recentRequests.add(request.ID)
        if (request.TTL > 0)  {
            request.TTL = request.TTL - 1
            for every v in V(db) {
                resultsTmp = v.requestResource(request)   /* executa considerando o          */
                results = merge(results, resultsTmp)      /* parâmetro broadcast probability */
            }
        }
    }
    return results
}
```

Figura 4.9: Algoritmo de pesquisa *peer-to-peer*

4.7 Considerando preferências de usuários na descoberta de recursos

Uma importante característica do modelo *PerDiS* é a sua adaptação ao usuário, ou personalização. Mais especificamente, o *PerDiS* permite que ao longo de sua utilização o serviço passe a considerar informações relativas aos seus usuários para melhorar o desempenho e/ou a qualidade dos resultados obtidos.

Essa funcionalidade é obtida através da interação com o *Ambiente Virtual do Usuário* (*AVU*) do *middleware* EXEHDA. A premissa siga-me definida no ISAM reflete-se não só nas aplicações que um usuário normalmente executa, mas no seu ambiente computacional como um todo (AUGUSTIN, 2003). Este engloba, além das aplicações em execução, as informações de personalização das aplicações definidas pelo usuário, o conjunto de aplicações instaladas (i.e. passíveis de serem disparadas por aquele usuário), como também seus arquivos privados. É atribuição do serviço *AVU* a manutenção do acesso *pervasivo* a este ambiente virtual, da forma mais eficiente possível (YAMIN, 2004). Por acesso *pervasivo* entende-se que as informações disponibilizadas no *AVU* de um usuário devem estar disponíveis para este usuário a qualquer hora, em qualquer lugar e através de qualquer dispositivo.

A interação com o serviço *AVU* permite que o *PerDiS* utilize determinadas preferências referentes aos seus usuários na realização do processo de descoberta. Essas preferências, previamente armazenadas no *AVU* de cada usuário, são usadas para refinar o processo de descoberta. Outras estratégias de descoberta de recursos, como o Allia (RATSIMOR et al., 2004), utilizam personalização por usuário, porém essa se aplica principalmente na parametrização do gerenciamento dos recursos disponibilizados no ambiente móvel (como taxa de envio de anúncios, por exemplo). O Allia utiliza uma personalização baseada em políticas definidas previamente para cada

usuário, o que em tese poderia ser utilizada também para parametrizar preferências nas operações de pesquisa de recursos. Em termos de expressividade, considera-se que a utilização das informações mantidas no *AVU* ou a definição prévia de políticas de personalização são estratégias equivalentes.

O emprego das informações de personalização por usuário é feito opcionalmente (caso informações relevantes para o usuário em questão estejam armazenadas no *AVU*) pela instância *base* do serviço de descoberta após essa ter processado completamente uma requisição de pesquisa. Após todo o processo de combinação de critérios ter sido realizado, as informações de personalização podem ser utilizadas para refinar o conjunto de resultados a ser retornado para o usuário (figura 4.10). Nesse momento, o módulo *preference manager* do *discoverer base* é responsável por interagir com o *AVU* do usuário e obter dados que possam ser aplicados sobre o retorno original da consulta, produzindo o conjunto de resultados final que será retornado ao usuário.

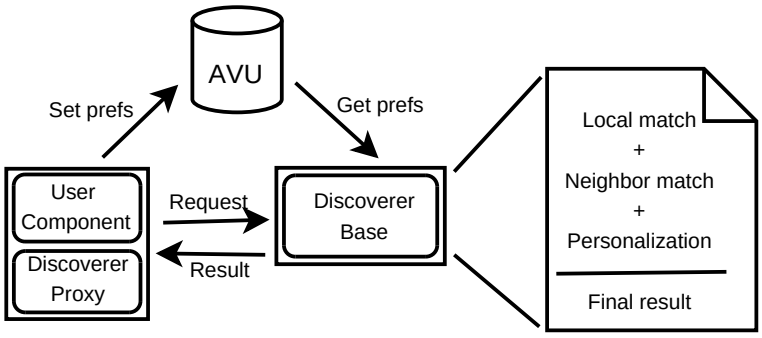

Figura 4.10: Utilização das informações de personalização no processo de descoberta

4.7.1 Exemplos de personalização do serviço de descoberta

Inicialmente, considera-se alguns cenários onde a personalização do serviço de descoberta de recursos pode ser útil: como já mencionado anteriormente, as estratégias de descoberta geralmente consideram apenas o "presente" no processo de descoberta, não levando em conta interações passadas entre usuários e recursos. Um incremento significativo nas funcionalidades de um serviço de descoberta seria a possibilidade de manter informações que permitissem o refinamento de uma consulta de recursos futura, baseado em informações obtidas a partir de consultas feitas no passado. A seguir, será apresentado como o *PerDiS* trata essa situação.

Ao receber os resultados de uma determinada pesquisa, o serviço de descoberta permite que o usuário sinalize uma opção de preferência (ou uma opção *default*) dentre os resultados retornados. Essa informação pode então ser usada para que consultas futuras, que potencialmente encontrem o mesmo conjunto de recursos, retornem para o usuário apenas seu recurso favorito, caso a preferência ainda seja válida. Como exemplo, pode ser citada uma consulta por impressoras que um determinado usuário realiza uma ou mais vezes por dia em seu ambiente de trabalho: supondo-se *(a)* que o serviço de descoberta encontre sempre três impressoras distintas para a pesquisa do usuário, e *(b)* que posteriormente o usuário necessite selecionar qual delas deseja utilizar, a utilização personalizada do serviço de descoberta poderia eliminar o segundo passo através da definição de uma opção padrão que seria

armazenada junto ao *AVU* daquele usuário. Caso o conjunto de resultados encontrado em uma consulta não esteja contido no conjunto de resultados armazenados no AVU, todo o novo conjunto de resultados será apresentado ao usuário (mesmo que a opção padrão esteja nele).

Outra situação que pode beneficiar-se da utilização de preferências seria a definição de critérios de ordenamento de recursos na listagem de resultados retornados nas pesquisas de cada usuário. Nesse caso, um determinado usuário poderia definir que gostaria que suas consultas por recursos do tipo *host* viessem ordenadas em ordem decrescente da velocidade do processador (atributo *cpuMhz*), enquanto que um outro usuário poderia definir algum outro critério de ordenamento para o mesmo tipo de recurso como, por exemplo, ordem decrescente da quantidade de memória RAM existente no nó (atributo *memoryRamMB*).

A figura 4.11 apresenta um exemplo do conjunto de dados mantido junto ao *AVU* para informar as preferências de um usuário em relação ao processo de descoberta de recursos, considerando as duas situações descritas anteriormente. Especificamente, ele determina que uma consulta por recursos do tipo *host* que encontre as três alternativas indicadas apresente ao usuário apenas o recurso marcado como opção padrão (no caso, o terceiro recurso da lista). Quanto aos critérios de ordenamento, é especificado que uma consulta por recursos do tipo *host* sempre irá trazer os resultados em ordem decrescente da velocidade de processamento (atributo *cpuMhz*), e que uma consulta do tipo *printer* sempre irá trazer os resultados em ordem decrescente da capacidade de impressão de páginas por minuto (atributo *ppm*).

```
<PREFERENCES>
  <RESULTSETS>
    <RESULTSET resourceType="Host" >
      <RESOURCE name="strokes:1:1555454522" />
      <RESOURCE name="razorlight:2:4455314527" />
      <RESOURCE name="libertines:3:8765451232" default="yes" />
    </RESULTSET>
  </RESULTSETS>
  <ORDERBY>
    <ORDER resourceType="Host" property="cpuMhz" orderType="desc" />
    <ORDER resourceType="Printer" property="ppm" orderType="desc" />
  </ORDERBY>
</PREFERENCES>
```

Figura 4.11: Exemplo de definição de preferências de um usuário

Esses são exemplos simples mas que podem agregar um grande valor às funcionalidades de um mecanismo de descoberta. A interação com o serviço *AVU* do *middleware* permite que o serviço de descoberta tenha acesso a uma ampla variedade de informações referentes a cada usuário do ambiente *pervasivo*. Pesquisas futuras irão explorar outros aspectos que podem ser personalizados para cada usuário no processo de descoberta.

5 PERDIS: ASPECTOS DE IMPLEMENTAÇÃO, ESTUDO DE CASO E RESULTADOS OBTIDOS

Esse capítulo tem por objetivo apresentar os principais aspectos relacionados à implementação do serviço *PerDiS*, bem como alguns resultados obtidos. Inicialmente, serão discutidos aspectos referentes à linguagem de programação utilizada e as principais classes do modelo. Será apresentado em seguida o *VisualUserComponent*, uma interface gráfica implementada para ilustrar e facilitar a utilização do serviço *PerDiS*. Em seguida, um estudo de caso ilustrará o uso em um cenário real das diferentes funcionalidades do serviço desenvolvido. Finalmente, serão apresentadas algumas medições e resultados referentes à avaliação do *PerDiS*.

5.1 A linguagem de programação Java

Assim como a grande maioria dos serviços do *middleware* EXEHDA, o *PerDiS* também está sendo implementado na linguagem de programação Java. Em função de sua importância no escopo dessa pesquisa, é apresentado a seguir um breve histórico de seu surgimento e uma descrição de suas principais características.

Java (SUN MICROSYSTEMS, 2005) foi criada pela Sun Microsystems a partir de uma pesquisa corporativa interna com o codinome Green que ocorreu no ano de 1991. Deste projeto resultou a linguagem que é baseada em C e C++ e que seu criador, James Gosling, chamou de Oak (Carvalho) em homenagem a uma árvore que havia em frente à janela de seu escritório na Sun. Descobriu-se mais tarde que já havia uma linguagem de programação que se chamava Oak. Quando uma equipe da Sun visitou uma cafeteria local, o nome Java (cidade de origem de um tipo de café importado) foi sugerido e teve seu uso consagrado pela prática.

Naquela época, o projeto Green atravessava algumas dificuldades. O mercado para dispositivos eletrônicos inteligentes destinados ao consumidor final não estava se desenvolvendo tão rapidamente como a Sun havia previsto e um contrato importante pelo qual esta competia fora concedido à outra empresa. Nessas condições, o projeto estava em risco de cancelamento. Porém, graças à popularidade alcançada pela *World Wide Web* em 1993, a equipe da Sun viu imediato potencial na utilização de Java para criar páginas da *Web* com o recurso de conteúdo dinâmico. Isso deu nova vida ao projeto.

Java apresenta inúmeras vantagens quando utilizada para a programação de um sistema distribuído como, por exemplo, a portabilidade de seu código. O compilador Java não gera instruções nativas. Ao invés disso, ele gera *bytecodes* para uma máquina virtual, a Máquina Virtual Java (JVM). Qualquer arquitetura que possua

uma JVM poderá processar os *bytecodes*, tornando o programa Java independente de plataforma.

O fato de Java ser orientada a objetos contribuiu para o sucesso alcançado pela linguagem. A orientação a objetos promove o encapsulamento e a reutilização de código, facilitando o desenvolvimento de projetos envolvendo um grande número de pessoas.

Os projetistas de Java, na Sun Microsystems, eram na verdade programadores C++. Eles entenderam o que havia de melhor na linguagem, suas características e suas limitações, e basearam toda a criação de Java em C++. Ao projetar a linguagem Java, eles copiaram a sintaxe de C++ e reutilizaram seus melhores elementos de projeto. Além disso, eliminaram a maioria dos geradores de erros: ponteiros e gerenciamento de memória via código. Em Java, a alocação de objetos é feita de forma automática, não sendo necessário código específico para essa tarefa. A desalocação de objetos, também é feita de forma transparente ao programador, através do *Garbage Collector* (GC). O GC é uma *thread* que executa em baixa prioridade, liberando objetos que não estejam mais sendo referenciados por outros objetos em um programa Java.

5.2 Parametrização do serviço *PerDiS*

Cada nó existente no ISAMpe é configurado através de um perfil de execução, o qual define e parametriza todos os serviços do *middleware* EXEHDA que devem ser ativados nesse nó (YAMIN, 2004). Esse perfil é descrito em um arquivo XML (*exehda-services.xml*). O formato genérico desse arquivo é apresentado na figura 5.1.

```
<EXEHDA>
    <PROFILE name="profileName">
        <SERVICE name="sName" impl="className" loadPolicy="boot"|"demand">
            <PROP name="paramName" value="paramValue" />
        </SERVICE>
    </PROFILE>
<\EXEHDA>
```

Figura 5.1: Formato do documento de definição do perfil de execução de um nó EXEHDA

Um bloco <PROFILE> define um perfil de execução, sendo o valor do atributo "name" quem define o nome associado àquele perfil. Internamente ao bloco <PROFILE>, blocos <SERVICE> são utilizados para descrever os serviços que integram o perfil de execução. Em relação ao bloco <SERVICE>, o atributo "name" indica o nome canônico do serviço, o atributo "impl" especifica o componente (uma classe Java na versão atual do protótipo do EXEHDA) que deve ser utilizado como implementação para o serviço, e o atributo "loadPolicy" define a política de carga que o núcleo do EXEHDA deve utilizar para o serviço. Nesse sentido, o atributo loadPolicy pode assumir os seguintes valores: *(i)* "boot", indicando que o serviço deve ser carregado durante o *bootstrap* do *middleware*, e *(ii)* "demand", significando que o serviço deve ser carregado somente quando for utilizado pela primeira vez. Finalmente, os elementos <PROP> são utilizados dentro de um bloco <SERVICE>

para definir propriedades que poderão ser recuperadas em tempo de execução pelo serviço, para parametrização de sua execução.

A figura 5.2 ilustra a configuração de uma instância do *middleware* no que se refere ao serviço *PerDiS* (executando como uma instância *base* do serviço de descoberta).

```
<EXEHDA>
    <PROFILE name="test-base">
        ...
        ...
        <SERVICE name="discoverer"
                 impl="org.isam.exehda.services.perdis.DiscovererBase"
                 loadPolicy="boot">
            <PROP name="unicastAddress" value="143.54.12.225"/>
            <PROP name="unicastPort" value="6666"/>
            <PROP name="multicastProxyAddress" value="228.5.6.7"/>
            <PROP name="multicastProxyPort" value="6667"/>
            <PROP name="multicastBaseAddress" value="228.5.6.8"/>
            <PROP name="multicastBasePort" value="6668"/>
            <PROP name="ackTimeout" value="1000"/>
            <PROP name="contactAddress" value="//143.54.12.225/discoverer"/>
        </SERVICE>
        ...
        ...
    </PROFILE>
</EXEHDA>
```

Figura 5.2: Definição do perfil de execução do EXEHDA referente ao serviço *PerDiS*

As propriedades utilizadas na parametrização do serviço *PerDiS* são descritas a seguir:

- *unicastAddress* e *unicastPort*: endereço IP e porta onde a instância local do serviço de descoberta estará executando;

- *multicastProxyAddress* e *multicastProxyPort*: endereço IP *multicast* e porta utilizados para enviar anúncios a instâncias do tipo *proxy* do serviço de descoberta (conforme o protocolo de anúncio descrito na seção 4.2.2);

- *multicastBaseAddress* e *multicastBasePort*: endereço IP *multicast* e porta utilizados para enviar anúncios a outras instâncias do tipo *base* do serviço de descoberta que possam estar executando na mesma EXEHDAcell (conforme o protocolo de anúncio descrito na seção 4.2.2);

- *ackTimeout*: parâmetro que configura o tempo em milisegundos que a instância descobridora deve aguardar após a execução do protocolo de anúncio, antes de prosseguir em operação;

- *contactAddress*: endereço de contato enviado a instâncias remotas (nas mensagens de anúncio) para a comunicação com a instância local do serviço em questão.

5.3 Principais classes do modelo

O código fonte da implementação atual do serviço *PerDiS* compreende 57 classes, com um total de aproximadamente 5500 linhas de código Java, distribuídas em 18 arquivos. A fim de manter um certo grau de modularidade na implementação do *PerDiS*, sempre que possível procurou-se um mapeamento de um-para-um entre os componentes/módulos do modelo e as classes usadas na implementação; muitas vezes, porém, um único módulo (modelo) deu origem a duas ou mais classes (implementação). Além disso, foram contabilizadas também as classes utilizadas na implementação da interface visual do *PerDiS* (descrita na seção 5.4). A figura 5.3 apresenta um diagrama de classes, desenvolvido utilizando-se a notação UML (*Unified Modeling Language*), com as principais classes que compõem o serviço.

As quatro classes descritas a seguir representam os quatro componentes que formam a arquitetura *PerDiS*:

- *ResourceComponent:* trata-se da classe utilizada por cada recurso que deseja se anunciar através do *PerDiS*. O programa executado por essa classe recebe como argumento o arquivo XML correspondente ao descritor do recurso a ser disponibilizado, e contata a instância local do serviço de descoberta. Cada recurso disponibilizado em um determinado *host* tem uma instância da classe *ResourceComponent* associada, porém todos compartilham a mesma referência do serviço local de descoberta;

- *UserComponent:* essa é a classe utilizada pelos usuários do serviço *PerDiS* para realizar o processo de descoberta de recursos, sendo que esses usuários podem ser pessoas, aplicações ou outros serviços do ambiente de execução. Assim como acontece com o *ResourceComponent*, o *UserComponent* também se comunica com a instância local do serviço de descoberta. Através do *UserComponent* o usuário pode especificar parâmetros de pesquisa, solicitar a descoberta de recursos, e verificar os resultados obtidos;

- *DiscovererProxy:* essa classe implementa o serviço local de descoberta de recursos, que executa junto a cada nó do ISAMpe. Apresenta os mesmos métodos para gerenciamento e pesquisa de recursos que aqueles existentes na classe *DiscovererBase* (instância celular do serviço de descoberta). Dessa forma, um recurso ou usuário pode invocar os métodos disponibilizados para gerenciamento/pesquisa de recursos sem necessariamente saber se está interagindo com uma instância local ou celular do serviço de descoberta. Nas operações de gerenciamento de recursos, o *DiscovererProxy* é responsável por atualizar sua *CIB* local e então propagar as requisições para a instância celular (ou mais de uma, caso instâncias replicadas estejam sendo usadas) existente em sua EXEHDAcell. Em relação à operação de pesquisa de recursos, essa requisição é normalmente propagada para a instância celular. Como descrito na seção 4.3, esse componente do modelo *PerDiS* pode ter seu comportamento adaptado para refletir mudanças no contexto de execução. Para isso, a classe *DiscovererProxy* implementa a interface *ContextListener* disponibilizada pelo *middleware*, e através do método *contextChanged()* tem condições de receber e tratar eventos que indiquem tais mudanças no contexto;

- *DiscovererBase:* corresponde à classe que implementa o componente celular do serviço de descoberta, catalogando e gerenciando os recursos no nível da

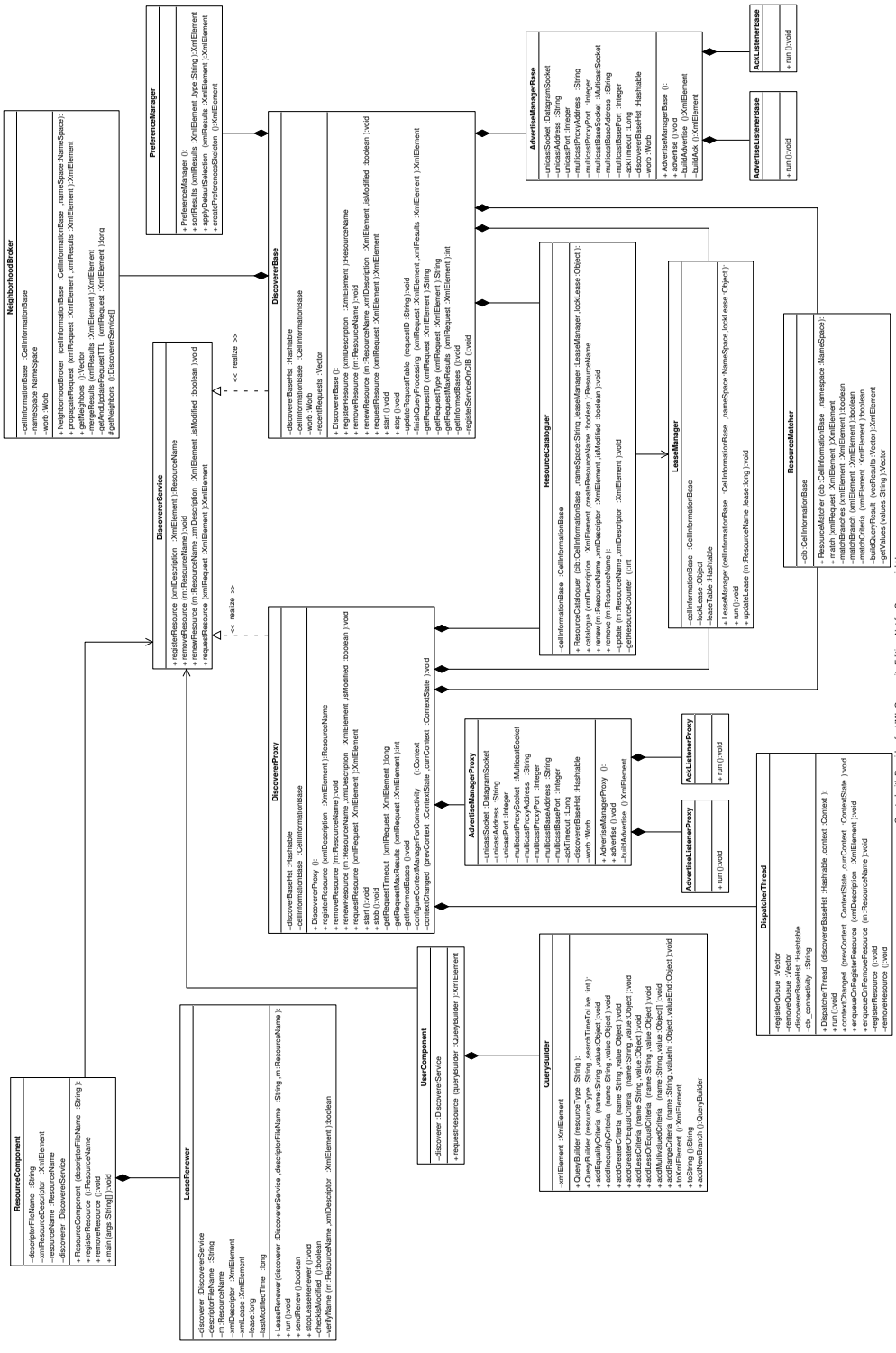

Figura 5.3: Diagrama de classes simplificado do serviço PerDiS

célula de execução. Recebe e processa as requisições enviadas pelas instâncias *DiscovererProxy* da EXEHDAcell. Além disso, possui uma instância da classe *NeighborhoodBroker*, usada para propagar as requisições de pesquisa para outras células de execução, caso necessário. De forma análoga, o *DiscovererBase* também pode receber requisições de pesquisa enviadas por outras instâncias celulares do serviço de descoberta residentes em células vizinhas, mas nessa caso a requisição é indistinguível daquelas enviadas por instâncias do tipo *proxy* residentes na mesma célula.

As principais classes que implementam as funcionalidades específicas dos componentes da arquitetura *PerDiS* são descritas a seguir:

- *AdvertiseManagerProxy:* implementa o protocolo de anúncio descrito na seção 4.2.2, referente à instância *proxy*. Para isso, utiliza um *socket multicast* para o envio de anúncios para a(s) instância(s) *base(s)* do serviço de descoberta, um *socket unicast* para o recebimento de *acks*, e um *socket multicast* para o recebimento de anúncios vindos diretamente de uma ou mais instâncias do tipo *base*;

- *AdvertiseManagerBase:* implementa o protocolo de anúncio descrito na seção 4.2.2, referente à instância *base*. Para isso, utiliza um *socket multicast* para o envio de anúncios para outras possíveis instâncias *base* do serviço de descoberta, um *socket unicast* para o recebimento de *acks*, e um *socket multicast* para o recebimento de anúncios vindos de instâncias do tipo *proxy*;

- *LeaseRenewer:* corresponde a uma *thread* que, junto a uma instância *ResourceComponent*, fica constantemente renovando as informações relacionadas a um determinado recurso. O envio da renovação é feito através do método *renewResource()* da instância *DiscovererProxy* local, e a periodicidade é definida no arquivo XML com as propriedades descritivas do recurso;

- *LeaseManager:* refere-se à *thread* que executa junto a cada instância *proxy* ou *base* do serviço de descoberta e que monitora o recebimento das mensagens de renovação do estado dos recursos. O *LeaseManager* mantém uma tabela *hash* associando cada recurso catalogado a uma data de expiração. Ao constatar que o *lease* relacionado a um recurso expirou sem que o mesmo tenha sido renovado, o *LeaseManager* remove a entrada da tabela *hash* e procede a efetiva remoção do recurso através de sua exclusão na *CIB*;

- *ResourceCataloguer:* classe utilizada para realizar a interface de uma instância do serviço de descoberta (*proxy* ou *base*) com a *CIB*. Possui métodos para a inclusão e alteração das propriedades descritivas de um recurso na *CIB*. Esses métodos recebem diretamente o arquivo XML contendo a descrição de um recurso, e mapeiam cada propriedade desse arquivo para entradas na *CIB*;

- *ResourceMatcher:* trata-se da classe utilizada para combinar a especificação de pesquisa de um usuário com as descrições dos recursos contidos na *CIB*. A comparação inicia-se pelo método *match()*, e prossegue recursivamente através dos métodos *matchCriteria()* e *matchBranches()*, varrendo a especificação do usuário e comparando com os recursos encontrados na *CIB*. Os recursos que

satisfizerem ao menos uma das ramificações da especificação (lembrando que esta pode ter diversos níveis de *branches* aninhados) são adicionados a um vetor de resultados. Finalmente, esses recursos e suas propriedades são utilizados na montagem de um XML que será retornado ao usuário;

- *NeighborhoodBroker:* essa classe implementa o algoritmo apresentado na seção 4.6.4, o qual habilita a descoberta de recursos envolvendo mais de uma célula de execução no ISAMpe. Ao processar uma requisição de pesquisa, uma instância *DiscovererBase* a encaminha para uma instância *NeighborhoodBroker*, que a examina e determina se esta deve ser propagada para instâncias do tipo *DiscovererBase* vizinhas. Mais especificamente, essa classe verifica o campo TTL da requisição; caso este seja igual a "1", nenhuma ação é tomada e os resultados encontrados na célula local são preparados para serem retornados ao usuário; caso contrário, o campo é decrementado, um conjunto de células vizinhas é selecionado da *CIB*, e a requisição de pesquisa é propagada para as instâncias descobridoras residentes nessas células. Na implementação atual, a escolha dos vizinhos é um processo puramente randômico, mas trabalhos futuros irão avaliar a utilização de meta-dados associados a cada célula na *CIB*, objetivando realizar uma seleção mais inteligente dos vizinhos. Finalmente, os resultados encontrados pelos vizinhos são combinados ao resultado encontrado na célula local em um único XML, e repassados a uma instância da classe *DiscovererProxy* para que esta repasse o retorno da consulta ao usuário requisitante;

- *PreferenceManager:* classe utilizada para realizar a interface com o serviço *AVU* do *middleware* a fim de ler, utilizar e escrever informações relacionadas às preferências dos usuários num repositório de dados *pervasivo*.

5.4 VisualUserComponent: uma interface gráfica para o *PerDiS*

O *VisualUserComponent* é uma facilidade implementada para testar e demonstrar visualmente a utilização do *PerDiS*. A classe *VisualUserComponent* (e uma série de classes auxiliares) provê uma interface gráfica para o *UserComponent*, permitindo a montagem de consultas e a visualização dos resultados retornados graficamente.

A figura 5.4 apresenta a tela de montagem de consultas do *VisualUserComponent*. Nessa tela devem ser informados obrigatoriamente o escopo da pesquisa (campo TTL) e o tipo de recurso a ser descoberto, e opcionalmente o número máximo de resultados que devem ser retornados ao usuário. Além disso, o usuário tem a opção de informar diferentes critérios de pesquisa (botão *addCriteria*), e incluir ramificações em sua especificação (botão *addBranches*). Por sua vez, sob cada ramificação, novos critérios são adicionados, assim como outras ramificações aninhadas também o podem, conforme descrito na seção 4.5.2. Os botões *Request* e *Clear* são usados para submeter a pesquisa e limpar a tela de especificação, respectivamente.

A figura 5.5 apresenta a tela de resultados de pesquisas do *VisualUserComponent*. Essa tela permite visualizar os recursos encontrados por uma pesquisa. Os recursos são listados na tela, junto com suas propriedades descritivas, permitindo ao usuário inspecionar o retorno da consulta. Os descritores dos recursos encontrados são utilizados pelo serviço *RBeing* do *middleware*, o qual provê a forma de acesso a esses recursos (conforme descrito na seção 4.2.1).

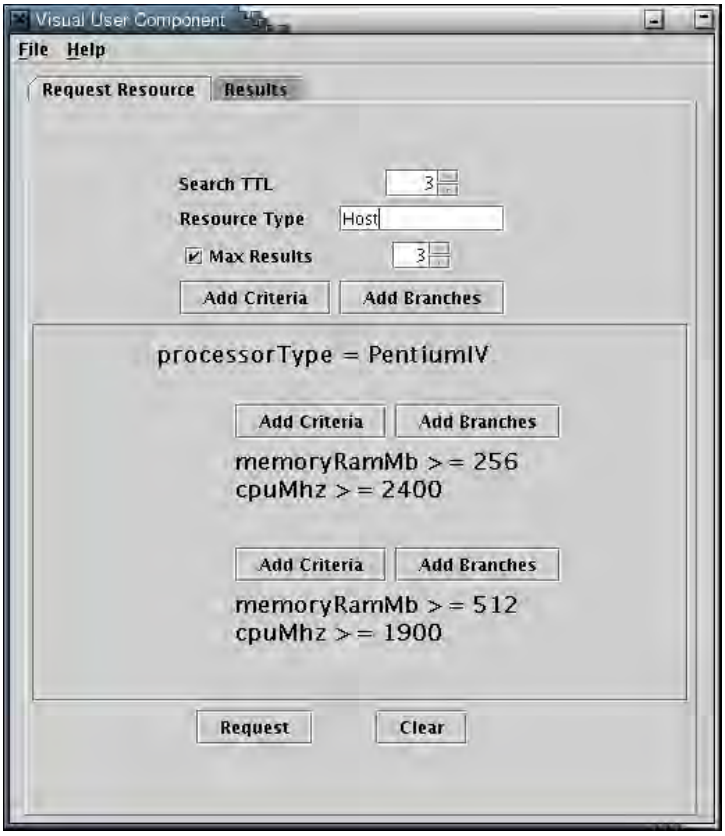

Figura 5.4: VisualUserComponent - Tela de pesquisa de recursos

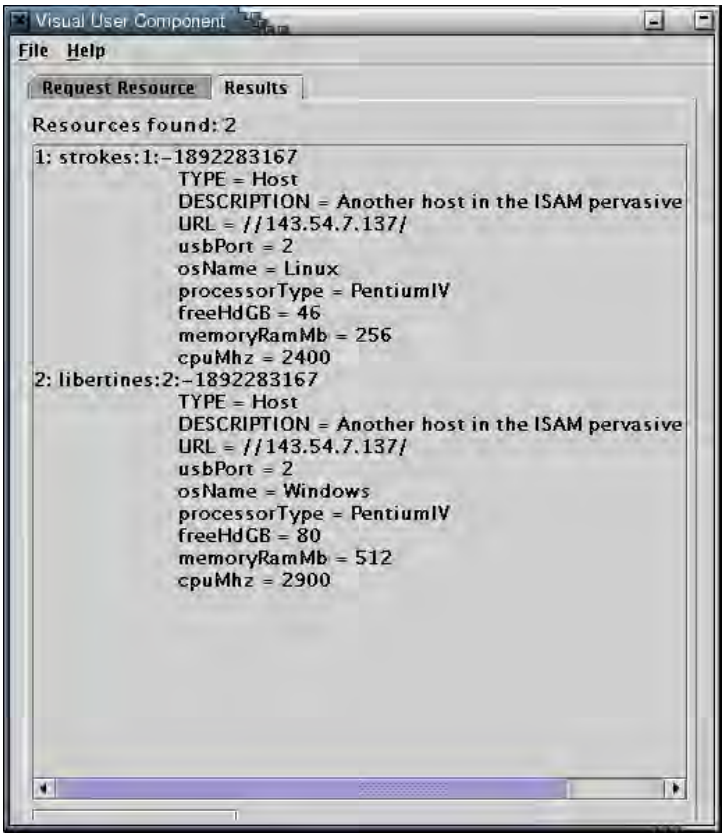

Figura 5.5: VisualUserComponent - Tela de visualização de resultados

5.5 Estudo de caso: aplicação GeneAl

Essa seção apresentará um estudo de caso em um cenário real, onde as contribuições do *PerDiS* podem ser exploradas, demonstrando as vantagens e o potencial da utilização do serviço de descoberta de recursos. Utilizando a aplicação *GeneAl* (SCHAEFFER FILHO et al., 2004, 2005) para alinhamento de seqüências genéticas, será demonstrado como esta pode beneficiar-se das funcionalidades providas pelo *PerDiS*.

O objetivo dessa seção é meramente ilustrativo, apresentando descritivamente como a aplicação *GeneAl* poderia usufruir de uma integração com o *PerDiS*. A implementação dos conceitos apresentados nessa seção, porém, será o foco de uma pesquisa futura, a ser desenvolvida na continuidade dos trabalhos.

5.5.1 A modelagem da aplicação GeneAl

O alinhamento de seqüências genéticas refere-se à operação de comparação de nucleotídeos que tenta encontrar similaridades locais usando bancos de dados de seqüências biológicas. Através de um esquema de pontuação sobre as seqüências comparadas é possível identificar quais são as mais semelhantes. A aplicação *GeneAl* (cujo nome vem de *Gene*tic *Al*ignment) realiza o alinhamento de seqüências genéticas sobre bases de dados dispersas geograficamente, encontrando os N melhores alinhamentos para uma dada seqüência de entrada.

Existem diversos algoritmos para realizar alinhamentos de seqüências genéticas em diferentes níveis de sofisticação, variando do algoritmo básico usado na implementação do programa BLAST (ALTSCHUL et al., 1990) até o algoritmo de Smith-Waterman (SMITH; WATERMAN, 1981) que consegue detectar similaridades fracas entre seqüências separadas por uma larga distância evolucionária (BUNDSCHUH, 2000). A aplicação *GeneAl* implementa uma variação do algoritmo de Smith-Waterman, conforme descrito por Meidanis e Setúbal (1994).

O problema a ser solucionado pela aplicação considera uma situação em que um pesquisador deseja investigar bancos de dados genéticos fisicamente distribuídos, procurando por seqüência genéticas que sejam similares a uma dada seqüência de interesse. Toma-se por hipótese que os bancos de dados não podem ser movidos, devido ao seu tamanho ou mesmo a restrições de privacidade.

5.5.1.1 Estratégia de distribuição

A fim de permitir a adição dinâmica de recursos computacionais ao ambiente de execução, o problema de alinhamento de seqüências genéticas foi modelado utilizando-se uma abordagem *master-worker* (YAMIN et al., 2002), associando um objeto mestre a cada banco de dados de seqüencias genéticas. Cada mestre é responsável por gerenciar um determinado número de trabalhadores, os quais irão realizar a procura pelos melhores alinhamentos em um banco de dados. Além disso, há uma camada de gerenciamento que coleta os resultados parciais junto a cada mestre para produzir o resultado final.

Um objeto mestre é responsável por dividir uma base de dados em um conjunto de tarefas (*jobs*), distribuí-las e manter controle delas, e por gerar o resultado final para uma dada célula de execução. Os trabalhadores obtêm tarefas de um mestre, e são responsáveis por processá-las. Ao término do processamento de uma tarefa, um trabalhador é responsável por enviar o resultado para seu mestre, e obter uma

nova tarefa a ser processada. O conteúdo genético usado nas bases de dados da aplicação *GeneAl* foi obtido de cromossomos humanos seqüenciados, disponibilizados na Internet (NATIONAL CENTER FOR BIOTECHNOLOGY INFORMATION, 2005).

5.5.1.2 Estratégias de particionamento de dados

Tarefas são compostas por um número de seqüências biológicas extraídas de uma base de dados. Elas são geradas por mestres e enviadas para trabalhadores. O tamanho de uma tarefa pode ser definido estaticamente antes do início da execução, ou ele pode ser ajustado dinamicamente durante a execução de acordo com a capacidade/carga dos *hosts* que realizam o processamento das tarefas.

Essa última estratégia é chamada de *time goal*, e as tarefas são redimensionadas dinamicamente de acordo com o cumprimento ou não de um dado objetivo em um período de tempo (reduzindo ou aumentando o número de seqüências por tarefa). O tamanho da primeira tarefa sempre é um valor estimado, mas o tamanho das tarefas seguintes é decidido em função do desempenho da tarefa imediatamente anterior, usando-se a relação entre o tempo esperado definido pelo usuário e o tempo real gasto na execução da tarefa anterior. No caso do tempo esperado ter sido ultrapassado, a tarefa seguinte será menor que a anterior; mas, no caso do tempo estimado ter sido maior que o necessário, a tarefa seguinte será maior que a anterior. Assim, a adaptação é realizada não somente em relação às características de *hardware*, mas também de acordo com flutuações na carga de trabalho dos *hosts* que compõem o ambiente de execução.

5.5.2 Utilizando o *PerDiS* como uma estratégia de descoberta de recursos para a aplicação *GeneAl*

Apresentados o cenário de uso e as principais características da aplicação *GeneAl*, nessa seção será descrita como a mesma pode beneficiar-se de uma estratégia de descoberta de recursos. Deve-se ressaltar que a aplicação, até o momento, não está integrada ao *PerDiS*, e esta tarefa ocorrerá como um dos trabalhos futuros relacionados a essa pesquisa. O valor do estudo sendo realizado nessa seção é meramente teórico, e permite a projeção de um cenário onde uma aplicação real pode beneficiar-se das funcionalidades providas pelo serviço *PerDiS*.

A seguir, serão apresentadas as principais considerações em relação à possibilidade de utilização do serviço *PerDiS* pela aplicação *GeneAl*, ilustrando as características e vantagens da solução desenvolvida para descoberta de recursos.

5.5.2.1 Descoberta de bases de dados genéticas e nós de processamento

No escopo da aplicação *GeneAl*, o serviço *PerDiS* seria usado na pesquisa de dois principais tipos de recursos: bases de dados de seqüências genéticas e nós de processamento onde seriam instanciados objetos trabalhadores que realizariam o processamento. O primeiro tipo de pesquisa seria responsável por descobrir as bases de dados disponíveis no ISAMpe segundo os critérios especificados pelo usuário, permitindo que esse pudesse escolher dentre as bases de dados que satisfizessem seus requisitos, qual ou quais seriam utilizadas efetivamente pela aplicação *GeneAl*. Dentre as informações retornadas no descritor de cada base de dados estaria a sua localização física, e esse dado seria utilizado na instanciação remota dos objetos do tipo mestre

(um mestre junto a cada base de dados). A partir daí, cada mestre seria responsável por interagir com o *PerDiS*, dessa vez para encontrar nós de processamento que potencialmente poderiam ser utilizados para abrigar os objetos trabalhadores.

5.5.2.2 Expressando necessidades de pesquisa

A linguagem de especificação de consultas provida pelo *PerDiS* poderia ser utilizada para restringir de diversas formas as pesquisas feitas para a localização de bases de dados ou possíveis nós de processamento, permitindo uma seleção que refletisse da melhor forma as intenções de pesquisa. Por exemplo, pode-se assumir que os nós de processamento a serem utilizados tenham um conjunto mínimo de requisitos no que se refere às combinações entre os atributos que definem a velocidade de CPU e a quantidade de memória RAM disponível no nó.

5.5.2.3 Utilização em larga-escala dos recursos disponíveis no ambiente de execução

A aplicação *GeneAl* utilizaria a descoberta de recursos em larga-escala basicamente para localizar as bases de dados de seqüências genéticas disponíveis no ISAMpe. Uma base de dados com as características desejadas por um pesquisador poderia estar a muitas células de distância da célula de execução local, e dessa forma o pesquisador interessado especificaria um número máximo de *hops* celulares para a pesquisa. A descoberta dos nós de processamento que atuam como trabalhadores sempre utilizaria uma pesquisa com escopo local à célula que abriga uma base de dados, pra manter o tráfego entre o mestre e os trabalhadores localizado internamente às fronteiras de uma célula de execução.

5.5.2.4 Possibilidade de escolha interativa ou não dos resultados de uma pesquisa

Novamente considerando a pesquisa por bases de dados, caso múltiplos resultados satisfizessem os requisitos de pesquisa informados, o *PerDiS* permitiria que o pesquisador navegasse entre os resultados obtidos para selecionar a base de dados a ser utilizada (ou mais de uma, se for o caso) pela aplicação *GeneAl*. Esse tipo de pesquisa é classificado como pesquisa interativa, pois exige que o pesquisador selecione o resultado desejado dentre os resultados encontrados. Porém, a utilização do mecanismo de preferências do *PerDiS* permitiria que o usuário armazenasse uma opção padrão dentre um possível conjunto de resultados encontrados. Nessa situação, o *PerDiS* automaticamente selecionaria a opção padrão, e eliminaria a necessidade da interação do usuário com o serviço após a pesquisa ter sido realizada, uma vez que seria trazida apenas a opção marcada como padrão.

5.5.2.5 Comportamento da descoberta em situações de desconexão

A adaptação do *PerDiS* quanto à conectividade permitiria que a aplicação *GeneAl* funcionasse, mesmo que de forma restrita, em uma situação de desconexão. Nesse caso, a pesquisa por recursos seria limitada ao escopo da máquina local, tanto no que se refere a bases de dados genéticas ou nós de processamento (nesse caso, apenas o próprio *host*). Caso houvesse recursos apropriados disponíveis localmente, a aplicação funcionaria utilizando-os, mesmo sabendo-se que esse comportamento seria bastante restrito em comparação ao potencial de processamento disponível no ISAMpe.

5.5.2.6 *Consistência do catálogo de recursos disponíveis*

Obviamente, a aplicação *GeneAl* apenas poderá usufruir efetivamente dos recursos descobertos através do *PerDiS* caso o último mantenha a devida consistência do catálogo de recursos disponíveis em cada célula, descartando do catálogo os recursos indisponíveis tão logo essa situação seja detectada. Nesse sentido, todos os recursos cadastrados no *PerDiS* seriam responsáveis pela renovação periódica de seus tempos de *lease*, utilizando a periodicidade informada no descritor de cada recurso. Apenas dessa forma o *PerDiS* poderia garantir que os recursos descobertos e informados à aplicação *GeneAl* realmente estejam disponíveis. Porém, a partir do momento em que se encerrasse o processo de descoberta e a aplicação passasse a utilizar os recursos encontrados, caberia a essa controlar a disponibilidade dos recursos sendo utilizados, uma vez que a atuação do *PerDiS* chega ao fim no momento em que o processo de descoberta é completado.

5.6 Resultados obtidos

Essa seção apresenta um conjunto de testes realizados para validar o funcionamento do *PerDiS*, executando-o como um dos serviços do *middleware* EXEHDA. As máquinas utilizadas nas execuções dos testes são listadas no quadro 5.1.

Máquina	Especificação
curly.inf.ufrgs.br	Pentium III 1 GHz 512 MB RAM
gppd.inf.ufrgs.br	Pentium III 860 MHz 640 MB RAM
gradep1.g3pd.ucpel.tche.br	AMD Athlon(tm) XP 2 GHz 256 MB RAM
rnp1.inf.ufsm.br	AMD Athlon(tm) XP 2 GHz 256 MB RAM

Quadro 5.1: Máquinas utilizadas na realização de testes com o *PerDiS*

A seguir, serão descritos cada um dos tipos de experimentos realizados, e os resultados obtidos.

5.6.1 Medições simples do tempo de execução das operações básicas de gerenciamento

Esse primeiro tipo de teste foi realizado a fim de que se pudesse obter algumas métricas relacionadas às operações básicas de gerenciamento de recursos. O quadro 5.2 apresenta os valores médios obtidos para as operações de inclusão, remoção e renovação do estado de um recurso ligado a um DP que por sua vez era responsável pela propagação dessas operações para uma instância celular do serviço de descoberta (DB). Cada uma das operações de gerenciamento é dividida em dois componentes: local e remoto. O componente local trata da preparação do recurso para a operação e a manipulação da *CIB* local, e ocorre junto à instância *proxy*. O componente remoto corresponde à propagação da informação para a instância *base*, e a manipulação da *CIB* remota. Atualmente, a implementação da *CIB* local funciona simplesmente como um *proxy* para a *CIB* remota, ou seja, não emprega

nenhuma estratégia de otimização ou *caching* de informações. Por esse motivo, na maioria dos casos o gerenciamento local de um recurso pode ser mais custoso que o componente remoto de gerenciamento, uma vez que no primeiro as informações são propagadas de qualquer forma, e sem nenhuma otimização no acesso à *CIB*. Porém, à medida que o projeto ISAM evoluir, o componente *CIB* sofrerá as modificações necessárias e o *PerDiS* poderá beneficiar-se dessas otimizações.

Nas execuções efetuadas, a instância *proxy* (DP) do serviço de descoberta executava no host *curly.inf.ufrgs.br*, enquanto a instância *base* (DB) executava no host *gppd.inf.ufrgs.br*. Nos testes, cada um dos recursos inseridos, removidos ou renovados possuía um descritor com 6 propriedades específicas, além das propriedades gerais comuns a qualquer tipo de recurso. As medições consideram o tempo médio de um conjunto de 25 execuções para cada tipo de teste, utilizando uma CIB contendo 25 recursos cadastrados.

	Operações de gerenciamento					
	Inclusão		Remoção		Renovação	
	Local	Remoto	Local	Remoto	Local	Remoto
Média (ms)	453,60	207,96	196,16	168,88	74,48	25,52
Desvio Padrão	155,66	84,78	44,99	13,64	20,04	5,60

Quadro 5.2: Métricas relacionadas às operações básicas de inclusão, remoção e renovação de um recurso

5.6.2 Impacto da expressividade da linguagem de especificação de consultas em relação a sua complexidade

Outro aspecto avaliado através de medições foi a variação do tempo médio de uma pesquisa em virtude do aumento de sua complexidade. Para isso, foram definidos quatro tipos de pesquisa, em ordem crescente de complexidade: *(A)* consultas sem adição de nenhum critério restritivo além do tipo de recurso a ser pesquisado, *(B)* consultas com dois critérios de pesquisa, *(C)* consultas com quatro critérios de pesquisa especificados, e *(D)* consultas com quatro critérios de pesquisa obrigatórios e dois *branches*, cada um com dois critérios de pesquisa adicionais. As pesquisas foram submetidas à instância *proxy* local (*curly.inf.ufrgs.br*) e então propagadas para a instância *base* do serviço de descoberta (*gppd.inf.ufrgs.br*). Nesses testes, foi utilizada uma *CIB* contendo 25 recursos cadastrados (10 do tipo *Host*, 10 do tipo *PDA* e 5 do tipo *Printer*), cada um com 6 propriedades específicas informadas no descritor. O retorno das pesquisas foi restrito a três recursos como valor máximo.

O quadro 5.3 apresenta o impacto do aumento da complexidade das consultas sobre o tempo de processamento. Considerando-se a variação da complexidade das pesquisas e o custo envolvido no processamento, constatou-se que o incremento no tempo de processamento encontra-se em uma faixa aceitável, não inviabilizando a realização de consultas envolvendo uma combinação mais rica de critérios de pesquisa. A diferença de tempo de pesquisa entre os tipos A, B e C (0, 2 e 4 critérios obrigatórios, respectivamente) foi de apenas 100 ms para cada incremento de complexidade. A diferença entre os tipos C e D foi um pouco maior (155 ms), uma vez que a pesquisa do tipo D envolvia o processamento de ramificações.

	Operações de pesquisa			
	Tipo A	Tipo B	Tipo C	Tipo D
Média (ms)	719,88	827,16	926,92	1081,32
Desvio Padrão	73,26	92,17	73,16	106,88

Quadro 5.3: Métricas relacionando o tempo de processamento de consultas em função de suas complexidades

5.6.3 Descoberta de recursos com escopo ampliado de pesquisa

Esse tipo de avaliação (quadro 5.4) refere-se ao incremento no tempo de pesquisa conforme o escopo da pesquisa é estendido. Para isso foram medidos os tempos de uma pesquisa envolvendo apenas os recursos na célula local (*ttl=1*), envolvendo os recursos alcançados até a célula diretamente conectada à célula local (*ttl=2*) e envolvendo os recursos alcançados até a célula a dois níveis de vizinhança da célula local (*ttl=3*). Além das instâncias de descoberta DP e DB (executando nos *hosts curly.inf.ufrgs.br* e *gppd.inf.ufrgs.br*, respectivamente) sob o domínio da instituição *UFRGS*, utilizou-se uma instância DB no *host gradep1.g3pd.ucpel.tche.br* sob o domínio da instituição *UCPEL* e outro DB no *host rnp1.inf.ufsm.br* sob o domínio da instituição *UFSM*. Na realização da pesquisa foi usada uma consulta com complexidade média (4 critérios), e cada célula foi populada com 8 recursos, sendo que todos os recursos encontrados eram retornados (8, 16 e 24 recursos para uma pesquisa de *ttl* 1, 2 e 3, respectivamente). Cada tipo de teste foi executado 25 vezes.

Os resultados obtidos com esse tipo de teste mostraram-se aceitáveis considerando a natureza geograficamente dispersa dos recursos envolvidos. A diferença entre os tempos de duas colunas corresponde ao tempo de propagação entre os *hops x - 1* e *x*, somado ao tempo de processamento do *hop x* (considerando que um *hop* seja equivalente a um DB).

	Pesquisa com escopo estendido		
	TTL = 1	TTL = 2	TTL = 3
Média (ms)	694,32	1.884,56	3.429,08
Desvio Padrão	61,91	253,07	1.313,41

Quadro 5.4: Métricas para a variação do tempo de consultas em função do escopo

5.6.4 Impacto da utilização de preferências no processo de descoberta

Nesses testes foram medidos os tempos relacionados à utilização de preferência do usuário na descoberta de recursos. Como mencionado anteriormente, existe implementada atualmente no *PerDiS* a possibilidade de utilização de dois tipos de preferências: ordenamento dos resultados encontrados e seleção de uma opção de recurso padrão dentre um conjunto de recursos encontrados (seção 4.7.1). A utilização de preferências de um usuário é uma etapa do processo de descoberta realizada apenas pela instância *base* do serviço, com o objetivo de realizar um processamento não essencial à descoberta apenas em um dispositivo com recursos computacionais suficientes para a execução de tal tarefa. Nos testes, consultas simples foram enviadas à instância *proxy* local (*curly.inf.ufrgs.br*) e então propagadas para a instância

base do serviço de descoberta (*gppd.inf.ufrgs.br*). A leitura e a interpretação do arquivo de preferências foram então realizadas junto à instância *base*, e o resultado pós-processado de cada pesquisa foi retornado ao usuário. Cada um dos tipos de teste foi executado 50 vezes.

Os testes de ordenamento (quadro 5.5) consideram o tempo necessário para interpretar o arquivo de preferências e ordenar um conjunto de 10, 20 e 30 *hosts* em ordem decrescente do valor do atributo *cpuMhz*. A versão atual do *PerDiS* utiliza uma implementação simples do conhecido algoritmo *BubbleSort* para realizar o ordenamento.

Os testes de seleção de uma opção padrão do conjunto de resultados (quadro 5.6) consideram o tempo necessário para interpretar o arquivo de preferências e fazer a combinação entre o conjunto de resultados encontrado e um possível conjunto de resultados no arquivo de preferências. Havendo a ocorrência no arquivo de preferências do usuário de todos os recursos encontrados na pesquisa, apenas o recurso indicado como padrão no arquivo de preferências é retornado ao usuário. Os testes utilizaram um conjunto de recursos no arquivo de preferência e um conjunto de resultados contendo 10, 20 e 30 recursos, e apresentam o tempo necessário para a identificação da opção padrão e montagem do novo retorno (contendo apenas um recurso).

	Ordenamento de resultados		
	10 recursos	20 recursos	30 recursos
Média (ms)	9,54	10,90	11,60
Desvio Padrão	5,08	6,62	5,95

Quadro 5.5: Métricas relacionadas à utilização de preferências para ordenamento do resultado

	Seleção de opção padrão		
	10 recursos	20 recursos	30 recursos
Média (ms)	20,68	60,02	103,90
Desvio Padrão	5,65	71,00	103,31

Quadro 5.6: Métricas relacionadas à utilização de preferências para seleção de opção padrão do resultado

O tempo necessário para o processo de ordenação pouco variou com o aumento da quantidade de recursos, sendo praticamente desprezível. Porém, o tempo necessário para a seleção da opção padrão no arquivo de preferências foi bastante afetado pelo aumento do tamanho do conjunto de resultados avaliado. No entanto, o tempo de pouco mais de 100 ms para a seleção da opção padrão em um conjunto de 30 recursos ainda pode ser considerado baixo frente ao valor agregado que a funcionalidade traz.

5.6.5 Medições relacionadas ao tempo de pesquisa variando a conectividade do usuário

Por fim, esses testes apresentam medições relacionadas ao tempo de pesquisa de recursos variando-se a conectividade do usuário. Quatro casos foram realizados, utilizando a interação direta com o *ContextManager* para simular os contextos

"conectado" e "desconectado": *(A)* pesquisa conectado, *(B)* pesquisa desconectado com *timeout* de 0 ms, *(C)* pesquisa desconectado com *timeout* de 500 ms, e *(D)* pesquisa conectado após uma reconexão (considerando que a pesquisa estava indefinidamente bloqueada até a chegada do evento de reconexão). Em todos os casos foram submetidas pesquisas de complexidade média (4 critérios) para a instância *proxy* e, dependendo do contexto, as pesquisas eram propagadas para a instância *base* (contexto "conectado") ou aguardava-se o *timeout* de reconexão, e caso isso não ocorresse dentro do intervalo definido, as pesquisas eram tratadas localmente pela instância *proxy*. Considerou-se uma população de 25 recursos a serem pesquisados (10 do tipo *Host*, 10 do tipo *PDA* e 5 do tipo *Printer*), e o retorno das pesquisas foi limitado a 3 recursos. O quadro 5.7 apresenta a média de 25 execuções para cada um dos tipos de teste.

	Pesquisas e desconexão			
	Tipo A	Tipo B	Tipo C	Tipo D
Média (ms)	1014,56	2468,64	3030,48	2252,76
Desvio Padrão	142,43	199,04	151,03	555,12

Quadro 5.7: Métricas relacionadas a pesquisas executando no contexto "conectado" e "desconectado"

Verificou-se um incremento do tempo de pesquisa entre os tipos *(A)* e *(D)*. Isso se deve ao protocolo de envio de anúncios executado sempre que uma instância do serviço de descoberta torna-se disponível no ambiente de execução (seção 4.2.2). Esse protocolo faz parte da etapa de reconexão, realizada quando uma instância do descobridor recebe um evento informando-o que este passou para o contexto conectado. O tempo predominante desse protocolo corresponde à espera realizada pelo *AdvertiseManager* do *PerDiS* após o envio de mensagens de anúncio em *multicast*. Esse tempo de espera permite que outras instâncias descobridoras respondam com um *ack* o anúncio enviado em *multicast*, antes que o protocolo do *PerDiS* prossiga em operação. O parâmetro que define quanto tempo deve-se aguardar pelo recebimento de *acks* é definido pelo perfil de configuração do nó. Nesses testes foi usado o tempo de 1000 ms, uma vez verificado que este era o tempo suficiente para o recebimento de respostas antes de seguir em operação. Porém, em situações onde haja um sensível atraso nas mensagens enviadas pela rede, esse parâmetro pode ser definido com um valor que corresponda a essas condições.

Outra questão a ser discutida refere-se ao tempo de acesso à *CIB* local no modo de operação desconectado (pesquisas dos tipos *(B)* e *(C)*): verificou-se que as pesquisas locais levaram muito mais tempo para executar do que as pesquisas remotas. Até o presente momento, o serviço *PerDiS* é adaptável ao contexto e permite a operação em modo desconectado. O serviço *CIB*, porém, tem sua implementação local atuando apenas como um *proxy* para a implementação remota, não sendo utilizadas técnicas de *caching* de informações atualmente. Nesse caso, mesmo com a adaptação à desconexão realizada pelo *PerDiS*, a *CIB* local ainda necessitava acessar a *CIB* remota (isso só foi possível porque nos testes foi utilizado um gerador de eventos de mudança de contexto que permitiu *simular* o contexto "desconectado"). Implementações futuras da *CIB* utilizarão uma *cache* local no serviço executando junto ao nó, o que se refletirá em um melhor desempenho na execução de pesquisas locais do *PerDiS*.

Uma medição adicional ainda foi feita a fim de computar o tempo de execução do protocolo de reconexão da instância *proxy* no processo de adaptação. O tempo médio de um conjunto de 25 execuções é apresentado no quadro 5.8. Observa-se que nesses valores o tempo predominante refere-se ao *timeout* para o recebimento de *acks* do protocolo de anúncio de recursos que, como mencionado anteriormente, foi configurado como 1000 ms.

	Tempo de reconexão
Média (ms)	1006,36
Desvio Padrão	4,29

Quadro 5.8: Métricas relacionadas ao tempo necessário para a execução do protocolo de reconexão da instância *proxy*

6 TRABALHOS RELACIONADOS

Este capítulo tem como objetivo apresentar uma breve descrição dos principais trabalhos desenvolvidos para descoberta de recursos, tanto na área acadêmica como também na área comercial. Diversos artigos sobre descoberta de recursos comparam os protocolos e as soluções existentes atualmente, identificando as principais características relacionadas a cada uma delas (BETTSTETTER; RENNER, 2000; HELAL, 2002; STOREY; BLAIR; FRIDAY, 2002). Após descrever os principais aspectos de cada um dos trabalhos pesquisados, é feita uma comparação entre as funcionalidades providas por essas soluções e aquelas identificadas como necessárias a um mecanismo de descoberta de recursos para uso em um ambiente *pervasivo*.

6.1 INS/Twine

O INS/Twine (BALAZINSKA; BALAKRISHNAN; KARGER, 2002) é um serviço para descoberta de recursos de forma escalável, onde entidades chamadas *resolvers* colaboram como *peers* para distribuir as informações relacionadas aos recursos e responder consultas. Na verdade, o Twine foi integrado ao INS (ADJIE-WINOTO et al., 1999), o Intentional Naming System desenvolvido pelo MIT, e agora forma o núcleo de sua arquitetura. Embora ele seja motivado pelo surgimento de infra-estruturas ubíquas, assim como o *PerDiS*, o INS/Twine não aborda a maioria dos requisitos para esses ambientes, como consciência do contexto, informações de personalização, e expressividade para especificação de consultas.

6.1.1 Visão geral da arquitetura

Para alcançar a escalabilidade proposta, o INS/Twine utiliza um particionamento *hash-based* das descrições de recursos entre um conjunto de *resolvers* simétricos. Inicialmente, o Twine transforma cada descrição de recurso em um conjunto de chaves numéricas. Cada subseqüência única de atributo/valor (chamada de *strand*) é extraída da descrição do recurso. Então, um valor *hash* é computado para cada *strand*, que constituem as chaves numéricas.

O INS/Twine utiliza uma rede de *resolvers* auto-organizáveis para rotear descrições de recursos entre eles para fins de armazenamento, e para colaborativamente responder consultas de usuários. Cada *resolver* conhece um conjunto de outros *resolvers* na rede.

Dispositivos e usuários comunicam-se com *resolvers* para anunciar recursos ou submeter consultas. Essa comunicação acontece na *"borda"* de uma rede INS/Twine, enquanto a comunicação entre *resolvers* acontece no *"núcleo"* da rede.

A arquitetura do INS/Twine apresenta três camadas:

- *Resolver*: é a camada superior da arquitetura, faz a interface com o repositório local de *AVTrees* (seção 6.1.3) e o *query engine*, retornando conjuntos de registros correspondentes a consultas (parciais). Ao receber um anúncio de um recurso, o *resolver* armazena-o localmente. Após, o *resolver* separa a descrição do recurso anunciado em *strands* e passa cada um deles para a camada inferior, a *StrandMapper*. O objetivo da divisão de descrições de recursos em *strands* é quebrá-las em pedaços significativos que permitam a *resolvers* especializarem-se em torno de subconjuntos de descrições;

- *StrandMapper*: esta camada mapeia um *strand* em uma ou mais chaves numéricas usando uma função de *hash*. Cada chave é então passada para a camada *KeyRouter*, juntamente com o anúncio completo do recurso ou a consulta;

- *KeyRouter*: é a camada inferior da arquitetura, e determina quais *resolvers* na rede devem armazenar a informação sobre o recurso, ou devem participar na resolução da consulta.

Como descrições completas de recursos são transmitidas durante o anúncio, qualquer *resolver* especializado em uma chave computada de uma consulta é capaz de responder a consulta sem requisitar junções ou transferências extras de dados. Dessa forma, quando um usuário submete uma consulta, o *resolver* que a recebe primeiro seleciona randomicamente um dos *strands* da consulta. Esse *strand* serve para determinar qual *resolver* deve tratar a consulta. Os resultados da consulta são então retornados para o *resolver* originador, o qual os envia para o usuário.

Como a maioria das estratégias para descoberta de recursos existentes, a manutenção da consistência na ocorrência de mudanças na rede é obtida através da revalidação periódica de informações, onde dados não renovados em um certo intervalo de tempo são automaticamente removidos da rede.

6.1.2 Escalabilidade da solução

A definição de descrições de recursos em *strands* é crítica para a escalabilidade do INS/Twine. Ela habilita *resolvers* a se especializarem em armazenar informações e responder consultas sobre um subconjunto de recursos. A escolha de um processo adequado de *hash* distribuído na camada *KeyRouter* é fundamental para alcançar alta taxa de sucesso nas consultas e minimizar o número de *resolvers* contatados em uma consulta.

O *KeyRouter* pode ser pensado como uma tabela *hash* distribuída, onde cada nó da rede mantém pares atributo/valor dentro de uma determinada faixa de chaves. Dada uma chave, o sistema encontra o nó da rede que deve armazenar o valor correspondente.

Para alcançar escalabilidade, esses sistemas requisitam que cada nó armazene em sua tabela de roteamento somente informações sobre um subconjunto de outros nós. Um nó para uma determinada chave é encontrado consultando de nó em nó em uma direção apropriada, até que o nó destino seja alcançado. A implementação do INS/Twine é construída sobre a infra-estrutura *peer-to-peer* provida pelo Chord (STOICA et al., 2001), o qual reconstrói de forma eficiente a rede de *peers* na presença de falhas. O Chord é usado para eficientemente identificar qual nó, ou conjunto de k nós consecutivos, deve armazenar uma determinada chave.

6.1.3 Definição de atributos e critérios de pesquisa

O esquema de representação de recursos e consultas no Twine é definido por um documento XML contendo uma hierarquia de pares atributo/valor. Essas descrições são convertidas em uma forma canônica, chamada de árvore atributo/valor (*attribute/value tree*, ou simplesmente *AVTree*). Um recurso satisfaz uma consulta se a *AVTree* formada pela consulta é a mesma que a descrição original do recurso, com zero ou mais pares atributo/valor truncados. Isto quer dizer que ele realiza combinações exatas de todos atributos e valores definidos ou de um subconjunto deles. Porém, este mecanismo é bastante limitado e de reduzida expressividade, uma vez que ele realiza apenas a comparação de igualdade entre atributos e valores.

6.2 Solar

O Solar (CHEN; KOTZ, 2003) é um *framework* para coleta, processamento e disseminação de informações de contexto para aplicações conscientes de contexto, desenvolvido em Dartmouth College. Ele permite que recursos anunciem nomes sensíveis ao contexto e que aplicações façam consultas sensíveis ao contexto. A linguagem de especificação do Solar permite a composição de operadores de processamento de contexto para calcular o contexto desejado.

6.2.1 Arquitetura do Solar

Um sistema Solar consiste de uma coleção de *peers*, chamados Planetas. Cada Planeta tipicamente reside em um *host* fixo com razoável poder computacional e uma conexão de rede permanente. Os Planetas são coletivamente responsáveis pela implantação e execução de um conjunto de operadores, disseminação de eventos, e implementação de um serviço de nomes. Recursos e aplicações obtêm serviços do Solar conectando-se a qualquer Planeta.

Um recurso pode contatar qualquer Planeta com uma requisição para anunciar um nome. Nessa situação, um Planeta cria um objeto interno como *proxy* para aquele recurso, e o *proxy* internamente registra o nome. De forma semelhante, uma aplicação pode contatar qualquer Planeta com uma consulta. O Planeta então cria um objeto *proxy* para servir a consulta. O uso de *proxies* permite que o gerenciamento de inscrições e nomes seja feito inteiramente dentro do Planeta, o qual é confiável e com boa conectividade, ao contrário dos *hosts* onde estão os recursos ou aplicações, os quais podem ser lentos e com conexão limitada.

O serviço de nomes do Solar foi construído no topo de um serviço de diretórios genérico. Essa abordagem deixa as questões relacionadas à escalabilidade, distribuição e flexibilidade do diretório de serviços a cargo de pesquisas fora do escopo do projeto Solar. Em particular, a implementação do Solar utiliza o Intentional Naming System (INS) como diretório de recursos. No INS, uma coleção distribuída de *resolvers* forma uma rede que realiza o roteamento de mensagens para seus destinos. Assim, o INS combina resolução de nomes e roteamento de mensagens em uma única abstração. Porém, para permitir consultas persistentes, o INS precisou ser modificado de forma a incluir um mecanismo pelo qual um cliente tivesse meios de registrar um *callback* que o notificasse sempre que um novo nome que satisfizesse a consulta fosse inserido, ou um nome com essa característica fosse removido do diretório de recursos.

Para receber mensagens, uma aplicação deve anunciar seu nome para algum dos *resolvers* do INS. Então, os *resolvers* disseminam o nome entre eles. Registros de nomes são descartados conforme eles envelhecem, e dessa forma aplicações e recursos precisam anunciar seus nomes periodicamente.

6.2.2 Organização distribuída de um sistema Solar

Um sistema Solar consiste de uma coleção de *peers*, os quais recebem o nome de Planetas, tipicamente em *hosts* separados mas bem conectados em uma rede distribuída. Os Planetas são escritos inteiramente na linguagem Java, e formam o ambiente de execução para *proxies* e demais componentes da arquitetura. Quando um cliente se conecta a um Planeta de sua escolha e submete sua requisição, o Planeta instala um objeto *proxy* para o cliente. O *proxy* retransmite mensagens e eventos entre a conexão *socket* do cliente e os demais componentes do sistema Solar.

Os Planetas de um sistema Solar conectam-se diretamente entre si, conforme a necessidade, formando uma rede para disseminação de eventos. Um Planeta utiliza uma fila e uma *thread* para controlar o fluxo de eventos em um *socket*, e uma *thread* para receber e distribuir os eventos no outro extremo do *socket*. Caso mais de um inscrito para um publicador esteja residindo no mesmo Planeta, apenas uma cópia do evento publicado será entregue no Planeta dos inscritos. Trabalhos futuros tratarão as questões relacionadas à otimização da disseminação de eventos, incluindo *batching* e compressão.

Como mencionado anteriormente, questões relacionadas à escalabilidade não são abordadas diretamente pelo Solar, uma vez que seu serviço de nomes foi construído no topo de um diretório de recursos genérico, e o Solar confia na implementação desse diretório para tratar esses e outros aspectos.

6.2.3 A linguagem de especificação sensível ao contexto

No Solar, a descrição de propriedades baseada no conceito atributo/valor foi estendida para suportar nomes e consultas sensíveis ao contexto. Recursos utilizam especificação de nomes para definir seus nomes; aplicações utilizam especificações de nomes para consultar nomes que combinem. Um recurso satisfaz uma consulta se todos os atributos da consulta estão presentes no nome, e se os valores dos atributos correspondentes são iguais. Embora tenha sido desenvolvido para um ambiente de computação *pervasiva*, o descobridor de recursos do Solar tem uma linguagem de especificação de critérios bastante limitada, suportando somente igualdade de atributos.

Para realizar a descoberta de recursos consciente de contexto, o Solar monitora constantemente o contexto de execução, de forma que recursos possam atualizar seus nomes e as aplicações possam atualizar as consultas realizadas. Uma especificação de nome sensível ao contexto pode ser usada tanto para se referir a um recurso como também a uma consulta. Esse tipo de especificação é caracterizada por ter ao menos um atributo sensível ao contexto, cuja informação é derivada de um *event stream*.

Informação de contexto é geralmente derivada pelo pós-processamento e agregação da saída de diversos sensores. Cada unidade de informação de contexto publicada por um sensor é um evento, estruturado como um conjunto de pares atributo/valor. Considerando um intervalo de tempo, a saída de um sensor é um *event stream*. Além disso, operadores agregam informações de contexto inscrevendo-se em um ou mais *event streams*, e publicando um novo *event stream*. Assim, todos os

sensores e operadores são publicadores de eventos. A agregação de um conjunto de sensores e operadores dá origem a um grafo de operadores, o qual pode ser utilizado diretamente pela aplicação para definir um nome ou fazer uma consulta persistente. Por sua vez, os Planetas monitoram o contexto produzido pelo grafo para atualizar o espaço de nomes e notificar aplicações interessadas em mudanças de nomes.

6.3 Allia

O Allia (RATSIMOR et al., 2002) é um mecanismo para descoberta de recursos em ambientes *ad-hoc* de *m-commerce*, e desenvolvido por pesquisadores da University of Maryland Baltimore County. A pesquisa com alvo em ambientes móveis para comércio eletrônico tem como principal motivação o fato de que nesses ambientes, geralmente com uma organização *ad-hoc*, não se poderá confiar na existência de um diretório centralizado de recursos devido à natureza dos dispositivos envolvidos. Além disso, como os dispositivos que hospedam os serviços são móveis, eles podem mover-se para fora das proximidades a qualquer momento. O Allia trata-se de um *framework* para descoberta de recursos baseado em agentes e governado por políticas, utilizando *caching* em uma rede *peer-to-peer*. Capacidades e limitações dos dispositivos, preferências de usuários e critérios específicos de aplicações são fatores aos quais o Allia pode se adaptar, sempre com base em políticas previamente especificadas.

6.3.1 Visão geral da arquitetura

Na arquitetura proposta pelo Allia, cada dispositivo móvel executa uma plataforma composta por quatro componentes principais, além do catálogo de recursos local:

- *Policy Manager:* controla o comportamento da plataforma. Políticas locais podem ser usadas para controlar diversos aspectos da plataforma, como freqüência dos anúncios, política de *caching*, etc. Durante a inicialização, políticas são registradas junto ao *Policy Manager* local, porém estas podem ser alteradas durante o funcionamento do sistema. Cada um dos demais componentes da arquitetura consulta o *Policy Manager* sobre questões específicas relativas ao seu comportamento. Tais políticas são usadas para restringir funcionalidades da plataforma e refletir as capacidades do dispositivo, especificar preferências relacionadas ao mecanismo de *caching* (taxa de atualização, estratégia de substituição, etc) ou à estratégia de envio de anúncios (freqüência, *time-to-live*, etc), e até mesmo para expressar preferências relacionadas a um usuário;

- *Cache Manager:* gerencia os anúncios enviados por recursos vizinhos. Tais anúncios podem descrever serviços executando em nós vizinhos ou anúncios que foram repassados por nós vizinhos. Os parâmetros para manutenção e gerenciamento da *cache* são obtidos junto ao *Policy Manager*, como tempo de expiração das entradas, estratégia de substituição e tamanho da *cache*;

- *Advertising Manager:* responsável por difundir as descrições dos recursos registrados no catálogo local. O componente *Policy Manager* é usado para controlar a taxa de anúncios enviados pela plataforma. Além disso, diversas políticas

podem ser usadas para ajustar a taxa de anúncios conforme as condições do ambiente de execução;

- *Forwarding Manager:* este componente recebe anúncios de recursos remotos e requisições de usuários, e decide baseando-se nas políticas locais se deve propagar ou ignorar tais mensagens. Para evitar a geração de um excesso de mensagens através de *broadcast*, o *Forwarding Manager* pode utilizar envio de mensagens em *multicast* para seletivamente propagar anúncios de recursos.

A seção a seguir apresentará o funcionamento do Allia considerando a arquitetura descrita.

6.3.2 Funcionamento do mecanismo de descoberta de recursos

Nessa abordagem, cada nó anuncia seus serviços para outros nós em sua proximidade, de acordo com a política local, através do envio de uma mensagem em *broadcast*. Ao receber um anúncio, o agente executando na plataforma remota decide baseado em sua política local se deve armazená-lo em sua *cache* ou rejeitá-lo.

A aliança de um nó A é formada pelo conjunto de nós que possuem informações relativas a seus serviços armazenados na *cache* do nó A. Assim, um nó explicitamente conhece os nós membros em sua aliança. Porém, cada nó não conhece as alianças das quais ele é membro. Sempre que um nó move-se de um determinado local para outro, ele constrói sua aliança ouvindo anúncios de recursos próximos. Ele também torna-se membro de outras alianças ao anunciar seus serviços locais.

Como já mencionado anteriormente, a política local define a forma que o nó utilizará para se anunciar para seus vizinhos. Além disso, políticas definem também o número de membros que um nó pode ter em sua aliança (limitando o número de anúncios que esse nó pode manter em *cache*).

Quando um agente necessita descobrir um certo serviço, ele inicialmente verifica se o serviço está disponível na plataforma local. Caso não o encontre, é então feita uma procura em sua *cache*, verificando se os membros de sua aliança possuem tal serviço. Se o serviço ainda não for encontrado, a plataforma de origem tenta realizar o envio em *broadcast* (ou *multicast*) da requisição para outras alianças em sua proximidade.

Através de um mecanismo genérico de políticas, o Allia permite não só a definição de parâmetros relacionados ao funcionamento do mecanismo de *caching* ou de anúncios, mas também a adaptação às condições de rede e capacidades de um dispositivo, e a utilização de preferências relacionadas a um determinado usuário. Não são explicitadas claramente as limitações da expressividade de preferências de usuários através da definição de políticas, mas podem ser citadas principalmente questões relacionadas ao gerenciamento dos recursos, como taxa do envio de anúncios e comportamento de propagação ou não de mensagens.

Quanto às questões relacionadas à descoberta de recursos dispersos geograficamente, o Allia apresenta algumas limitações no sentido em que não são utilizados quaisquer mecanismos de organização por escopos. Trata-se de uma abordagem puramente *ad-hoc*, acrescida de *caches* que mantém informações sobre recursos fisicamente próximos. A descoberta de um determinado recurso localizado fora das proximidades do requisitante, poderia empregar uma quantidade extremamente elevada de *hops*, o que a tornaria inviável.

6.3.3 Especificação de recursos e critérios de pesquisa

Diferentemente de outras estratégias, o *framework* Allia não aborda a questão relacionada a mecanismos de descrição e protocolos que possam ser usados para descrever recursos. Um mecanismo para combinação de descrições durante o processo de descoberta de recursos não é especificado ou limitado pelo *framework* Allia (RATSIMOR et al., 2004). Recursos podem seguir a especificação Jini e ser descritos e acessados através de *proxies* Jini. XML, DAML (DARPA Agent Markup Language, 2005) ou OWL (WEB Ontology Language, 2004) podem também ser usados para descrever recursos. Além disso, outras alternativas ainda podem ser empregadas.

6.4 Jini

O Jini (SUN MICROSYSTEMS, 2003a) consiste de uma arquitetura para definição, anúncio e descoberta de recursos baseada em Java, e desenvolvida pela Sun Microsystems. Recursos Jini são definidos por interfaces da linguagem Java, e podem representar dispositivos de *hardware*, programas de *software* ou uma combinação dos dois. Ao serem disponibilizados na rede, recursos enviam anúncios em *multicast* para um ou mais serviços de diretórios (*Jini Lookup Services*). Uma das grandes limitações encontradas no Jini refere-se a sua falta de expressividade na representação de propriedades de um recurso e critérios de pesquisa, baseando seu processo de filtragem apenas na presença ou não de um valor para cada um dos atributos definidos numa interface Java.

6.4.1 Arquitetura Jini

Os princípios da arquitetura Jini são similares aos encontrados no SLP (seção 6.6). Um componente chave do Jini é o *Jini Lookup Service (JLS)*, que mantém informações dinâmicas sobre os serviços disponíveis. Através de um conjunto de protocolos específicos, dispositivos ou aplicações registram-se em uma rede Jini. Para tornar-se membro da rede Jini, o recurso deve colocar uma entrada na *Lookup Table* do *Lookup Service*, o qual é uma base de dados com todos os recursos da rede.

Todos os recursos devem localizar um ou mais *Jini Lookup Services* antes que possam tornar-se parte da rede Jini. Essa localização pode ser conhecida à priori, ou pode ser descoberta através de um mecanismo de *multicast* (protocolo *discovery*, figura 6.1). Depois de descoberto, o recurso deve fazer um *upload* de seu objeto *proxy* serializado no JLS (protocolo *join*, figura 6.2), o qual será usado pelos usuários para contatar o recurso original e invocar seus métodos.

Quando um recurso registra o seu objeto *proxy*, é retornado um *lease*. O recurso deve renovar periodicamente esse *lease* para manter sua presença no *lookup service*. Através desse mecanismo, é possível ter uma visão atualizada dos recursos disponíveis, uma vez que recursos que por algum motivo não estejam mais aptos a serem utilizados não terão o seu *lease* renovado.

Um usuário que esteja procurando por um recurso na rede inicialmente deve encontrar o JLS através de uma consulta em *multicast* na rede, ou se comunicar diretamente com ele utilizando um identificador previamente conhecido. Em seguida, ele pode consultar no *lookup service* um ou mais recursos que combinem com um determinado *template*. Ao final, um *proxy* do recurso será carregado na máquina do usuário e poderá ser usado para comunicar-se com o recurso (protocolo *lookup*,

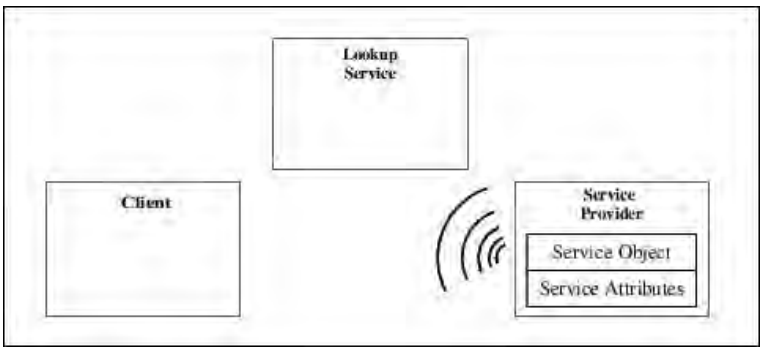

Figura 6.1: Protocolo *discovery* do Jini (SUN MICROSYSTEMS, 2003b)

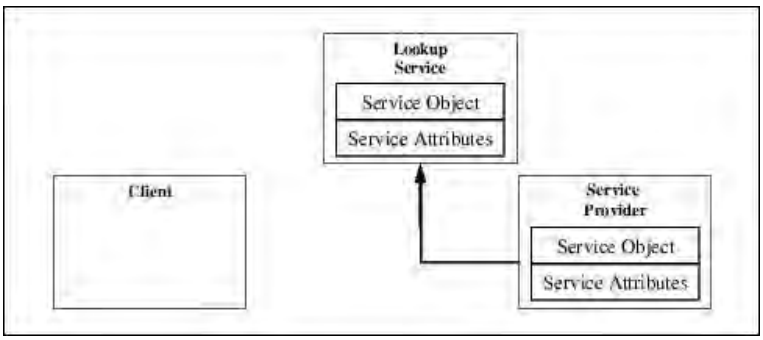

Figura 6.2: Protocolo *join* do Jini (SUN MICROSYSTEMS, 2003b)

figura 6.3).

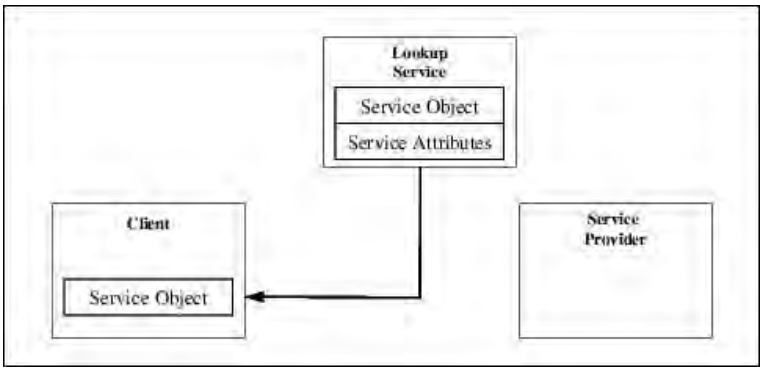

Figura 6.3: Protocolo *lookup* do Jini (SUN MICROSYSTEMS, 2003b)

Nas situações na qual nenhum *lookup service* pode ser encontrado, os usuários podem utilizar uma técnica chamada *peer lookup*. Em tal situação, o usuário pode enviar o mesmo pacote de identificação utilizado pelos *lookup services* para requisitar que recursos se registrem. Dessa forma, eles se registrarão junto ao usuário pensando que este é um *lookup service*. O usuário pode selecionar os recursos que ele necessita através das respostas recebidas, e descartar o restante (SUN MICROSYSTEMS, 2003b).

6.4.2 Grupos de lookup

Os *lookup services* podem ser configurados para serem parte de um ou mais grupos. De forma semelhante, recursos também podem ser configurados para que se registrem junto a *lookup services* que fazem parte de determinados grupos. Essa característica permite que o administrador agrupe conjuntos de recursos em agrupamentos lógicos gerenciados por *lookup services* específicos em uma rede. Além de grupos específicos que podem ser criados (uma motivação seria questões relacionadas à proximidade física onde, por exemplo, um grupo poderia agrupar os recursos disponíveis em uma sala de conferências), existe o grupo padrão (público) junto ao qual os recursos podem se registrar. Esse tipo de separação em grupos permite uma administração mais prática de um conjunto de serviços. Porém, o Jini não define claramente a forma de interação automática entre componentes existentes em grupos diferentes, e como é feita a interação entre os *lookup services*.

6.4.3 Atributos de pesquisa

Freqüentemente, apenas a utilização do tipo do recurso não será suficiente para permitir a expressão de critérios de pesquisa mais complexos. Os atributos de um recurso podem ser determinados pelo administrador local ou mantidos pelo próprio recurso. Esses atributos serão utilizados quando o usuário consultar o *service lookup* caso os mesmos tenham sido especificados no *template* do recurso. Dessa forma, as consultas são limitadas aos recursos que possuam um atributo em particular, possivelmente com um valor especificado.

O protocolo de combinação de atributos é bastante simples: é possível dizer que um atributo deve estar presente no serviço consultado, ou então que sua presença é irrelevante. Além disso, caso o atributo deva estar presente, é possível dizer que cada campo do atributo deve ter um determinado valor, ou que o valor do atributo e de seus campos também são irrelevantes. Essa é a mesma estratégia utilizada no modelo de tuplas JavaSpaces (SUN MICROSYSTEMS, 1998).

6.5 UPnP

O Universal Plug and Play (UPnP) (MICROSOFT CORPORATION, 2000) é uma solução para descoberta de recursos projetada para estender o modelo original Microsoft Plug and Play para periféricos, tornando-o mais dinâmico, flexível, fácil de usar e com suporte a redes de dispositivos de diferentes fabricantes. Atualmente, o UPnP é mantido pelo Universal Plug and Play Forum, o qual é formado por um conjunto de diversos fabricantes de produtos nas áreas de eletrônicos, computação, automação e segurança, redes e dispositivos móveis.

6.5.1 Visão geral da arquitetura

Em uma rede UPnP, dispositivos são modelados como objetos com uma tabela de estados associada. Essa tabela de estados fornece uma representação do estado interno do objeto. Os clientes podem fazer mudanças nessas tabelas de estados, fazendo com que operações sejam invocadas no objeto associado para que seu estado interno reflita as mudanças correspondentes. A arquitetura permite que os objetos gerem eventos sempre que seu estado for modificado, e que clientes se registrem para o recebimento desses eventos para que todas as visões de um dado dispositivo

estejam consistentes. Os três blocos básicos de construção de uma rede UPnP são: dispositivos, serviços e *control points*.

Um dispositivo UPnP é um elemento que pode fornecer um conjunto de serviços, ou aninhar um conjunto de dispositivos (um equipamento que funcione como uma televisão e um vídeo cassete, por exemplo). Um dispositivo sempre está associado a um descritor XML que provê uma série de informações, como serviços disponibilizados, dispositivos embutidos, lista de propriedades e qualquer outra informação que possa ser pertinente.

Um serviço é a menor unidade de controle em uma rede UPnP. Ele disponibiliza ações que permitem ao mundo externo controlá-lo, e mantém seu estado interno em variáveis de estado. De forma semelhante ao que ocorre com os dispositivos UPnP, serviços também são associados a descritores XML que armazenam todo o tipo de informação relacionada. Um serviço em um dispositivo UPnP consiste de três componentes: (1) uma tabela de estados, (2) um servidor de controle e (3) um servidor de eventos. A tabela de estados tem a função de manter o estado do serviço. O servidor de controle recebe e executa requisições de ações, atualizando o estado interno do serviço e retornando ao chamador o resultado do processamento. Por último, o servidor de eventos publica eventos a cada vez que o estado do serviço muda. Esses eventos podem ser capturados por outros serviços interessados em executar algum procedimento específico em determinada condição sinalizada pelo evento gerado.

Por último, um *control point* desempenha um papel de controlador em uma rede UPnP. Após descobrir um outro dispositivo, um *control point* pode obter informações descritivas desse dispositivo e dos serviços associados a este, invocar ações ou inscrever-se no gerador de eventos de um determinado serviço (que será responsável por notificar o *control point* nas situações em que o estado interno do serviço for alterado). Incorporando-se as funcionalidades dos *control points* nos dispositivos permite-se a interação *peer-to-peer* entre os componentes de uma rede UPnP.

6.5.2 Funcionamento do UPnP

O UPnP utiliza o Simple Service Discovery Protocol (SSDP) para descobrir dispositivos em redes IP. O SSDP permite que dispositivos anunciem seus serviços utilizando datagramas enviados em IP *multicast*. Esses anúncios contêm o tipo do serviço e a *Universal Resource Locator* (URL) do serviço sendo anunciado.

O SSDP é utilizado tanto por *control points* como também por dispositivos. No momento do *boot* de um *control point*, este pode enviar requisições SSDP para descobrir se determinados tipos de dispositivos e serviços (possivelmente utilizando critérios de consulta) estão disponíveis na rede. Dispositivos UPnP aguardam por tais requisições em portas *multicast*, e ao receberem uma mensagem desse tipo eles verificam se satisfazem os critérios de consulta especificados. Em caso afirmativo, os dispositivos respondem a requisição *multicast* enviando uma mensagem SSDP em *unicast* para o *control point* que enviou a requisição. Além disso, no momento em que um dispositivo é conectado à rede ele envia mensagens SSDP em *multicast* que anunciam a sua presença para que possíveis *control points* que estejam conectados na rede tenham conhecimento disso. Tanto essas mensagens de anúncio como também aquelas enviadas em resposta a uma requisição originada de um *control point* contêm informações que permitem a localização do descritor do dispositivo, que contém informações sobre suas propriedades e sobre os serviços suportados. O SSDP

provê mecanismos para que os dispositivos (e os serviços associados) possam enviar notificações de desconexão da rede UPnP, assim como também inclui a utilização de *timeouts* para evitar que dados incorretos ou antigos fiquem armazenados nos *control points*. Em contrapartida, a solução provida pela Microsoft apresenta baixa escalabilidade, sendo apropriada para uso em escopos restritos, como ambientes domésticos ou pequenos escritórios.

6.5.3 Definição de atributos

Dentro do UPnP, o XML é utilizado na definição dos descritores de dispositivos e serviços, e nas especificações de pesquisa de recursos. Essa é uma das principais vantagens proporcionadas pelo UPnP em relação ao Jini, provendo um mecanismo mais rico para a representação de recursos. Além disso, a utilização do XML permite uma possível interoperabilidade com outras estratégias de descrição de recursos, sendo possível a conversão entre documentos XML de forma automática através de transformações XSL. Mesmo assim, a combinação de critérios de pesquisa usada pelo UPnP suporta unicamente igualdade de atributos, não sendo empregados outros operadores na seleção de recursos.

6.6 SLP

O Service Location Protocol (SLP) (BETTSTETTER; RENNER, 2000) foi 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.

6.6.1 Visão geral da arquitetura

Os agentes definidos pelo SLP são os seguintes: *User Agents (UA)*, *Service Agents (SA)* e *Directory Agents (DA)*.

Um *User Agent* atua junto a uma aplicação para adquirir as informações necessárias por essa aplicação para conectar-se a um recurso de rede. Cada recurso é representado por um *Service Agent*. Um UA descobre um SA para um dado tipo de serviço através da transmissão de uma requisição de serviço (*Service Request*). Em contrapartida, o UA aguarda por alguma resposta de serviço (*Service Reply*).

Quando houver múltiplas redes, um UA geralmente desejará conectar-se a serviços disponibilizados em outras redes. Para evitar um excesso de mensagens em *multicast* e para prover um *handler* comum para as mensagens de *Service Request* enviadas pelos UAs, o SLP define um componente chamado *Directory Agent*, que responde a essas requisições (figura 6.4).

O SLP permite também a comunicação entre *User Agents* e *Service Agents* em redes próximas sem a necessidade de qualquer *Directory Agent* (figura 6.5). Para isso, os UAs devem ser configurados para que enviem mensagens de requisição de serviços com um TTL maior que um. Essa estratégia pode ser apropriada em muitas instalações onde não se espera que tal tráfego vá ultrapassar uma pequena porcentagem do tráfego total da rede. O número de DAs utilizados também é variável, e o protocolo foi definido de forma que os SAs possam selecionar um ou mais DAs em particular para que registrem seus serviços.

Figura 6.4: SLP com a utilização de um DA (BETTSTETTER; RENNER, 2000)

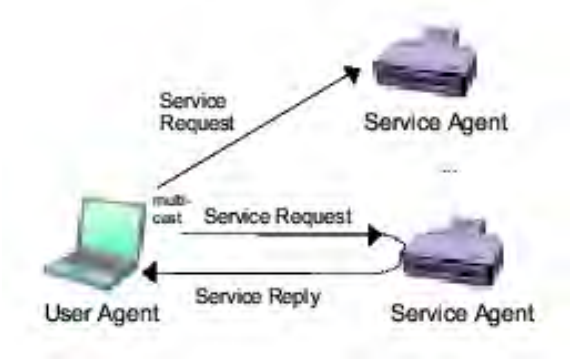

Figura 6.5: SLP sem a utilização de um DA (BETTSTETTER; RENNER, 2000)

6.6.2 Facilidade administrativa e definição de escopos

Um dos elementos que simplifica a administração do SLP é referente ao fato de que foram removidos quaisquer requisitos de organização hierárquica dos recursos em rede. Outro aspecto no que diz respeito à facilidade administrativa refere-se à natureza dinâmica da informação fornecida para as requisições dos UAs. O agente responsável por um determinado serviço somente irá responder a requisição se estiver em operação no momento da transmissão da mensagem de requisição. Apesar do fato de que DAs podem manter entradas antigas em *cache*, o tempo de vida de um registro de serviço é tipicamente pequeno o suficiente para que esse problema não dure muito tempo. Além disso, como um UA pode receber respostas de múltiplos SAs para uma mesma requisição, ele pode tentar acessar outro serviço (talvez mesmo de forma automática) caso a primeira tentativa resulte em fracasso.

Os serviços disponibilizados aos usuários podem ser agrupados em sub-conjuntos chamados de escopos, os quais são uma maneira de agrupar uma coleção de SAs sob os critérios de definição do administrador da rede. Aos *User Agents* é permitido a utilização dos serviços de um determinado escopo. Essa associação de um escopo a um UA pode ser feita através do uso de DHCP, ou através de configuração estática dos UAs. Porém, o SLP pode ser utilizado sem a definição de qualquer escopo, situação na qual os agentes utilizarão um escopo padrão.

Apesar dessa divisão de recursos em domínios separados de administração proporcionar uma maior escalabilidade à arquitetura, o modo de interação entre com-

ponentes existentes em escopos distintos não é bem definido pelo protocolo. Essa questão compromete a aplicabilidade do SLP na descoberta de recursos em larga-escala.

6.6.3 Especificação de atributos

Para que um *User Agent* crie uma requisição que enumere atributos de serviço apropriados para selecionar um determinado serviço, deve haver uma maneira comum de especificação desses atributos. A combinação entre os valores dos atributos de um serviço (providos pelo SA) e os valores da requisição do usuário só é possível se ambos, UA e SA, concordam no que diz respeito ao significado desses atributos.

A enumeração e descrição desses atributos é chamada de *service template*, e inclui as seguintes informações: o tipo do serviço, o qual caracteriza o tipo geral do serviço (por exemplo, "impressora"); o número da versão para o *template*; a linguagem na qual os nomes e valores dos atributos são fornecidos; uma descrição do tipo do serviço; e uma lista de atributos.

Para cada atributo, o *template* especifica seu tipo de valor, que pode ser *integer*, *string*, *boolean*, ou *opaque* (binário). Atributos do tipo *string* são formados por caracteres UTF-8, e assim podem incluir qualquer caractere ASCII. Existem caracteres reservados utilizados para formular a sintaxe booleana das requisições. Além disso, cada atributo pode ser especificado como *optional*, *multi-valued*, *literal*, ou *expected*. Atributos do tipo *optional* podem ter valores padrão. Atributos *multi-valued* podem receber mais de um valor quando definidos por um SA. Atributos do tipo *literal* não podem ser traduzidos para outras linguagens. Por último, um atributo marcado como *expected* deve fazer parte de qualquer resposta para uma dada requisição, ou o UA pode receber uma resposta contendo informações que ele não possa usar.

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

6.7 Análise comparativa de funcionalidades

As seções anteriores descreveram brevemente algumas das principais estratégias para descoberta de recursos existentes atualmente, abordando tanto projetos acadêmicos como também soluções desenvolvidas por empresas ou consórcios. Apresenta-se no quadro 6.1 um comparativo entre as funcionalidades requisitadas para a descoberta de recursos na arquitetura ISAM e aquelas providas pelas estratégias de descoberta de recursos apresentadas anteriormente. Os critérios considerados nessa comparação são os seguintes:

1. *Utilização de informações do contexto de execução*;
2. *Expressividade na descrição de recursos e critérios de pesquisa*;
3. *Possibilidade de interoperabilidade com outras estratégias de descoberta*;
4. *Suporte à descoberta de recursos em larga-escala (geograficamente dispersos)*;
5. *Utilização de estratégias para manutenção automática da consistência*;

Funcionalidades desejáveis	Estratégias de descoberta						
	Twine	Solar	Allia	Jini	UPnP	SLP	PerDiS
1	–	√	√	–	–	–	√
2	–	–	–	–	–	√	√
3	√	–	√	–	√	–	√
4	√	–	–	–	–	–	√
5	√	√	√	√	√	√	√
6	–	–	√	–	–	–	√

Quadro 6.1: Comparação de funcionalidades entre estratégias de descoberta

6. *Utilização de preferências por usuário.*

Dentre as estratégias de descoberta de recursos pesquisas, as únicas que abordam a utilização de informações do contexto de execução são o Solar e o Allia. O Twine, apesar de ser uma solução desenvolvida para utilização em ambientes *pervasivos*, não trata essa questão, e os protocolos utilizados como solução de mercado (Jini, UPnP e SLP) também não o fazem.

Quanto à expressividade na descrição de recursos e especificações de pesquisas, apenas o SLP provê uma solução satisfatória, utilizando uma notação baseada em expressões regulares para especificar requisitos de pesquisa. Essa notação é muito semelhante àquela empregada pelo *PerDiS*, no sentido que permite a utilização de diversos operadores sobre os atributos de pesquisa, e o agrupamento de condições. Todas as demais estratégias utilizam a comparação simples de igualdade entre atributos e valores contidos nas especificações, o que limita muito a expressividade.

Porém, uma notação adequada para utilização em um cenário caracterizado não somente pela riqueza computacional, mas também pela heterogeneidade de recursos e soluções, seria aquela que além de ser expressiva permitisse alguma possibilidade futura de interoperabilidade com outras soluções ou serviços de descoberta que eventualmente possam existir. Essa possibilidade é proporcionada pela utilização de XML na especificação das propriedades de um recurso, e na facilidade de conversão existente entre documentos XML de diferentes formatos através da utilização de documentos de estilo XSLT. Twine, Allia e UPnP são as soluções que utilizam o XML para a especificação de propriedades de recursos. Como descrito anteriormente, o SLP é a única estratégia com níveis satisfatórios de expressividade, e assim foi identificado que dentre as soluções pesquisadas, nenhuma combina as características de uma linguagem com alta expressividade e com possibilidade de interoperabilidade.

Quanto ao suporte à descoberta de recursos em larga-escala, apenas o Twine provê uma solução abordando diretamente esse aspecto. O Solar não trata essa questão, no sentido em que confia na utilização de um serviço de diretórios escalável, que dê o suporte necessário. Allia e UPnP são soluções desenvolvidas para utilização em cenários mais restritos e não se preocupam com essa questão. Por fim, apesar de o Jini e o SLP permitirem a utilização de diretórios de recursos que cubram os recursos de uma determinada área, eles não especificam adequadamente como esses diretórios distribuídos poderiam ser combinados para realizar a descoberta de recursos dispersos geograficamente de forma eficiente.

A manutenção automática da consistência do diretório de recursos foi o único dos requisitos levantados que é tratado por todas as estratégias de descoberta de

recursos pesquisadas. O mecanismo usualmente empregado é baseado na revalidação periódica de informações relacionadas aos recursos disponibilizados no meio, onde os dados não renovados em um certo intervalo de tempo são automaticamente removidos dos catálogos disponíveis na rede.

Por último, o Allia foi a única estratégia pesquisada que abordou a utilização de preferências de usuários no processo de descoberta. Esta utiliza um mecanismo de definição de políticas para especificar as preferências de um usuário. Porém, diferentemente do que ocorre com o *PerDiS*, essas políticas estão mais associadas a preferências de gerenciamento do que a preferências de pesquisa. As demais estratégias pesquisadas não tratam esse aspecto.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo tem por objetivo apresentar as conclusões sobre esse trabalho, assim como suas principais contribuições de pesquisa. Também são enumeradas as publicações do autor relacionadas à pesquisa desenvolvida durante o curso de mestrado. Por fim, é indicada uma série de atividades que são propostas como trabalhos futuros.

7.1 Conclusões

Mecanismos para descoberta de recursos permitem a localização automática de dispositivos ou serviços disponibilizados em rede, e a pesquisa nessa área vem ganhando maior importância devido ao aumento da complexidade dos ambientes computacionais em rede, seja esta uma infra-estrutura com ou sem-fio. Associado à popularização de dispositivos pessoais móveis, um novo paradigma conhecido como computação *pervasiva* vem surgindo, caracterizado pela dispersão de recursos computacionais no meio e pela profunda integração entre essas tecnologias.

Porém, um ambiente com essas características, associado à inerente dinamicidade e heterogeneidade do meio, impõe uma série de novos requisitos e desafios a uma estratégia de descoberta de recursos. O *PerDiS* tem como objetivo prover um serviço para descoberta de recursos considerando tal cenário, bem como o escopo de pesquisa do projeto ISAM. O projeto ISAM, apresentado no capítulo 2, provê uma plataforma integrada, da linguagem ao ambiente de execução, para construir e executar aplicações *pervasivas*, que em sua visão são distribuídas, móveis e conscientes do contexto por natureza.

Uma etapa fundamental nessa pesquisa foi o estudo das necessidades de uma estratégia para descoberta de recursos em um ambiente *pervasivo*, e o levantamento dos requisitos (apresentados no capítulo 3), que guiaram a modelagem e a implementação do *PerDiS* (descritas nos capítulos 4 e 5, respectivamente). Desde o início ficou identificado que seria oportuno para a qualificação da pesquisa desenvolvida que o *PerDiS* tivesse total integração com outros serviços do *middleware* do ISAM, aproveitando-se das funcionalidades providas por tais serviços para aumentar as potencialidades do mecanismo de descoberta. Essa estratégia permitiu ao *PerDiS* ir além de outras propostas independentes para descoberta de recursos, as quais não possuem o suporte de um ambiente de execução próprio e por esse motivo não abordam o conjunto de requisitos identificados em sua totalidade. Uma comparação do *PerDiS* com outras propostas para descoberta de recursos foi apresentada no capítulo 6, o que permitiu verificar claramente o conjunto de requisitos considerados por cada uma delas.

O protótipo implementado, assim como a maior parte dos demais serviços que compõem o *middleware*, foi construído na linguagem de programação Java, em especial devido a sua portabilidade, o que permite a execução do *middleware* sobre plataformas heterogêneas. Além do modelo proposto, identificou-se a necessidade de modelar um componente que permitisse uma interação visual com o *PerDiS*, facilitando sua depuração e a execução de testes. Esse foi concebido como um *front-end* para o *UserComponent*, e foi chamado de *VisualUserComponent*.

Para exemplificar o uso do *PerDiS* em um cenário real, foi criado um estudo de caso sobre uma aplicação para alinhamento de seqüências genéticas denominada *GeneAl*. Nesta aplicação, as funcionalidades do serviço de descoberta de recursos puderam ser demonstradas, explicitando as vantagens proporcionadas pelo *PerDiS* em relação a outras estratégias de descoberta. Enquanto o estudo de caso apresentou uma exemplificação teórica do uso das funcionalidades proporcionadas pelo *PerDiS*, um conjunto de testes utilizando medições específicas sobre as diferentes operações realizadas pelo serviço permitiu a obtenção de algumas métricas para avaliação do seu desempenho.

O desenvolvimento dessa pesquisa partiu de um amplo estudo das necessidades que deveriam ser supridas por um mecanismo de descoberta de recursos na computação *pervasiva*. A identificação de tais requisitos forneceu toda a base necessária para a concepção do *PerDiS*. Esses requisitos foram definidos a partir do estudo das funcionalidades providas pelas principais estratégias existentes para descoberta de recursos e a partir do estudo das características presentes em um ambiente de computação *pervasiva*. Esse levantamento bibliográfico dispendeu um grande esforço inicial de pesquisa, mas mostrou-se muito valioso na continuidade da concepção e desenvolvimento do *PerDiS*.

Considera-se que os objetivos definidos na etapa inicial do trabalho foram alcançados. Porém, sabe-se que o *PerDiS* pode ser estendido e melhorado de diversas formas. Algumas evoluções que podem ser realizadas como trabalhos futuros a partir da implementação atual são discutidas na seção 7.2. Em tese, esse trabalho demonstrou o fruto de uma pesquisa que levantou e supriu uma série de necessidades imediatas, mas que de forma alguma podem ser classificadas como únicas e absolutas, e a identificação de novos requisitos é uma etapa natural que pode acontecer no ciclo de vida de qualquer serviço do *middleware* do ISAM.

7.1.1 Contribuições de pesquisa ao projeto ISAM

Nessa seção são revisitadas as principais contribuições dessa pesquisa ao projeto ISAM, a saber:

- Levantamento bibliográfico das principais estratégias já existentes para descoberta de recursos;

- Levantamento dos requisitos de uma estratégia para descoberta de recursos apropriada para uso na computação *pervasiva*;

- Elaboração de um modelo para descoberta de recursos na computação *pervasiva*, além da implementação de um protótipo para validação do modelo proposto e da realização de testes de funcionalidade da solução desenvolvida. Também foi realizada a implementação de um *front-end* gráfico para a sub-

missão de consultas e visualização de resultados, a fim de facilitar os testes e a depuração do *PerDiS*;

- Integração do protótipo à plataforma ISAM, como um dos serviços providos por seu *middleware*, e validação do funcionamento de outros serviços já existentes no *middleware*, através da interação desses com o *PerDiS* durante a realização de testes.

Como uma contribuição principal, englobando todas as anteriores, pode-se colocar a agregação de um novo serviço ao *middleware* do ISAM para a realização da descoberta de recursos no ambiente de execução. A concepção desse serviço levou em consideração os novos desafios impostos pelos ambientes *pervasivos* a uma estratégia de descoberta de recursos. Sua importância traduz-se na inexistência, até então, de um mecanismo para desempenhar tal tarefa de uma forma adequada em um ambiente *pervasivo*, o que motivou originalmente o desenvolvimento dessa pesquisa.

7.1.2 Publicações

A seguir são listadas as publicações em que o autor participou ao longo do curso de mestrado, envolvendo especificamente o desenvolvimento do *PerDiS* ou sobre a pesquisa realizada no escopo do projeto ISAM.

7.1.2.1 Artigos publicados sobre o PerDiS ou no âmbito do projeto ISAM (como primeiro autor)

- **PerDiS: A Scalable Resource Discovery Service for the ISAM Pervasive Environment.** In: *International Workshop on Hot Topics in Peer-to-Peer Systems (Hot-P2P)*, Volendam, 2004. IEEE Computer Society. p. 80-85.

- **A Practical Grid Experiment Using the ISAM Architecture for Genetic Sequence Alignment.** In: *2nd International Workshop on Middleware for Grid Computing (MGC)*, Toronto, 2004. Middleware 2004 Companion - Workshop Proceedings. ACM. p. 99.

- **PerDiS: Um Modelo para Descoberta de Recursos na Arquitetura ISAM.** In: *VI Workshop de Comunicação Sem-Fio e Computação Móvel (WCSF)*, Fortaleza, 2004. p. 98-107.

- **Um Serviço Adaptativo para Descoberta de Recursos em Larga-Escala na Arquitetura ISAM.** In: *I Workshop de Peer-to-Peer (WP2P)*, Fortaleza, 2005. p. 85-96.

- **Applying the ISAM Architecture for Genetic Alignment in a Grid Environment.** In: *III Workshop de Grade Computacional e Aplicações (WGCA)*, Petrópolis, 2005.

- **Descoberta de Recursos na Arquitetura ISAM.** In: *V Escola Regional de Alto Desempenho (ERAD)*, Canoas, 2005. p. 89-90.

- **Uma Proposta para Descoberta de Recursos no ISAM Pervasive Environment.** In: *II Workshop de Processamento Paralelo e Distribuído (WSPPD)*, Porto Alegre, 2004. p. 137-142.

7.1.2.2 *Artigos publicados sobre o PerDiS ou no âmbito do projeto ISAM (como co-autor)*

- **Disseminando Informações na Arquitetura ISAM.** In: *Seminário Integrado de Software e Hardware (SEMISH) do XXV Congresso da SBC*, São Leopoldo, 2005. p. 1802-1816.
- **A Scalable Dissemination Service for the ISAM Architecture.** In: *17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)* (aceito para publicação), Rio de Janeiro, 2005.
- **GRADEp: Towards Pervasive Grid Executions.** In: *III Workshop de Grade Computacional e Aplicações (WGCA)*, Petrópolis, 2005.
- **Pervasive Computing with ISAM: joining Context-Aware, Mobile and Grid Computing.** In: *II Workshop de Grade Computacional e Aplicações (WGCA)*, Petrópolis, 2004.
- **Implementação de um Mecanismo para Descoberta de Recursos no Projeto ISAM.** In: *XVI Salão de Iniciação Científica*, Porto Alegre, 2004. p. 114-114.

7.2 Trabalhos futuros

Nessa seção, são descritas atividades relacionadas ao desenvolvimento e evolução do *PerDiS*, as quais são sugeridas como trabalhos futuros. Essas atividades são sintetizadas a seguir:

- Utilização de heurísticas na seleção da vizinhança P2P;
- Avaliação de novas possibilidades a serem expressas como preferências de usuários;
- Integração do *PerDiS* à aplicação *GeneAl*, e avaliação da nova estratégia junto à aplicação;
- Realização de testes adicionais no escopo do projeto GRADEp.

Uma contribuição significativa ao *PerDiS* seria a sua extensão, incorporando técnicas e heurísticas para uma seleção eficiente e inteligente na propagação de requisições para os vizinhos (*super-peers*) na rede P2P (seção 4.6.4). Atualmente, o protocolo probabilístico utiliza uma seleção randômica dos vizinhos de uma célula. Uma seleção mais eficiente, considerando métricas sobre meta-dados das células vizinhas seria uma questão de pesquisa interessante a ser explorada junto ao *PerDiS*.

Outro aspecto que poderia ser abordado trata da identificação, avaliação e utilização de novas possibilidades a serem expressas como preferências de usuários (seção 4.7). A utilização de preferências considerando informações do contexto corrente do usuário é uma atividade de pesquisa que também pode ser considerada na evolução do *PerDiS*.

Na linha da avaliação prática do *PerDiS* em um estudo de caso, um trabalho a ser desenvolvido consistiria da integração do *PerDiS* à aplicação *GeneAl* (seção 5.5).

Atualmente, a descoberta de informações pela aplicação é feita de forma estática, a partir de dados providos em um arquivo de configuração. A integração permitiria uma descoberta dinâmica dos recursos computacionais e bases de dados utilizados pela aplicação.

Por fim, cabe ressaltar a necessidade de testes adicionais do protótipo desenvolvido, o que deve ocorrer no escopo do projeto GRADEp (GEYER et al., 2005). O GRADEp é um trabalho em andamento do Grupo de Trabalho GRADEp, patrocinado pela RNP (Rede Nacional de Pesquisas), o qual objetiva estender a proposta tradicional de computação em grade com a noção de execuções em uma grade *pervasiva*, através da incorporação de aspectos de mobilidade ao cenário da computação em grade.

REFERÊNCIAS

ABRAHAM, I.; DOLEV, D.; MALKHI, D. LLS: a locality aware location service for mobile ad hoc networks. In: JOINT WORKSHOP ON FOUNDATIONS OF MOBILE COMPUTING, 2004. **Proceedings...** New York: ACM Press, 2004. p.75–84.

ADJIE-WINOTO, W. et al. The design and implementation of an intentional naming system. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 1999. **Proceedings...** New York: ACM Press, 1999. p.186–201.

ALTSCHUL, S. F. et al. Basic local alignment search tool. **Journal of Molecular Biology**, [S.l.], v.215, p.403–410, 1990.

ANDERSON, T.; LEE, P. A. **Fault Tolerance - Principles and Practice**. [S.l.]: Prentice-Hall, 1981.

ARNOLD, K. The Jini architecture: dynamic services in a flexible network. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION CONFERENCE, 36., 1999. **Proceedings...** New York: ACM Press, 1999. p.157–162.

AUGUSTIN, I. **Abstrações para uma Linguagem de Programação Visando Aplicações Móveis Conscientes do Contexto em um Ambiente de Pervasive Computing**. 2003. 193p. Tese (Doutorado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

AUGUSTIN, I.; YAMIN, A.; BARBOSA, J.; GEYER, C. ISAM - a Software Architecture for Adaptive and Distributed Mobile Applications. In: IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS, 7., 2002, Taormina, Italy. **Proceedings...** [S.l.]: IEEE Computer Society, 2002.

AUGUSTIN, I.; YAMIN, A.; GEYER, C. Distributed Mobile Applications With Dynamic Adaptive Behavior. In: INTERNATIONAL CONFERENCE ON COMPUTERS AND THEIR APPLICATIONS, CATA, 2002, San Francisco, Califórnia. **Proceedings...** [S. l.]: ISCA, 2002. p.4–6.

BAKRE, A.; BADRINATH, B. R. Handoff and System Support for Indirect TCP/IP. In: USENIX SYPOSIUM ON MOBILE AND LOCATION-INDEPENDENT COMPUTING, 2., 1995, Ann Arbor, MI. **Proceedings...** [S.l.: s.n.], 1995.

BALAZINSKA, M.; BALAKRISHNAN, H.; KARGER, D. INS/Twine: a scalable peer-to-peer architecture for intentional resource discovery. In: INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING, PERVASIVE, 2002, Zurich, Switzerland. **Proceedings...** [S.l.: s.n.], 2002.

BARBOSA, J. L. V. **Holoparadigma**: um modelo multiparadigma orientado ao desenvolvimento de software distribuído. 2002. 213p. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

BELLAVISTA, P.; CORRADI, A.; STEFANELLI, C. The Ubiquitous Provisioning of Internet Services to Portable Devices. **IEEE Pervasive Computing**, [S.l.], v.1, n.3, p.81–87, July-Sept. 2002.

BETTSTETTER, C.; RENNER, C. A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. In: EUNICE OPEN EUROPEAN SUMMER SCHOOL, EUNICE, 6., 2000, Twente, Netherlands. **Proceedings...** [S.l.: s.n.], 2000.

B'FAR, R. **Mobile Computing Principles** : designing and developing mobile applications with UML and XML. [S.l.]: Cambridge University Press, 2004. 878 p.

BHAGWAT, P.; PERKINS, C.; TRIPATHI, S. Network layer mobility: an architecture and survey. **IEEE Personal Communications**, [S.l.], v.3, n.3, p.54–64, June 1996.

BIRRELL, A. D.; NELSON, B. J. Implementing remote procedure calls. **ACM Transactions on Computer Systems**, [S.l.], v.2, n.1, p.39–59, 1984.

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. Extensible Markup Language (XML). **The World Wide Web Journal**, [S.l.], v.2, n.4, p.29–66, 1997.

BRESLER, J.; AL-MUHTADI, J.; CAMPBELL, R. Gaia mobility: extending active space boundaries to everyday devices. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS WORKSHOPS, 24., 2004. **Proceedings...** [S.l.: s.n.], 2004. p.430–433.

BUNDSCHUH, R. An analytic approach to significance assessment in local sequence alignment with gaps. In: COMPUTATIONAL MOLECULAR BIOLOGY, 2000. **Proceedings...** New York: ACM Press, 2000. p.86–95.

CHAKRABORTY, D.; CHEN, H. Service discovery in the future for mobile commerce. **ACM Crossroads**, [S.l.], v.7, n.2, p.18–24, 2000.

CHALMERS, D.; DULAY, N.; SLOMAN, M. A Framework for Contextual Mediation in Mobile and Ubiquitous Computing. **Personal and Ubiquitous Computing**, [S.l.], v.8, n.1, p.1–18, 2004.

CHALMERS, D.; DULAY, N.; SLOMAN, M. Meta Data to Support Context Aware Mobile Applications. In: IEEE INTL. CONFERENCE ON MOBILE DATA MANAGEMENT, MDM, 2004. **Proceedings...** [S. l.]: IEEE CS, 2004. p.199–210.

CHALMERS, D.; DULAY, N.; SLOMAN, M. Towards Reasoning About Context in the Presence of Uncertainty. In: WORKSHOP ON ADVANCED CONTEXT MODELLING, REASONING AND MANAGEMENT AT UBICOMP, 2004. **Proceedings...** [S.l.: s.n.], 2004.

CHEN, G.; KOTZ, D. Context-Sensitive Resource Discovery. In: IEEE INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING AND COMMUNICATIONS, 1., 2003. **Proceedings...** [S.l.: s.n.], 2003. p.243–252.

CLUET, S.; KAPITSKAIA, O.; SRIVASTAVA, D. Using LDAP directory caches. In: ACM SIGMOD-SIGACT-SIGART SYMPOSIUM ON PRINCIPLES OF DATABASE SYSTEMS, 1999. **Proceedings...** New York: ACM Press, 1999. p.273–284.

CODDINGTON, P. D.; LU, L.; WEBB, D.; WENDELBORN, A. L. Extensible job managers for grid computing. In: AUSTRALASIAN COMPUTER SCIENCE CONFERENCE ON RESEARCH AND PRACTICE IN INFORMATION TECHNOLOGY, 26., 2003. **Proceedings...** [S.l.: s.n.], 2003. p.151–159.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Distributed systems** : concepts and design. 3rd ed. [S.l.]: Addison-Wesley, 2001.

DABROWSKI, C.; MILLS, K. Understanding self-healing in service-discovery systems. In: SELF-HEALING SYSTEMS, 1., 2002. **Proceedings...** New York: ACM Press, 2002. p.15–20.

DARPA Agent Markup Language. Disponível em: <http://www.daml.org>. Acesso em: jul. 2005.

FLINN, J.; SATYANARAYANAN, M. Energy-aware adaptation for mobile applications. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, SOSP, 17., 1999. **Proceedings...** New York: ACM Press, 1999. p.48–63.

FLINN, J.; SATYANARAYANAN, M. Managing battery lifetime with energy-aware adaptation. **ACM Transactions on Computer Systems**, [S.l.], v.22, n.2, p.137–179, 2004.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The Anatomy of the Grid: enabling scalable virtual organization. **The International Journal of High Performance Computing Applications**, [S.l.], v.15, n.3, p.200–222, Fall 2001.

FOX, G. Peer-to-peer networks. **Computing in Science & Engineering**, [S.l.], v.3, n.3, p.75–77, May-June 2001.

FRAINER, G. C. **RBEING**: uma biblioteca de acesso a recursos na plataforma ISAM. 2004. Trabalho de Conclusão (Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

FRIDAY, A.; DAVIES, N.; CATTERALL, E. Supporting Service Discovery, Querying and Interaction in Ubiquitous Computing Environments. In: ACM INTERNATIONAL WORKSHOP ON DATA ENGINEERING FOR WIRELESS AND MOBILE ACCESS, MOBIDE, 2., 2001, Santa Barbara, U.S. **Proceedings...** [S.l.: s.n.], 2001. p.7–13.

GEYER, C. F. R.; SILVA, L. C. da; YAMIN, A. C.; AUGUSTIN, I.; SCHAEFFER FILHO, A. E.; MORAES, M. C.; REAL, R. A.; FRAINER, G. C.; PIRES, R. P. GRADEp: towards pervasive grid executions. In: WORKSHOP DE GRADE COMPUTACIONAL E APLICAÇÕES, 3., 2005, Petrópolis, Brazil. **Proceedings...** [S.l.: s.n.], 2005.

GEYER, C. F. R.; YAMIN, A. C.; SILVA, L. C. da; VARGAS, P. K.; DUTRA, I.; PETEK, M.; ADAMATTI, D.; AUGUSTIN, I.; BARBOSA, J. L. V. Regional Center And Grid Development In Brazil. In: LAFEX INTERNATIONAL SCHOOL ON HIGH ENERGY PHYSICS, LISHEP, 2002, Rio de Janeiro, Brazil. **Proceedings...** [S.l.: s.n.], 2002. 24 p.

GNUTELLA WEBSITE. Disponível em: <http://www.gnutella.com>. Acesso em: out. 2004.

GRAY, J.; REUTER, A. **Transaction processing**: concepts and techniques. [S.l.]: Morgan Kaufmann, 1993.

GUTTMANN, E. et al. **Service Location Protocol, Version 2**: RFC 2608. [S. l.]: IETF, 1999.

HARBIRD, R.; HAILES, S.; MASCOLO, C. Adaptive Resource Discovery for Ubiquitous Computing. In: WORKSHOP ON MIDDLEWARE FOR PERVASIVE AND ADD-HOC COMPUTING, 2., 2004, Toronto, Canadá. **Proceedings...** New York: ACM, 2004. p.155–160.

HELAL, S. Standards for service discovery and delivery. **IEEE Pervasive Computing**, [S.l.], v.1, n.3, p.95–100, July-Sept. 2002.

HELAL, S. Programming pervasive spaces. **IEEE Pervasive Computing**, [S.l.], v.4, n.1, p.84–87, Jan.-Mar. 2005.

HENRY, K. Objective ViewPoint: distributed computation with java remote method invocation. **Crossroads**, [S.l.], v.6, n.5, p.2, 2000.

HOLLIDAY, J.; AGRAWAL, D.; ABBADI, A. E. Disconnection modes for mobile databases. **Wireless Networks**, [S.l.], v.8, n.4, p.391–402, 2002.

HSU, C.-H.; KREMER, U. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In: ACM SIGPLAN CONFERENCE ON PROGRAMMING LANGUAGE DESIGN AND IMPLEMENTATION, PLDI, 2003. **Proceedings...** New York: ACM Press, 2003. p.38–48.

ISAM. **Projeto ISAM**. Disponível em: <http://www.inf.ufrgs.br/ isam/>. Acesso em: out. 2004.

JALOTE, P. **Fault tolerance in distributed systems**. [S.l.]: Prentice Hall, 1994.

KAASINEN, E. User needs for location-aware mobile services. **Personal Ubiquitous Comput.**, [S.l.], v.7, n.1, p.70–79, 2003.

KISTLER, J. J.; SATYANARAYANAN, M. Disconnected operation in the Coda File System. **ACM Transactions on Computer Systems**, [S.l.], v.10, n.1, p.3–25, 1992.

LEBECK, A. R. et al. Power aware page allocation. In: INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, ASPLOS, 9., 2000. **Proceedings...** New York: ACM Press, 2000. p.105–116.

LI, X. et al. Performance directed energy management for main memory and disks. In: INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, ASPLOS, 11., 2004. **Proceedings...** New York: ACM Press, 2004. p.271–283.

LYNCH, N. A. **Distributed algorithms**. [S.l.]: Morgan Kaufmann, 1996.

MAHGOUB, I.; ILYAS, M. **Mobile Computing Handbook**. [S.l.]: CRC Press, 2004. 200 p.

MALLICK, M. **Mobile and Wireless Design Essentials**. [S.l.]: Wiley, 2003. 480 p.

MCGRATH, R. E. **Discovery and Its Discontents**: Discovery Protocols for Ubiquitous Computing. Urbana-Champaign: Department of Computer Science University of Illinois, 2000.

MCKNIGHT, L. e. a. Wireless grids: distributed resource sharing by mobile, nomadic, and fixed devices. **IEEE Internet Computing**, [S.l.], v.8, n.4, p.24–31, 2004.

MEIDANIS, J.; SETÚBAL, J. C. **Uma introdução à Biologia Computacional**. Recife, Brasil: UFPE/di, 1994. p.15–54. (IX Escola de Computação).

MENASCE, D. A. Scalable P2P search. **IEEE Internet Computing**, [S.l.], v.7, n.2, p.83–87, Mar.-Apr. 2003.

MICROSOFT CORPORATION. **UPnP Device Architecture**. [S.l.], 2000.

MISCHKE, J.; STILLER, B. A methodology for the design of distributed search in P2P middleware. **IEEE Network**, [S.l.], v.18, n.1, p.30–37, Jan.-Feb. 2004.

MOCKAPETRIS, P. V.; DUNLAP, K. J. Development of the domain name system. In: SIGCOMM, 1988. **Proceedings...** [S.l.: s.n.], 1988. p.123–133.

MORAES, M. C.; SILVA, L. C. da; SCHAEFFER FILHO, A. E.; YAMIN, A. C.; AUGUSTIN, I.; GEYER, C. F. R. Disseminando Informações na Arquitetura ISAM. In: SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE, SEMISH; CONGRESSO DA SBC, 25., 2005, São Leopoldo, Brasil. **Anais...** [S.l.: s.n.], 2005. p.1802–1816.

MULLENDER, S. J. **Distributed systems**. 2nd ed. [S.l.]: Addison-Wesley, 1993.

MUMMERT, L. B.; EBLING, M. R.; SATYANARAYANAN, M. Exploiting weak connectivity for mobile file access. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, SOSP, 1995. **Proceedings...** New York: ACM Press, 1995. p.143–155.

NAPSTER WEBSITE. Disponível em: <http://www.napster.com>. Acesso em: out. 2004.

NATIONAL CENTER FOR BIOTECHNOLOGY INFORMATION. **Human Reference Sequence**. Disponível em: <http://www.ncbi.nlm.nih.gov/genome/seq/>. Acesso em: mar. 2005.

NEEDHAM, R. M.; SCHROEDER, M. D. Using encryption for authentication in large networks of computers. **Commun. ACM**, [S.l.], v.21, n.12, p.993–999, 1978.

NOBLE, B. D. et al. Agile application-aware adaptation for mobility. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, SOSP, 16., 1997. **Proceedings...** New York: ACM Press, 1997. p.276–287.

OZSU, M. T.; VALDURIEZ, P. **Principles of Distributed Database Systems**. [S.l.]: Prentice Hall, 1998.

PEDONE, F.; DUARTE, N. L.; GOULART, M. Probabilistic Queries in Large-Scale Networks. In: EUROPEAN DEPENDABLE COMPUTING CONFERENCE ON DEPENDABLE COMPUTING, EDCC, 4., 2002, London, UK. **Proceedings...** [S.l.]: Springer-Verlag, 2002. p.209–226.

PERKINS, C. E.; HARJONO, H. Resource discovery protocol for mobile computing. **Mobile Network Applications**, [S.l.], v.1, n.4, p.447–455, 1996.

POPEK, G. J.; KLINE, C. S. Encryption and Secure Computer Networks. **ACM Comput. Surv.**, [S.l.], v.11, n.4, p.331–356, 1979.

RAATIKAINEN, K.; CHRISTENSEN, H. B.; NAKAJIMA, T. Application requirements for middleware for mobile and pervasive systems. **SIGMOBILE Mob. Comput. Commun. Rev.**, [S.l.], v.6, n.4, p.16–24, 2002.

RAKOTONIRAINY, A.; GROVES, G. **Resource Discovery for Pervasive Environments.** [S.l.]: Springer, 2002. p.866–883. (Lecture Notes in Computer Science, v.2519).

RANGANATHAN, A.; AL-MUHTADI, J.; CAMPBELL, R. Reasoning about Uncertain Contexts in Pervasive Computing Environments. **IEEE Pervasive Computing**, [S.l.], v.3, n.2, p.62–70, Apr. 2004.

RATSIMOR, O. et al. Allia: alliance-based service discovery for ad-hoc environments. In: INTERNATIONAL WORKSHOP ON MOBILE COMMERCE, WMC, 2., 2002. **Proceedings...** New York: ACM Press, 2002. p.1–9.

RATSIMOR, O. et al. Service discovery in agent-based pervasive computing environments. **Mobile Network Applications**, [S.l.], v.9, n.6, p.679–692, 2004.

REAL, R. A. **Uma proposta de escalonamento para Computação Pervasiva**. 2004. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

ROMAN, M. **An Application Framework for Active Space Applications**. 2003. Dissertação (Mestrado em Ciência da Computação) — University of Illinois at Urbana-Champaign, Illinois, USA.

ROMAN, M. et al. A middleware infrastructure for active spaces. **IEEE Pervasive Computing**, [S.l.], v.1, n.4, p.74–83, Oct.-Dec. 2002.

ROUSSOS, G.; MARSH, A. J.; MAGLAVERA, S. Enabling pervasive computing with smart phones. **IEEE Pervasive Computing**, [S.l.], v.4, n.2, p.20–27, Apr.-June 2005.

ROYER, E. M.; TOH, C.-K. A review of current routing protocols for ad hoc mobile wireless networks. **IEEE Personal Communications**, [S.l.], v.6, n.2, p.46–55, Apr. 1999.

SATYANARAYANAN, M. **A Survey of Distributed File Systems**. [S.l.]: Department of Computer Science, Carnegie Mellon University, 1989. (Technical Report CMU-CS-89-116).

SATYANARAYANAN, M. Fundamental challenges in mobile computing. In: ANNUAL ACM SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, PODC, 15., 1996. **Proceedings...** New York: ACM Press, 1996. p.1–7.

SATYANARAYANAN, M. Pervasive Computing: vision and challenges. **IEEE Personal Communications**, [S.l.], p.10–17, Aug. 2001.

SCHAEFFER FILHO, A. E. **EXEHDA-HM**: Uma Contribuição à Gerência da HoloTree. 2002. 65 p. Trabalho de Conclusão (Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

SCHAEFFER FILHO, A. E.; MORAIS, L. L. de; REAL, R. A.; SILVA, L. C. da; YAMIN, A. C.; AUGUSTIN, I.; GEYER, C. F. R. Applying the ISAM Architecture for Genetic Alignment in a Grid Environment. In: WORKSHOP DE GRADE COMPUTACIONAL E APLICAÇÕES, 3., 2005. **Proceedings...** [S.l.: s.n.], 2005.

SCHAEFFER FILHO, A. E.; MORAIS, L. L. de; REAL, R. A.; SILVA, L. C. da; YAMIN, A. C.; AUGUSTIN, I.; VARGAS, P. K.; GEYER, C. F. R. A Practical Grid Experiment Using the ISAM Architecture for Genetic Sequence Alignment. In: INTERNATIONAL WORKSHOP ON MIDDLEWARE FOR GRID COMPUTING, 2., 2004, Toronto, Canada. **Proceedings...** New York: ACM, 2004. p.99.

SILVA, L. C. da. **Primitivas para Suporte à Distribuição de Objetos Direcionados à Pervasive Computing**. 2003. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

SMITH, F.; WATERMAN, M. S. Comparison of Biosequences. **Advances in Applied Mathematics**, [S.l.], v.2, p.482–489, 1981.

STOICA, I. et al. Chord: a scalable peer-to-peer lookup service for internet applications. In: CONFERENCE ON APPLICATIONS, TECHNOLOGIES, ARCHITECTURES, AND PROTOCOLS FOR COMPUTER COMMUNICATIONS, SIGCOMM, 2001. **Proceedings...** New York: ACM Press, 2001. p.149–160.

STOREY, M.; BLAIR, G.; FRIDAY, A. MARE: resource discovery and configuration in ad hoc networks. **Mobile Networks and Applications**, [S.l.], v.7, n.5, p.377–387, 2002.

SUN MICROSYSTEMS. **JavaSpace Specification**. [S.l.], 1998.

SUN MICROSYSTEMS. **Jini Technology Core Platform Specification**: version 2.0. [S.l.], 2003.

SUN MICROSYSTEMS. **Jini Technology Architecture Specification**: version 2.0. [S.l.], 2003.

SUN MICROSYSTEMS. **Java - The Source for Developers**. Disponível em: <http://java.sun.com>. Acesso em: out. 2004.

TALIA, D.; TRUNFIO, P. Toward a synergy between P2P and grids. **IEEE Internet Computing**, [S.l.], v.7, n.4, p.94–96, July-Aug. 2003.

TANENBAUM, A. S. **Distributed systems - principles and paradigms**. [S.l.]: Prentice-Hall, 2002.

TERRY, D. B. et al. Managing update conflicts in Bayou, a weakly connected replicated storage system. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, SOSP, 15., 1995. **Proceedings...** New York: ACM Press, 1995. p.172–182.

TSOUMAKOS, D.; ROUSSOPOULOS, N. A Comparison of Peer-to-Peer Search Methods. In: INTERNATIONAL WORKSHOP ON THE WEB AND DATABASES, 6., 2003, San Diego, USA. **Proceedings...** [S.l.: s.n.], 2003.

UMAR, A. **Mobile Computing And Wireless Communications**. [S.l.]: Nge Solutions, 2004. 712 p.

VINOSKI, S. Service discovery 101. **IEEE Internet Computing**, [S.l.], v.7, n.1, p.69–71, Jan.-Feb. 2003.

W3C. **W3C Recommendation - XSL Transformations (XSLT) Version 1.0**. Disponível em: <http://www.w3.org/TR/xslt/>. Acesso em: jul. 2005.

WEB Ontology Language. Disponível em: <http://www.w3.org/2001/sw/WebOnt/>. Acesso em: jul. 2005.

WEISER, M. The computer of the 21st Century. **Scientific American**, [S.l.], v.265, n.3, p.66–75, Sept. 1991.

WEISER, M.; WELCH, B.; DEMERS, A.; SHENKER, S. Scheduling for Reduced CPU Energy. In: USENIX SYMPOSIUM ON OPERATING SYSTEM DESIGN AND IMPLEMENTATION, 1., 1994, Monterey, CA. **Proceedings...** [S.l.: s.n.], 1994.

YAMIN, A. C. **Arquitetura para um Ambiente de Grade Computacional Direcionado às Aplicações Distribuídas, Móveis e Conscientes do Contexto da Computação Pervasiva**. 2004. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

YAMIN, A. C.; AUGUSTIN, I.; BARBOSA, J. L. V.; SILVA, L. C. da; GEYER, C. F. R. Collaborative Multilevel Adaptation in Distributed Mobile Applications. In: INTERNATIONAL CONFERENCE OF THE CHILEAN COMPUTER SCIENCE SOCIETY, 22., 2002, Atacama, Chile. **Proceedings...** [S.l.]: IEEE-CS, 2002.

YAMIN, A. C.; BARBOSA, J. L. V.; AUGUSTIN, I.; SILVA, L. C. da; REAL, R. A.; GEYER, C. F. R.; CAVALHEIRO, G. A framework for exploiting adaptation in high heterogeneous distributed processing. In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, SBAC-PAD, 14., 2002, Vitória, Brazil. **Proceedings...** [S.l.: s.n.], 2002.

YAMIN, A. C.; BARBOSA, J. L. V.; AUGUSTIN, I.; SILVA, L. C. da; REAL, R.; GEYER, C.; CAVALHEIRO, G. Towards Merging Context-aware, Mobile and Grid Computing. **The International Journal of High Performance Computing Applications**, [S.l.], v.17, n.2, p.191–203, 2003.

YANG, B.; GARCIA-MOLINA, H. Designing a super-peer network. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 19., 2003. **Proceedings...** [S.l.: s.n.], 2003. p.49–60.