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

MAURÍCIO COUTINHO MORAES

DIMI: um Disseminador *Multicast* de Informações para a Arquitetura ISAM

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

Prof. Dr. Cláudio F.R. Geyer
Orientador

Porto Alegre, outubro de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Moraes, Maurício Coutinho

DIMI: um Disseminador *Multicast* de Informações para a Arquitetura ISAM / Maurício Coutinho Moraes. - Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

87 f.:il.

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

1.ISAM. 2. Disseminação de dados. 3.Computação pervasiva. 4. Roteamento de mensagens. 5. *Multicast* no nível de aplicação. I. Geyer, Cláudio F.R. II. Título.

"Information is about talking to each other: yes, cars and shoes and especially genetic cures are important, but in the end those are just trappings. What truly matters is how we relate to the world. And so much of that is about communication... As we talk to each other more, amazing things begin to happen, possibly more amazing even than the promise of genetic engineering... I think that perhaps the results of the communication revolution will not be seen from the effects of moving large quantities of bits around; we shall see the true revolution because we will all be able to talk to each other more easily: one-on-one, but also in groups and, as a planet. I've heard it suggested that the next revolution is the formation of a kind of global mind that results from enough people and enough interconnectedness."

BRUCE ECKEL, NO PREFÁCIO DE SEU LIVRO "THINKING IN JAVA"

AGRADECIMENTOS

– À minha família, pelos incentivos moral e financeiro.

– Ao meu orientador, pela orientação e pelas várias portas abertas durante o curso.

– Especialmente ao meu co-orientador informal, Luciano Cavalheiro da Silva, pelas valiosíssimas contribuições para esta dissertação, em todos os seus estágios de desenvolvimento.

– Aos integrantes do projeto ISAM, pela acolhida e pela participação ativa na contrução de artigos.

– Ao colega Alberto Egon, profissional exemplar, pelas diversas críticas contrutivas a esta dissertação e por incentivar-me a escrever artigos.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CP	Computação Pervasiva
IP	Internet Protocol
ISAM	Infra-estrutura de Suporte às Aplicações Móveis
ISAM*pe*	ISAM pervasive environment
LPA	Lista Parcial de Ascendência
LPU	Lista Parcial de Unidades de Disseminação
MB	Megabytes
MDC	Multiple Description Coding
MNA	Multicast no Nível de Aplicação
NAT	Network Address Translator
NS-2	Network Simulator-2
NTP	Network Time Protocol
P2P	Peer-to-Peer
PE	Ponto de Encontro
PEC	Ponto de Entrada de Consumo
PEP	Ponto de Entrada de Produção
RMI	Remote Method Invocation
TCP	Transmission Control Protocol
UD	Unidade de Disseminação
UDP	User Datagram Protocol
UML	Unified Modeling Language
XML	Extensible Markup Language

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O projeto ISAM apresenta uma plataforma para o desenvolvimento e a execução de aplicações pervasivas. O ambiente de execução proposto na arquitetura ISAM, denominado ISAM*pe* (ISAM *pervasive environment*), foi concebido para ser implantado em escala global, com elevado número de componentes. Esses componentes podem ser móveis ou fixos e podem apresentar limitações em seus recursos computacionais. Muitas das aplicações pervasivas que podem ser executadas no ISAM*pe* têm necessidade de um serviço de disseminação capaz de distribuir informações de um produtor para um grande número de consumidores. Esta dissertação apresenta um serviço de disseminação de informações para a arquitetura ISAM, denominado DIMI (Disseminador Multicast de Informações). O DIMI apresenta uma arquitetura de *multicast* no nível de aplicação.

O DIMI propõe um algoritmo de formação da topologia de disseminação que tem o objetivo de alcançar maior escalabilidade pelo alívio de sobrecarga em participantes específicos do canal, durante os momentos em que houver um grande número de novos consumidores querendo iniciar o consumo simultaneamente. No seu ambiente-alvo, o ISAM*pe*, o DIMI também oferece suporte à desconexão planejada, permitindo que dispositivos computacionais com limitações de conectividade participem da disseminação, e ainda oferece suporte à mobilidade de usuários. A topologia de comunicação do DIMI adapta-se às condições da rede física subjacente a ela, de acordo com critérios específicos da aplicação que utilizar o serviço. Os resultados obtidos com a simulação de alguns aspectos do funcionamento do DIMI e com a execução do protótipo do mesmo validam os argumentos usados para justificar a necessidade e a forma de construção do serviço.

Palavras-Chave: ISAM, disseminação de dados, computação pervasiva, roteamento de mensagens, *multicast* no nível de aplicação.

.

DIMI: A Multicast Information Disseminator for the ISAM Architecture

ABSTRACT

The ISAM project presents a platform to the development and to the execution of pervasive applications. The execution environment proposed by the ISAM architecture, named ISAM*pe* (ISAM pervasive environment) was conceived to be deployed on global scale, having a large number of components. These components may be mobile or static and may present computational resource limitations. Many of the pervasive applications that may be executed on the ISAM*pe* need a dissemination service capable of distribute information from one producer to many consumers. This dissertation presents a information dissemination service for the ISAM architecture, named DIMI (Multicast Information Disseminator - *Disseminador Multicast de Informações*). DIMI presents an application-level multicast architecture.

DIMI proposes an algorithm to create the dissemination topology which objective is to achieve scalability through the relief of overload on specific participants of the channel, during the moments where many new consumers want to start consumption simultaneously. Inside its target-environment, DIMI also offer support to user mobility and to planned disconnection, allowing resource limited computational devices to be participants of the dissemination. DIMI's dissemination topology adapt itself to the conditions of fabric network, accordingly to applications' criteria. The results obtained with the simulation of some DIMI's characteristics and with prototipation validate the arguments used to justify the necessity and the way of construction of the service.

Keywords: ISAM, data dissemination, pervasive computing, message routing, application-level multicast.

1 INTRODUÇÃO

As linhas de pesquisa em ambientes distribuídos que surgiram nos últimos anos, como a computação em grade (FOSTER; KESSELMAN, 1999), as redes *peer-to-peer* (ORAM, 2001), a computação móvel (ADACHI, 2002), a computação consciente do contexto (CHEN; KOTZ, 2003) e a computação pervasiva (SATYANARAYANAN, 2001), (SAHA; MUKHERJEE, 2003), somadas aos recentes avanços tecnológicos obtidos na construção de *hardware* de redes de computadores (STANFORD, 2003) indicam que os ambientes computacionais do futuro terão um número de componentes de *hardware* e de *software*, tanto homogêneos quanto heterogêneos, muito maior do que o existente na atual Internet e um número muito grande de relacionamentos entre esses componentes. Além disso, espera-se que a criação, a manutenção e a destruição de grande parte desses relacionamentos seja feita dinamicamente. Para que esses ambientes computacionais tornem-se amplamente utilizados por usuários ordinários, assim como ocorre hoje com a Internet, é necessário que eles apresentem, entre outras características, elevada escalabilidade associada a um desempenho aceitável (O'SULLIVAN; LEWIS, 2003).

Entre as novas linhas de pesquisa em ambientes computacionais distribuídos, a computação pervasiva (CP) é a proposta de um novo paradigma computacional que permite ao usuário o acesso ao seu ambiente computacional em qualquer lugar, o tempo todo, através de diferentes tipos de *hardware*, móveis ou não. A CP tem foco nos usuários e em suas tarefas e não em dispositivos e tecnologias. As aplicações e o ambiente de execução da CP são conscientes do contexto, ou seja, eles realizam esforços para pró-ativamente monitorar e controlar as condições do ambiente, reagindo às modificações acontecidas neste mesmo ambiente através de processos de adaptação (YAMIN, 2004) (AUGUSTIN, 2004) (SATYANARAYANAN, 2001).

1.1 Contexto

O projeto ISAM (Infra-estrutura de Suporte às Aplicações Móveis Distribuídas) (ISAM, 2005), em desenvolvimento no Grupo de Processamento Paralelo e Distribuído do Instituto de Informática da Universidade Federal do Rio Grande do Sul, apresenta uma plataforma para o desenvolvimento e a execução de aplicações pervasivas. O principal objetivo desse projeto é explorar alternativas para simplificar a tarefa de projetar aplicações para ambientes pervasivos. A pesquisa contida nesta dissertação está inserida no projeto ISAM, tratando as questões relativas à disseminação de informações na arquitetura ISAM.

1.2 Motivação

O ambiente de execução da arquitetura ISAM apresenta mecanismos de obtenção de informações a respeito dos elementos de contexto, como, por exemplo, a utilização de CPU ou de memória. Essas informações devem ser disponibilizadas tanto para o *middleware* do ambiente computacional quanto para as aplicações pervasivas executadas sobre este *middleware*, para que os mecanismos de adaptação possam ser alimentados com dados sobre o contexto e tomar as decisões adaptativas pelas quais eles são responsáveis. Devido ao elevado número de componentes de software e de usuários dos ambientes de CP, é possível que haja muitos consumidores de informações a respeito de elementos de contexto em um determinado instante. Além disso, muitas das aplicações pervasivas que podem ser executadas no ambiente de execução da arquitetura ISAM têm necessidade de um serviço de disseminação capaz de distribuir informações de um produtor para um grande número de consumidores. Exemplos de tais aplicações são as executadas em *smart spaces* (WANG et al., 2004) como salas de aula, hospitais, estádios, serviços de emergência, centros de convenção e campos de batalha, entre outros (YAU; AHAMED, 2003). Devido ao grande número de componentes e de usuários que podem participar do ambiente de execução da arquitetura ISAM (YAMIN, 2004), é necessário que haja a disponibilidade de um serviço de disseminação de informações escalável nesse ambiente de execução.

A proposição de um serviço de disseminação para a arquitetura ISAM baseado na utilização em escala global do protocolo IP Multicast (DEERING; CHERITON, 1990) não é viável, pois a escalabilidade desse protocolo é inversamente proporcional ao número de grupos ou canais concorrentemente ativos (PENDARAKIS et al., 2001) (CHU; RAO; ZHANG, 2000). Além disso, a utilização generalizada do protocolo IP Multicast implicaria problemas de segurança e alterações nos roteadores IP (CHU; RAO; ZHANG, 2000) (ALMEROTH, 2000).Os benefícios trazidos pelo uso do protocolo IP Multicast em termos de economia de largura de banda devem compensar a complexidade associada à criação e à manutenção dos grupos ou canais *multicast*, o que inviabiliza a sua utilização generalizada em uma rede global para dar suporte a ambientes que possuam um grande número de grupos ou canais existindo simultaneamente e que sejam criados e destruídos dinamicamente, como, por exemplo, jogos multi-jogador, salas de reunião e canais de informações de contexto dos ambientes pervasivos.

Devido às limitações do protocolo IP Multicast, o *multicast* no nível de aplicação (MNA) tem surgido como uma alternativa para a disseminação de informações em ambientes largamentes distribuídos, oferecendo inicialização acelerada, configuração simplificada e melhor controle de acesso, ao custo de carga adicional na rede, em relação ao protocolo IP Multicast, devido à transferência de controle da camada de rede para a camada de aplicação (CHU; RAO; ZHANG, 2000) (PENDARAKIS et al., 2001). O MNA não exige que a infra-estrutura da rede física seja alterada em quaisquer de seus componentes (roteadores, protocolos e *software* de gerência). A topologia de disseminação utilizada nesse tipo de *multicast* é determinada por uma rede lógica composta de *endhosts* (i.e os *hosts* onde residem as aplicações que consomem as informações) que trocam informações entre si via transmissões *unicast*.

Diversas arquiteturas de disseminação MNA foram propostas nos últimos anos, como, por exemplo, o HMTP (ZHANG; JAMIN; ZHANG, 2002), o ALMI (PENDARAKIS et al., 2001), o Overcast (JANNOTTI et al., 2000) o TAO (KWON;

FAHMY, 2002), o SALM (BANNERJEE; BHATTACHARJEE; KOMMAREDDY, 2002), o Yoid (FRANCIS, 2000) e o CoopNet (PADMANABHAN et al., 2002). Estas arquiteturas possuem o objetivo de fornecer os benefícios disponibilizados pelo protocolo IP Multicast sem necessitar de alterações ou controle fino da infra-estrutura da rede física e seus componentes. Elas, no entanto, não conseguem aliviar alguns participantes pré-determinados e específicos do grupo ou canal de disseminação de uma potencial sobrecarga caso um grande número de novos consumidores queiram iniciar o consumo simultaneamente.Essa característica restringe sua escalabilidade. O oferecimento de maior escalabilidade para atender um grande número de novos consumidores simultâneos é útil para as classes dos provedores de informações que eventualmente enfrentam picos de demanda por determinado tipo de informação.

Além da disseminação de informações escalável, o *middleware* do ambiente de execução da arquitetura ISAM e as aplicações pervasivas executadas sobre ele precisam de um serviço de disseminaçãode informações capaz de (YAMIN, 2004):

1. adaptar-se à entrada e à saída dinâmica dos usuários e adaptar-se às características dinâmicas da infra-estrutura de rede sobre a qual o serviço é executado, de acordo com critérios específicos dos usuários do serviço, passados via parâmetros;
2. oferecer suporte à movimentação de usuários no ambiente de execução, de forma transparente. O suporte transparente à mobilidade de usuário é importante para que se alcance a *invisibilidade* necessária aos ambientes de computação pervasiva (SATYANARAYANAN, 2001). Dessa forma, uma vez participando de um canal *multicast*, os usuários não necessitam preocupar-se com a manuntenção desta participação em função da sua movimentação pelo ambiente;
3. oferecer suporte à desconexão planejada de dispositivos computacionais no ambiente de execução, de forma transparente aos usuários. A desconexão planejada é aquela desconexão de rede que não é causada por falhas mas pelo comportamento natural da aplicação em um determinado contexto (YAMIN, 2004) (SADOK; KELNER; CORDEIRO, 1999). O suporte transparente à desconexão planejada de usuários, assim como ocorre com o suporte transparente à mobilidade de usuários, é importante para que se alcance a *invisibilidade* necessária aos ambientes de computação pervasiva. Dessa forma, uma vez participando de um canal *multicast*, os usuários não necessitam preocupar-se com a manutenção desta participação em função das restrições de conectividade de rede do dispositivo que estiverem utilizando.

1.3 Objetivo

Em função da motivação discutida, o objetivo geral deste trabalho é propor, desenvolver e testar um serviço de disseminação de informações para ambientes distribuídos em geral, que satisfaça as necessidades da arquitetura ISAM. Tais necessidades são as seguintes: a escalabilidade, o suporte à mobilidade de usuários e o suporte à desconexão planejada. Entre os objetivos específicos deste trabalho, destaca-se a utilização de filtros para aumentar o desempenho da disseminação no serviço proposto.

1.4 Contribuição do Autor

As principais contribuições do autor deste trabalho são a criação, a modelagem e a prototipação de um serviço de disseminação MNA para a arquitetura ISAM. Além disso, o autor propõe um novo protocolo de formação da topologia de disseminação MNA com o objetivo de aumentar a escalabilidade do serviço pelo aumento de capacidade de aceitação simultânea de novos consumidores.

Os resultados provenientes de simulações com o simulador de rede *Network Simulator-2* (NS-2, 2005) e os resultados provenientes da execução do protótipo também destacam-se como contribuições, pois validam os argumentos utilizados para justificar o serviço proposto.

1.5 Estrutura da Dissertação

Esta dissertação está estruturada em 6 capítulos. O capítulo 2 delinea contexto contexto deste trabalho, detalhando (**i**) os aspectos gerais da CP; (**ii**) o projeto ISAM, com sua arquitetura, seu ambiente de execução e seu middleware (**iii**) os fundamentos da disseminação *multicast* e os meios utilizados atualmente para realizá-la em ambientes distribuídos; e (**iv**) os requisitos de um serviço de disseminação para a arquitetura ISAM. O capítulo 3: discute o modelo do serviço de disseminação para a arquitetura ISAM proposto nesta dissertação, com seus objetivos, sua arquitetura, suas funcionalidades e seu comportamento. O capítulo 4 valida os argumentos existentes no capítulo 3 pela apresentação de resultados obtidos com a simulação e com a prototipação do serviço proposto. O capítulo 5 discute o estado da arte da disseminação MNA e realiza uma comparação do DIMI com alguns de seus trabalhos relacionados no que diz respeito às necessidades da arquitetura ISAM. Finalmente, o capítulo 6 finaliza esta dissertação, apresentando a sua conclusão e as direções dos seus trabalhos futuros.

2 CONTEXTO DO TRABALHO

Este capítulo caracteriza os principais aspectos da computação pervasiva e da arquitetura-alvo desta pesquisa, a arquitetura ISAM. A compreensão das características e das necessidades da arquitetura ISAM é necessária para o entendimento da motivação e da modelagem de um serviço de disseminação de informações para esta mesma arquitetura.

Este capítulo também caracteriza os fundamentos da disseminação em canais ou em grupos e discute o estado da arte das estratégias de disseminação de informações utilizadas atualmente. No final deste capítulo, é feita uma síntese dos argumentos expostos para caracterizar os requisitos de um serviço de disseminação de informações para a arquitetura ISAM.

2.1 Computação Pervasiva

A computação pervasiva (CP) é a proposta de um novo paradigma computacional que permite ao usuário o acesso ao seu ambiente computacional em qualquer lugar, o tempo todo, através de diferentes tipos de *hardware*, móveis ou não. A CP tem foco nos usuários e em suas tarefas e não em dispositivos e tecnologias, representando a evolução de duas áreas de pesquisa distintas: a computação distribuída e a computação móvel. Além das linhas de pesquisa existentes nessas duas áreas, a computação pervasiva incorpora outras quatro linhas de pesquisa (SATYNARAYANAN, 2001):

- **uso efetivo de *smartspaces* e interoperabilidade entre *smartspaces***: um *smartspace* é uma área bem definida no espaço, que possui infra-estrutura computacional embarcada capaz de monitorá-la e controlá-la. Exemplos de *smartspaces* podem ser bairros, estádios, casas, carros, geladeiras e salas que contenham hardware e software que interajam com seu ambiente interior e exterior. Um ambiente pervasivo deve ser capaz de permitir a interoperabilidade entre diferentes *smartspaces*;
- **invisibilidade**: um ambiente computacional *invisível* deve causar a mínima distração possível nos usuários a ponto de interagir com eles no nível de subconsciência. Para atingir invisibilidade, o ambiente e seus componentes precisam ajustar seu funcionamento sem que haja necessidade de intervenção humana;
- **escalabilidade local**: devido ao elevado número de usuários e de dispositivos que se espera que exista em um ambiente de computação pervasiva, o número de interações entre os componentes do ambiente deverá ser muito grande. A infra-estrutura pervasiva deve, então, buscar ao máximo a proximidade dos componentes

que interagem entre si consideravelmente. Caso contrário, o sistema poderá ficar sobrecarregado com interações de pouca importância feitas a longas distâncias;

- **adaptação à heterogeneidade e às condições do ambiente de execução**: as aplicações e a infra-estrutura pervasiva devem adaptar-se em tempo de execução às condições técnicas e não-técnicas em que o ambiente de execução se encontra. Esse desafio é fundamental para que o ambiente pervasivo atinja a invisibilidade desejada. Um exemplo de adaptação é a capacidade de lidar com desconexões planejadas, que são aquelas desconexões de rede causadas não por falhas, mas pelo comportamento natural dos dispositivos em um determinado contexto (SADOK; KELNER; CORDEIRO, 1999) (YAMIN, 2004).

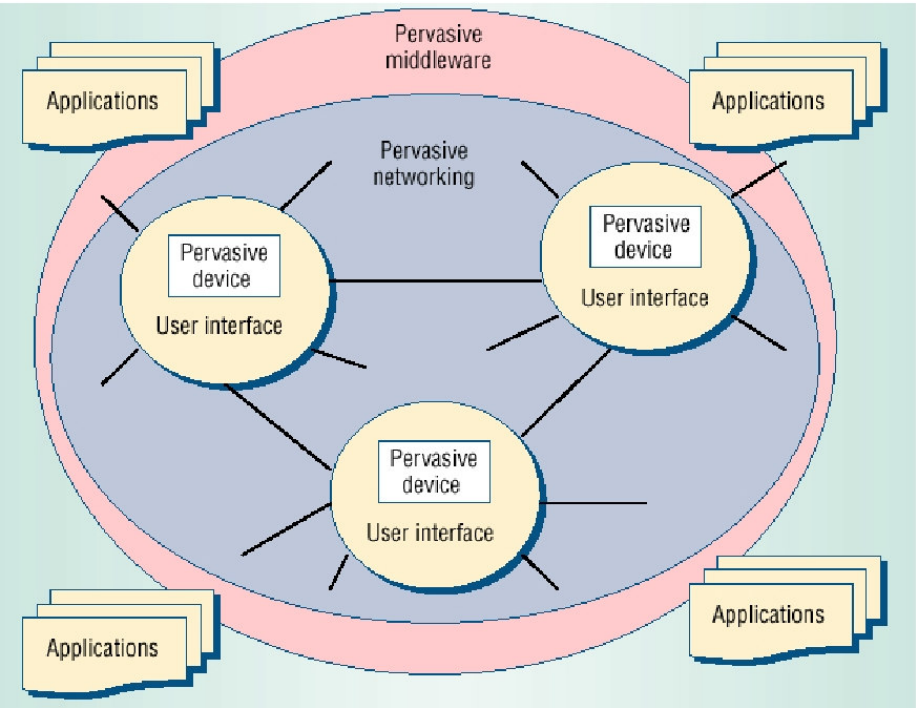

Figura 2.1: Modelo de um ambiente pervasivo.

Os avanços tecnológicos necessários para a construção de um ambiente pervasivo pertencem a quatro grandes áreas (ver figura 2.1) (SAHA; MUKHERJEE, 2003):

- **dispositivos**: diferentes tipos de dispositivos devem existir em um ambiente pervasivo. Entre eles, pode-se citar (**i**) os tradicionais, como monitores, teclados e mouses; (**ii**) os móveis, como celulares e PDAs; e (**iii**) os embarcados, como sensores de ambiente ou bio-sensores;
- **redes**: o número de nós e de conexões entre os nós da rede pervasiva deve ser muito maior do que o que existe hoje na Internet. São necessárias, portanto, mudanças nas tecnologias de rede atuais para possibilitar a integração escalável e eficiente desses componentes em uma rede pervasiva;

- ***middleware***: é necessária a existência de um *middleware* que faça a mediação das interações entre a rede e as aplicações com o objetivo de manter os usuários imersos no ambiente pervasivo;
- **aplicações**: aplicações pervasivas precisam responder e adaptar-se ao ambiente de forma transparente aos usuários, através da interação com o *middleware*.

O hardware e a infra-estrutura de rede necessários para a realização da computação pervasiva estão rapidamente tornando-se uma realidade (STANFORD, 2003). A criação de *software* pervasivo, no entanto, está um pouco mais distante. Esse atraso no desenvolvimento de programas pervasivos é causada pela dificuldade de projetar, construir e executar aplicações em um ambiente pervasivo. É difícil, sem a existência de um ambiente de execução e de suporte às aplicações pervasivas, construir aplicações que se adaptem às constantes mudanças no ambiente de execução e às movimentações de usuários no espaço global (YAMIN, 2004).

Várias propostas de ambientes de computação pervasiva tentam vencer alguns dos desfios de construção de *software* pervasivo, como, por exemplo, o projeto Aura (AURA, 2005) e o projeto Gaia (GAIA, 2003). Um projeto de destaque do Grupo de Processamento Paralelo Distribuído da Universidade Federal do Rio Grande do Sul, que tenta vencer alguns dos desafios de construção de *software* pervasivo, é o projeto ISAM (ISAM, 2005), no qual este trabalho está inserido, que será discutido a seguir.

2.2 O projeto ISAM

O projeto ISAM (Infra-estrutura de Suporte às Aplicações Móveis) apresenta uma plataforma para o desenvolvimento e a execução de aplicações pervasivas (ISAM, 2005). O principal objetivo desse projeto é explorar alternativas para simplificar a tarefa de projetar aplicações para ambientes pervasivos, permitindo, dessa forma, aos projetistas das aplicações, a concentração de esforços na solução do problema e a diminuição de esforços de implementação da solução encontrada para os recursos disponíveis (AUGUSTIN, 2004) (YAMIN, 2004).

2.2.1 A arquitetura

A arquitetura ISAM é apresentada na figura 2.2. Tal arquitetura está organizada em camadas lógicas, com níveis diferenciados de abstração que enfatizam dois momentos:

- o desenvolvimento das aplicações pervasivas (AUGUSTIN, 2004);
- o gerenciamento do ambiente de execução das aplicações pervasivas (YAMIN, 2004).

A representação da consciência do contexto como um módulo virtual tem por objetivo ressaltar sua importância na arquitetura, bem como caracterizar sua presença na concepção em todos os componentes.

Na arquitetura ISAM, as aplicações pervasivas são distribuídas, móveis e sensíveis ao contexto. A arquitetura ISAM foi construída com base na tese do projeto ISAM: uma infra-estrutura para o desenvolvimento e a execução de aplicações pervasivas é obtida pela integração das soluções existentes para computação em grade (FOSTER; KESSELMAN, 1999) (FORUM, 2002), computação móvel (ADACHI, 2002) e computação consciente do contexto (CHEN; KOTZ, 2003). O principal conceito envolvido nessa integração é o comportamento de adaptação dinâmica ao contexto das

aplicações e do próprio ambiente de execução, o qual viabiliza a implementação da semântica *siga-me* definida na arquitetura ISAM, onde a aplicação segue o usuário em sua movimentação pelo espaço global.

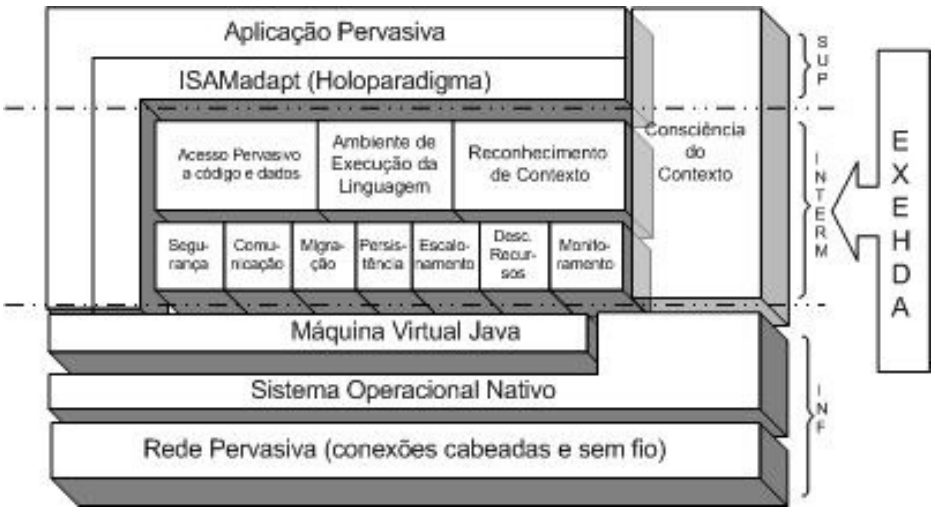

Figura 2.2: Arquitetura ISAM.

A arquitetura ISAM, ilustrada na figura 2.2, é composta de três camadas. A camada superior (SUP) contém as aplicações pervasivas e as abstrações que a arquitetura ISAM fornece para a construção e a execução das mesmas. As aplicações pervasivas são construídas de acordo com o modelo de programação adotado pelo projeto ISAM, o ISAMAdapt (AUGUSTIN, 2004). O ISAMAdapt partiu das abstrações do Holoparadigma (BARBOSA, 2002) e adicionou novas abstrações e construções com uma semântica adequada ao ambiente pervasivo, como, por exemplo, facilidades para a codificação de aplicações adaptativas e sensíveis ao contexto. A utização do modelo de programação ISAMAdapt justifica-se pela insuficiência e inapropriação de outros modelos existentes para tratar as necessidades pervasivas das aplicações.

A camada intermediária (INTERM) representa o ambiente de execução, codificado no *middleware* do ISAM, discutido na seção 2.2.3. A camada intermediária contém os mecanismos de suporte à execução das aplicações pervasivas e às estratégias de adaptação. Tal camada é dividida em dois níveis. O primeiro nível é composto por três módulos de serviço à aplicação, responsáveis pelo acesso pervasivo a dados e código, pela execução das aplicações e pelo serviço de reconhecimento de contexto. O segundo nível da camada intermediária contém os módulos básicos do ambiente de execução, dos quais provêm as funcionalidades necessárias para o primeiro nível, cobrindo vários aspectos, tais como **comunicação**, migração, persistência, escalonamento, monitoramento, segurança e descoberta de recursos. O serviço proposto nesta dissertação faz parte do módulo de comunicação do segundo nível da camada intermediária.

A camada inferior (INF) é composta pelos sistemas e linguagens nativos que integram o meio físico de execução. Por questões de portabilidade, a plataforma base de implementação é a Máquina Virtual Java (JVM). A arquitetura supõe a existência de

uma rede global com suporte à operação sem fio, interligada à outra cabeada, que disponibilize uma infra-estrutura de equipamentos e serviços em escala global.

2.2.2 O ambiente Computacional

Na arquitetura ISAM, o ambiente computacional é a união de todos os recursos computacionais colaborando entre si e agindo coordenamente em escala global. Tal ambiente, ilustrado na figura 2.2.3, é denominado ISAM *pervasive environment* (ISAM*pe*). Nele, todos os componentes (dados, código, dispositivos, serviços e recursos) são pervasivos e gerenciados por um *middleware* que fornece acesso aos mesmos. Neste cenário, os dispositivos móveis são interfaces que funcionam como portais que recebem as tarefas a serem executadas, podendo transferir essa execução segundo critérios próprios.

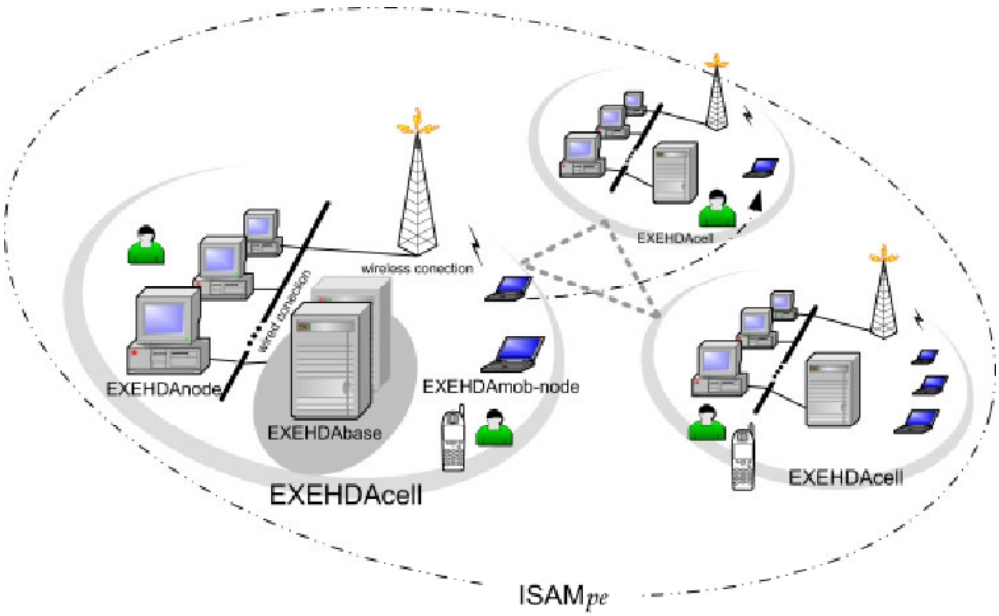

Figura 2.3: O ISAM*pe*.

Cada usuário, no ISAM*pe*, possui um ambiente virtual que pode ser acessado de qualquer lugar em que ele se encontrar, a partir de qualquer tipo de dispositivo. Além disso, esse ambiente adapta-se às movimentações do usuário no espaço global, de acordo com a semântica *siga-me* das aplicações pervasivas. outro comportamento presente nas aplicações pervasivas do ISAM*pe* é capacidade de adaptação aos períodos de desconexão planejada.

As aplicações pervasivas no ISAM*pe* não devem fazer qualquer suposição a respeito do contexto em que serão executadas. Ao invés disso, elas devem apresentar comportamento adaptativo ao contexto. Contexto é definido como toda a informação relevante para a aplicação que pode ser obtida do ambiente de execução. Alterações de estado no ambiente são gatilhos que disparam o comportamento adaptativo das aplicações. Os programadores das aplicações explicitamente identificam os aspectos da entidade de onde provém a informação e definem seus atributos (elementos de contexto), os quais passam a integrar o contexto da aplicação. Para exemplificar, um nó de processamento poderá ter como elementos de contexto a ocupação de memória e o

tamanho da filas de processos. Uma alteração em um desses atributos pode ser usada para disparar um procedimento de adaptação, tanto na aplicação quanto no ambiente de execução.

O ISAM*pe* oferece suporte à mobilidade lógica e física dos componentes do ambiente pervasivo. Mobilidade lógica é a reorganização da relação dos componentes de *software* de uma aplicação ou do serviço do ambiente pervasivo. Nessa reorganização poderá ou não ser disparada uma migração de componentes de *software*. Já a mobilidade física é entendida como a mobilidade de usuário, empregando ou não um dispositivo computacional móvel.

2.2.3 O *Middleware*

O *middleware* da arquitetura ISAM, denominado *Execution Environment for Highly Distributed Applications* (EXEHDA) - Ambiente de Execução para Aplicações Largamente Distribuídas - fornece suporte à adaptação no nível de aplicação e possui capacidade de adaptação em seu próprio funcionamento (EXEHDA, 2005) (YAMIN, 2004). A capacidade de adaptação faz com que o sistema se ajuste às mudanças no estado dos recursos disponíveis.

O EXEHDA oferece assistência ao gerenciamento do ISAM*pe*, através de serviços de controle do meio físico onde a computação é executada, e também assistência à execução das aplicações, através de serviços e abstrações necessários para a implementação e a execução das mesmas. De forma resumida, os principais objetivos do EXEHDA são:

- gerenciar aspectos funcionais e não-funcionais da execução das aplicações;
- suportar adaptações dinâmicas na execução das aplicações;
- fornecer mecanismos para construir, gerenciar e **disseminar** informações de contexto;
- usar as informações de contexto em seus mecanismos de tomada de decisão;
- decidir, juntamente com as aplicações, a respeito de ações de adaptação;
- oferecer aos usuários da arquitetura ISAM um comportamento que expresse a semântica *siga-me*, onde a aplicação segue o usuário em sua movimentação pelo espaço global;
- manter-se consistente e operacional nos momentos em que houver desconexão planejada em um nó.

O EXEHDA pode ser visto a partir de duas grandes perspectivas. A primeira delas é o ponto de vista das aplicações da computação pervasiva, para as quais o EXEHDA é o provedor dos serviços. Nesse caso, o EXEHDA proporciona às aplicações a visão do ISAM*pe*. A segunda perspectiva é o ponto de vista dos recursos que compõem o meio físico distribuído, para os quais o EXEHDA define as políticas de organização, gerência e utilização.

A premissa de integrar os cenários da computação em grade, da computação móvel e da computação sensível ao contexto, no EXEHDA, é mapeada para uma organização composta de células de execução, conforme ilustrado na figura 2.3. De acordo com essa agregação de células, os recursos da infra-estrutura física são mapeados para três abstrações básicas, que compõem o ISAM*pe*:

EXEHDAcell ou célula: é a área de atuação de uma EXEHDAbase, e é composta de EXEHDAnodes. A determinação da abrangência de uma EXEHDAcell é feita de acordo com algum critério, como, por exemplo, proximidade geográfica e/ou lógica;

EXEHDAbase ou base: é o ponto de contato para os EXEHDAnodes. É responsável por todos os serviços disponibilizados no ISAM*pe*. Por questões de escalabilidade, uma EXEHDAbase pode estar replicada por diversos dispositivos. Essa replicação, no entanto, é transparente aos EXEHDAnodes. Estruturalmente, uma base dispõe de recursos sem limitações severas de processamento, armazenamento, transmissão e conectividade. As ligações entre as EXEHDAbases são robustas o bastante para que desconexões entre duas delas sejam consideradas infreqüentes. Uma EXEHDAbase é uma entidade estável dentro da EXEHDAcell, permitindo que os demais integrantes da célula tenham um caráter mais dinâmico no que se refere à sua disponibilidade na célula;

EXEHDAnode ou nó de processamento: é o equipamento de processamento disponível em uma base, responsável pela execução das aplicações. Os EXEHDAnodes podem apresentar mobilidade, estando localizados na EXEHDAcell que detiver seu ponto de acesso em um determinado momento. Um EXEHDAnode pode apresentar limitações severas de recursos de processamento, armazenamento, transmissão ou conectividade.

2.3 Disseminação *Multicast*

A disseminação *multicast* em um canal ou em um grupo ocorre quando uma mensagem é enviada ao mesmo tempo para muitos destinos, através da utilização de um nome lógico, identificador do canal, caracterizando uma comunicação do tipo 1-para-N ou N-para-N (EL-SAYED; ROCHA; MATHY, 2003) (CHOCKLER; KEIDAR; VITENBERG, 2001). Apesar de mais de uma década já haver passado desde 1990, quando Deering propôs a primeira versão do IP Multicast (DEERING; CHERITON, 1990), a disseminação *multicast* é uma área de pesquisa considerada em fase inicial de desenvolvimento (ALMEEROTH, 2000).

O conjunto de origens e destinos de uma informação disseminada via *multicast* é chamado **canal**, grupo ou sessão. Cada um dos elementos de um canal é chamado de **participante** ou membro. Roteadores são participantes que retransmitem a informação após recebê-la.

Uma disseminação *multicast* é ótima quando possui as seguintes características:

- todos os destinos recebem todas as mensagens da origem;
- são feitas apenas as duplicações estritamente necessárias da informação para que todos os destinos a recebam;
- os roteadores e os destinos recebem uma determinada mensagem uma única vez;
- as ligações entre os parcipantes são as mais curtas possível. O critério de medida de distância entre os participantes é específico de cada aplicação, sendo os mais comuns a largura de banda e a latência das ligações de rede (PENDARAKIS et al., 2001) (FRANCIS, 2000) (JANNOTTI et al. 2000).

De acordo com as características citadas, o fluxo de uma disseminação *multicast* ótima, exemplificado na figura 2.4, pode ser representado por uma topologia de árvore onde o nó raiz é a origem, os nós internos da árvore são os roteadores, as folhas são os destinos da mensagem e as arestas da árvore são as ligações de rede mais curtas entre pais e filhos. A partir dessa representação de árvore de uma disseminação *multicast* ótima, algumas constatações podem ser feitas:

- sem qualquer tipo de restrição, um roteador pode guardar uma mensagem, após retransmití-la, para dela fazer uso como se fosse um de seus destinos finais;
- cada nó da árvore não precisa conhecer nada mais na topologia além de seu pai e seus filhos. Ou seja, para cada um dos nós da árvore, seu pai é a origem dos dados e seus filhos são os destinos. Essa informação é suficiente para que a transmissão seja levada a cabo e os dados cheguem a todos os seus consumidores, sem perda alguma para o serviço.

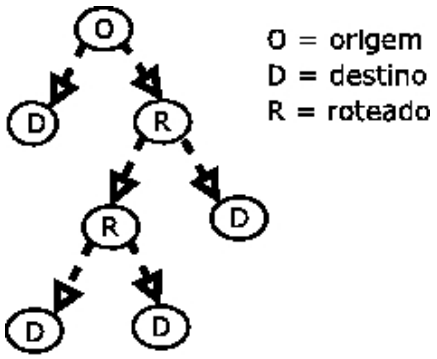

Figura 2.4: Exemplo uma árvore de disseminação *multicast*.

As constatações feitas levam à seguinte conclusão: um mecanismo que se proponha a realizar a disseminação *multicast* pode ser baseado em uma unidade básica de (re)transmissão, que incorpore as funções das origens, dos destinos e dos roteadores de uma árvore de disseminação *multicast*. Tais unidades básicas de (re)transmissão apresentam o seguinte comportamento: ao receber uma mensagem, retransmita-a para todos os destinatários cadastrados para esse tipo de mensagem. O tipo de mensagem pode ser o identificador da árvore de disseminação em questão.

São dois os tipos de disseminação *multicast* realizados atualmente: (**i**) a disseminação feita através do protocolo IP Multicast e gerenciada pelos roteadores IP existentes na infra-estrutura da Internet; e (**ii**) a disseminação feita no nível de aplicação, gerenciada pelos *endhosts* participantes do canal ou grupo (i.e, os *hosts* onde residem as aplicações que consomem as informações disseminadas). Esses dois tipos de disseminação serão discutidos nas próximas subseções.

2.3.1 O Protocolo IP *Multicast*

O protocolo IP Multicast foi a primeira característica funcional importante adicionada ao protocolo IP (IP, 2005) (CHU; RAO; ZHANG, 2000), possuindo as seguintes características (DEERING; CHERITON, 1999) (ALMEROTH, 2000):

uma origem pode enviar pacotes *multicast* para o canal a qualquer tempo, sem qualquer tipo de restrição;

- o protocolo IP Multicast foi construído sobre o protocolo UDP (*User Datagram Protocol*) (UDP, 2005), logo os pacotes são entregues aos destinos segundo a política do melhor esforço, sem garantias de entrega;
- as origens só precisam conhecer o endereço *multicast* do canal. As origens não precisam conhecer os participantes do canal para o qual estão enviando pacotes. Um canal pode ter um número qualquer de origens e destinos;
- os participantes podem entrar ou sair de um canal de acordo com a sua vontade, sem necessidade de registro, sincronização ou negociação com qualquer outro participante do canal ou componente centralizador.

O protocolo IP Multicast introduziu a idéia da comunicação em canais sobre IP, mas não encontrou condições de ser largamente empregado (PENDARAKIS et al., 2001) (FRANCIS, 2000) (ZHANG; JAMIN; ZHANG, 2002). O protocolo IP Multicast foi proposto como uma extensão da arquitetura da Internet para dar suporte a entrega de pacotes a múltiplos destinos no nível de rede, constituindo-se uma solução para a disseminação em canais que contenham um grande número de participantes. Apesar disso, o protocolo IP Multicast possui as seguintes desvantagens que o impedem de ser implantado em larga escala na Internet (CHU; RAO; ZHANG, 2000):

- é necessário que os roteadores mantenham o estado do canal, o que não apenas viola o princípio arquitetural *sem estado* do projeto original de roteadores IP, mas também introduz maior complexidade e sérias limitações de escalabilidade à camada IP, no que diz respeito ao número de grupos simultaneamente ativos;
- o protocolo IP Multicast permite que uma origem qualquer envie pacotes para um grupo arbitrário, o que torna a rede vulnerável a ataques do tipo *flooding* e complica o gerenciamento da rede;
- é necessário que cada canal dinamicamente obtenha um endereço globalmente único do espaço de endereçamento *multicast*, o que é difícil de fazer de uma maneira escalável, distribuída e consistente;
- o protocolo IP Multicast possui a semântica de entrega pelo melhor esforço. Isso torna mais difícil a implantação de características como garantia de entrega e controle de fluxo em aplicações baseadas no IP Multicast do que em aplicações baseadas no protocolo TCP (*Transmission Control Protocol*) (TPC, 2005)

2.3.2 *Multicast* no Nível de Aplicação

Devido às características negativas citadas do IP Multicast, o *multicast* no nível de aplicação (MNA) tem surgido como uma alternativa para a disseminação de informações em canais.

No MNA, os participantes do canal são *endhosts* (i.e, os *hosts* onde residem as aplicações que consomem as informações) que interconectam-se, formando topologias lógicas, denominadas redes *overlay* (JANNOTTI et al., 2000) (CHU; RAO; ZHAANG, 2000) (FRANCIS, 2000). Nessas topologias, as transmissões são do tipo *unicast*. No MNA, o gerenciamento da rede permanece inalterado e a implantação do serviço é acelerada em comparação com o IP Multicast, pois não há a necessidade de qualquer

alteração na infra-estrutura de rede disponível hoje na Internet. O MNA mantém a natureza *sem estado* dos roteadores IP transferindo a manutenção dos canais para a camada de aplicação dos seus *endhosts* participantes. Além disso, o MNA delega às aplicações a tarefa de atribuir nomes globalmente únicos aos canais, simplificando a construção de mecanismos de nomeação escaláveis, distribuídos e consistentes, de acordo com as necessidades de cada aplicação.

Em compensação, em relação ao IP Multicast, os aspectos negativos do MNA são (CHU; RAO; ZHAANG, 2000) (PENDAARAKIS et al., 2001):

- a impossibilidade de prevenir a construção de múltiplas ligações *unicast* sobre a mesma ligação física. Dessa forma, não é possível evitar o desperdício de recursos de rede pela redundância de transmissão de informações;
- o potencial aumento de latência das transmissão, devido ao fato da mensagem possivelmente ter de passar por outros *endhosts* em seu trajeto, antes de chegar ao seu destino final.

2.3.2.1 Topologias Utilizadas em Multicast no Nível de Aplicação

Como já foi explicado, no MNA, a topologia de comunicação é determinada por uma rede lógica composta de *endhosts*. O principal desafio da comunicação no MNA é a construção e a manutenção de uma topologia bem estruturada para a entrega das informações, de acordo com as características desejadas na execução da disseminação. A topologia indicada para uma aplicação que dê preferência a alto desempenho na disseminação pode não ser a mais indicada para uma aplicação que precise de maiores garantias de entrega de dados (FRANCIS, 2000).

As duas topologias utilizadas pelos mecanismos de disseminação encontradas nos trabalhos relacionados são as árvores e os grafos ou *meshes*. Uma árvore tem a característica principal de oferecer eficiência e economia de recursos na entrega dos dados, pois cada mensagem vai diretamente a todos os seus destinos, sendo duplicada apenas nos pontos onde isso se faz necessário, sem passar mais de uma vez em cada ligação lógica entre os participantes da topologia. A fragilidade, no entanto, também é uma característica de uma árvore, pois uma falha ou parada de serviço em um único participante não-folha é capaz de parti-la, impedindo alguns dos destinos de receber as mensagens.

Grafos ou *meshes* não apresentam a eficiência e a economia de recursos das árvores pois podem levar um participante a receber e eventualmente processar mais de uma vez a mesma mensagem ou podem permitir que uma mensagem seja transmitida mais de uma vez por uma ligação na topologia, inclusive em ciclos. Apesar disso, grafos são mais robustos que árvores, pois é possível que diversos de seus participantes simultaneamente parem de funcionar sem que a disseminação *multicast* seja necessariamente prejudicada.

A escolha de uma ou outra topologia é ditada pela relação de troca entre o ganho de desempenho e busca por robustez. Nada impede, entretanto, que soluções de mesclagem de árvores e grafos sejam utilizadas em mecanismos MNA. Um exemplo de mesclagem poderia ser a utilização, em um mesmo canal, de uma árvore para a disseminação de dados e de um grafo para a troca de mensagens de controle para gerência e manutenção da topologia.

2.3.2.2 Métricas de Avaliação de Desempenho para Multicast no Nível de Aplicação

Os principais argumentos contra a utilização do MNA no lugar de IP Multicast fazem menção ao consumo redundante de recursos e à perda de desempenho imposta pelo MNA. Para embasar ou refutar tais argumentos, diversas métricas de avaliação do desempenho de MNA e do impacto de sua utilização na rede podem ser utilizadas (EL-SAYED; ROCHA; MATHY, 2003) (CHU; RAO; ZHANG, 2000):

- ***stress***: métrica que define o número de cópias de um pacote que passam por uma ligação física de rede. O valor ótimo, alcançado pelo IP Multicast, é 1. Isso quer dizer que, em uma disseminação *multicast*, o ideal é que não haja duplicação de mensagens em uma ligação física. Esta métrica de avaliação mostra a qualidade da distribuição de carga na rede física pelo MNA;
- **uso de recursos**: soma de todos os produtos (latência* *stress*) em todas as ligações de rede que participam da disseminação de dados no MNA. Esta métrica de avaliação fornece uma idéia do uso dos recursos utilizados pelo MNA, assumindo que ligações com latências maiores são mais custosas.
- ***stretch***: também chamada de penalidade relativa de atraso *relative delay penalty* entre a raiz e um participante qualquer do canal. É a razão entre o atraso de entrega de um pacote na topologia MNA e o atraso de uma entrega para o mesmo participante através de uma transmissão *unicast*. Esta métrica indica o aumento de atraso obtido quando uma transmissão MNA é utilizada no lugar de uma transmissão *unicast*;
- **perda de pacotes após falha**: número médio de pacotes perdidos devido à falha de um único participante. Esta métrica tenta mostrar a robusteza do MNA perante a ocorrência de eventos imprevisíveis em um dos participantes do canal.

2.4 Requisitos do Serviço de Disseminação para a Arquitetura ISAM

As características citadas da arquitetura ISAM, somadas à abrangência global do ISAM*pe*, requerem um serviço de disseminação altamente escalável, capaz de lidar com a mobilidade dos usuários e capaz de ter consciência do contexto em que for executado. Também é necessário que o serviço adapte-se ao contexto de acordo com as necessidades dos usuários. A proposição de um serviço de disseminação para a arquitetura ISAM baseado principalmente em IP Multicast (DEERING; CHERITON, 1990) não é viável, pois a escalabilidade deste é inversamente proporcional ao número de grupos concorrentemente ativos. Além disso, a utilização generalizada de IP *multicast* implicaria problemas de segurança e mudanças substanciais na infra-estrutura de hardware existente e no software dos roteadores IP (CHU; RAO; ZHANG, 2000) (ALMEROTH, 2000).

Uma solução possível para a disseminação de informações no ISAM*pe* é a utilização de *multicast* no nível de aplicação (MNA), em que os participantes do canal conectam-se em redes lógicas e se comunicam através de transmissões *unicast*. Sistemas como o HMTP (ZHANG; JAMIN; ZHANG, 2002), o CoopNet (PADMANABHAN et al., 2002), o Yoid (FRANCIS, 2000) e o Almi (PENDARAKIS et al. 2001) utilizam o MNA para disseminar informações em ambientes distribuídos. Esses sistemas, no entanto, não preenchem os requisitos da arquitetura ISAM, porque

não possuem consciência do contexto, não apresentam suporte à mobilidade de usuários e apresentam escalabilidade limitada, pois delegam a responsabilidade da aceitação de novos participantes no canal a participantes específicos existentes no canal. Essa delegação de responsabilidade representa um gargalo nas situações em que muitos novos consumidores querem iniciar o consumo simultaneamente.

O oferecimento de maior escalabilidade para atender um grande número de novos consumidores simultâneos é útil para as classes dos provedores de informações que eventualmente enfrentam picos de demanda por determinado tipo de informação. Essas situações de pico de demanda são comuns em ambientes de computação pervasiva, devido ao enorme número existente de usuários, de dispositivos e de interações entre os mesmos (SATYANARAYANAN, 2001). Essas mesmas situações, no entanto, não pertencem exclusivamente aos ambientes de computação pervasiva. Existem alguns sítios da Internet atual que se enquadram nessa situação, como o SlashDot (SLASHDOT, 2005), a Receita Federal do Brasil (RECEITA, 2005) e o MSNBC (MSNBC, 2005), entre outros. Um exemplo de situação desse tipo foi a busca de informações a respeito dos ataques com aviões nos Estados Unidos, ocorridos em 11 de setembro de 2001 (PADMANABHAN et al., 2002).

De forma resumida, os principais requisitos do serviço de disseminação para a arquitetura ISAM são: elevada escalabilidade, adaptação dinâmica às modificações do ambiente de execução, suporte à mobilidade de usuários e suporte à desconexão planejada. O próximo capítulo apresenta uma proposta de serviço de disseminação de informações que preenche esses requisitos da arquitetura ISAM.

3 DIMI: PROPOSTA DE UM SERVIÇO DE DISSEMINAÇÃO PARA A ARQUITETURA ISAM

Neste capítulo são apresentados os objetivos do serviço DIMI, as características dos dados disseminados por ele, seu modelo conceitual, os componentes de sua arquitetura e seu funcionamento. Além disso, é discutida a forma como o DIMI se adequa às características do ambiente de execução da arquitetura ISAM, o ISAM*pe*, para atingir seus objetivos.

3.1 Objetivos

Para atender as necessidades da arquitetura ISAM (YAMIN, 2004) (AUGUSTIN, 2004), o modelo de disseminação de dados do serviço DIMI tem os seguintes objetivos:

- **escalabilidade**: o serviço deve ser escalável e evitar a formação de gargalos em função da admissão concorrente de novos participantes em um canal. Ou seja, o serviço deve possuir escalabilidade para atender um grande número de novos consumidores simultâneos;
- **adaptação dinâmica**: o serviço deve adaptar-se em tempo de execução às características dinâmicas do ambiente de execução, redirecionando dados e criando ou destruindo ligações conforme o estado em que se encontrem os participantes do canal, inclusive de acordo com seu consumo corrente de CPU e de memória;
- **suporte à mobilidade de usuários**: o serviço deve ser capaz de dar suporte à mobilidade de usuários, de forma transparente. Isso é importante para que se alcance a *invisibilidade* necessária ao ISAM*pe*. Dessa forma, uma vez participando de um canal, os usuários não necessitam preocupar-se com a manutenção desta participação em função da sua movimentação pelo ambiente computacional;
- **suporte à desconexão planejada**: o serviço deve ser capaz de dar suporte à desconexão planejada de dispositivos, de forma transparente aos usuários. O suporte transparente à desconexão planejada de usuários também é importante para que se alcance a *invisibilidade* necessária ao ISAM*pe*. Com isso, uma vez participando de um canal, os usuários não necessitam preocupar-se com a manuntenção desta participação em função das restrições de conectividade de rede do dispositivo que estiverem utilizando;
- **utilização de filtros**: o serviço deve poder realizar pré e pós-processamento dos dados disseminados com o objetivo de diminuir o tamanho dos mesmos e,

conseqüentemente, economizar recursos e aumentar o desempenho da disseminação.

3.2 Tipos de Dados Disseminados e Usuários

Os usuários do serviço são quaisquer componentes que desejem transmitir ou receber informações originadas em um produtor e destinadas a uma quantidade indeterminada de consumidores. Exemplos de usuários do serviço são aplicações de vídeo-transmissão e o serviço de reconhecimento de contexto da arquitetura ISAM.

Os dados disseminados formam um fluxo seqüencial de mensagens independentes entre si. Isso quer dizer que um consumidor em bom funcionamento recebe as mensagens na mesma ordem em que elas são produzidas e que os consumidores podem processar uma mensagem imediatamente após sua recepção, sem necessidade de esperar por outras mensagens ou pelo fim do fluxo de mensagens. O serviço se esforça na entrega das mensagens mas não garante a entrega das mesmas, devido a eventuais limitações de espaço nos *buffers* de mensagens, discutidos na seção 3.8.3, ou devido a eventuais falhas nos integrantes de um canal.

3.3 Arquitetura

A arquitetura do serviço DIMI está ilustrada na figura 3.1. Os participantes da disseminação, ou seja, as origens, os destinos e os roteadores da disseminação feita no DIMI são indistintamente chamados de **Unidades de Disseminação (UDs)**. As UDs podem estabelecer relacionamentos entre si, chamados **conexões**. As conexões entre as UDs são relativas a um único canal, de tal forma que duas UDs podem estar conectadas em um canal *A* mas não possuir qualquer conexão em um canal *B*, onde *A* é diferente de *B*. As UDs podem participar de zero ou mais canais simultaneamente. Pode haver apenas uma conexão entre duas UDs em um canal.

Uma UD X entra em um canal *A* através do estabelecimento de uma conexão com outra UD *Y*. Nesse caso, *Y* é dito um **ponto de entrada** para *X* ou **pai** de *X* em *A* e *X* passa a ser **filho** de *Y* em *A*. Uma UD pode ter muitos pais e muitos filhos em um canal, o que permite a redundância na disseminação. Um relacionamento do tipo pai-filho em *A* estabelece que o pai envia os dados da disseminação para o filho, que os recebe. Apesar dos dados de disseminação partirem do pai para o filho em um canal, um filho pode iniciar uma comunicação de controle com seu pai sem restrição alguma.

Pontos de Entrada de Produtores (PEPs) e Pontos de Entrada de Consumidores (PECs) são os componentes de utilização do serviço por parte dos usuários e estão ligados a uma UD. PEPs e PECs são componentes de fachada (GAMMA et al., 2003) para o acesso a uma UD. Uma UD pode ter muitos PEPs e PECs conectados a si.

Um PEP é utilizado para a criação e a destruição de um canal e para o envio de dados pelo mesmo. Pode haver um único PEP por canal e um PEP pode estar ligado a um único canal. A UD do PEP de um canal é denominada a **raiz do canal**.

Um PEC é utilizado pelos usuários para a recepção de dados em um canal. Cada consumidor possui um PEC no canal. Pode haver um número arbitrário de PECs em um canal, mas um PEC somente pode receber dados de um único canal.

Os produtores enviam dados para os consumidores através de **mensagens**. Mensagens são estruturas semelhantes a pacotes de rede, contendo cabeçalho e corpo.

Os dados fornecidos a um PEP são disseminados da seguinte forma: a raiz repassa a mensagem para seus filhos no canal e estes repassam a mensagem para os seus filhos, que repetem o comportamento até que a mensagem chegue a todos os seus consumidores.

As UDs são programas executados em dispositivos computacionais físicos como PCs, PDAs, celulares, etc. Esses dispositivos físicos que executam as UDs são chamados de **dispositivos disseminadores**.

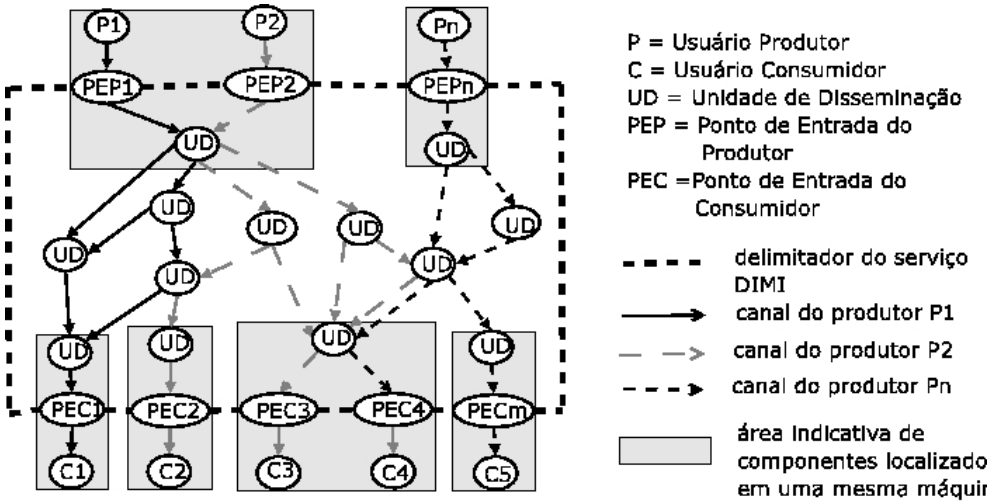

Figura 3.1: Arquitetura do serviço DIMI.

As UDs possuem, associadas a cada canal de que participam, uma *Lista Parcial de UDs (LPU)*, composta de um conjunto conhecido de UDs que participam do canal e ordenada crescentemente pela distância de cada uma dessas UDs em relação à UD que possui a LPU. Uma LPU é uma estrutura de dados auxiliar utilizada para a manutenção de um canal, conforme será discutido na seção 3.7.7.

3.4 Modelo Conceitual

O modelo conceitual do DIMI foi exposto em forma de diagrama de classes UML (*Unified Modeling Language)* (UML, 2005). Tal notação foi utilizada pois através dela é possível indicar quais são os principais componentes do serviço, quais as suas funcionalidades e como esses componentes estão relacionados entre si. As classes do modelo conceitual do DIMI e seus relacionamentos são mostrados na figura 3.2. As próximas subseções descrevem com detalhes cada uma dessas classes.

3.4.1 A Classe *Message*

Um objeto da classe *Message* representa uma mensagem, que é a estrutura de dados que contém a informação que deve ser disseminada. Uma mensagem é composta por um cabeçalho (atributo *header*) e um corpo (atributo *content*). O cabeçalho contém informações sobre a mensagem, distribuídas nos seguintes campos:

- *CHANNEL-NAME*: campo que contém o nome do canal a que pertence a mensagem. Essa informação é necessária para o roteamento da mensagem;

- *ORIGIN-NAME*: campo que contém o nome da origem da mensagem. A origem de uma mensagem é o nome da última UD pela qual a mensagem passou. Esta informação é necessária para a manutenção da topologia de disseminação em um canal, que será discutida na seção 3.7.7;

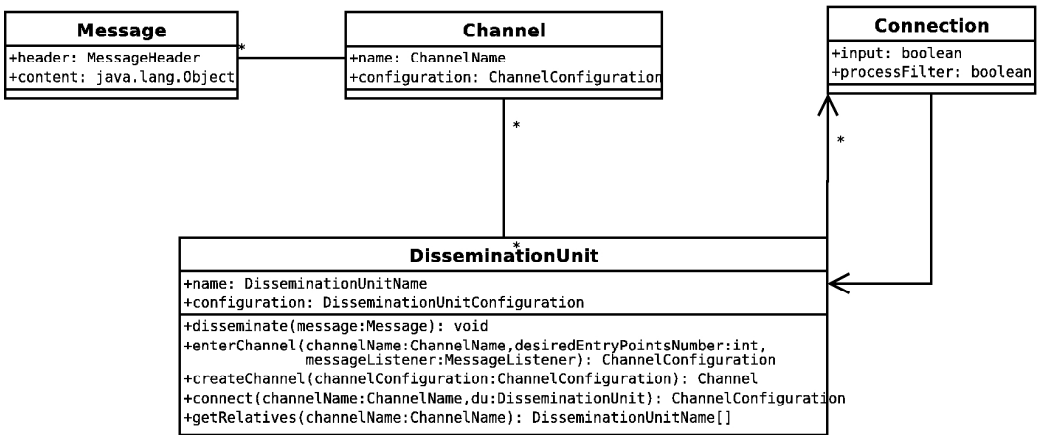

Figura 3.2: Principais classes do modelo DIMI.

- *SEQUENCE-NUMBER*: campo que contém o número identificador de uma mensagem de dados no canal a que ela pertence. As mensagens de dados em um canal são numeradas sequencialmente, sendo que o número de sequência da primeira mensagem é zero. O valor deste campo em mensagens que não são de dados é insignificante;
- *RODUCTION-TIMESTAMP*: campo que contém a data e a hora obtida no produtor da mensagem, no momento do envio da mensagem. Este campo pode servir para medir o atraso da mensagem no canal, desde que todos os dispositivos disseminadores possuam seus relógios sincronizados;
- *ORIGIN-TIMESTAMP*: campo que contém a data e a hora obtida na origem da mensagem, no momento do envio da mensagem. Este campo pode servir para medir o atraso da mensagem na conexão pela qual foi entregue, desde que os dispositivos disseminadores relativos à conexão possuam seus relógios sincronizados;
- *PATH*: campo que armazena o nome de todas as UDs pelas quais a mensagem passou desde a sua produção. Este campo é utilizado para a detecção de ciclos de disseminação, conforme será discutido na seção 3.7.6;
- *TYPE*: campo identificador de tipo de mensagem. Este campo é utilizado pelo serviço para realizar funções de controle. Detalhes sobre este campo serão discutidos mais adiante nesta mesma subseção;
- *FILTERED*: este campo indica se a mensagem foi ou não filtrada. Filtros servem para aumentar o desempenho da disseminação em um canal e serão discutidos na seção 3.6;

- *CONTENT-SIZE*: campo que contém o número de *bytes* do corpo da mensagem. Esta informação é necessária para gerenciar a utilização dos *buffers* de mensagens das UDs.

O campo *TYPE* do cabeçalho de uma mensagem serve para permitir que o serviço realize sua própria manutenção, através do envio de mensagens de controle pelas suas UDs. Ao receber uma mensagem, uma UD executa as operações relativas ao tipo da mensagem, que pode ser um dos seguintes:

- *DATA*: tipo indicativo de mensagem de dados;
- *HEART-BEAT*[1]: uma mensagem deste tipo é enviada de um pai para um filho em um canal. Essa mensagem tem a função de avisar o filho que o pai continua participando do canal e que, por isso, o filho pode permanecer conectado;
- *DESTROY-CHANNEL*: tipo de mensagem que avisa o destino que o canal foi destruído. Ao receber uma mensagem desse tipo, uma UD deve disseminá-la e, após isso, apagar todas as estruturas alocadas para o funcionamento do canal da mensagem;
- *LEAVE*[2]: uma mensagem deste tipo avisa o destino que a origem saiu do canal. Ao receber essa mensagem de um filho, a UD deve liberar todas as estruturas alocadas para a manutenção daquela conexão naquele canal e verificar se ainda possui interesse em participar do canal. Caso não haja tal interesse, a UD destino também deve desconectar-se de seu(s) pai(s) no canal. A verificação de interesse de permanência de uma UD em um canal será discutida na seção 3.7.5. Ao receber uma mensagem desse tipo de qualquer um de seus pais em um canal, a UD destino deve procurar um pai substituto no canal;
- *LPA-CHANGED*: uma mensagem deste tipo contém o novo valor da LPA (lista parcial de ascendência) da origem. O conceito de LPA será discutido na seção 3.7.6.

3.4.2 A Classe DisseminationUnit

A classe *DisseminationUnit* representa o elemento principal do serviço: a UD. O papel de uma UD é o de responsável pela criação e manutenção cooperativa dos canais de que fizer parte e o de gerente de disseminação. Uma UD *A* pode executar as seguintes operações, que serão referenciadas adiante no texto, nas seções 3.5, 3.7 e 3.8:

- *disseminate*: método responsável pela disseminação de uma mensagem de *A* para seus filhos. A mensagem disseminada é aquela passada como parâmetro na chamada deste método (parâmetro *message*);
- *createChannel*: método de criação de um canal em *A*, a partir de um conjunto de configurações de canal (parâmetro *channelConfiguration*);
- e*nterChannel*: método que causa a execução do protocolo de entrada em um canal, que será discutido na seção 3.7.2. Este método recebe como parâmetros o nome do canal do qual *A* deve consumir mensagens (*channelName*) e o número de pais que *A* deve procurar ter no canal (*desiredEntryPointsNumber*);

---

[1] As mensagens deste tipo **não** são retransmitidas para os filhos das UDs que as receberem

[2] Idem ao anterior.

- *connect*: este método conecta uma instância de *DisseminationUnit* (parâmetro *du*) como filho de *A* em um canal cujo nome é fornecido pelo parâmetro *channelName*;
- *getRelatives*: método de obtenção dos nomes das UDs que são pais ou filhos de *A* em um canal cujo nome é fornecido pelo parâmetro *channelName*. A informação obtida através deste método é necessária para o funcionamento do protocolo de entrada em um canal, que será discutido na seção 3.7.2.

Instâncias diferentes da classe *DisseminationUnit* interconectam-se para formar os canais do serviço DIMI, havendo, obrigatoriamente, apenas uma instância de *DisseminationUnit* por dispositivo disseminador. Uma instância de *DisseminationUnit* possui um conjunto de configurações (atributo *configuration*) e um nome (atributo name). O nome de uma *DisseminationUnit* é representado por uma URL com o seguinte formato: **DIMI://IP:port**, que contém o endereço IP e a porta de trabalho da UD. O conjunto de configurações de uma *DisseminationUnit*, definido via parâmetros, no momento de disparo do serviço em um dispositivo disseminador, contém os seguintes campos:

- *FILTER-LIST*: este campo contém a lista de todos os filtros que a UD é capaz de processar. Cada elemento desta lista é um par-ordenado do tipo <*nome-do-filtro, nome-da-classe-que-implementa-o-filtro*>. Filtros serão discutidos na seção 3.6;
- *DISTANCE-METRICS-LIST*: este campo contém a lista de todas as métricas de distância que a UD é capaz de processar. Cada elemento desta lista é um par-ordenado do tipo <*nome-da-metrica-de-distância, nome-da-classe-que-implementa-a-metrica-de-distância*>. Métricas de distância são os critérios utilizados para a medida de distância entre UDs em um canal, podendo ser, por exemplo, a latência, a largura de banda ou o número de passos (*hops*) entre duas UDs;
- *BUFFER-SIZE*: este campo contém o tamanho máximo do *buffer* de mensagens do dispositivo disseminador. *Buffers* são estruturas de dados usadas no suporte à desconexão planejada de usuários, conforme será discutido na seção 3.8.3;
- *NEW-CHILD-ACCEPTANCE*: este campo contém a lista dos valores máximos permitidos de utilização de CPU e de memória, em porcentagem, para que possa haver aceitação de novos filhos pela UD.

Um exemplo de arquivo de configuração de uma UD *A* é mostrado na listagem de arquivo da figura 3.3. O arquivo XML (*eXtended Markup Language*) (XML, 2005) da referida figura indica que *A*:

- é capaz de calcular o filtro chamado *diff*, implementado pela classe *dimi.channel.filter.Diff* e o filtro chamado *zip*, implementado pela classe *dimi.channel.filter.Zip*;
- reconhece a métrica de distância chamada *latency*, implementada pela classe *dimi.network.metrics.Latency*, e a métrica de distância chamada *bandwidth*, implementada pela classe *dimi.network.metrics.Bandwidth*;
- possui um *buffer* de 2 MB de espaço;

- aceitará novos filhos no canal desde que, no momento da requisição feita pelo novo filho potencial, a utilização de sua CPU não ultrapasse 70% de sua capacidade total e a utilização de sua memória não ultrapasse 60% de sua capacidade total.

3.4.3 A Classe *Channel*

A classe *Channel* representa um canal do serviço DIMI. Pode haver mais de um canal co-existindo, sendo que canais co-existentes são totalmente independentes entre si. Um canal possui um nome (atributo *name*), globalmente único, e um conjunto de configurações (atributo *configuration*) fornecido como parâmetro pelo usuário no momento da sua criação, que contém os seguintes campos:

```
<DIMI-DU-CONFIG>
 <FILTER-LIST>
  <FILTER>
   <NAME> diff </NAME>
   <IMPL> dimi.channel.filter.Diff </IMPL>
  </FILTER>
  <FILTER>
   <NAME> zip </NAME>
   <IMPL> dimi.channel.filter.Zip </IMPL>
  </FILTER>
 </FILTER-LIST>
 <DISTANCE-METRICS-LIST>
  <DISTANCE-METRIC>
   <NAME> latency </NAME>
   <IMPL> dimi.network.metrics.Latency </IMPL>
  </DISTANCE-METRIC>
  <DISTANCE-METRIC>
   <NAME> bandwidth </NAME>
   <IMPL> dimi.network.metrics.Bandwidth </IMPL>
  </DISTANCE-METRIC>
 </DISTANCE-METRICS-LIST>
 <MESSAGE-BUFFER-SIZE> 2MB </MESSAGE-BUFFER-SIZE>
 <NEW-CHILD-ACCEPTANCE>
  <CPU-MAX-USAGE> 70% </CPU-MAX-USAGE>
  <MEMORY-MAX-USAGE> 60% </MEMORY-MAX-USAGE>
 </NEW-CHILD-ACCEPTANCE>
</DIMI-DU-CONFIG>
```

Figura 3.3: Exemplo de arquivo XML utilizado para parametrização de uma UD.

- *FILTER*: este campo contém o nome do filtro utilizado no canal. Filtros serão discutidos na seção 3.6;
- *DESCRIPTION*: este campo contém uma descrição textual do conteúdo das mensagens de um canal. Um exemplo de descrição pode ser: "*Canal auxiliar de monitoração de sensores de temperatura da célula X*";
- *PRIORITY*: este campo contém a prioridade do canal representada por um número inteiro positivo, onde 1 identifica a prioridade máxima. Mensagens de canais com prioridade maior, quando *bufferizadas* em uma UD, tem precedência sobre mensagens de canais com prioridade menor, e, portanto, são transmitidas em primeiro lugar. As mensagens de canais com igual prioridade

são transmitidas em ordem de chegada. A *bufferização* de mensagens será discutida na seção 3.8.3;

- *HEART-BEAT-PERIOD*: este campo contém o período, em milisegundos, de emissão das mensagens de tipo *HEART-BEAT*. Essas mensagens indicam quais UDs conectadas permanecem funcionando e, por isso, são importantes para a manutenção da topologia de disseminação. O valor deste campo deve ser escolhido com cuidado, pois um intervalo pequeno de envio de mensagens *HEART-BEAT* pode sobrecarregar a rede e, consequentemente, diminuir o desempenho do serviço e de outras aplicações. Um intervalo grande para este parâmetro pode, no entanto, causar a não-recepção de um número significativo de mensagens por parte de alguns consumidores caso as suas respectivas UDs precisem encontrar novos pontos de entrada devido a falhas em seus pais. A escolha do melhor valor deve ser baseada na previsão da taxa de produção de mensagens de dados em um canal. Quanto maior for a frequência de produção de mensagens de dados em um canal, maior deve ser a taxa de transmissão de mensagens *HEART-BEAT*;

```
<DIMI-CHANNEL-CONFIG>
  <DESCRIPTION> This channel contains data about ...</DESCRIPTION>
  <PRIORITY> 3 </PRIORITY>
  <FILTER> diff </FILTER>
  <HEART-BEAT-PERIOD> 1000 </HEART-BEAT-PERIOD>
  <DISTANCE-METRIC> latency </DISTANCE-METRIC>
  <PRODUCER-BUFFER-OVERFLOW-BEHAVIOR> throw-exception
  </PRODUCER-BUFFER-OVERFLOW-BEHAVIOR>
</DIMI-CHANNEL-CONFIG>
```

Figura 3.4: Exemplo de arquivo XML para parametrização da criação de um canal.

- *DISTANCE-METRIC*: este campo contém o nome da medida de distância utilizada no canal.

Um exemplo de arquivo de configuração de um canal *C* é mostrado na listagem de arquivo da figura 3.4. O arquivo XML da referida figura indica que *C*:

- possui uma descrição textual "*This channel contains data about ...*";
- possui prioridade 3;
- possui um filtro de nome *diff*;
- possui um período de envio de mensagens *HEART-BEAT* de 1000 milisegundos;
- possui uma métrica de distância chamada *latency*;
- dispara uma exceção ao usuário produtor toda vez que houver estouro de *buffer* na UD raiz do canal, conforme será discutido na seção 3.8.3.

3.4.3.1 Nomeação de Canais

Um nome de canal é representado por uma URL que contém o nome de uma UD que faz parte do canal concatenado ao nome local do canal na mesma UD, no seguinte formato: ***dimi://nome_da_UD/nome_local_do_canal***. Segundo essa formação de

nomes, um canal possui tantas URLs quantas são as UDs que dele fazem parte. Todas essas URLs, globalmente únicas, referenciam o mesmo canal, em UDs diferentes. Por exemplo, digamos que a UD *X*, de URL *dimi://143.12.54.230:1000*, seja pai da UD *Y*, de URL *dimi://143.12.54.231:1001*, no canal *C*, e que o nome local de *C* em *X* seja *C1* e em *Y* seja *C2*. Assim, *C* pode ser referenciado por dimi*://143.12.54.230:1000/C1* e por *dimi://143.12.54.231:1001/C2*. Uma eventual UD *Z* que queira entrar em *C* pode referenciar *C* por qualquer uma de suas URLs. A existência de diversos nomes para o mesmo canal, onde cada nome referencia uma UD distinta, é importante para o aumento da escalabilidade do serviço, de acordo com o protocolo de entrada, que será discutido na seção 3.7.2.

3.4.4 A Classe *Connection*

A classe *Connection* representa uma conexão entre duas UDs em um canal. Pode haver mais de um conexão entre duas UDs, desde que em canais distintos. Conexões são totalmente independentes entre si. Uma conexão indica se a UD conectada pode ou não processar o filtro do canal, através do atributo *processFilter*. Uma conexão também indica se a UD conectada é um pai ou um filho no canal, através do atributo *input*.

3.5 Interfaces do Serviço para os Usuários

Esta seção discute as interfaces do serviço com o objetivo de tornar claro quais são as possibilidades dos usuários ao utilizar o serviço DIMI. As interfaces estão divididas em dois grupos: as de utilização do serviço e as de definição do serviço. As interfaces de utilização do serviço contêm os métodos relativos à participação dos usuários em um canal. As interfaces de definição do serviço são as que oferecem meios para os usuários definirem o comportamento do serviço. De forma resumida, ao utilizar o serviço DIMI, o usuário pode:

- criar e destruir canais;
- entrar e sair de canais;
- enviar mensagens por um canal;
- definir métodos de consumo de mensagens;
- definir o filtro de um canal;
- definir a métrica de distância entre as UDs de um canal.

As interfaces de utilização do serviço, mostradas na figura 3.5, são as seguintes:

- *DisseminationService*: interface de criação de canais, através do método *createChannel*, e de entrada em um canal, através método *enterChannel*. Esses dois métodos são padrões de projeto do tipo *facade* ou fachada (GAMMA et al., 2003) para os métodos de mesmo nome da classe *DisseminationUnit*;
- *ProducerEntryPoint*: interface de um PEP, através da qual é possível disseminar dados em um canal, pelo método *disseminate*, e destruir um canal, pelo método *destroy*. O método *disseminate* é uma fachada para o método de mesmo nome da classe *DisseminationUnit*. O método *destroy* também é uma fachada para o método *disseminate* da classe *DisseminationUnit* passando a este uma mensagem do tipo *DESTROY-CHANNEL* como parâmetro;

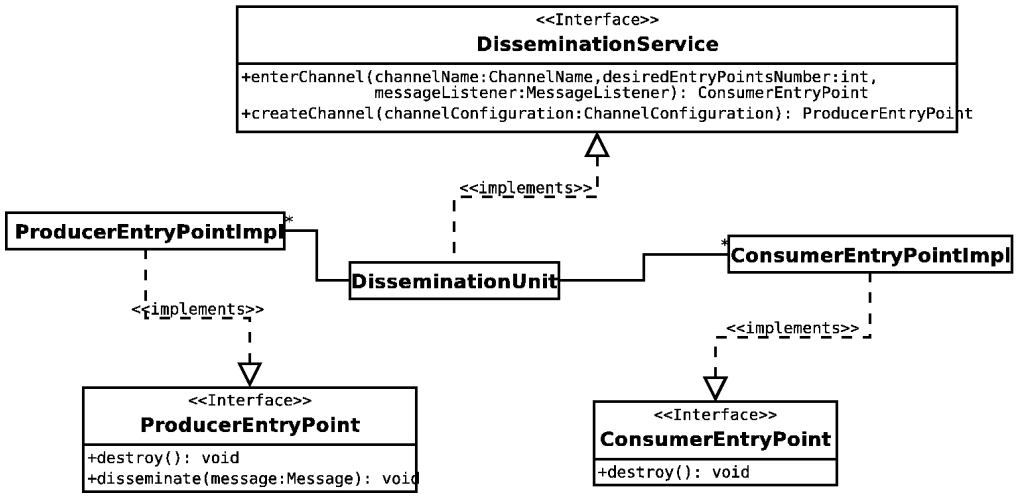

Figura 3.5: Interfaces de utilização do serviço DIMI.

- *ConsumerEntryPoint*: interface de um PEC, através da qual é possível terminar o consumo de dados em um canal, pelo método *destroy*, que destrói o PEC do usuário, impossibilitando qualquer tipo de utilização posterior.

As interfaces de definição do serviço, mostradas na figura 3.6, são as seguintes:

- *MessageListener*: interface utilizada para a definição do comportamento de recepção das mensagens pelos usuários consumidores. Esta interface funciona como um mecanismo de *callback*, em que o método *onMessage*, definido e implementado pelo usuário, é executado toda vez que uma mensagem chega por um canal.
- *Filter*: a interface *Filter* permite aos usuários definir a implementação do filtro utilizado em um canal. O método *apply* deve aplicar o filtro à mensagem passada como parâmetro (parâmetro *message*), ajustar o campo *FILTERED* do cabeçalho da mesma com o valor *true* e retornar a mensagem filtrada. O método *remove* deve remover o filtro da mensagem passada como parâmetro (parâmetro *message*), ajustar o campo *FILTERED* do cabeçalho da mesma com o valor *false* e retornar a mensagem desfiltrada. Filtros serão discutidos na seção 3.6;

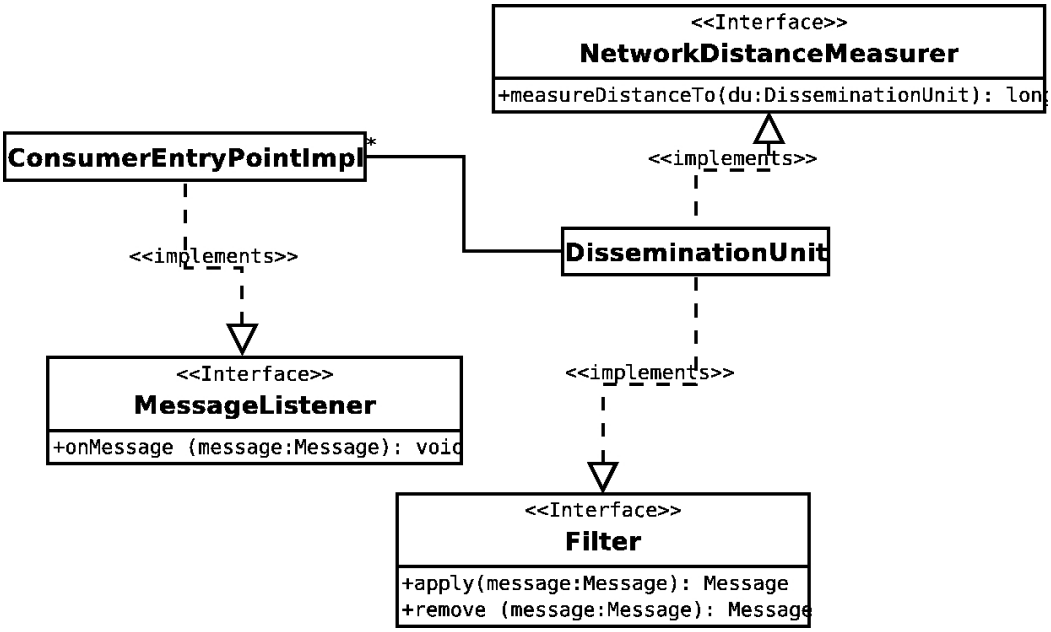

Figura 3.6: Interfaces de definição do serviço DIMI.

- *NetworkDistanceMeasurer*: interface através da qual é possível definir a implementação do medidor de distância entre as UDs de um canal. O método *measureDistanceTo* recebe um objeto (parâmetro *du*) da classe *DisseminationUnit* e retorna um número inteiro. Quanto maior for o valor do retorno, maior será considerada a distância até a UD passada como parâmetro.

3.5.1 Exceções

Exceções são geradas aos usuários sempre que não for possível executar algum dos métodos das interfaces do serviço de forma satisfatória. As exceções do serviço são as seguintes:

- *CantCreateChannelException*: uma exceção de criação de canal é disparada toda vez que não for possível criar um canal. Um canal não é criado caso não haja disponibilidade de recursos na UD que é o ponto de entrada do PEP do novo canal. Nos casos em que isso ocorrer, o usuário deve esperar algum tempo e tentar executar novamente a criação do canal. Um canal também não é criado caso o arquivo XML que contém as configuração do novo canal não esteja bem formado;
- *CantEnterChannelException*: esta exceção é disparada no ínicio do consumo, sempre que uma UD não conseguir entrar em um canal. Uma UD pode não conseguir entrar em um canal caso não encontre nenhum ponto de entrada que a aceite como filha no canal. Isso pode ocorrer caso todas as UDs do canal não aceitem novas conexões devido a terem atingido o limite máximo de utilização de recursos dos seus dispositivos disseminadores, conforme foi discutido na seção 3.4.2;
- *CantFindChannelException*: exceção disparada no início do consumo, caso o canal procurado não exista ou não possa ser localizado;

- *CantFindDisseminationUnitException*: exceção disparada no início do consumo, quando não for possível localizar a UD referenciada no nome do canal desejado;
- *CantDisseminateException*: exceção disparada ao usuário produtor quando a UD do PEP não for capaz de disseminar uma mensagem. Uma UD pode não conseguir disseminar uma mensagem em um canal caso o canal tenha sido previamente destruído ou quando ocorrer o estouro do *buffer* de mensagens, caso este comportamento tenha sido escolhido pelo usuário no momento da criação do canal.

3.6 Uso de Filtros

Um filtro aplicado a um canal é um processamento feito nas mensagens do mesmo com o objetivo de economizar recursos e otimizar a transmissão de dados, de forma transparente aos usuários. Um exemplo de filtro de canal é o filtro diferencial. Um filtro diferencial causa a disseminação das diferenças entre o corpo de duas mensagens consecutivas, em vez da disseminação das mensagens completas. Supondo que *m1* seja a primeira mensagem a ser enviada em um canal, como não há mensagem anterior a *m1*, seu conteúdo é disseminado completamente. Sendo *m2* a segunda mensagem no canal, a mensagem disseminada *m2*' contém apenas a diferença entre *m2* e *m1*, calculada logo após o envio de *m2* pelo produtor. Imediatamente antes do consumidor receber *m2*', *m2* é restaurada pelo serviço a partir de *m1*. A utilização do filtro diferencial em um canal traz potenciais ganhos de desempenho na disseminação, pois o tamanho da diferença entre o corpo duas mensagens *m2-m1=m2*' é provavelmente menor que o tamanho de *m2*. Como o filtro é aplicado pelo serviço, o processo é transparente ao usuário. Um exemplo de canal em que a utilização de filtros diferenciais é bastante eficiente é o canal cujas mensagens são arquivos XML com DTD fixa, pois estes geralmente contêm um alto grau de redundância em seu conteúdo. Exemplos de ferramentas que podem ser utilizadas em uma implementação do filtro diferencial são as ferramentas de software livre *DIFF* (DIFF, 2005) e *PATCH* (PATCH, 2005).

Outro exemplo de filtro é o compactador. Um filtro compactador realiza a compactação das mensagens de acordo com um algoritmo qualquer de compactação, como, por exemplo o Zip (ZIP, 2005) ou o Gzip (GZIP, 2005). Assim como ocorre no caso do filtro diferencial, é provável o ganho de desempenho na disseminação de dados pois uma mensagem com dados compactados é potencialmente menor que uma mensagem com dados não-compactados, utilizando menos recursos de rede e dimininuindo o tempo de transmissão.

```
public Message process(
        Message msg,     // mensagem a ser disseminada
        Connection conn, // conexão pela qual a mensagem deve ser transmitida
        Filter filter) { // filtro do canal

 if(msg.isFiltered()) { // Se a mensagem já está filtrada.
  if(conn.processFilter()) {// Se a UD da conexão processa o filtro do canal.
    return msg; // Retorna a mensagem.
  } else { // Se a UD da conexão não processa o filtro do canal.
    return filter.remove(msg); // Retorna a mensagem filtrada.
  }
 } else { // se a mensagem não está filtrada
  if (conn.processFilter()) {// Sea UD da conexão processa o filtro do canal.
    return filter.apply (msg); // Retorna a mensagem filtrada.
```

```
  } else { // Se a UD da conexão processa o filtro do canal.
   return msg; // Retorna a mensagem.
  }
 }
}
```

Figura 3.7: Algoritmo de aplicação de filtro sobre uma mensagem de um canal

Como visto na seção 3.4.2, cada UD é capaz de processar apenas um conjunto limitado de filtros, indicado em suas configurações. UDs, entretanto, podem participar de canais que possuam filtros que elas sejam incapazes de calcular. Isso é possível com a utilização do algoritmo mostrado na figura 3.7, sob a forma de método escrito na linguagem Java. Tal algoritmo efetua a filtragem da mensagem de acordo com o estado da mesma (filtrada ou não-filtrada) e de acordo com a capacidade que a UD destino tem de processar o filtro. Note-se que, da forma como está escrito, este algoritmo é executado para cada conexão com cada filho de uma UD em uma disseminação. O método foi exposto assim com o objetivo de ser didático. Em uma implementação real, com o objetivo de ganho de desempenho, não há restrições em se alterar o método para que a operação de filtragem seja feita uma única vez, se necessário, para todos os filhos de uma UD.

3.7 Funcionamento

Esta seção discute todos os processos relativos à criação, à gerência e à manutenção dos canais do serviço. Diagramas de seqüência UML são utilizados para ilustrar o funcionamento desses processos.

3.7.1 Criação de um Canal e Início de Produção

A criação de um canal *C* é iniciada por um usuário do serviço de disseminação, através da chamada do método *createChannel* da interface *DisseminationService* da UD do seu dispositivo disseminador, que cria uma instância de *channel* e o PEP de *C*, conforme ilustrado na figura 3.8. A chamada para *createChannel* fica bloqueada até que *C* seja criado. Após isso, o controle retorna ao usuário e *C* torna-se acessível aos consumidores. O nome de *C* é gerado localmente pela UD do dispositivo disseminador do usuário.

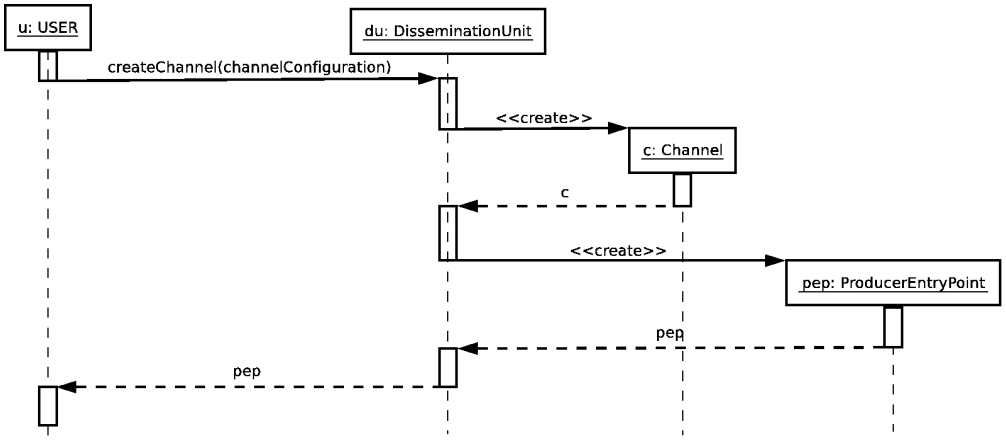

Figura 3.8: Passos envolvidos na criação de um canal.

3.7.2 Início de Consumo

Novos consumidores obtêm um nome do canal desejado por meios externos ao serviço, como, por exemplo, por email ou por *banners*. A partir do momento em que o usuário obtém um nome do canal, ele está apto a entrar no canal. O usuário pode entrar no canal existente apenas como consumidor, pois o produtor único do canal já entrou no mesmo no momento da sua criação.

Um usuário que deseja consumir informações de um canal deve executar o método *enterChannel* da interface *DisseminationService* da UD do seu dispositivo disseminador, de acordo com o diagrama de seqüência da figura 3.9. Este método é responsável pela descoberta de um pai no canal e pela criação de uma instância da classe *ConsumerEntryPoint* para o usuário consumidor. O referido método fica bloqueado até que a UD do dispositivo decida se é ou não possível localizar o canal desejado e entrar no mesmo. Caso isso seja possível, o método é então desbloqueado e o controle é repassado ao usuário, que torna-se apto a receber as mensagens do canal. Caso contrário, é disparada uma exceção ao usuário, conforme discutido na seção 3.5.1.

O processo de entrada em um canal, denominado *protocolo de entrada em um canal*, serve para descobrir o melhor ponto de entrada no canal para a UD que quer iniciar o consumo. O melhor ponto de entrada é, em um subconjunto de todas as UDs do canal, aquele que apresenta a menor distância até a nova UD. O protocolo de entrada proposto aqui é inpirado no HMTP (ZHANG; KAMIN; ZHANG, 2002), que obriga que todas as novas requisições sejam feitas na raiz do canal, sobrecarregando-a. A principal diferença entre o protocolo de entrada em um canal do DIMI e do HMTP é a desnecessidade, no DIMI, de interação obrigatória do novo consumidor com a raiz e seus filhos diretos, liberando-os de uma possível sobrecarga caso muitos novos consumidores desejem entrar em um canal simultaneamente. Os detalhes do protocolo de entrada em um canal no HMTP serão discutidos na seção 5.7.

O protocolo de entrada em um canal no DIMI é mostrado na figura 3.10. Nele, uma UD *X* começa a entrada em um canal através da UD *Y* cujo nome é referenciado no nome do canal passado como parâmetro ao método *enterChannel*. *X* ajusta *Y* como seu pai potencial e constrói uma lista com os nomes de *Y* e de todos os pais e filhos de *Y*. Dessa lista, *X* obtém a UD mais próxima de si, *Z*. Se *Z* for diferente de *Y*, *X* ajusta *Z*

como seu pai potencial e repete recursivamente o algoritmo até encontrar a UD apropriada. Caso contrário, se *Y* for igual a *Z*, *X* tenta entrar no canal como filho de *Y*. *Y* pode ou não aceitar um novo filho no canal. Caso *Y* não aceite o novo filho, *X* marca *Y* como uma UD inválida, retorna um nível e continua o algoritmo recursivo.

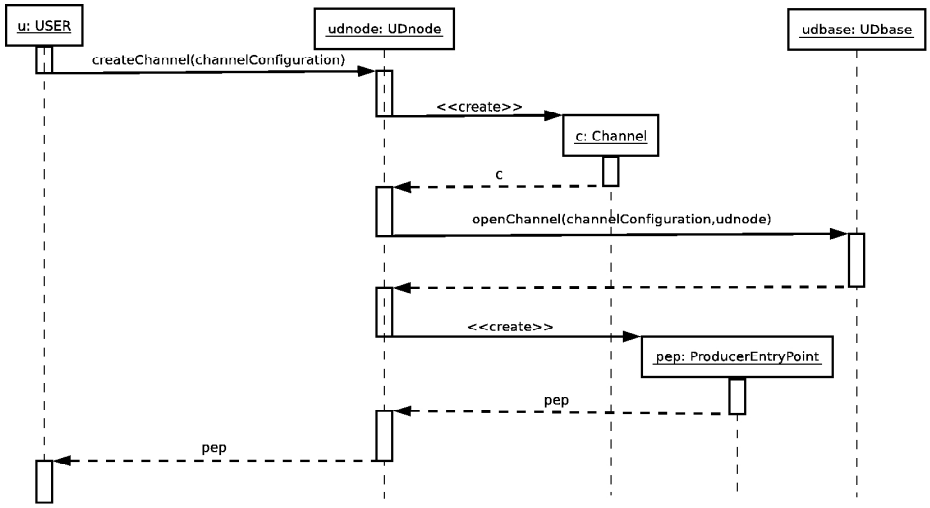

Figura 3.9: Passos envolvidos na entrada em um canal.

A cada iteração do protocolo, as referências das UDs identificadas são armazenadas na LPU do canal em *X*. As referências armazenadas na LPU são utilizadas após a entrada no canal para manter e otimizar a topologia do mesmo. Como o protocolo de entrada pode impor um atraso significante na entrada no canal, caso este possua muitas UDs participantes, é utilizado o mecanismo de paternidade temporária proposta no HMTP (ZGANG; JAMIN; ZHANG, 200). Por esse mecanismo de paternidade temporária, uma UD *X* é adotada temporariamente por outra UD *Y* como filha, até que consiga executar completamente o protocolo de entrada em um canal. Assim, até que *X* estabeleça o melhor ponto de entrada em um canal, ela recebe as mensagens do canal por *Y*. Uma UD pode aceitar ou não um filho temporário de acordo com a sua capacidade no momento da requisição de filiação temporária.

3.7.3 Disseminação de Mensagens em um Canal

A partir do momento em que se criou o canal e se obteve uma instância da classe *ProducerEntryPoint*, é possível disseminar mensagens. O produtor dissemina as mensagens para os consumidores através do método *disseminate* da interface oferecida pela sua instância de *ProducerEntryPoint*, conforme mostrado na figura 3.11. Esse método realiza a entrega da mensagem à sua UD através do método *disseminate*. A transmissão da mensagem do PEP para a sua UD é síncrona, o que significa que o PEP ficará bloqueado até que sua UD receba a mensagem. Todas as retransmissões seguintes, porém, são assíncronas, sendo feitas nos momentos mais adequados a cada uma das UDs do canal. Dessa forma, não se garante que a mensagem seja retransmitida imediatamente após sua recepção em cada uma das UDs.

A disseminação de mensagens do produtor para os consumidores segue o seguinte fluxo: **produtor −> PEP −> [UD]+ −> PEC −>** consumidor. Isso significa que uma mensagem do produtor passa por seu PEP no canal. Do PEP, a mensagem é retransmitida por quantas UDs existirem no canal, sendo finalmente entregue aos consumidores através de seus PECs. A cada entrega de mensagem, o método *onMessage* da interface *MessageListener*, implementado pelo usuário, é executado no dispositivo disseminador onde o PEC está localizado.

1.Uma pilha *S*, inicialmente vazia, é utilizada para armazenar todas as UDs válidas, identificadas durante o algoritmo;

2.Encontre a UD referenciada pelo nome do canal, marque-a como o pai potencial e calcule a sua distância, usando a métrica de distância do canal desejado;

3.Obtenha uma lista com todos os pais e filhos do pai potencial e calcule a distância de todos eles, usando a métrica de distância do canal desejado;

4.Encontre a UD mais próxima entre o pai potencial e todos os seus pais e filhos, exceto aqueles marcados como inválidos. Se todos eles forem inválidos, retire o elemento do topo da pilha *S*, torne-o o novo pai potencial e retorne ao passo 3. Se não houver elementos em *S*, gere uma exceção indicando que não foi possível entrar no canal;

5.Se a UD mais próxima não for o pai potencial, coloque o pai potencial na pilha *S*, ajuste a UD mais próxima como pai potencial e retorne ao passo 3; Caso contrário, tente conectar-se ao pai potencial. Se isso não for possível, marque o pai potencial como inválido e retorne ao passo 4. Se a conexão foi estabelecida, termine.

Figura 3.10: protocolo de entrada em um canal do DIMI.

Ao receber um mensagem, uma UD executa as seguintes ações:

- verifica o tipo da mensagem;
- executa o processamento correspondente ao tipo da mensagem, conforme discutido na seção3.4.1;
- dissemina a mensagem, se for o caso, conforme discutido na seção 3.4.1.

3.7.4 Destruição de um canal

Um canal no DIMI é destruído através da disseminação de uma mensagem do tipo *DESTROY-CHANNEL*. Essa mensagem desaloca as estruturas de dados relativas ao canal nas UDs pelas quais passa. Uma mensagem do tipo *DESTROY-CHANNEL* pode se originar no usuário produtor através da execução do método *destroy* do PEP, ou pode se originar em uma UD que detecte falha na raiz do canal.

3.7.5 Saída de um canal

Existem duas formas de uma UD sair de um canal: por decisão ou por falha. Pela primeira opção, a UD envia uma mensagem do tipo *LEAVE* para seus pais e filhos. A

cada nova desconexão detectada em um canal, uma UD verifica se ainda possui interesse em participar do mesmo. Caso isso não ocorra, ela sai do canal, propagando este comportamento de verificação e saída para outras UDs. Uma UD possui interesse em um canal quando possui mais de uma conexão no mesmo ou um usuário esteja utilizando o serviço através dela (se houver um PEP ou um PEC conectado a ela).

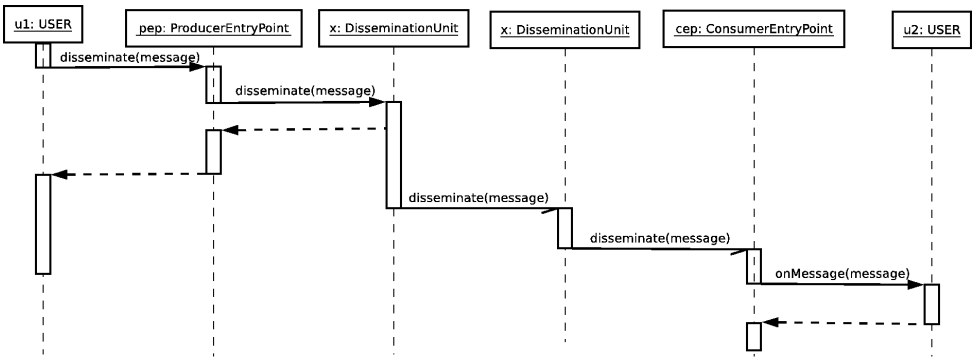

Figura 3.11: Passos envolvidos na disseminação de uma mensagem em um canal.

A segunda maneira de sair de um canal, por falha, acontece quando uma UD pára de enviar mensagens do tipo *HEART*-BEAT para os seus filhos e gera exceções quando seus pais tentam estabelecer alguma comunicação. Nesse caso, as UDs que tiverem a UD falha como pai procuram um novo pai no canal, conforme será discutido na seção 3.7.7.

3.7.6 Detecção, Prevenção Parcial e Ruptura de Ciclos

Um problema que pode ocorrer na topologia de disseminação é a formação de ciclos. Um ciclo é formado quando uma UD recebe uma mensagem mais de uma vez. Isso ocorre quando duas UDs tornam-se ascendentes uma da outra, isto é, quando um dos caminhos da raiz *R* até uma UD *X* do canal contém a UD *Y* e um dos caminhos de *R* até *Y* contém *X*. O DIMI possui mecanismos de detecção e de prevenção parcial de ciclos.

No DIMI, como ocorre no HMTP, todas as mensagens armazenam em seu cabeçalho o caminho pelo qual passam. O caminho de uma mensagem é composto pela lista seqüencial dos nomes das UDs que retransmitiram a mensagem desde a raiz do canal. Ao receber uma mensagem, uma UD verifica se seu nome já existe no caminho da mensagem, isto é, se aquela mensagem já passou por ela. Se a UD encontrar seu nome no caminho, existe um ciclo no canal. Ciclos precisam ser desfeitos, pois desperdiçam recursos computacionais. Ao detectar um ciclo dessa forma, a UD descarta a mensagem e destrói a conexão pela qual a mensagem foi recebida.

Além da detecção de ciclos através da verificação do caminho de uma mensagem, é utilizado um mecanismo de prevenção parcial de ciclos. Esse mecanismo trabalha com o conceito de **Lista Parcial de Ascendencia (LPA)**, que é semelhante ao conceito de *root path* utilizado no HMTP (ZHANG; JAMIN; ZHANG, 2002), no Yoid (FRANCIS, 2000) e no ALMI (PENDARAKIS et al., 2001), porém adaptado para a existência de

múltiplos pais em um canal. Esse mecanismo permite a detecção relativamente rápida da existência certos tipos de ciclos em um canal. Uma LPA é definida da seguinte forma:

- a LPA da raiz é composta pelo nome da própria raiz;

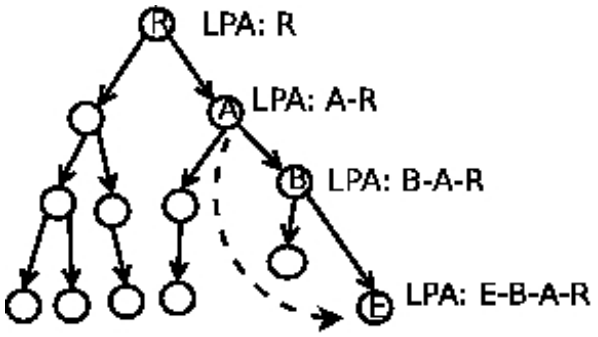

Figura 3.12: Exemplo de ciclo evitado pela prevenção parcial de ciclos.

- a LPA de uma UD *X* é composta pelo nome de *X*, concatenado com a LPA do seu primeiro-pai. O primeiro-pai de uma UD é o seu pai mais próximo, de acordo com a métrica de distância do canal.

O primeiro-pai de uma UD é recalculado periodicamente devido às modificações dinâmicas no estado da rede física subjacente ao serviço e no estado das UDs do canal. Caso o primeiro-pai de uma UD *X* em um canal seja alterado, *X* atualiza sua LPA com base na LPA do novo primeiro-pai e envia uma mensagem do tipo *LPA-CHANGED* para todos os seus filhos. Esta mensagem possui em seu conteúdo a LPA de *X*. Cada filho, ao receber uma mensagem *LPA-CHANGED* de seu primeiro-pai, modifica sua própria LPA e também envia para seus filhos uma mensagem *LPA-CHANGED*, repetindo o processo até que todas as UDs que tiveram sua LPA alterada tenham feito as modificações necessárias. Caso uma UD receba uma mensagem *LPA-CHANGED* de uma UD que não seja seu primeiro-pai, a mensagem é descartada.

O mecanismo de prevenção parcial de ciclos, ilustrado na figura 3.12, funciona da seguinte forma: antes de uma UD *A* aceitar um novo primeiro-pai *B* no canal, ela verifica a LPA de *B* no canal. Caso a LPA de *B* contenha o nome de *A*, este desiste de entrar no canal como filho de *B* e procura um outro primeiro-pai no canal. Esse mecanismo não previne completamente a formação de ciclos, devido à natureza distribuída da criação de conexões em um canal, mas serve como uma verificação eficiente que pode evitar que alguns tipos de ciclos ocorram (FRANCIS, 2000).

3.7.7 Ajustes Dinâmicos na Topologia para Aumentar o Desempenho da Disseminação

No DIMI, a topologia de disseminação adapta-se às modificações no estado da rede física subjacente ao serviço e à dinâmica de entrada e saída dos usuários do serviço, em tempo de execução. Para isso, uma UD *X* apresenta o seguinte comportamento:

- periodicamente, *X* calcula a sua distância até uma UD aleatória *Y* do canal, obtida da sua LPU. Caso a distância de *X* até *Y* seja menor que a distância de *X* até o seu primeiro-pai (o pai mais próximo) *Z*, *X* desconecta-se de *Z* e conecta-se em *Y* no canal;
- periodicamente, *X* descobre novas UDs do canal por meio da comunicação com as UDs referenciadas na LPU do canal. Essas UDs recentemente descobertas são adicionadas à LPU do canal;
- para substituir um pai falho que não seja seu primeiro-pai ou para encontrar um novo pai no canal, *X* escolhe uma UD aleatória da LPU do canal, e torna-se sua filha, sem executar o protocolo de entrada descrito na seção 3.7.2;
- Para substituir seu primeiro-pai em caso de falha deste, *X* escolhe a UD mais próxima existente em sua LPU, e executa, a partir dela, o protocolo de entrada descrito na seção 3.7.2.

3.8 O DIMI no ISAM*pe*

> O requisito do middleware de manter-se operacional durante os períodos de desconexão planejada motivou, além da concepção de primitivas de comunicação adequadas a essa situação, a separação dos serviços que implementam operações de natureza distribuída em intâncias locais ao *EXEHDAnodo* (instância nodal), e instâncias locais a *EXEHDAbase* (instância celular). Nesse sentido, o relacionamento entre instância de nodo e celular assemelha-se à estratégia de proxies, enquanto que o relacionamento entre instâncias celulares assemelha-se à estratégia P2P. A abordagem P2P nas operações inter-celulares vai ao encontro do requisito de escalabilidade.

O trecho acima, obtido da tese de doutorado que modelou o EXEHDA (YAMIN, 2004), mostra a necessidade de localização dos serviços distribuídos do EXEHDA tanto nos nós de processamento quanto nas bases das células do ISAM*pe*. Para que o DIMI possa tornar-se um serviço do EXEHDA, é necessário, portanto, realizar algumas adequações em seu modelo P2P a fim de contemplar também a abordagem de *proxies* dos relacionamentos entre os nós de processamento e a base da célula. Nesta seção são discutidas as adequações do modelo do serviço DIMI para permitir a participação dos nós de processamento do ISAM*pe* no processo de disseminação. Também é discutida a integração do serviço DIMI com o serviço de descoberta de recursos do ISAM*pe*, o PerDis (SCHAEFFER FILHO et al., 2004).

3.8.1 UDbases e UDnodes

Devido à característica estrutural do ISAM*pe* de realizar a comunicação entre células através de suas respectivas bases, a UD foi especializada, conforme a figura 3.14, em dois tipos: a UD da célula, denominada **UDbase**, e a UD do nó de processamento, denominada **UDnode**. Uma UDbase é executada em uma EXEHDAbase e é responsável pela interação com outras células no processo de disseminação, segundo a estratégia P2P. O nome de uma UDbase segue a formação de nomes definida na seção 3.4.3.1. Uma UDnode é executada em um EXEHDAnode e interage com a UDbase da célula à que pertence, conforme ilustrado na figura 3.13. O nome de uma UDnode é representado por uma URL que contém o nome da sua UDbase

concatenado ao seu endereço IP e à sua porta de trabalho, no seguinte formato: **dimi://nome_da_UDbase/IP:porta**.

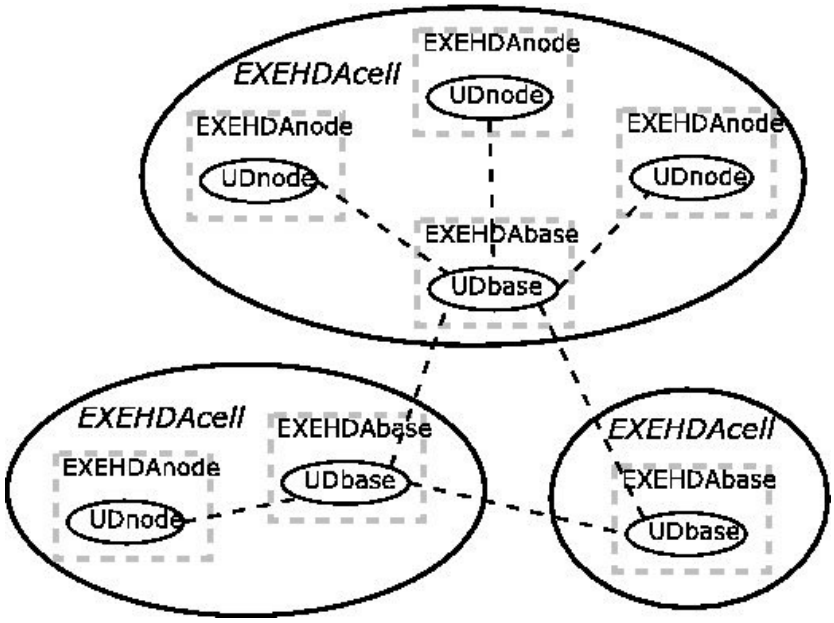

Figura 3.13: UDs do DIMI no ISAMpe.

Devido ao seu papel de *proxy* para as UDnodes, conforme ilustrado na figura 3.14, foram adicionados à UDbase três métodos em relação aos existentes na interface de *DisseminationUnit*, os métodos *enterChannel*, *openChannel* e *getChannelRootName*. O método *openChannel* tem a função de tornar acessível às outras células um canal criado por uma UDnode de sua célula. Os parâmetros desse método são o nome do novo canal (parâmetro *channelName*), suas configurações (parâmetro *channelConfiguration*) e uma referência para a UDnode raiz do canal (parâmetro *udnode*). O método *enterChannel* da UDbase é polimórfico ao método de mesmo nome de *DisseminationUnit*. Esse método serve para permitir a entrada de uma UDnode em um canal, através da UDbase responsável por essa UDnode. O método *getChannelRootName*, por sua vez, é necessário para a integração do DIMI com o serviço de descoberta de recursos da arquitetura ISAM, conforme será discutido na seção 3.8.5.

3.8.2 Funcionamento

3.8.2.1 Criação de um Canal e Início de Produção em uma UDnode

A criação de um canal em uma UDnode, ilustrada na figura 3.15, é realizada pela chamada do método *createChannel* da interface *DisseminationService* da UDnode. Esse método cria um canal e, através do método *openChannel*, pede à sua UDbase que o torne disponível para o acesso externo. Caso a UDbase consiga tornar o canal disponível para acesso externo, a UDnode cria um PEP para o canal e o retorna ao usuário. Caso contrário, é lançada para o usuário uma exceção do tipo *CantCreateChannelException*, conforme foi discutido na seção 3.5.1.

3.8.2.2 Entrada em um canal e Início de Consumo em uma UDnode

A entrada de uma UDnode em um canal, ilustrada na figura 3.1.6, é realizada pela chamada do método *enterChannel* da interface *DisseminationService* da UDnode. Esse método chama o método *enterChannel* da UDbase da célula. A UDbase executa então o protocolo de entrada em um canal, discutido na seção 3.7.2, utilizando como ponto inicial a UDbase da célula da UD referenciada no nome do canal. Após a execução com sucesso do protocolo de entrada e do conseqüente encontro de um ponto de entrada no canal desejado, a UDbase da célula cria uma instância de *channel* com um nome globalmente único e retorna o controle à UDnode que, por sua vez, instancia um objeto *channel* com um nome globalmente único, de acordo com o que foi descrito na seção 3.4.3.1 e retorna o controle ao usuário. A partir desse momento, o usuário está apto para receber as mensagens do canal.

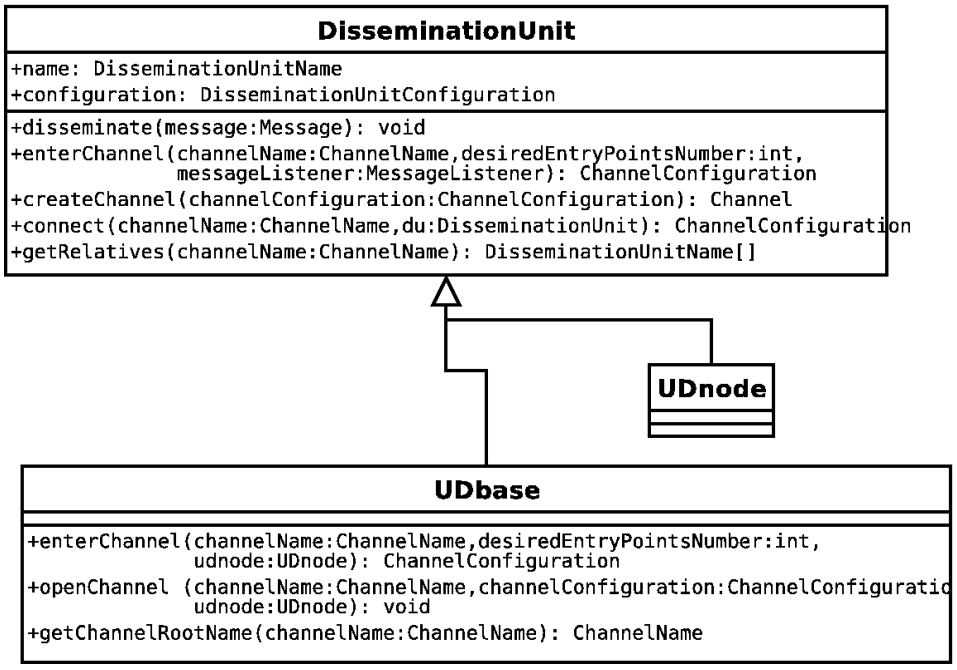

Figura 3.14: Especialização da classe *DisseminationUnit* em UDbase e UDnode.

3.8.2.3 Disseminação de Mensagens por um Canal

Toda disseminação no ISAM*pe* segue o seguinte fluxo: **produtor –> PEP [–> UDnode] –> [UDbase]+ [–> UDnode] –> PEC–> consumidor**. Isso significa que uma mensagem do produtor passa por seu PEP no canal, conectado a uma UDnode ou a uma UDbase. Da UDnode ou da UDbase a que o PEP está conectado, a mensagem é retransmitida por quantas UDbases existirem no canal. A mensagem é finalmente entregue aos usuários através de seus PECs, localizados em uma UDnode ou em um UDbase.

3.8.3 Suporte à Desconexão Planejada de UDnodes

O suporte à desconexão planejada é oferecido somente às UDnodes, porque as UDbases são executadas em EXEHDAbases e, por isso, sua conectividade é constante. Para isso, é utilizada a *bufferização* de mensagens. Ao receber uma notificação de desconexão de rede do serviço de contexto do EXEHDA, uma UDnode *X* avisa a UDbase *Y* de sua célula que ficará desconectada temporariamente. A partir deste momento, todas as mensagens relativas a *X* são armazenadas nos *buffers* de *Y*, se *X* for uma consumidora, ou nos *buffers* de *X*, se *X* for a produtora do canal. Ao receber uma notificação de reconexão de rede do serviço de contexto do EXEHDA, *X* avisa *Y* que reconectou-se à rede. A partir deste momento, as mensagens *bufferizadas* relativas àquela desconexão são entregues, obedecendo o critério de prioridade de canais.

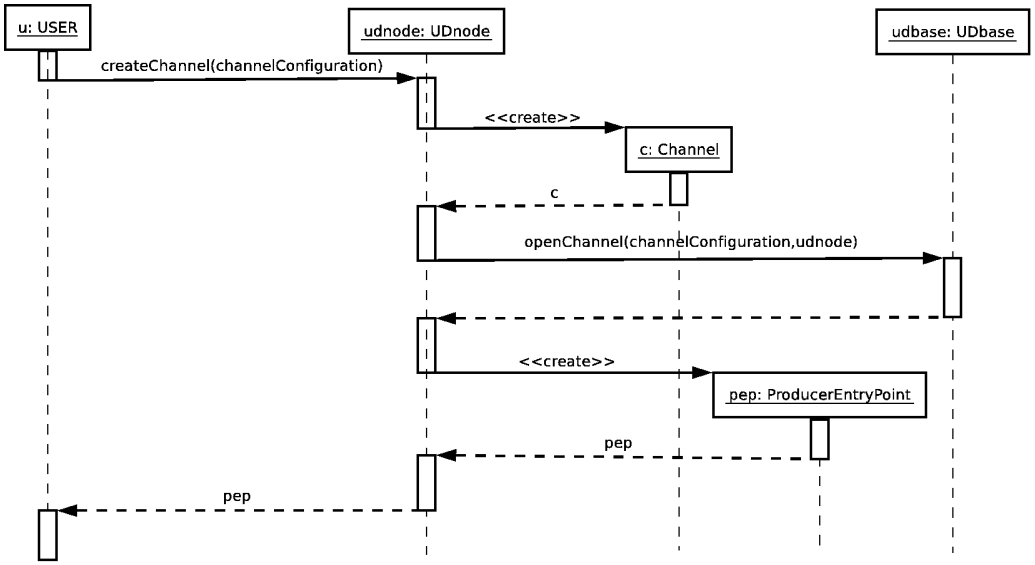

Figura 3.15: Passos envolvidos na criação de um canal no ISAM*pe*.

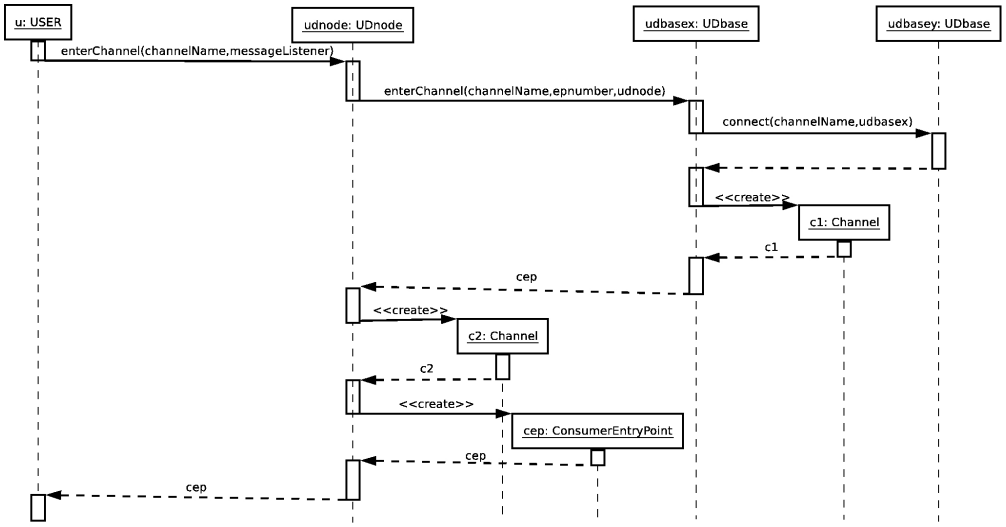

Figura 3.16: Passos envolvidos na entrada em um canal no ISAM*pe*

A simples utilização de *buffers*, no entanto, é utilizada em outros mecanismos de disseminação de mensagens como, por exemplo, o JMS (HAPNER et al., 2002), mas não representa uma solução a ser utilizada em dispositivos disseminadores que apresentem limitações de espaço em mémoria. O DIMI apresenta uma solução flexível para os dispositivos com limitações de memória. O comportamento do serviço em relação ao estouro do *buffer* da UDnode produtora ocorrido nos momentos desconexão planejada é definido pelo usuário produtor, no momento da criação do canal, através do parâmetro *PRODUCER-BUFFER-OVERFLOW-BEHAVIOR* (figura 3.4).

Quando houver estouro de *buffer* em uma UDnode produtora *Z* devido à produção de mensagens em períodos de desconexão planejada, o serviço pode se comportar de três formas distintas:

- *Z* descarta silenciosamente as mensagens sendo enviadas. Esse comportamento é indicado pelo valor *discard* do parâmetro *PRODUCER-BUFFER-OVERFLOW-BEHAVIOR* da configuração do canal, conforme discutido na seção 3.4.3;
- *Z* gera uma exceção ao usuário, avisando que a produção da mensagem causou um estouro de *buffer*. Esse comportamento é indicado pelo valor *throw-exception* do parâmetro *PRODUCER-BUFFER-OVERFLOW-BEHAVIOR*;
- *Z* bloqueia a chamada de disseminação, até que a conexão seja reestabelecida e todas as mensagens do *buffer* sejam enviadas, quando então o controle retorna ao produtor. Esse comportamento é indicado pelo valor *block* do parâmetro *PRODUCER-BUFFER-OVERFLOW-BEHAVIOR.*

3.8.4 Suporte à Mobilidade de UDnodes

O suporte à mobilidade é oferecido somente às UDnodes, porque as UDbases são executadas em EXEHDAbases e, por isso, não são móveis. Isso é feito através do uso do serviço de contexto do EXEHDA, que notifica a UDnode toda vez que seu

respectivo EXEHDAnode for conectado a uma nova célula. Quando a UDnode móvel for uma consumidora *C*, esta continua recebendo as mensagens do canal através de sua antiga UDbase *A* até que a sua nova UDbase *N* entre no canal e, após a entrada de *N*, até que *A* e *N* entreguem a mesma mensagem, o que pode ser verificado pela comparação do campo *SEQUENCE-NUMBER* do cabeçalho das mensagens. Dessa forma, é possível garantir que *C* não perderá mensagens devido à movimentação entre as células.

Quando a UDnode móvel for uma produtora, é utilizado um mecanismo semelhante ao utilizado no HMTP (ZHANG; JAMIN; ZGABG, 2002) e no TAO (KWON; FAHMY, 2002), baseado na transmissão *unicast* da nova UDbase do produtor para a UDbase à qual o produtor estava conectado no momento da criação do canal, onde é feita a disseminação. A figura 3.17 exemplifica o funcionamento desse mecanismo. No exemplo, a UDnode produtora, representada pelo retângulo, movimenta-se da UDbase raiz do canal *A* para a UDbase *B*. A disseminação é feita então da seguinte forma: *B* repassa as mensagens do produtor diretamente para *A*, que realiza a disseminação normalmente, como se o produtor continuasse conectado a ele. Segundo Kwon (KWON; FAHMY, 2002), esse mecanismo é viável em termos de desempenho para grande parte das aplicações de produtor único.

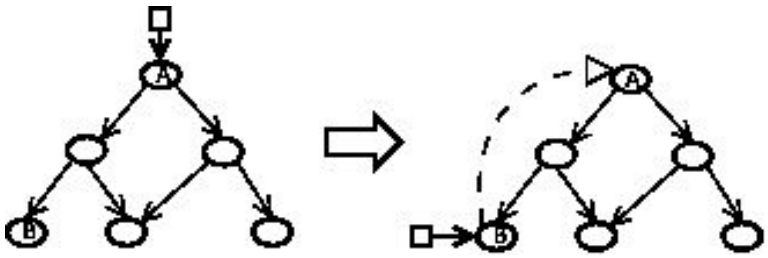

Figura 3.17: Mobilidade de uma UDnode no ISAM*pe*.

3.8.5 Integração com o Serviço de Descoberta de Recursos

No ISAM*pe*, os usuários obtêm um nome do canal desejado através do serviço de descoberta do EXEHDA, o PerDiS (*Pervasive Discovery Service)* (SCHAEFFER et al., 2004. O PerDiS permite a localização de recursos computacionais ou serviços presentes no ISAM*pe* através da definição de uma especificação de pesquisa. Uma especificação de pesquisa que busque a localização de um canal do DIMI permite que um usuário encontre os nomes do canal desejado que referenciam as UDs mais próximas para iniciar o consumo.

A localização das UDs do canal que estão próximas do novo consumidor é importante para diminuir o tempo de execução do protocolo de entrada, discutido na seção 3.7.2, e melhorar o desempenho de disseminação, conforme será discutido na seção 4.1. A figura 3.18 mostra um exemplo de canal, com as suas UDbases participantes. Nesse exemplo, um novo consumidor localizado na célula da UDbase *N*, disparando o protocolo de entrada, a partir da UDbase *I*, descobre que o melhor ponto de entrada para ele no canal é a UDbase *F*. Integrado com o Perdis, o DIMI pode, antes

de iniciar a execução do protocolo de entrada, realizar a busca por UDbases vizinhas[3] de *N* que já participem do canal. Dessa forma, é possível que *N* obtenha uma referência para *F* de forma mais rápida através do PerDiS do que pela execução direta do protocolo de entrada.

Por outro lado, é possível que uma consulta feita no PerDiS demore muito para ser executada. Por isso, é importante que as consultas tenham limites de tempo pequenos para que não causem o efeito contrário ao desejado e acabem aumentando o tempo de entrada em um canal. O PerDiS permite a realização de consultas criterizadas pelo grau de vizinhança[4] das células do ISAM*pe* e por um limite de tempo de execução da pesquisa. Assim, usando o PerDiS, é possível buscar todas as UDbases existentes em um canal dentro de um escopo de vizinhança de grau *n* e com tempo máximo de pesquisa *t*, onde *n* e *t* são parâmetros de consulta.

Por meio do acréscimo do método *enterChannelUsingPerdis* à interface *DisseminationService* (discutida na seção 3.5), conforme mostra a figura 3.19, pode-se parametrizar e utilizar o PerDiS através do DIMI. Foi adicionado à interface *DisseminationService* o método *enterChannelUsingPerdis*, implementado pela classe *DisseminationUnit*, que realiza a entrada da UD em um canal utilizando o PerDiS para descobrir a UDbase mais próxima para iniciar o protocolo de entrada, discutido na seção 3.7.2. O método *enterChannelUsingPerdis* recebe, além dos parâmetros do método *enterChannel*, discutidos na seção 3.4.2, o grau de vizinhança (parâmetro *degree*) e o tempo máximo de execução da busca (parâmetro *timeout*) do PerDiS.

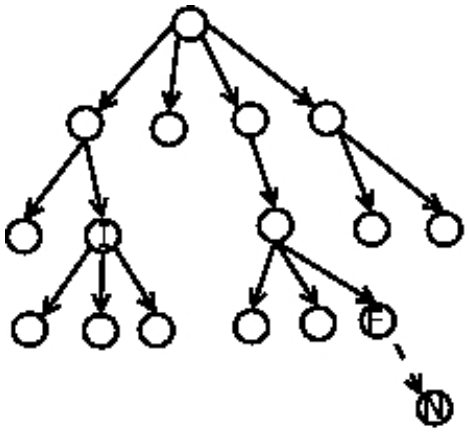

Figura 3.18: Canal do DIMI no ISAM*pe*. Os círculos representam UDbases.

[3] A vizinhança de uma célula *X* é o conjunto de células que *X* conhece no ISAM*pe*.

[4] O grau de vizinhança de uma célula *X* em relação a uma célula *Y* é o número de células existentes entre *X* e *Y*. Assim, os vizinhos de *X* com grau de vizinhança 0 são os vizinhos diretos de *X*, os vizinhos de *X* com grau de vizinhança 1 são os vizinhos dos vizinhos diretos de *X*, e assim por diante.

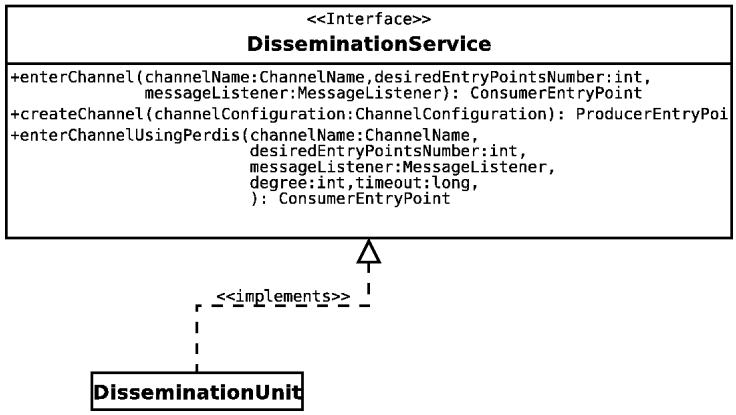

Figura 3.19: Interface DisseminationService integrada com o PerDis.

3.8.5.1 Anunciação das UDbases

Para que o PerDiS possa encontrar recursos computacionais no ISAM*pe*, é necessário que esses recursos se anunciem ao PerDiS, informando-lhe suas características e suas disponibilidades. Das UDs do DIMI, somente as UDbases precisam se anunciar ao PerDiS, pois elas são as responsáveis pela disseminação intercelular. A anunciação das UDbases ao PerDiS ocorre logo após a sua entrada no canal, com uso de arquivos XML. Um exemplo de arquivo XML de anunciação de uma UDbase em um canal pode ser visto na figura 3.20. Na referida figura, é informada a existência de uma UDbase participante de um canal com a descrição *A DUbase on a channel in the ISAMpe*, o tipo *DUBASE-ON-A-CHANNEL*, o intervalo de notificações de disponibilidade (*tag LEASE*)[5] de 60000 milissegundos e a URL *dimi://andrews.inf.ufrgs.br:40000/channel3*. O atributo *value* da tag PROPERTY com valor de atributo *root-url* indica o nome do canal na UDbase raiz, no caso do exemplo, *dimi://saloon.inf.ufrgs.br:40000/coisa.inf.ufrgs.br:40000/channel5*.

```
<RESOURCE>
 <DESCRIPTION>A DUbase on a channel in the ISAMpe</DESCRIPTION>
 <TYPE>DUBASE-ON-A-CHANNEL</TYPE>
 <LEASE>60000</LEASE>
 <URL>dimi://andrews.inf.ufrgs.br:40000/channel3</URL>
 <PROPERTIES>
 <PROPERTY name="root-url"
value="dimi://saloon.inf.ufrgs.br:40000/coisa.inf.ufrgs.br:40000/chann
el5" />
 </PROPERTIES>
</RESOURCE>
```

Figura 3.20: Arquivo XML de anunciação de uma UDbase em um canal no PerDis.

[5] O intervalo de notificação de disponibilidade é o período de tempo que um recurso tem para avisar novamente o serviço de descoberta que continua disponível no ambiente de execução.

```
<QUERY>
 <TTL>1</TTL>
 <TYPE>DUBASE-ON-A-CHANNEL</TYPE>
 <TIMEOUT>1000</TIMEOUT>
 <CRITERIA name="root-url"
value="dimi://saloon.inf.ufrgs.br:40000/coisa.inf.ufrgs.br:40000/chann
el5" />
</QUERY>
```

Figura 3.21: Exemplo de um arquivo que define uma consulta por UDbase no PerDis.

3.8.5.2 Realização de Consultas

Os recursos do ISAM*pe* podem, após a sua anunciação ao PerDiS, ter seus endereços entregues aos usuários. Para que um endereço de recurso possa ser entregue a um usuário, é necessário que o recurso satisfaça os critérios de busca definidos pelo usuário no momento da consulta ao PerDiS.

A consulta por UDbases participantes de um canal no PerDiS também é feita com o uso de arquivos XML. Um exemplo de uma consulta por UDbases participantes de um canal pode ser visto na figura 3.21. O exemplo mostrado na figura informa ao PerDiS que se busca obter o maior número de recursos do tipo *DUBASE-ON-A-CHANNEL* (*tag TYPE*) com grau de vizinhança 1 no ISAM*pe* (*tag TTL*) e que participem do canal cujo endereço na UD raiz seja *dimi://saloon.inf.ufrgs.br:40000/coisa.inf.ufrgs.br:40000/channel5* (*tag CRITERIA*), desde que o tempo de pesquisa não ultrapasse 1000 milissegundos (*tag TIMEOUT*).

3.8.5.3 Entrada em um Canal

Conforme foi discutido anteriormente e está ilustrado no diagrama de seqüência da figura 3.22, um usuário inicia o consumo em um canal do DIMI utilizando o PerDiS através da execução do método *enterChannelUsingPerdis*. Como a consulta pelas UDbases vizinhas participantes do canal é feita com base no endereço do canal em sua UD raiz, é necessário que, antes da utilização do PerDiS, o nome do canal na sua UD raiz seja descoberto. Isso é feito através da execução do método *getChannelRootName* da UDbase referenciada pelo parâmetro *channelName* do método *enterChannelUsingPerdis*.

Após a descoberta do nome do canal em sua raiz, um arquivo XML de consulta semelhante ao da figura 3.21 é gerado automaticamente com os valores referentes àquela consulta: as *tags TTL* e *TIMEOUT* recebem, respectivamente, os valores dos parâmetros *degree* e *timeout* do método *enterChannelUsingPerdis*. O atributo *value* da *tag CRITERIA* recebe o nome do canal em sua raiz obtido com a execução do método *getChannelRootName*.

O serviço PerDiS é parametrizado com o arquivo XML de consulta construído automaticamente. Se o PerDiS retornar alguma referência para uma ou mais UDbases do canal, a mais próxima dessas UDbases é escolhida como ponto inicial para a execução do protocolo de entrada discutido na seção 3.7.2. Caso contrário, se o PerDiS não retornar nenhuma referência para UDbases do canal, é utilizada como ponto de inicial para o protocolo de entrada a UDbase referenciada pelo nome do canal contido

no parâmetro *channelName* do método *enterChannelUsingPerdis*. Após isso, o comportamento do DIMI ocorre como foi descrito na seção 3.7.2.

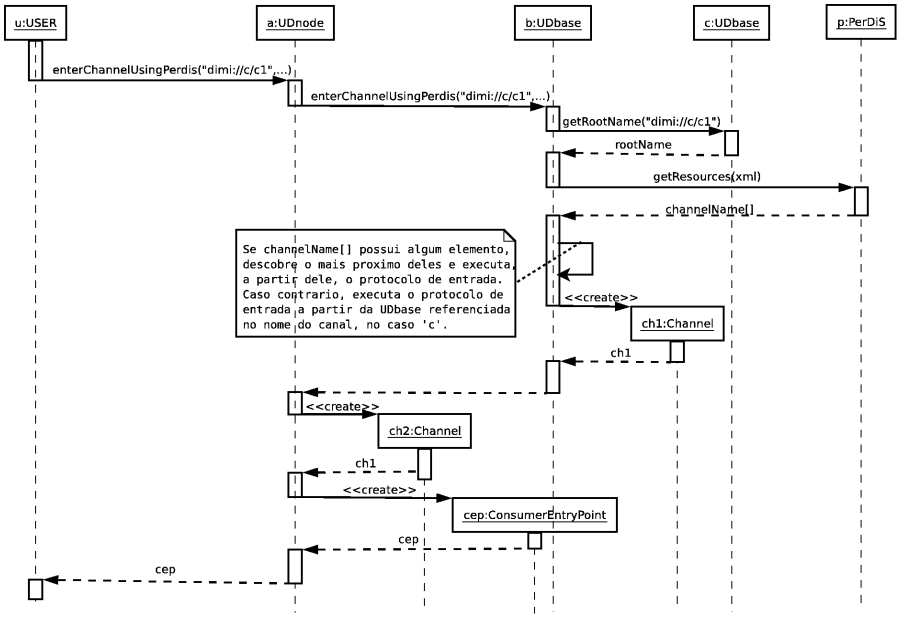

Figura 3.22: Passos de entrada em um canal utilizando o PerDis através do DIMI.

4 RESULTADOS

Este capítulo discute os resultados do DIMI e os métodos utilizados para a obtenção dos mesmos: a simulação e a prototipação. O objetivo deste capítulo é validar os argumentos de construção do DIMI, discutidos no capítulo anterior.

4.1 Simulação

Foi realizada a comparação entre o DIMI e o HMTP (ZHANG; JAMIN; XHANG, 2002) através de simulação para que pudessem ser medidas as diferenças de escalabilidade e desempenho de disseminação entre essas duas arquiteturas de disseminação MNA. O protocolo de entrada proposto no DIMI é inspirado no funcionamento do HMTP. principal diferença entre eles é que, no HMTP, um novo participante inicia a entrada em um canal a partir da raiz, descendo no canal pelos participantes que forem mais próximos de si, até encontrar um ponto de entrada adequado em termos de distância. No DIMI, um novo participante inicia a entrada a partir de um participante qualquer do canal, e se desloca para cima e para baixo no canal pelos participantes que forem mais próximos de si, até encontrar um ponto de entrada adequado em termos de distância. O HMTP será discutido na seção 5.7.

A simulação do serviço DIMI em uma rede com centenas nós é justifica pela inviabilidade de executar o protótipo em uma rede de tamanha grandeza. O protótipo do DIMI será discutido na seção 4.2.

4.1.1 Ferramentas Utilizadas

A ferramenta escolhida para realizar a simulação foi o simulador de redes Network Simultator-2 (NS-2) (NS-2, 2005). A utilização de outros simuladores de sistemas distribuídos em geral, como o MONARC-2 (MONARC-2, 2005) e o Neko (URBÁN; DÉFAGO; SCHIPER, 2001), foi cogitada. Esses simuladores, no entanto, não oferecem abstrações diretas para o envio de pacotes. Eles também não oferecem abstrações diretas para a definição de topologias de rede e para a medição das características dinâmicas das redes, como, por exemplo, a latência das transmissões. Essas limitações tornam a simulação do serviço DIMI mais fácil no NS-2.

No ambiente de simulação NS-2, é possível criar uma rede arbitrária de nós capazes de comunicarem-se entre si através de protocolos amplamente conhecidos, como o TCP (TPC, 2005), assim como através de protocolos criados e implementados pelos usuários. As linguagens utilizadas para a realização da simulação do DIMI no NS-2 foram a linguagem OTcl (WETHERALL, 1995) e a linguagem C++ (WALLACE, 1994). A linguagem OTcl, que é a versão orientada a objetos da linguagem interpretada Tcl (OUSTERHOUT, 1993), foi utilizada na criação e configuração da rede simulada, assim

como na implementação dos protocolos de entrada em um canal e disseminação. A linguagem C++, por sua vez, foi utilizada na definição das mensagens disseminadas na rede simulada e no tratamento das mensagens nos participantes do canal.

Para criar a topologia de rede utilizada na simulação, foi utilizado o gerador de topologias BRITE (MEDINA et al., 2001). Tentou-se também utilizar o gerador de topologias INET (INET, 2005), mas seu uso não se mostrou possível. O INET apresenta uma limitação rígida no número mínimo de nós da rede gerada em aproximadamente 3000 nós. Isso impossibilitou a rede gerada de ser utilizada no simulador NS-2, por motivos desconhecidos. Ao tentar-se realizar a simulação com uma rede de 3000 nós, a execução do NS-2 era abortada com a seguinte mensagem de erro: "*Stack error*".

4.1.2 Metodologia

A topologia de rede utilizada na simulação foi um grafo aleatório não-particionado *R* de 500 nós, gerado pelo método de Waxman (WAXMAN, 1988), com os parâmetros *default* fornecidos pelo gerador BRITE. Nesse grafo, as ligações de rede possuiam largura de banda uniformemente distribuída entre 10 MB/s e 1024 MB/s e latência uniformemente distribuída entre 0,1 e 4,0 milisegundos.

A utilização de 500 nós de rede é arbitrária e, como tal, pode dar margem a, no mínimo, dois questionamentos. O primeiro questionamento é a respeito da variação dos resultados obtidos e mostrados na próxima seção caso o número de nós da rede simulada seja diferente. A variação do número de nós da rede não deve afetar o comportamento relativo do DIMI e do HMTP. É esperado que o comportamento observado, caso o número de nós da rede varie, seja o mesmo dos resultados apresentados na próxima seção, que mostram que quanto maior o número de participantes de um canal, maior é a diferença de desempenho entre as propostas. Esse comportamento é justificado pelos argumentos discutidos na próxima seção.

O segundo questionamento a respeito dos resultados é se é válido simular uma disseminação em um ambiente distribuído em escala global como o ISAM*pe* com apenas algumas centenas de participantes. Como este trabalho está inserido no projeto ISAM, é possível alegar que cada um dos participantes da disseminação simulada representa uma EXEHDAbase, que tem sob o seu gerenciamento um número razoável de EXEHDAnodes (possivelmente dezenas), e que, por isso, a simulação estaria expressando o comportamento do DIMI e do HMTP com dezenas de centenas de participantes.

Cada resultado foi obtido pela média de 20 execuções, tanto do DIMI quanto do HMTP. A cada execução, um número arbitrário de participantes do serviço, passado como parâmetro, foi disposto aleatoriamente sobre *R*, que permaneceu inalterado em todas as execuções. Este comportamento, ilustrado na figura 4.1, representa as inúmeras execuções de uma aplicação qualquer executada no DIMI e no HMTP sobre a infra-estrutura da Internet. Na figura 4.1, vê-se duas instâncias da execução do serviço (DIMI ou HMTP) sobre uma infra-estrutura de rede com 9 nós em que um produtor (*p*) e dois consumidores (*c1* e *c2*) são dispostos aleatoriamente. Em cada uma das possíveis execuções sobre a infra-estrutura, as medições das características do serviço são potencialmente diferentes porque podem ser diferentes as ligações de rede que compõem as conexões entre os participantes do serviço. A escolha da quantidade 20 para o número de execuções foi empírica, baseada em algumas execuções feitas

arbitrariamente, em que se notou que a média a partir de 15 repetições já não apresentava diferenças significativas nos resultados.

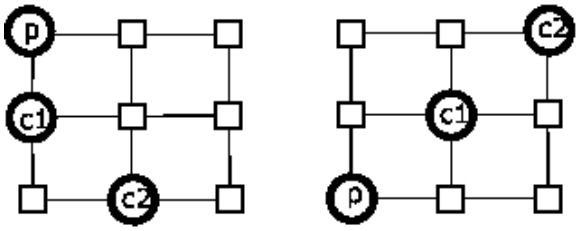

Figura 4.1: Instâncias do serviço com um produtor (*p*) e dois consumidores (*c1* e *c2*).

Em qualquer das execuções havia apenas um produtor e, consequentemente, apenas um canal. Cada participante do canal possuía apenas um pai e um limite máximo de 4 filhos. O intervalo de produção de novas mensagens utilizado em todas as execuções da simulação foi de 0,01 segundo. As mensagens disseminadas tiveram o tamanho arbitrário de 10 Kbytes. Cada execução da simulação era terminada 3 segundos após a entrada do último participante no canal, o que fez com que o número de mensagens em um canal variasse a cada execução. No entanto, sabe-se, pelos dados fornecidos, que, após a entrada de todos as participantes no canal, o número máximo de mensagens disseminadas no mesmo não ultrapassou 300, que é o número de mensagens que um produtor que produz uma mensagem a cada 0,01 segundo pode produzir em 3 segundos. Semelhantemente à forma como foi escolhido o número de execuções da simulação, o espaço de tempo de 3 segundos após a entrada de todos os participantes no canal foi escolhido empiricamente, pois se observou que essa quantia era suficiente para que a medida do tempo de disseminação fosse realizada.

4.1.3 Resultados Obtidos

Esta seção mostra os resultados obtidos com a comparação de simulação do DIMI e do HMTP. Os critérios usados para a comparação foram o desempenho de disseminação e a escalabilidade relativa ao número de novos participantes simultâneos.

4.1.3.1 Tempo de Disseminação

O primeiro resultado da simulação, mostrado na figura 4.4, demonstra a diferença do tempo de disseminação entre o DIMI e o HMTP. O DIMI apresenta pior desempenho que o HMTP porque a simulação não leva em conta a manutenção do canal, discutida na seção 3.7.7, e escolhe participantes completamente aleatórios como ponto inicial para execução do protocolo de entrada. Assim, a topologia de um canal no HMTP ficou mais próxima à árvore mínima de disseminação do que a topologia de um canal do serviço DIMI. Tal comportamento é explicado com auxílio das figuras 4.2 e 4.3, onde é mostrada a topologia de um canal hipotético que assume a distância geográfica como métrica de distância. O canal do exemplo possui sete participantes localizados em sete capitais brasileiras. A figura 4.2 mostra onde um novo participante localizado em São Paulo entraria através do serviço DIMI, caso o protocolo de entrada utilizasse o participante localizado em Aracaju como ponto inicial. A figura 4.3 mostra onde este

mesmo novo participante entraria no canal através do HMTP, que concentra todas as requisições de novos participantes na raiz do canal.

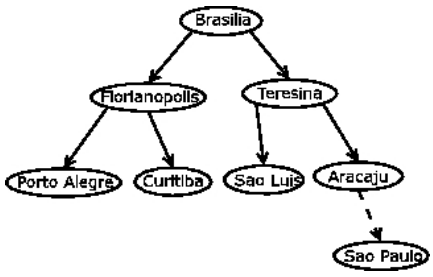

Figura 4.2: Exemplo de topologia de disseminação do DIMI.

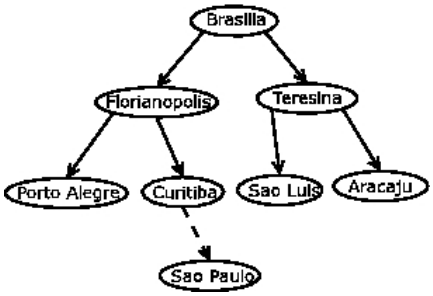

Figura 4.3: Exemplo de topologia de disseminação do HMTP.

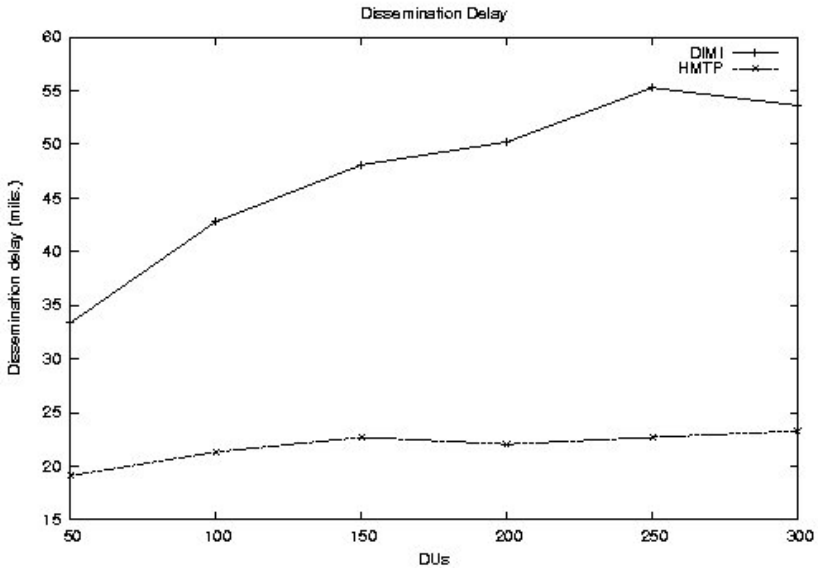

Figura 4.4: Comparação dos tempos de disseminação no DIMI e no HMTP.

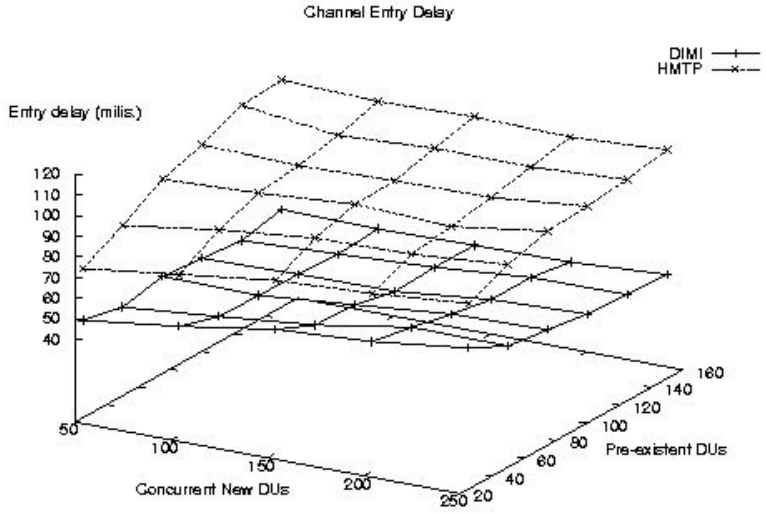

Figura 4.5: Comparação dos tempos de entrada de novos participantes concorrentes.

4.1.3.2 Tempo de Entrada em um Canal

O segundo resultado da simulação, mostrado na figura 4.5, mostra a diferença do tempo de entrada em um canal entre o DIMI e o HMTP. Os valores foram obtidos levando em consideração o número de participantes pré-existentes no canal e o número de participantes que desejavam entrar concorrentemente no canal. O DIMI apresentou melhor desempenho que o HMTP porque, no DIMI, o processo de entrada no canal era iniciado em participantes aleatórios da rede, enquanto que, no HMTP, este processo era iniciado única e obrigatoriamente na raiz do canal.

4.1.3.3 Considerações a Respeito dos Resultados

Os dois resultados mostram que o DIMI atinge maior escalabilidade que o HMTP quando novos consumidores desejam entrar simultaneamente em um canal, apresentando o custo da perda de desempenho na disseminação de mensagens no canal. Espera-se, entretanto, que, com a aplicação do protocolo de manutenção de canal e com a integração do DIMI com o serviço de descoberta de recursos da arquitetura ISAM, esta perda seja minimizada, tornando o desempenho de disseminação do DIMI equivalente ao desempenho do HMTP.

4.2 Prototipação

Assim como ocorreu com os outros serviços do EXEHDA, o protótipo do DIMI foi implementado com a linguagem Java (SUN, 2005a). Para realizar a comunicação entre as UDs, foi utilizado o Java/RMI (*Java/Remote Method Invocation*) (SUN, 2005b). O serviço JMS foi cogitado para realizar a comunicação entre as UDs no lugar do Java/RMI, por ser um serviço de troca de mensagens maduro e amplamente conhecido (HAPNER et al., 2002). No entanto, a complexidade de implantação de uma aplicação baseada no JMS, devido à necessidade de existência de um ou mais *providers* JMS no ambiente de execução, fez com que a opção de construir o protótipo do DIMI sobre o Java/RMI fosse a escolhida.

No protótipo do DIMI foram implementadas as características e funcionalidades descritas no capítulo anterior, com exceção daquelas relacionadas à priorização dos canais, ao gerenciamento do tamanho dos *buffers*, à manutenção de canais, à de consciência do contexto e ao suporte à mobilidade. O protótipo, cujo diagrama UML de classes está mostrado nas figuras 4.6 e 4.7, é constituído de 41 classes e interfaces, aproximadamente 70 arquivos e mais de 6000 linhas de código. Na figura 4.7, por motivos didáticos, são mostrados apenas os métodos e atributos relevantes das classes para a implementação do modelo do DIMI. Dessa forma, os métodos *getters* e *setters* do padrão JavaBeans (SUN, 2005c) e os métodos privados não são mostrados. Além disso, por motivos de espaço, na figura 4.7, não foram mostrados os compartimentos vazios de atributos e métodos das classes e as interfaces da figura 4.6 estão representadas de forma compacta por círculos.

Foi adicionada ao protótipo uma interface gráfica para uma UD, mostrada na figura 4.8. Através dessa interface é possível ao usuário:

- visualizar informações sobre a UD, tais como o seu nome e as suas configurações;
- visualizar os canais de que a UD faz parte, suas LPAs, seus nomes e suas configurações;
- visualizar as conexões da UD em um canal;
- visualizar o cabeçalho das mensagens disseminadas em um canal;
- sair de um canal.

4.2.1 Testes com o Protótipo

Para testar o protótipo e realizar medições nos tempos de disseminação, de entrada em um canal e de ganho de desempenho através do uso de filtros, foi realizada a disseminação de dados de monitoramento do subsistema de monitoramento do EXEHDA (YAMIN, 2004). Esses dados de monitoramento são armazenados em *arrays* de objetos da classe *MonitoringData*, mostrada na figura 4.9. Um objeto dessa classe contém o nome do sensor que o gerou (atributo *sname*), a data e hora de sua geração (atributo *timestamp*), os valores possíveis de monitoração (atributos *intValue*, *longValue*, *doubleValue* e *StringValue*) e o tipo do valor de monitoração contido no objeto (atributo *dataType*), que indica se o valor do objeto é um inteiro, um inteiro longo, um real ou um *string*. Para entender o funcionamento do filtro utilizado nos testes discutidos a seguir, deve ser notado que um objeto dessa classe, por conter apenas um tipo de valor, apresenta 3 atributos não utilizados.

Os testes foram feitos com a participação de 2 células, de acordo com a figura 4.10. Umas das células estava localizada na Universidade Federal do Rio Grande do Sul (UFRGS) e a outra célula na Universidade Católica de Pelotas (UCPEL). A descrição das máquinas envolvidas na disseminação e de seus papéis são mostrados na tabela 4.1.

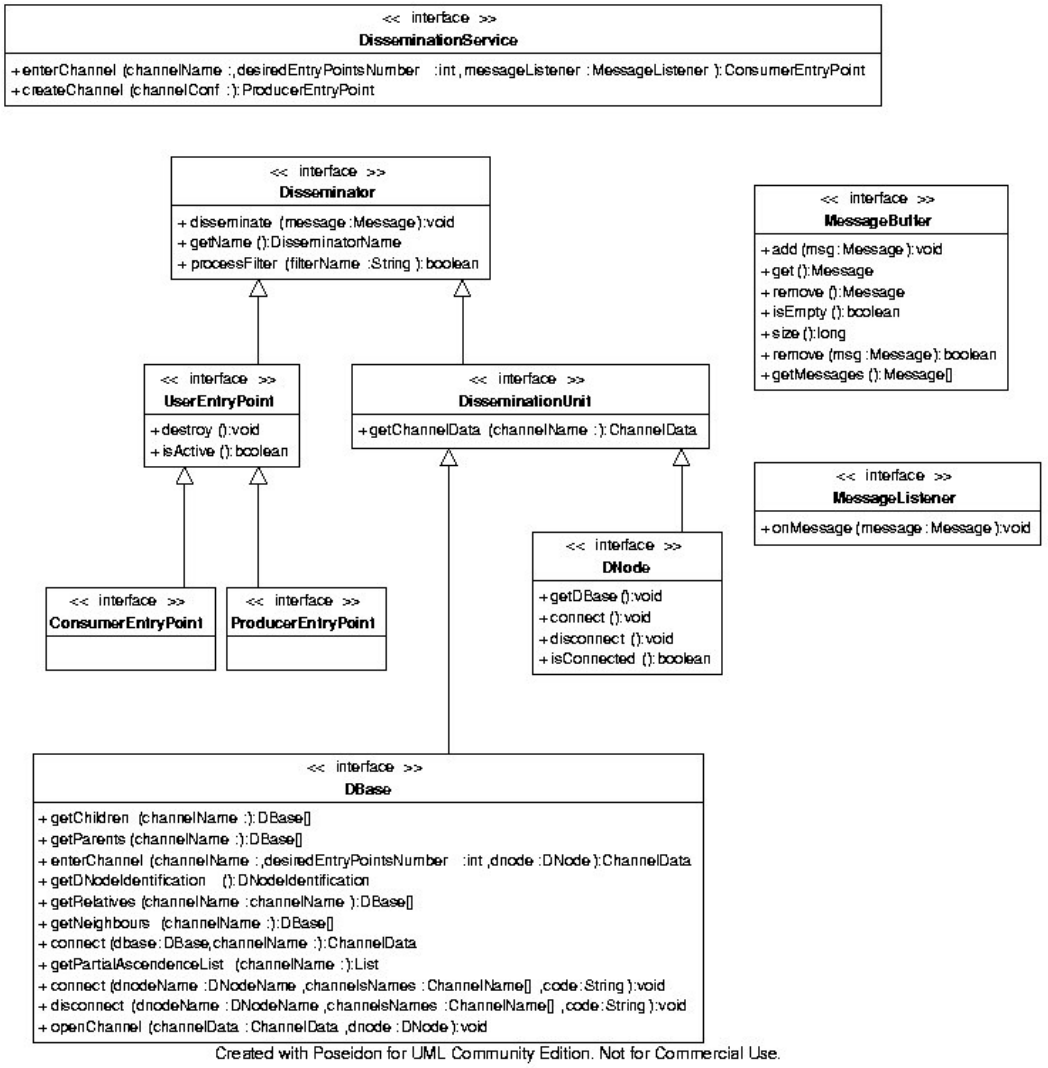

Figura 4.6: Interfaces do protótipo do DIMI.

Tabela 4.1: Máquinas participantes da experimentação com o protótipo do DIMI

Nome	Papel	processador/Memória
saloon.inf.ufrgs.br	UDbase/consumidor	AMD Athlon XP 2400 / 512MB
indiana.inf.ufrgs.br	UDnode/produtor	Pentium III / 512 MB
coisa.inf.ufrgs.br	UDnode/consumidor	AMD Athlon 1 GHz / 512
gradepl.g3pd.ucpel.tche.br	UDbase/consumidor	AMD Athlon XP 2400 / 256 MB

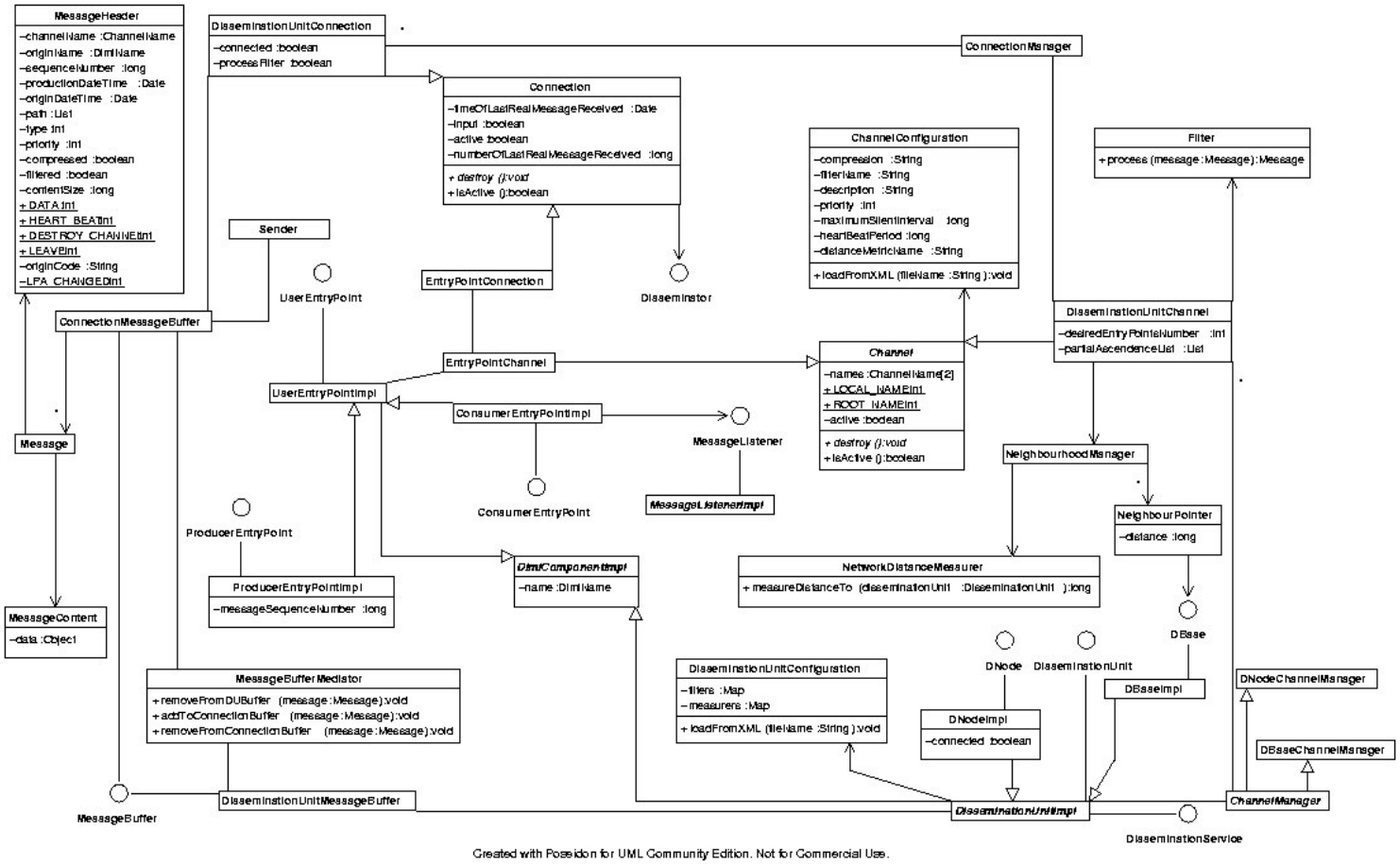

Figura 4.7: Diagrama de classes do protótipo do DIMI

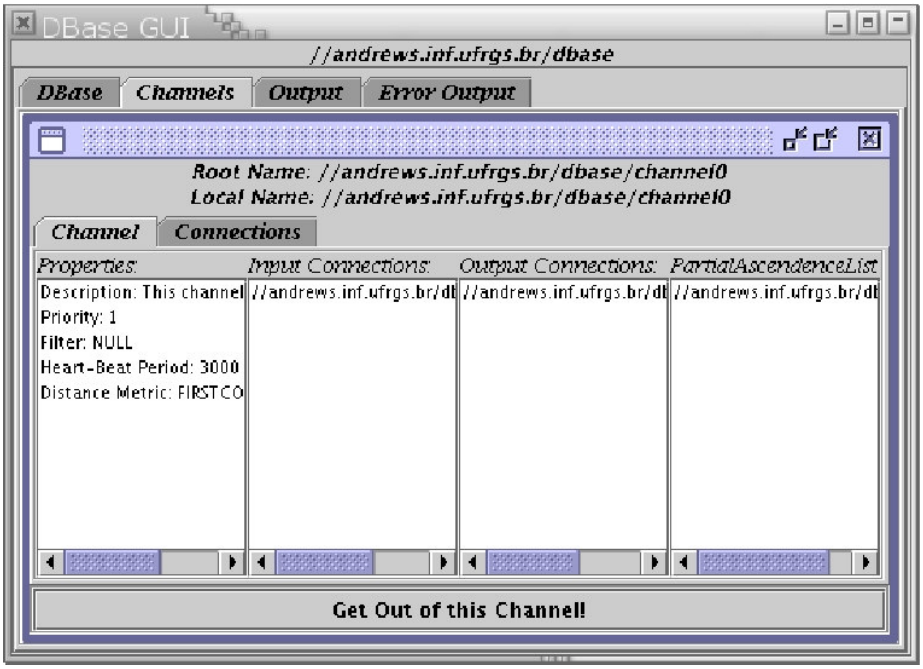

Figura 4.8: Interface gráfica de uma UD no protótipo do DIMI.

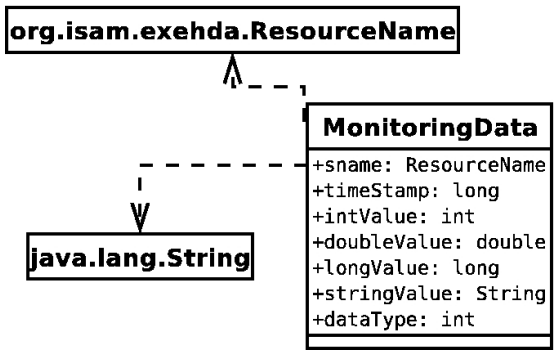

Figura 4.9:Definição de classe de monitoramento no EXEHDA.

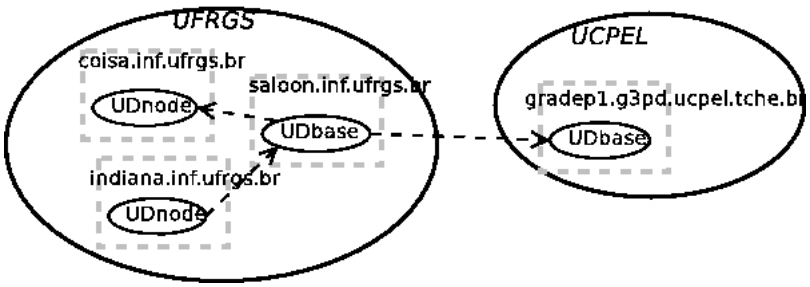

Figura 4.10: Células participantes da disseminação com o protótipo.

Para poder medir com um certo grau de precisão o tempo de disseminação usando o protótipo, todas as máquinas tiveram seus relógios sincronizados através do protocolo NTP (*Network Time Protocol*) (NTP, 2005). Esse protocolo, quando executado na Internet, oferece um atraso de relógio normalmente localizado entre 5 e 100 milisegundos em relação aos servidores NTP. Portanto, todos os tempos de disseminação mostrados nas figuras 4.11 e 4.12 e na tabela 4.3 foram acrescidos de 200 milisegundos, que, a princípio, é o erro máximo que pode ser inserido pelo NTP no cálculo do tempo da disseminação.

A verificação do ganho de desempenho na disseminação através do uso de filtros foi feita com o uso de um filtro *F* específico para os dados de monitoramento do EXEHDA. Na operação de filtragem, *F*, recebe um *array O* de objetos *MonitoringData* e o transforma em um *array* de bytes, levando em conta apenas as informações referentes ao tipo do valor de cada objeto de *O* e portanto diminuindo o tamanho de cada mensagem. A filtragem feita com *F* compacta cada mensagem em até aproximadamente 60% de seu tamanho original. A tabela 5.1 mostra os tamanhos, em *bytes*, de uma mensagem contendo um *array* de número variável de objetos da classe *MonitoringData* em um canal que usa o filtro *F* e em um canal que não usa nenhum filtro.

Tabela 4.2: Comparação do tamanho de mensagens (*bytes)* em canais com filtros.

Número de Elementos do Array no Conteúdo Mensagem	Tamanho da Mensagem em um Canal sem Filtro	Tamanho da Mensagem em um Canal com o Filtro *F*
100	8220	3564
300	21020	7564
500	33820	11564
1000	65820	22564

4.2.2 Resultados Obtidos

A figura 4.11 mostra o atraso médio de entrega de uma mensagem para o consumidor da UDnode da máquina *coisa.inf.ufrgs.br* e a figura 4.12 mostra o atraso médio de entrega de uma mensagem para o consumidor da UDbase da máquina *gradep1.g3pd.ucpel.tche.br*. Os resultados mostrados nessas figuras foram obtidos pela média de 20 execuções e mostram que a utilização do filtro *F* aumenta o desempenho da disseminação de dados de monitoramento do EXEHDA. Os tempos médios de entrada em um canal para o consumidor localizado na máquina *coisa.inf.ufrgs.br* e para o consumidor localizado na máquina *gradep1.g3pd.ucpel.tche.br* são mostrados na tabela 4.3.

Tabela 4.3: Tempo médio de entrada no canal, em milissegundos.

Nome	Tempo
coisa.inf.ufrgs.br	334,55
gradepl.g3.ucpel.tche.br	941,656

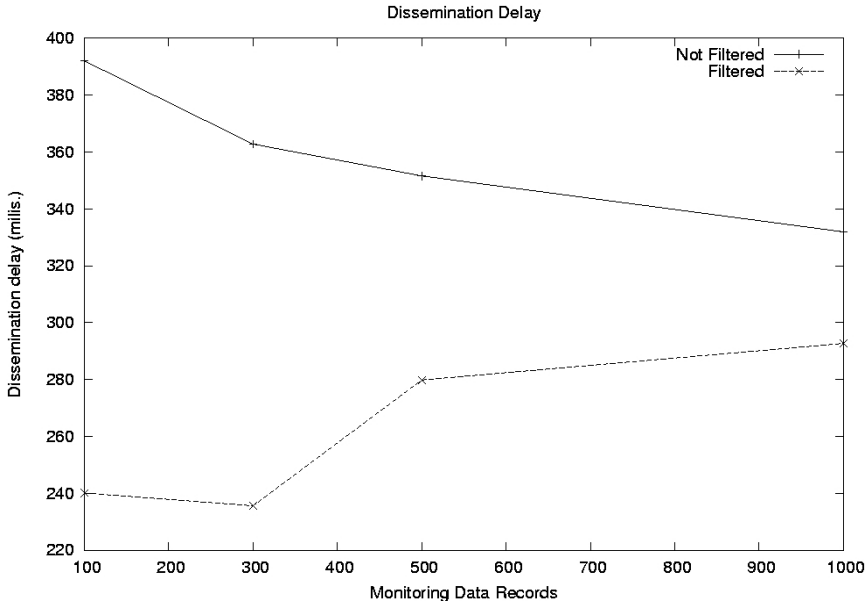

Figura 4.11: Tempo de disseminação para consumo em *coisa.inf.ufrgs.br*.

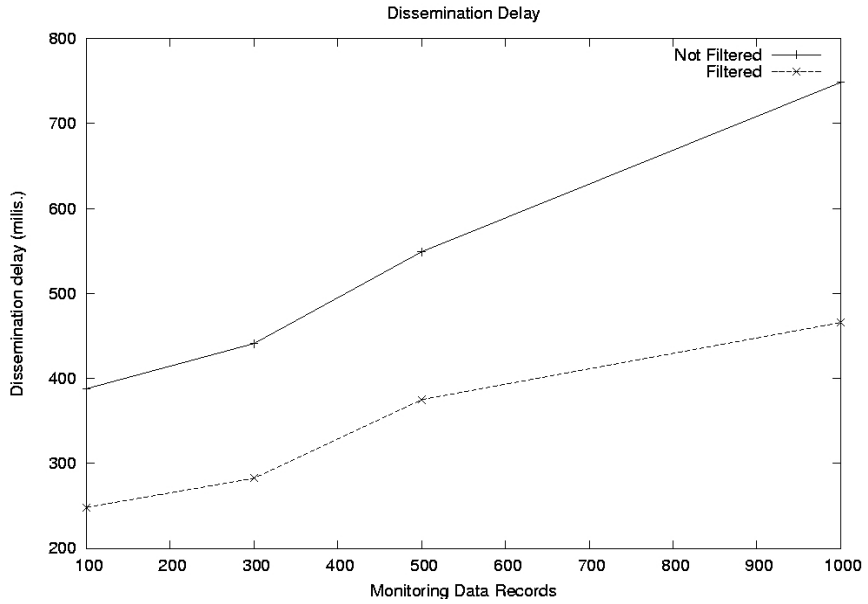

Figura 4.12: Tempo de disseminação para consumo em *gradep1.g3pd.ucpel.tche.br*.

5 TRABALHOS RELACIONADOS

Neste capítulo, são discutidos alguns trabalhos relacionados ao DIMI, nominalmente o ALMI (PENDARAKIS et al., 2001), o Overcast (JANNOTTI et al., 2001), o SALM (BANERJEE; BHATTACHARJEE; KOMMAREDDY, 2002), o Yoid (FRANCISC, 2000) o CoopNet (PADMANABHAN et. al., 2002), o TAO (KEON; FAHMY, 2002) e o HMTP (ZHANG; JAMIN; ZHANG, 2002). Todos esses trabalhos realizam *multicast* no nível de aplicação, de acordo com o que foi discutido no capítulo 3. No final deste capítulo é feita uma comparação entre esses trabalhos e o DIMI no que diz respeito às necessidades da arquitetura ISAM.

5.1 ALMI

O ALMI (*Application Level Multicast Infrastructure*) apresenta uma infra-estrutura de comunicação *multicast* no nível de aplicação para aplicações distribuídas que possuam, no máximo, algumas dezenas de participantes (PENDARAKIS et al., 2001). A topologia de um canal ALMI forma uma árvore do tipo *Minimal Spaning Tree* (MST) (ANTONOIU; SRIMANI, 1997), onde o custo associado a cada aresta é uma métrica específica de cada aplicação. Um canal ALMI consiste de um controlador e um ou mais participantes. Mensagens de dados são transferidas pela árvore, sofrendo retransmissões em cada um dos seus participantes, enquanto que mensagens de controle são transmitidas diretamente entre os participantes e o controlador, em transmissões *unicast*. A comunicação entre dois participantes vizinhos na árvore é bidirecional. Pais podem enviar mensagens para filhos e vice-versa. Quando um participante recebe uma mensagem, ele a retransmite para todos os seus vizinhos, exceto para aquele do qual ele recebeu a mensagem. O conceito de pais e filhos na árvore é utilizado para a criação e a manutenção do canal e para detectar ciclos na disseminação. Um canal ALMI pode possuir muitos produtores de dados.

O controlador do canal é responsável pela entrada de novos participantes e pela manutenção da topologia de disseminação. É o controlador que garante a conectividade da topologia quando os participantes entram e saem do canal, ou quando apresentam falhas. Também é o controlador que calcula a MST periodicamente, a partir de medições feitas localmente por cada um dos participantes do canal. O cálculo da MST, após ser executado pelo controlador, é repassado aos participantes do canal em forma de uma lista de pares ordenados <*pai, filho*>. A partir dessa lista, os participantes sabem a quem devem se conectar para formar a topologia de disseminação.

Um participante do canal possui um endereço formado pelo seu endereço IP e uma porta. Os novos participantes se registram no canal por meio de uma mensagem de controle do tipo *JOIN* enviada para o controlador. Uma vez aceito, o novo participante recebe do controlador seu identificador e o identificador de seu pai no canal. Esse

indentificador contém o endereço de rede do participante. O novo participante então envia uma mensagem do tipo *GRAFT* para seu pai e recebe em resposta a porta de comunicação para o envio de dados. A ligação filho-pai na árvore é periodicamente testada por ambas as partes. Caso seja detectada alguma falha no pai, o filho envia uma mensagem *REJOIN* para o coordenador, pedindo para entrar novamente no canal. Caso um participante detecte falha em um filho, ele apenas destrói a respectiva conexão.

Um limite mínimo de ganho de desempenho *lmg* é definido pelos usuários por meio de parâmetros, com o objetivo de alcançar estabilidade na formação da topologia e evitar o custo de recalcular e reconstruir constantemente a MST, a cada variação nos dados de monitoramento computados por cada participante. A cada nova detecção de variação nos custos das ligações entre os participantes, a MST só é recalculada se as modificações resultantes causarem ganhos que superem *lmg*.

5.2 Overcast

Um canal do Overcast (*Overlay Network Multicast*) consiste de um único produtor de mensagens, que pode estar replicado, e um número arbitrário de consumidores (JANNOTTI et al., 2000). O produtor é o responsável pela gerência e o controle de todos os participantes do canal. O Overcast utiliza a largura de banda como critério de métrica de custo para a montagem da topologia de um canal. Por isso, ele não possui o objetivo de oferecer comunicação *multicast* a aplicações interativas como jogos multi-jogadores ou vídeo-conferências.

Uma característica interessante da arquitetura do Overcast é a sua capacidade de armazenamento de mensagens enviadas para um canal, de forma que um consumidor pode pedir para receber todas as últimas *n* mensagens de um canal. Assim, por exemplo, um consumidor pode pedir os últimos dez minutos de um fluxo de vídeo correntemente transmitido.

A criação e a manutenção da topologia de um canal no Overcast busca a maximização da largura de banda disponível entre o produtor e os consumidores. Dessa forma, tenta-se colocar os novos participantes o mais distante possível da raiz, sem sacrificar a largura de banda desde a raiz. Para medir a largura de banda em uma conexão, é medido o tempo de *download* de 10 Kbytes. Periodicamente, um participante reavalia sua posição na topologia, podendo reestabelecer conexões caso encontre um pai mais apropriado para si no canal.

O produtor de um canal Overcast periodicamente obtém informações a respeito da topologia de disseminação do canal. Essas informações são utilizadas pelo produtor para aumentar a velocidade do processo de aceitação de novos participantes e obter estatísticas de consumo de mensagens no canal. Cada participante de um canal Overcast, incluindo o produtor, mantém uma tabela de informações sobre todos os participantes abaixo de si na topologia do canal, além de um *log* com todas as mudanças ocorridas nessa tabela. Dessa forma, o produtor sempre possui informações atualizadas sobre todos os participantes do seu canal.

Para aliviar o problema da escalabilidade que surge da concentração de responsabilidades no produtor, é utilizada uma técnica baseada na replicação do produtor, na qual o nome de DNS (*Domain Name System*) (DNS, 2005) do produtor é direcionado, segundo um comportamento *round-robin*, para as réplicas do produtor que existirem no canal.

A concentração de responsabilidades no produtor cria, além da limitação da escalabilidade, um ponto único de falhas no sistema. Para lidar com essa característica não desejável, o Overcast utiliza uma configuração linear de réplicas do produtor, ilustrada na figura 5.1, onde cada réplica possui apenas um filho. Nessa configuração linear, cada uma das réplicas do produtor está apta a substituí-lo em caso de falha.

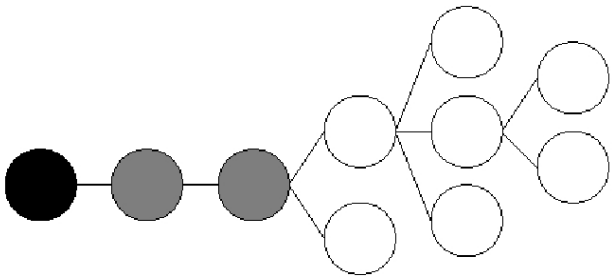

Figura 5.1: Exemplo de configuração linear de réplicas no Overcast.

5.3 SALM

O SALM (*Scalable Application Layer Multicast*) cria e gerencia canais que apresentam uma hierarquia de produtor único, em que os participantes mantém o estado detalhado dos participantes próximos e o estado resumido dos participantes distantes (BANERJEE; BHATTACHARJEE; KOMMAREDDY, 2002). A latência das ligações de rede é utilizada como a métrica de distância entre os participantes para a criação e a manutenção da topologia de disseminação.

A hierarquia utilizada nos canais do SALM é constituída pela agregação dos participantes em diferentes níveis. Esses níveis são identificados seqüencialmente, iniciando pelo nível zero *L0*. Os participantes de cada nível são reunidos em agregados, que podem ter tamanho mínimo de *k* participantes e tamanho máximo de 3k-1 participantes, onde *k* é um parâmetro definido pelos usuários. Adicionalmente, cada agregado possui um líder. Um algoritmo distribuído escolhe o participante localizado no centro do agregado como líder. A determinação do centro do agregado é baseada na teoria dos grafos. O centro do agregado é o participante que possui a menor distância em relação a todos os outros participantes do agregado.

Os participantes de um canal são reunidos em agregados da seguinte maneira: todos os participantes são parte do mais baixo nível *L0*. O protocolo de agregação do SALM particiona os participantes do nível *L0* em um conjunto de agregados. Os líderes dos agregados do nível *L0* são encontrados, passando a fazer parte do nível *L1*, onde novos agregados são formados. Iterativamente, os lideres do nível *Li* passam a fazer parte do nível *L*(*i*+*1*) e formam novos agregados, até ser encontrada a raiz do canal, conforme ilustrado na figura 5.2. Isso origina as seguintes propriedades para a distribuição dos participantes nos diferentes níveis:

- um participante pertence a apenas um único agregado em um nível;

- se um participante está presente em um nível *Li*, então ele deve estar presente em cada um dos níveis inferiores a *Li* e ser, obrigatoriamente, um líder em cada um dos agregados dos níveis inferiores a *Li*;
- se um participante não está presente em um nível i, ele não pode estar presente em nenhum nível *j*, onde *j>i*;
- existem, no máximo, *logkN* níveis, onde *k* é o parâmetro citado anteriormente e *N* é o número de participantes do canal.

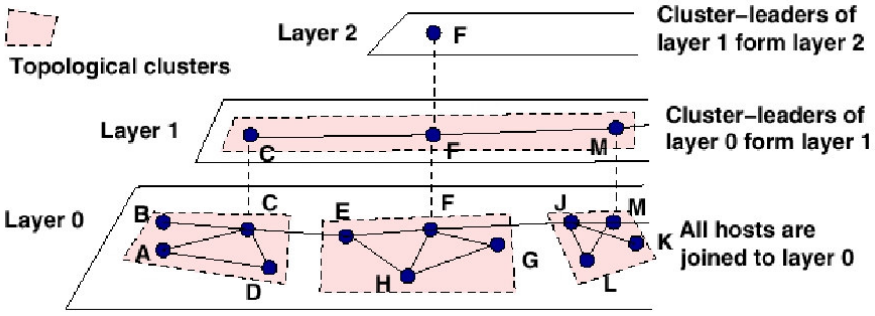

Figura 5.2: Exemplo de um canal do SALM, com sua hierarquia de níveis.

A hieraquia de participantes é usada para definir diferentes topologias de disseminação de mensagens de controle e de dados, pois os envios de mensagens de controle, em sua maior parte, são feitos apenas entre os integrantes do agregado e as mensagens de dados precisam ser entregues a todos os participantes do canal.

Para gerenciar a entrada de novos participantes em um canal, é utilizado um componente especial, que todos os participantes devem conhecer antecipadamente, chamado ponto de encontro (PE) ou *rendezvous point*, que pode ou não ser um participante do canal. Qualquer novo participante que queira entrar em um canal, deve antes contatar o PE. Para que um novo participante entre no canal, ele deve ser inserido em algum dos agregados existentes na camada *L0*.

Se um novo participante *A* quiser entrar no grupo, ele deverá entrar em contato com o PE do canal, que lhe retornará os participantes presentes no mais alto nível da hierarquia do canal. *A* então identificará qual dos participantes do nível mais alto da hierarquia é o mais próximo de si. Uma vez encontrado o participante mais próximo, este informará a *A* quais são os participantes que integram o seu agregado no nível imediatamente inferior. *A* então repetirá recursivamente esse processo até que encontre seu agregado no nível *L0*.

Cada participante *H* de um agregado *C* envia periodicamente uma mensagem a cada um de seus vizinhos em *C* contendo a distância estimada de todos os seus vizinhos em *C*. Para cada líder de agregado vizinho, o líder de *C* envia periodicamente uma mensagem contendo os detalhes de sua vizinhança em seu próprio agregado. É através dessas mensagens que cada participante mantém informações detalhadas a respeito de seus vizinhos no agregado e informações resumidas a respeito dos participantes de

outros agregados. Um participante *H* é declarado *morto* por todos os outros participantes de seu agregado se nenhum deles recebe um aviso de vida através de mensagens de aviso de vida durante um determinado intervalo de tempo.

Um líder de agregado periodicamente verifica o tamanho do seu agregado e trata da divisão ou da combinação com outros agregados se o mesmo ultrapassa seu valor máximo ou mínimo permitidos. Um agregado que ultrapassa o tamanho máximo de *3k-1* é simplesmente dividido em dois. Se qualquer agregado resultante dessa divisão tiver tamanho maior que *3k-1*, o processo se repete, até que todos os agregados estejam dentro dos limites de tamanho. Se o tamanho de um agregado *C* em um nível *Li*, com um líder *J* ficar abaixo de *k*, então é iniciada a operação de combinação. *J* escolhe seu vizinho mais próximo, *K*, no nível *L(i+1)*. *K* é também líder de um agregado em *L(i)*. *J* inicia então a combinação de seu agregado com o agregado de *K*.

5.4 Yoid

Yoid (*Your own internet distribution)* possui um conjunto de protocolos que permitem seu funcionamento sobre IP multicast e sobre IP unicast (TCP ou UDP). Tais protocolos apresentam finalidades variadas que incluem identificação, controle de fluxo, seqüenciamento, recuperação e retransmissão, entrega com probabilidade nula de perda de pacotes e modos de transmissão. O principal protocolo de Yoid é, no entanto, o de gerenciamento de topologias que permite que participantes dinamicamente se auto-cofigurem em uma topologia de duas partes chamada *tree-mesh* (FRANCIS, 2000).

Cada canal em Yoid possui um ou mais pontos de encontro (PE) ou *rendezvous points*, que não fazem parte da topologia, mas servem como mecanismo de descoberta para permitir que novos participantes se juntem ao canal. PEs, portanto, são contatados quando um novo participante deseja entrar ou sair de um canal e, além disso, mantêm uma lista com todos os integrantes de um canal. Além disso, PEs periodicamente verificam se cada um dos integrantes de um canal estão em bom funcionamento.

O nome de um canal consiste de três componentes: (**i**) o nome do PE, (**ii**) as portas de trabalho do PE, e (**iii**) o nome do canal. Esse esquema de nomeação permite que qualquer participante crie localmente um nome de grupo globalmente identificado e permite a identificação do PE no nome do canal. Uma URL pode codificar um nome de canal da seguinte forma: **yoid://PE_host_name:port/channel_name**.

Um canal do Yoid é construído sobre duas topologias: uma árvore e uma malha *mesh*. A árvore é utilizada para a disseminação eficiente de dados para os participantes do canal. A malha é, por sua vez, utilizada para a disseminação robusta de dados e de mensagens de controle para os participantes do canal. Para implementar a topologia de malha no Yoid, cada participante mantém um pequeno número de endereços de vizinhos (três ou quatro). Esses vizinhos são escolhidos aleatoriamente entre todos os participantes do canal, por um participante *N*, com as seguintes exceções:

- não é possível incluir participantes que sejam vizinhos na árvore;
- não é possível incluir um participante que já tenha estabelecido uma ligação de malha com *N*.

A documentação do Yoid (FRANCIS, 2000) não é muito clara a respeito dos mecanismos utilizados para a formação da topologia de disseminação e para a aceitação de novos participantes em um canal. Apesar disso essa documentação discute

detalhadamente alguns aspectos importantes da disseminação de dados usando *multicast* no nível de aplicação, como a realização da detecção e da prevenção de ciclos na topologia de disseminação.

5.5 CoopNet

O CoopNet (*Cooperative Network*) propõe um protocolo de comunicação *multicast* no nível de aplicação que combina características de comunicação cliente-servidor com características de comunicação *peer-to-peer* (P2P) (PADMANABHAN et al., 2002). O CoopNet utiliza o modelo P2P para complementar o modelo cliente-servidor e não para substituí-lo. A comunicação P2P toma lugar no CoopNet quando o servidor ficar sobrecarregado com diversas chamadas simultâneas. Nos momentos de sobrecarga do servidor, este delega aos clientes a tarefa de prover as informações em seu lugar. A topologia de disseminação de um canal é uma árvore que possui sua raiz no servidor. Cada participante da árvore não é uma máquina dedicada, podendo, por isso, decidir em um determinado instante se quer fazer parte ou não da redistribuição de informações. Além disso, os participantes do canal podem falhar. Por essas razões, o CoopNet adota múltiplas árvores de disseminação para a mesma informação disseminada. Isto permite que a disseminação de dados seja mais robusta, pois não a torna dependente da participação confiável de todos os participantes do canal. O custo da existência de múltiplas árvores de disseminação é, entretanto, a sobrecarga na rede, devido à redundância de dados.

As árvores de distribuição são criadas e gerenciadas de forma centralizada pelo servidor CoopNet. Esse comportamento centralizado é justificado pelos projetistas do CoopNet com os seguintes argumentos:

- o servidor não fica sobrecarregado em casos de picos de consumo, pois a distribuição é compartilhada com diversos clientes CoopNet;
- a criação e manutenção centralizada das árvores de distribução é mais simples do que a descentralizada e, consequentemente, calcula entradas e saídas de participantes mais rapidamente;
- o ponto único de falha localizado no servidor não é considerado um problema, pois no contexto no qual o CoopNet trabalha, o servidor é a própria fonte de informações e, se ele falhar, nenhum outro participante poderá substituí-lo em sua função.

O servidor tem controle e conhecimento total das árvores de disseminação. Quando um novo participante deseja entrar no canal, ele entra em contato com o servidor, informando sua largura de banda disponível para a disseminação dos dados. O servidor responde para o novo participante com uma lista de participantes aos quais ele deve se conectar no canal. É utilizado um mecanismo de codificação de audio e vídeo em *N* fluxos independentes chamado *Multiple Description Coding* (MDC). Cada um dos fluxos criados por MDC é transmitido em uma das árvores de disseminação de um canal.

Os pais de um novo participante são escolhidos da seguinte forma: a partir da raiz, vai-se descendo em direção às folhas no canal verificando a capacidade de aceitação de novos filhos em cada um dos participantes. O próprio servidor pode aceitar um novo filho. Após recebidas as respostas dos participantes que possuírem capacidade para aceitar novos filhos, alguns desses participantes são escolhidos aletoriamente pelo

servidor. Os projetistas de CoopNet argumentam que esse procedimento *top-down* com escolha aleatória garante árvores balanceadas de disseminação. Em termos de custos de envio de mensagens para a entrada de um novo participante em um canal, o servidor recebe e envia uma mensagem. Cada pai do novo participante recebe e envia uma mensagem. O novo participante, por sua vez, envia e recebe $M + 1$ mensagens, onde M é o número de árvores de distribuição do canal em questão.

Saídas de participantes de uma ou mais árvores de disseminação são feitas de duas formas: por decisão ou por falha. No primeiro caso, o participante avisa o servidor que sairá e o servidor trata então de reinserir os filhos do participante no canal através do método descrito anteriormente. Ao reinserir um participante, o servidor implicitamente reinsere as subárvores das quais esse participante é raiz.

Uma falha em um participante corresponde ao caso em que um participante sai do canal sem avisar o servidor CoopNet. A descoberta de um participante falho é feita através da verificação da degradação do recebimento de dados por seus filhos. Os filhos de um participante falho não recebem nenhum pacote. Cada participante é capaz de verificar a degradação de recebimento de dados através da comparação da entrega de dados entre as diversas árvores das quais faz parte. Quando um participante detecta a degradação de entrega de mensagens em um nível inaceitável, onde o limite de aceitabilidade é definido como parâmetro pelos usuários, ele entra em contato com seu pai para saber se ele confirma o mesmo problema. Se o pai confirmar, a origem do problema localiza-se acima na árvore e o participante deixa que seu pai lide com o problema. Caso o pai não confirme, o participante que detectou o problema pede à raiz para entrar novamente em outra árvore de disseminação, reexecutando o algoritmo citado anteriormente.

No CoopNet, é utilizado um conjunto de participantes pré-determinados para otimizar o cálculo das distâncias entre participantes do canal. A determinação da proximidade entre dois participantes é feita de forma que cada participante determine suas *coordenadas* através da medida de sua latência em relação ao referido conjunto de participantes pré-determinados. Para comparar a proximidade de dois participantes, o servidor mantém todas as coordenadas dos participantes do canal e realiza uma comparação quando necessário.

5.6 TAO

No TAO (*Topology Aware Overlay for Group Communication*), é proposta a exploração da rede física subjacente ao canal com o objetivo de formar a topologia de disseminação do canal, que é uma árvore (KWON; FAHMY, 2002). O critério de distância entre os participantes explorado no TAO é o caminho mais curto entre os roteadores IP, em termos de latência. Para entrar em um canal, o novo participante escolhe um pai cujo caminho até a raiz do canal se sobreponha ao máximo com o seu próprio caminho. No TAO, um produtor diferente da raiz precisa enviar, via transmissões *unicast*, dados diretamente para a raiz para que esta os dissemine pela árvore para todos os consumidores do canal. Os projetistas de TAO argumentam que esse tipo de comportamente é viável em termos de desempenho quando existe apenas um pequeno número de produtores. O TAO trabalha com as duas seguintes definições:

- o caminho do participante A ao participante B, denotado por $P(A,B)$, é uma sequência de roteadores que fazem parte do caminho mais curto entre A e B de

acordo com o protocolo de roteamento utilizado. O tamanho do caminho *P(A,B)* ou *len(P(A,B)* é o número de roteadores que *P(A,B)* possui;

- *A*::*B* se *P(S,A)* é um prefixo de *P(S,B)*, onde *S* é a raiz da árvore dedisseminação.

Cada participante do canal mantém uma tabela chamada tabela de familiares (TF), em que ficam definidas as suas conexões de ascendência e descendência. Uma entrada da tabela TF é reservada para o pai e as outras para os filhos. Uma entrada de TF consiste de uma tupla (*endereco, caminho*), onde *endereco* é o endereço IP do participante referenciado e *caminho* é o caminho mais curto do participante referenciado até a raiz da árvore.

O algoritmo de entrada na árvore de comunicação é o seguinte: se um novo participante *N* quiser entrar no canal através do participante *C*, é selecionado, se possível, um participante *A*, tal que *A* seja filho de *C*, *A*::*N* e *len(P(S,N)) > len(P(S,A)) > len(P(S,C)*. Caso isso não seja possível, se existir *A* filho de *C* tal que *N::A*, *N* torna-se filho de *C* e *A* torna-se filho de *N*. No caso de não haver filho de *C* satisfazendo a primeira ou a segunda condição, *N* torna-se filho de *C*.

Um novo participante que deseja entrar em um canal envia uma mensagem do tipo *JOIN* para a raiz *S* do canal. Após o recebimento dessa mensagem, *S* computa o caminho do novo participante e executa o algoritmo de entrada no canal, visto anteriormente. Se o novo participante tornar-se filho de *S*, a TF de *S* é atualizada. Caso contrário, *S* propaga uma mensagem do tipo *FIND* para seu filho que contém o maior prefixo do caminho do novo participante. A mensagem *FIND* carrega o endereço IP e o caminho do novo participante e é propagada para cada participante que compartilhe o maior prefixo de caminho. O participante que receber uma mensagem do tipo *FIND* repropaga essa mensagem ou assume o novo participante como filho, alterando a sua própria TF.

A saída de um participante *A* do canal é iniciada pelo envio de uma mensagem do tipo *LEAVE* de *A* para seu pai. Uma mensagem *LEAVE* contém a TF de *A*. Ao receber uma mensagem *LEAVE*, o pai de *A* atualiza sua TF, retirando *A* e adicionando os filhos de *A*. Falhas em participantes do canal são detectadas através de mensagens periódicas de aviso de vida. Quando um filho falha, seu pai simplesmente o descarta da sua TF. Quando um pai falha, seu filhos precisam entrar novamente no canal, através da reexecução do algoritmo descrito anteriormente.

Se as condições da rede mudarem durante a existência do canal e a árvore de disseminação tornar-se ineficiente, esta é adaptada às novas condições da rede. Essa adaptação é feita pela verificação periódica dos caminhos de todos os participantes do canal até a raiz do canal. Isso é feito da seguinte forma: os participantes calculam os caminhos de seus filhos e agregam informações de caminho para os participante que contenham prefixos de caminho iguais. Grupos de participantes que contenham prefixos semelhantes são então formados. O caminho de um único participante de um grupo é calculado de cada vez. Quando uma alteração é detectada em um participante, todos os caminhos dos integrantes do seu grupo são alterados.

O algoritmo apresentado de entrada em um canal reduz o atraso de transmissão de mensagens e reduz o número de pacotes duplicados em uma ligação de rede. Esse algoritmo, entretanto, não leva em consideração o congestinamento ou a largura de banda das ligações de rede. Para lidar com congestionamentos e levar em cosideração a

largura de banda das ligações de rede, uma otimização pode ser feita no algoritmo. Essa otimização é a seguinte: um novo participante participante *B* pode tornar-se filho de outro participante *A* se *A* tiver um prefixo de caminho comum com *B* de tamanho *len(P(S,A))-k*, mesmo que os últimos *k* participantes do caminho de *A* até a raiz não participem do prefixo de *B*. Essa adaptação é chamada de *minus-k*. *Minus-k* permite que novos filhos de de um participante *A* com escassez de largura de banda assumam como pai um outro participante, não deixando, dessa forma, piorar o problema de falta de largura de banda em *A*, pela aceitação de novos filhos no canal. Quando a disponibilidade de largura de banda em um participante cair abaixo de um certo limite, definido através de parâmetros, o algoritmo *minus-k* é ativado, indicando que um caminho alternativo pode ser explorado. O parâmetro *k* controla o desvio permitido na entrada de um novo participante. Um valor grande para *k* pode aumentar o atraso na disseminação, mas aproveita melhor a largura de banda. Através do algoritmo *minus-k*, um novo participante pode examinar vários caminhos possíveis permitidos por *k* e selecionar aquele que maximize a largura de banda.

5.7 HMTP

O HMTP *(Host Multicast Tree Protocol)* propõe uma solução híbrida de *multicast* no nível de aplicação e de IP Multicast. O HMTP tenta ser compatível com o IP Multicast o máximo possível, sem que sejam necessárias quaisquer modificações no *hardware* e no *software* existentes hoje na Internet (ZHANG; JAMIN; ZHANG, 2002). O HMTP utiliza o conceito de *ilhas* IP Multicast. Uma *ilha* IP Multicast é uma rede de qualquer tamanho que suporte o IP Multicast sem necessidade de quaisquer tipos de ajustes. Dentro de uma dessas *ilhas*, o IP Multicast nativo é utilizado para a disseminação de dados. Um participante específico de uma *ilha* é eleito seu Membro Designado (MD). *Ilhas* diferentes são conectadas por túneis UDP (*User Datagram Protocol*) que ligam seus respectivos MDs. Ao chegar em uma *ilha*, os pacotes UDP de outra *ilha* são retransmitidos através do IP Multicast para os seus participantes e para as *ilhas* vizinhas fazendo-se uso de outros túneis UDP.

Os MDs organizam-se em uma árvore de disseminação, através da execução do protocolo de entrada em um canal no HMTP, detalhado na figura 5.3. Nessa árvore, dois DMs são vizinhos se eles estão conectados por um túnel UDP. Como a topologia de disseminação é uma árvore, todos os participantes têm apenas um pai, não havendo redundância de entrega de dados em um canal. Cada canal requer a existência de um ponto de encontro (PE) ou um *Host Multicast Rendezvous Point* (HMRP), onde novos participantes podem aprender sobre a topologia da árvore de disseminação do canal. A localização do PE não faz parte do HMTP e, por isso, é feita por meios externos ao mesmo. O HMTP utiliza a latência como critério de medida de distância entre dois participantes.

Periodicamente, o estado dos participantes é atualizado fazendo-se uso do envio de mensagens. Cada filho envia mensagens do tipo *REFRESH* para seu pai na árvore de disseminação. O pai, em resposta, envia uma mensagem do tipo *PATH*, contendo o caminho do pai à raiz. Ao receber uma mensagem do tipo *PATH* de seu pai, o filho automaticamente atualiza seu caminho até a raiz. Quando um participante deixa a árvore, ele avisa seu pai e seus filhos. Seu pai, então, simplesmente o descarta da lista de filhos e seus filhos entram novamente na árvore através da reexecução do protocolo de entrada. Quando um participante falha, seu pai e seus filhos detectam a falha e comportam-se como se tivessem recebido um aviso de saída do participante falho.

Para adaptar-se às condições da rede, HMTP periodicamente reestrutura a àrvore de disseminação através do reestabelecimento de conexões entre os participantes do canal. Esse reestabelecimento de conexões é executado periodicamente através da reexecução do algoritmo de entrada por cada participante.

No HMTP, é feita a detecção e não a prevenção dos ciclos de disseminação. Quando um ciclo é formado, um participante detecta esse ciclo quando seu caminho à raiz for atualizado. Após a detecção do ciclo, o participante reexecuta o protocolo de entrada para encontrar um novo pai na árvore, de modo que o ciclo seja desfeito.

5.8 Comparação dos Trabalhos Relacionados com o DIMI

Nesta seção é feita uma comparação entre os trabalhos citados anteriormente neste capítulo e o DIMI no que diz respeito às necessidades da arquitetura ISAM. Essa comparação é resumida na tabela 5.1.

---

1. Uma pilha *S*, inicialmente vazia, é utilizada para armazenar todas os participantes válidos, identificados durante o algoritmo;

2. Encontre a raiz do canal, marque-a como o pai potencial e calcule a sua distância, usando a latência das ligações de rede como métrica de distância;

3. Obtenha uma lista com todos os filhos do pai potencial e calcule a distância de todos eles, usando a latência das ligações de rede como métrica de distância;

4. Encontre o participante mais próximo entre o pai potencial e todos os seus filhos, exceto aqueles marcados como inválidos. Se todos eles forem inválidos, retire o elemento do topo da pilha *S*, torne-o o novo pai potencial e retorne ao passo 3. Se não houver elementos em *S*, gere uma exceção indicando que não foi possível entrar no canal;

5. Se o participante mais próximo não for o pai potencial, coloque o pai potencial na pilha *S*, ajuste o participante mais próximo como pai potencial e retorne ao passo 3; Caso contrário, tente conectar-se ao pai potencial. Se isso não for possível, marque o pai potencial como inválido e retorne ao passo 4. Se a conexão foi estabelecida, termine.

Figura 5.3: Algoritmo de entrada do HMTP.

O ALMI, o Overcast e o Coopnet não são escaláveis pois possuem gerenciamento centralizado dos canais, o que não ocorre com o SALM, o Yoid, o TAO e o HMTP pois estes possuem gerenciamento distribuído. Comparados com o DIMI, estes últimos, no entanto, potencializam a sobrecarga nos nós envolvidos diretamente na entrada de novos participantes em um canal: a raiz, no HMTP, e os pontos de encontro (*rendezvous points*), no SALM, no TAO e no Yoid. O DIMI não obriga a requisição de acesso a um integrante específico do canal. De fato, no DIMI todos os integrantes de um canal são potenciais pontos de entrada para novos consumidores. O protocolo de entrada proposto nesta dissertação é inspirado no HMTP. A principal diferença entre eles é que, no HMTP, um novo consumidor inicia a entrada no canal a partir da raiz, descendo no canal pelos participantes que forem mais próximos de si, até encontrar um ponto de entrada adequado em termos de distância. No DIMI, um novo consumidor inicia a entrada a partir de um participante qualquer do canal, e se desloca para cima e para

baixo no canal pelos participantes que forem mais próximos de si, até encontrar um ponto de entrada adequado em termos de distância.

Além da escalabilidade, a consciência do contexto é uma característica necessária aos serviços do ISAM*pe*. Todos os trabalhos citados apresentam adaptações em relação ao estado da rede e em relação ao estado dos integrantes dos canais. O DIMI, porém, é o único a observar, além destes, outros aspectos do contexto, como a utilização de CPU e de memória. O suporte à desconexão planejada e o suporte à mobilidade de usuários também são características apresentadas somente pelo DIMI.

O suporte ao protocolo IP Multicast e a possibilidade de existência de mais de um produtor em um canal, no entanto, são características existentes no Yoid, no HMTP e no ALMI que não existem no DIMI. O suporte ao IP Multicast é desejado devido à existência de diversas *ilhas* IP Multicast operacionais atualmente (ZHANG; JAMIN; ZHANG, 2000) (CHU; RAO; ZHANG, 2000), que poderiam ser utilizadas dentro de uma célula no ISAM*pe* para a comunicação entre a base e os nós de processamento. A possibilidade de existência de mais de um produtor em um canal, por sua vez, é importante para algumas aplicações, como jogos multi-jogador e vídeo-conferências. Conforme será discutido na seção 6.1, pretende-se estender o modelo do serviço DIMI para possibilitar a existência de múltiplos produtores e o suporte ao IP Multicast.

Tabela 5.1: Comparação dos trabalhos relacionados com o DIMI.

	ALMI	SALM	Yoid	CoopNet	TAO	HMTP	DIMI
Escalabilidade	–	±	±	–	±	±	±
Consciência de Contexto	±	±	±	±	±	±	±
Desconexão Planejada	–	–	–	–	–	–	+
Mobilidade de Usuários	–	–	–	–	–	–	+
Suporte a IP Multicast	–	–	+	–	–	–	–
Múltiplos Produtores	+	–	–	–	–	–	–

6 CONCLUSÃO

Os ambientes de computação pervasiva terão um número de componentes de *hardware* e de *software*, tanto homogêneos quanto heterogêneos, muito maior do que o existente na atual Internet. Por isso, esses ambientes devem apresentar elevada escalabilidade, entre outras características.

Muitas das aplicações pervasivas têm necessidade de disseminar informações. As propostas atuais de mecanismos disseminadores não suprem as necessidades de escalabilidade dos ambientes de computação pervasiva. Esta dissertação propôs um serviço de disseminação de informações para a arquitetura ISAM, que é uma arquitetura para o desenvolvimento e a execução de aplicações pervasivas. A arquitetura ISAM possui um ambiente de execução próprio (ISAM*pe*), cujas funcionalidades e gerência são providas por um *middleware* (EXEHDA) capaz de oferecer, entre outras funcionalidades, primitivas de comunicação para objetos distribuídos. O serviço proposto, denominado DIMI (Disseminador *Multicast* de Informações), possui o objetivo principal de disseminar informações com elevada escalabilidade.

A escalabilidade no DIMI é alcançada pela não-formação de gargalos em função da admissão concorrente de novos participantes em um canal. Isso ocorre porque, no DIMI, é feita a distribuição da responsabilidade de aceitação de novos participantes entre todos os participantes existentes um canal. Tal comportamento é possível pela definição e uso de um novo algoritmo de entrada em um canal, proposto nesta dissertação, que permite ao novo participante a interação com um subconjunto de todos os participantes do canal desejado, em busca de um ponto de entrada adequado em termos de distância. Cabe ressaltar que o DIMI é parametrizável o bastante para deixar a cargo das aplicações a definição de métricas de distância que lhes sejam mais adequadas através de interfaces bem definidas. Dessa forma, as métricas de distância no DIMI não se restringem à latência ou à largura de banda das ligações de rede, como ocorre em trabalhos relacionados (PADMANABHAN et al., 2002) (JANNOTTI et al., 2000).

Os ganhos em escalabilidade resultantes da aplicação do algoritmo de entrada em um canal proposto nesta dissertação foram demonstrados nos resultados das simulações discutidos na seção 4.1. Esses resultados mostram que o DIMI atinge maior escalabilidade que o trabalho relacionado HMTP quando novos consumidores desejam entrar simultaneamente em um canal, apresentando o custo da perda de desempenho na disseminação de mensagens no canal. Como as simulações não levaram em conta a manutenção dos canais nem a integração do DIMI com o serviço de descoberta de recursos da arquitetura ISAM, é provável que, em uma situação real ou simulada da execução do DIMI, esta perda de desempenho seja minimizada, tornando o desempenho de disseminação do DIMI equivalente ao desempenho do HMTP.

Além de apresentar elevada escalabilidade, o DIMI oferece suporte à desconexão planejada e à mobilidade de usuários no ISAM*pe*. Essas funcionalidades levam em conta as peculiaridades do ISAM*pe* de comunicação P2P entre as bases das células e de comunicação através de *proxies* entre os nós de processamento das células. Por oferecer essas funcionalidades, pode-se dizer que o DIMI contribui para a invisibilidade necessária ao EXEHDA, pois a necessidade de intervenção do usuário para o funcionamento do serviço é diminuída. Cabe ressaltar que o DIMI foi cuidadosamente projetado para não perder mensagens disseminadas devido à movimentação dos usuários e de seus dispositivos no ISAM*pe*. Além disso, o serviço é parametrizável o bastante para que os usuários possam definir o comportamento do Ponto de Entrada de Produção (PEP) nas situações em que houver estouro de *buffer* em períodos de conexão planejada, conforme foi discutido na seção 3.8.3.

O funcionamento do serviço pode ser definido como adaptável ao contexto, pois são feitas, em tempo de execução, ações periódicas e eventuais de adaptação às condições da infra-estrutura física da rede subjacente ao serviço e ao estado das Unidades de Disseminação (UDs) participantes dos canais, levando em conta características dinâmicas como *status* de funcionamento, utilização de memória e utilização de CPU. Essa manutenção do funcionamento dos canais tende a causar o aumento do desempenho da disseminação durante o ciclo de vida dos canais. Outra medida para otimizar o desempenho de disseminação nos canais é a capacidade de utilização de filtros, conforme foi discutido na seção 3.2. O ganho em desempenho obtido com a utilização de filtros foi demonstrado com a execução do protótipo, conforme foi discutido na seção 4.2. Os resultados mostram que o uso de filtros aumenta o desempenho da disseminação inter e intra-celular.

Conforme discutido no capítulo 3, o DIMI apresenta uma arquitetura bem definida, composta de elementos distribuídos que se adequam à estrutura do ISAM*pe*, levando em conta seus tipos de relacionamentos P2P e *proxy*. O acesso dos usuários às funcionalidades dessa arquitetura, inclusive em casos de exceção, foram claramente definidos na seção 3.5. Também foram claramente definidos as principais classes do modelo conceitual e os mecanismos funcionais do DIMI, que contêm os meios através dos quais o serviço alcança seus objetivos, definidos na seção 3.1. No ISAM*pe*, definiu-se a integração do DIMI com o serviço de descoberta de recursos PerDiS e os ganhos para os usuários resultantes dessa integração O DIMI mescla soluções próprias e soluções de trabalhos relacionados para lidar com problemas existentes no *multicast* no nível de aplicação, como a criação e a manutenção da topologia de disseminação, o possível surgimento de ciclos de disseminação e a definição de métricas de distância entre os participantes de um canal.

Conforme discutido no capítulo 5, o DIMI satisfaz as necessidades da arquitetura ISAM como os trabalhos relacionados não são capazes de fazer. O serviço, em relação aos trabalhos relacionados, além das diferenças mostradas na seção 5.8, apresenta maior grau de parametrização, visando o funcionamento de acordo com as necessidades dos usuários. Os parâmetros do DIMI são: (**i**) as definições das métricas de distância e dos filtros dos canais; (**ii**) a consumo de CPU e memória a ser observado durante o funcionamento de uma UD em um dispositivo disseminador; (**iii**) o comportamento do PEP de um canal em caso de estouro de *buffer*; e (**iii**) o período de envio de mensagens de aviso de aviso de vida (*HEART-BEAT*) em cada UD do serviço.

O desenvolvimento do DIMI até agora resultou em cinco publicações de abrangências local e nacional, das quais quatro o autor desta dissertação foi o primeiro autor e de uma o mesmo foi co-autor. Essas publicações são as seguintes:

GEYER, C. F. R.; SILVA, L. C.; YAMIN, A. C.; AUGUSTIN, I.; SCHAEFFER FILHO, A. E.; MORAES, M. C.; REAL, R. A.; FRAINER, G.; PIRES, R.. *GRADEp*: *Towards Pervasive Grid Executions*. In: **III Workshop de Grade Computacional e Aplicações (WGCA)**, 2005, Petrópolis, 2005.

MORAES, M. C.; GEYER, C. F. R.. *DIMI: Disseminador Multicast de Informações para Ambientes de Computação Pervasiva*. In: **V Escola Regional de Alto Desempenho (ERAD),** 2005, Canoas, 2005. p. 99-100.

MORAES, M. C.; SILVA, L. C.; SCHAEFFER FILHO, A. E.; YAMIN, A. C.; AUGUSTIN, I.; SILVA, G. F.; GEYER, C. F. R..*Disseminando Informações na Arquitetura ISAM*. In: **Seminário Integrado de Software e Hardware (SEMISH)** (aceito para publicação), 2005, São Leopoldo, 2005.

MORAES, M. C.; SILVA, L. C.; YAMIN, A. C.; AUGUSTIN, I.; GEYER, C. F. R.. *Técnicas de Disseminação de Informações de Monitoramento em Sistemas Largamente Distribuídos.* In: **II Workshop de Processamento Paralelo e Distribuído (WSPPD)**, 2004, Porto Alegre. Evangraf, 2004. p. 131-136.

MORAES, M. C.; SILVA, L. C.; YAMIN, A. C.; AUGUSTIN, I.; SILVA, G. F.; GEYER, C. F. R.. *DIMI: um Disseminador de Informações para a Arquitetura ISAM* In: **I Workshop de Peer-to-Peer (WP2P)**, 2005, Fortaleza, 2005).

6.1 Trabalhos Futuros

Apesar de apresentar um modelo rico em funcionalidades e uma arquitetura bem definida, o serviço DIMI ainda possui melhoramentos a serem realizados. Esses melhoramentos podem ser efetuados tanto em seu modelo quanto em seus resultados. Os trabalhos futuros identificados são os seguintes:

- **criar canais com mútiplos produtores**: a possibilidade de existência de muitos produtores em um canal é importante para alguns tipos de aplicações, como por exemplo vídeo-conferências e jogos multi-jogadores. A característica de múltiplos produtores não foi encorporada no modelo do DIMI pois não foi encontrada uma solução escalável para essa necessidade. Uma solução possível, proposta no TAO (KWON; FAHMY, 2002) e no HMTP (ZHANG; JAMIN; ZHANG, 2002) e adaptada no DIMI para o suporte à movimentação do produtor, é a transmissão *unicast* dos diversos produtores para a raiz do canal, onde é feita a disseminação. Essa solução, no entanto, cria um gargalo na raiz do canal nos casos em que houver muitos produtores. Essa solução não é satisfatória para a arquitetura ISAM, e, por isso, não foi utilizada no DIMI;

- **utilizar IP Multicast para a disseminação intracelular**: o DIMI poderia realizar a disseminação intracelular fazendo uso das diversas *ilhas* IP Multicast existentes hoje na Internet. Uma *ilha* IP Multicast é uma rede de qualquer tamanho que suporte o IP Multicast sem necessidade de quaisquer tipos de ajustes. Dentro de uma

dessas *ilhas*, o IP Multicast nativo poderia ser utilizado para a disseminação de dados em um canal do DIMI. A utilização de IP Multicast economizaria recursos e aumentaria o desempenho da disseminação pois as UDbases não teriam necessidade de manter conexões TCP com as UDnodes de sua célula.

– **complementar as simulações discutidas no capítulo 4 com a ntegração do DIMI com o serviço de descoberta de recursos da arquitetura ISAM e com o acréscimo da manutenção dos canais, discutida na seção 3.7.7**: a integração do DIMI com o serviço de descoberta levaria as UDs do DIMI a encontrar pontos de entrada mais próximos de si no canal desejado, o que teria a consequência de construir a topologia de disseminação de forma mais balanceada, resultando em melhores tempos de disseminação de mensagens. A manutenção dos canais, por sua vez, também contribuiria para a melhoria dos tempos de disseminação, já que novas conexões entre UDs próximas tenderiam a ser criadas em tempo de execução.

– **complementar o protótipo com as características que ainda não foram implementadas**: seria importante para a obtenção de novos resultados se fossem implementadas e testadas as características do modelo do DIMI que ainda estão ausentes no atual estágio do protótipo. Essas características são: (**i**) a priorização dos canais; (**ii**) o gerenciamento do tamanho dos *buffers* de mensagens; (**iii**) a manutenção dos canais; (**iv**) a consciência do contexto; e (**v**) o suporte à mobilidade;

– **integrar o DIMI com o serviço Gateway do EXEHDA**: se os nomes das UDs (UDbases e UDnodes) forem construídos com o identificador global de nó de processamento, o *HostId*, disponível no EXEHDA, e for feito o uso do serviço Gateway do EXEHDA, que faz a intermediação das comunicações entre os nós externos à célula e os recursos internos a ela (YAMIN, 2004), é possível fazer com que as UDs participem da disseminação mesmo quando localizadas em redes que utilizem *Network Address Translator* (NAT). A capacidade de funcionamento em redes que utilizam NAT é uma característica desejável em um serviço de disseminação *multicast* no nível de aplicação (GANJAM; ZHANG, 2004);

– **definir mecanismos que ofereçam garantias de privacidade e autenticação dos dados disseminados**: garantias de que os dados consumidos em um canal do DIMI não foram alterados ou observados por terceiros são desejáveis em alguns casos. O oferecimento dessas garantias tornaria o DIMI atraente para um número maior de aplicações.

REFERÊNCIAS

ADACHI, F. Evolution Towards BroadBand Wireless Systems. In: INTERNACIONAL SYMPOSIUM ON WIRELESS PERSONAL MULTIMEDIA COMMUNICATIONS, 5., 2002. **Proceedings…** [S.l.:s.n.], 2002. v.1, p. 19-26.

ALMEROTH, K. The evolution of multicast: from the Mbone to interdomain multicast to Internet2 deployment. In: IEEE NETWORK, 2000. **Proceedings…** [S.1.:s.n.], 2000. p. 10-20.

ANTONOIU, G.; SRIMANI. P.K. Distributed Self-Stabilizing Algorithm for Minimum Spanning Tree Construction. In: EUROPEAN CONFERENCE ON PARALLEL PROCESSING, 1997. **Proceedings…** [S.1.:s.n.], 1997. p. 480-487.

AUGUSTIN, I. **Abstrações para uma linguagem de programação visando aplicações móveis em um ambiente de *pervasive computing***. 2004. 194 p. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

AURA. **Distraction Free Ubiquitous Computing**. Disponível em: <http//www-2.cs.cmu.Edu/aura/>. Acesso em: maio 2005.

BANERJEE, S.; BHATTACHARJEE, B.; KOMMAREDDY, C. Scalable application layer multicast. In: APPLICATIONS, TECHNOLOGIES, ARCHITECTURES, AND PROTOCOLS FOR COMPUTER COMMUNICATIONS, 2002. **Proceedings…** New York: ACM Press, 2002. p. 205-217.

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

BRASIL. Receita Federal. Disponível em: <http//www.receita.fazenda.gov.br>. Acesso em: maio 2005.

CHEN, G.; KOTZ, D. **A Survey of Context-Aware Mobile Computing Research**. (Technical Report TR2000-381). Disponível em: <http//www.cs.dartmouth.ed/solar/>. Acesso em: maio 2005.

CHOCKLER, G. V.; KEIDAR, I.; VITENBERG, R. Group communications specifications: a comprehensive study. **ACM Computing Surveys**, [S.1.], v. 33, n. 4, p. 427-469, 2001.

CHU, Y.H.; RAO, S.G.; ZHANG, H. A case for end system multicast. In: SIGMETRICS INTERNATIONAL CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 2000. **Proceedings…** New York: ACM Press, 2000. p. 1-12.

DEERING, S.E.; CHERITON, D.R. Multicast routing in datagram internetworks and extended LANs. **ACM Computing Surveys**, [S.1.], v. 8, n. 2, p. 85-110, 1990.

DIFF. **Comparing and Merging Files**. Disponível em: <http//www.gnu.org/software/diffiutils/manual/diff.html,>. Acesso em: maio 2005.

DNS. **Domain Name System**. Sítio na Internet. Disponível em <http//www.scit.wlv.ac.uk/jphb/comms/dns.html,>. Acesso em: maio 2005.

EL-SAYED, A.; ROCA, V.; MATHY, L. A Survey of Proposals for an Alternative Group Communication. **IEEE Network**, [S.1.], v. 17, p. 46-51, 2003.

EXEHDA. **Execution Environment for Highly Distributed Applications**. Disponível em: <http//www.inf.ufrgs.br/~exehda>. Acesso em: maio 2005.

FORUM G.G. **Grid Computing Environments Working Group**. Disponível em: <http//www.gridforum.org/>. Acesso em: maio 2005.

FOSTER, I.; KESSELMAN, C. (Ed.). **The Grid**: Blueprint for a New Computing Infrastructure. San Francisco, CA: Morgan Kaufmann, 1999.

FRANCIS, P. **Yoid**: extending the internet multicast architecture. Disponível em: <http//www.aciri.org/yoid/docs/index.html>. Acesso em: maio 2005.

GAIA. **Active Spaces for Ubiquitous Computing**. Disponível em: <http//www.gaia.cs.uiuc.edu/>. Acesso em: maio 2005.

GAMMA, E. et al**. Design Patterns**. [S.l.]: Addison-Wesley, 2003.

GANJAM, A.; ZHANG, H. Connectivity restrictions in overlay multicast. In: NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, NOSSDAV, 14., 2004. **Proceedings…** New York: ACM Press, 2004. p. 54-59.

GZIP. **GNU ZIP**. Disponível em: <http//www.gzip.com>. Acesso em: maio 2005.

HAPNER, M. et al. **Java Message Service**. Palo Alto, CA: [s.n.], 2002.

INET. **Autonomous System (AS) Level Internet Topology Generator**. Disponível em: <http//www.topology.eecs.umich.Edu/inet/>. Acesso em: maio 2005.

IP: Internet Protocol. Disponível em: <http//www.freesoft.org/CIE/ Topics/79.htm>. Acesso em: maio 2005.

ISAM: Infraestrutura de Suporte a Aplicações Móveis. Disponível em: <http//www.inf.ufrgs.br/~isam>. Acesso em: maio 2005.

JANNOTTI, J. et al. Overcast: Reliable Multicasting with an Overlay Network. In: SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLMENTATION, OSDI, 4., 2000. **Proceedings…** [S.l. : s.n.], 2000.

KWON, M.; FAHMY, S. Topology-aware overlay networks for group communication. In: NETWORK AND OPERATINGS SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, 12., 2002. **Proceedings…** New York: ACM Press, 2002. p. 127-136.

MEDINA, A. et al. **Universal Topology Generation from a User's Perspective**. [S.l.], 2001. Disponível em: <citeseer.ist.psu.Edu/medina01brite.html>. Acesso em: maio 2005.

MONARC-2: Models of Networked Analysis at Regional Centers. Disponível em: <http//www.monarc.cacr.caltech.Edu:8081/monarc/monarc.html>. Acesso em: maio 2005.

MSNBC**. Microsoft Today's News**. Disponível em: <http//www.msnbc.msn.com>. Acesso em: maio 2005.

NS-2: Network Simulator-2. Disponível em: <http//www.isi.Edu/nsnam/ns>. Acesso em: maio 2005.

NTP: Network Tome Protocol. Sítio na Internet. Disponível em: <http//www.ntp.org/>. Acesso em: maio 2005.

ORAM, A. **Peer-to-peer**: Harnessing the Benefits of a Disruptive Technology. [S.l.]: O'Reilly & Associates, 2001.

O'SULLIVAN, D.; LEWIS, D. Semantically driven service interoperability for pervasive computing. In: ACM INTERNACIONAL WORKSHOP ON DATA ENGINNERING FOR WIRELESS AND MOBILE ACCESS, MOBILE, 3., 2003. **Proceedings…** New York: ACM PREES, 2003. p. 17-24.

OUSTERHOUT, J. K. **An Introduction to Tcl and Tk**. Reading, MA, USA: Addison-Wesley, 1993.

PADMANABHAN, V.N. et al. Distributing streaming media content using cooperative networking. In: INTERNATIONAL WORKSHOP ON NETWORK AND OPERATING SYSTEMS SUPORT FOR DIGITAL AUDIO AND VIDEO, 12., 2002. **Proceedings…** New York: ACM Prees, 2002. p. 177-186.

PANDARAKIS, D. et al. An Application Level Multicast Infrastructure. In: USNIX SYMPOSIUN ON INTERNET TECHNOLOGIES AND SYSTEMS, USITS, 3., 2001. San Francisco, CA. USA. **Proceedings…** [S.l. : s.n.], 2001. p. 49-60.

PATCH. **Comparing and Meging Files**. Disponível em: <http//www.gnu.org/software/diffutils/manual/diff.html>. Acesso em: maio 2005.

SADOK, D.; KELNER, J.; CORDEIRO, C. Disconnection Protocol Support in Mobile Access**. Journal of the Brazilian Society,** [S.l.], v. 5, n. 3, Feb. 1999.

SAHA, D.; MUKHERJEE, A. Perspectives: Pervasive Computing: a Paradigm for the 21 st Century. **Computer**, [S.l.], v. 36, n. 3, p. 25-31, Mar. 2003.

SATYANARAYANAN, M. **Pervasive Computing**: Vision and Challenges. Disponível em: <http//www.citeseer.ist.psu.Edu/490127.html>. Acesso em: maio 2005.

SCHAEFFER FILHO, A. E. et al. PerDis: a Scalable Resource Discovery Service for the ISAM Pervasive Environment. In: INTERNACIONAL WORKSHOP ON HOT TOPICS IN PEER-TO-PEER SYSTEMS, 2004, Voledam, The Nederlands. **Proceedings…** [S.l.]: IEEE Computer Society, 2004. p. 80-85.

SLASHDOT. **News for Nerds. Stuff that matters**. Disponível em: <http//www.slashdot.org/>. Acesso em: maio 2005.

STANFORD, V. Pervasive computing goes the last hundred feet with RFID systems. **Pervasive Computing,** Los Alamitos**,** v. 2, n. 2, p. 9-14, 2003.

SUN. **Java Remote Method InvocationTecnology**. Sítio na Internet. Disponível em <http//www.java.sun.com/products/jdk/rmi>. Acesso em: maio 2005.

SUN. **Java Tecnology**. Disponível em: <http//www.java.sun.com/>. Acesso em: maio 2005.

SUN. **JavaBeans Technology.** Disponível em: <http//www.java.sun.com/products/javabeans>. Acesso em: maio 2005.

TPC: Transmission Control Protocol. Disponível em: <http//www.freesoft.org/CIE/RFC/793/index.htm>. Acesso em: maio 2005.

UDP: User Datagram Protocol. Disponível em: <http//www.freesoft.org/CIE/RFC/768/index.htm>. Acesso em: maio 2005.

UML: Unified Modeling Language. Disponível em: <http//www.uml.org>. Acesso em: maio 2005.

URBÁN, P.; DÉFAGO, X.; SCHIPER, A. Neko. Single Environment to Simulate and Prototype Distributed Algorithms. In: CONF. ON INFORMATION NETWORKING, ICOIN, 15., 2001, Beppu City, Japan. **Proceedings…** [S.l.: s.n.], 2001. Best Student Paper Award.

WALLACE, C. **The semantics of the C++ programming Language.** [S.1.]: Oxford University Press, 1994.

WANG, X. et al. Semantic Space: an Infraestructure for Smart Spaces. In: PERVASIVE COMPUTING, 2004. **Proceedings…** [S.l.: s.n.], 2004.

WAXMAN, B.M. Routing of Multipoint Connections. IEEE JSA. **Communications (Special Issue: Broadband Packet Communications),** [S.l.], v. 6, n. 9, p. 1617-1622, Dec. 1988.

WETHERALL, D. **MIT Object Tcl**. 1995. Disponível em: < ftp://ftp.tns.lcs.mit.edu/pub/otcl/ >. Acesso em: maio 2005.

XML: Extensible Markup Language. Sítio na Internet. Disponível em <http//www.sml.org>. Acesso em: maio 2005.

YAMIN, A. **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.

YAU, S.; AHAMED, S. An Approach to Developing Information Dissemination Service for Ubiquitous Computing Applications. In: INTERNACIONAL SYMPOSIUM ON AUTONOMOUS DECENTRALIZED SYSTEMS, 6., 2003. **Proceedings…** [S.l.:s.n.], 2003. p. 240-247.

ZHANG, B.; JAMIN, S.; ZHANG, L. Host multicast: a framework for delivering multicast en users. In: IEEE Infocom, 2002. **Proceedings…** [S.l.:s.n.], 2002.

ZIP. **Winzip**. Sítio na Internet. Disponível em <http//www.winzip.com>. Acesso em: maio 2005.