UNIVERSIDADE DO VALE DO RIO DOS SINOS
CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

Daniel Torres Bonatto

HNS: Uma Solução para Suporte à Execução Distribuída Considerando Aspectos da Pervasividade

Sao Leopoldo
2006

Daniel Torres Bonatto

HNS: Uma Solução para Suporte à Execução Distribuída Considerando Aspectos da Pervasividade

Dissertação submetida a avaliação como requisito parcial para a obtenção do grau de Mestre em Computação Aplicada

Orientador: Prof Dr. Jorge Luis Victória Barbosa

São Leopoldo
2006

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Bonatto, Daniel Torres

HNS: Uma Solução para Suporte à Execução Distribuída Considerando Aspectos da Pervasividade
/ por Daniel Torres Bonatto. — São Leopoldo: Ciências Exatas e Tecnologicas da UNISINOS, 2006.

98 f.: il.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos. Ciências Exatas e Tecnológicas Programa de Pós-Graduação em Computação Aplicada, São Leopoldo, BR–RS, 2006. Orientador: Barbosa, Jorge Luis Victória

1. Computação Móvel. 2. Computação Pervasiva. 3. Ambiente de Execução. 4. Sistema de nomes. 5. Holoparadima. I. Barbosa, Jorge Luis Victória. II. Título.

“Os problemas jamais podem ser resolvidos usando
os mesmos padrões de pensamento que os criaram.”
Albert Einstein

Agradecimentos

Aos meus orientadores, Prof. Jorge Luis Victória Barbosa e Prof. Gerson Geraldo H. Cavalheiro, pelo incentivo, dedicação, paciência e amizade, os quais foram imprescindíveis para a conclusão desta dissertação.

Aos demais professores do Programa de Pós-Graduação em Computação Aplicada (PIPCA), pelo conhecimento transmitido durante esta caminhada, meu muito obrigado.

Ao Prof. Adenauer Correa Yamin, por sua amizade e por seus concelhos que tantas vezes apontaram os caminhos a serem seguidos nesta vida.

A minha namorada, Renata, por seu amor, carinho e principalmente pelo ser compreensiva com meus momentos de ausência e de *stress*, TE AMO MUITO!

A meu pai Ademar e minhas mães Ceres, Almira e Cleusa, a todos meu amor, carinho e meu agradecimento por todo apoio que recebi.

A minha irmã Mônica, pelos momentos de descontração, pela preocupação e pelo amor, também es parte essencial desta caminhada, é muito bom te ter por perto.

Aos ótimos amigos que fiz durante o mestrado, em especial Fernando Cáprio, Dario Franz e Marcelo Cardozo, por que os momentos passam, mas bons amigos são eternos.

Aos bolsistas e amigos Solon e Felipe K. por sua participação direta no desenvolvimento deste trabalho, meu muito obrigado.

A um grande amigo Otávio, que por um acaso do destino voltou para a convivência cotidiana, pela ajuda, por estar sempre disposto a ouvir reclamações e pela amizade, meu muito obrigado.

Aos colegas de Mobilab, pela ajuda na realização de meus objetivos e pelos inúmeros momentos de descontração, com certeza sentirei saudades deste ambiente.

A Hewlett-Packard Computadores (HP), pela concessão da bolsa de mestrado e apoio financeiro.

Resumo

Nos últimos anos, tem-se observado a crescente evolução dos dispositivos portáteis, bem como de diversas novas tecnologias de comunicação sem fio. Esse avanço tecnológico propicia o surgimento de um cenário ideal para o desenvolvimento de ambientes que suportam a criação de aplicações pervasivas. Porém, um ambiente altamente dinâmico como este demanda a utilização de abstrações mais poderosas do que as existentes. O Holoparadigma propõe uma nova abstração, criada pensando em aplicações distribuídas executando em ambientes móveis. Nesta dissertação é apresentada a proposta para uma arquitetura de suporte a aplicações pervasivas para o Holoparadigma. Esta proposta estende as funcionalidades da HoloVM e define novos serviços para atender às demandas da computação pervasiva. Dentre estes serviços é definido um como sendo essencial para a arquitetura, que é o suporte à distribuição, composto por um servidor de nomes e uma camada para possibilitar a execução distribuída e transparente de programas. Para o modelo do servidor de nomes é definida uma estratégia de distribuição escalável e tolerante a falhas, conforme os princípios da computação pervasiva. Além disso, são mostrados resultados de experimentos realizados com este suporte.

Palavras-chave: Computação Móvel, Computação Pervasiva, Ambiente de Execução, Sistema de nomes, Holoparadima.

TITLE: “HSN: A Solution for Supporting Distributed Execution Considering Pervasiveness Aspects”

Abstract

Over the last few years, we have observed the growing evolution of portable devices, such as new technologies for wireless communication. This technological advance makes possible the emergence of an ideal scenery for developing environments supporting the creation of pervasive applications. However, such a highly dynamic environment demands the use of more powerful abstractions than those available today. Holoparadigm proposes a new form of abstraction, created aiming distributed applications running on mobile environments. In the present dissertation, we propose an architecture for Holoparadigm designed to support pervasive applications. This proposal extends the functionalities of HoloVM and defines new services to respond to the demand of pervasive computing. Among those services, one is defined as essential to the architecture, the support for distribution, which is composed by a name server and a layer for supporting distributed and transparent execution of programs. For the name server, a scalable and fault tolerant distribution strategy is defined, following the principles of pervasive computing. Furthermore, we show the results of experiments performed using this support.

Keywords: Mobile Computing, Pervasive Computing, Execution Environment, Naming System and Holoparadigm.

Sumário

Resumo **6**

Abstract **7**

Lista de Abreviaturas **11**

Lista de Figuras **13**

Lista de Tabelas **15**

1 Introdução **16**
1.1 Definição do Problema 17
1.2 Objetivos 18
1.3 Metodologia 18
1.4 Organização da Dissertação 19

2 Computação Pervasiva **20**
2.1 Áreas Relacionadas 20
2.1.1 Sistemas Distribuídos 21
2.1.2 Computação Móvel 21
2.2 Desafios 22
2.2.1 Elevada Heterogeneidade 22
2.2.2 Mobilidade Lógica e Física 23
2.2.3 Disponibilidade de Serviços e Dados 23
2.2.4 Adaptação (Sistema e Aplicação) 24
2.3 Sistemas para Suporte à Computação Pervasiva 24
2.3.1 Aura 24
2.3.2 Gaia 26
2.3.3 one.world 27
2.3.4 ISAM 29
2.4 Conclusão 30

3 Projeto *Mobile Holoparadigm* (MHolo) 32
3.1 Holoparadigma . . . 32
3.2 Ambiente de Execução MHolo . . . 35
3.3 Pervasive Holoparadigm (PHolo) . . . 36
3.3.1 Arquitetura Proposta . . . 37
3.3.2 Localização Física . . . 37
3.3.3 Contexto . . . 38
3.3.4 Tipos de Mobilidade . . . 39
3.3.5 Descoberta de Serviços . . . 40
3.4 Conclusão . . . 41

4 Trabalhos Relacionados 42
4.1 Sistemas de Nomes . . . 42
4.1.1 Domain Name System (DNS) . . . 42
4.2 Tabelas Hash Distribuídas (DHT) . . . 43
4.2.1 Content-Addressable Network (CAN) . . . 44
4.2.2 Tapestry . . . 45
4.2.3 Chord . . . 46
4.2.4 Kademlia . . . 47
4.3 Descoberta de Serviços . . . 48
4.3.1 SLP . . . 48
4.3.2 Jini . . . 50
4.3.3 UPnP . . . 50
4.3.4 Bonjour . . . 51
4.4 Conclusões . . . 51

5 Holo Naming System (HNS) 54
5.1 Objetivos do Modelo . . . 54
5.2 Modelo HNS . . . 55
5.2.1 Estratégia de distribuição dos servidores . . . 56
5.2.2 Suporte à execução distribuída . . . 57
5.3 Adaptando o sistema de DHT à abordagem do HNS . . . 59
5.3.1 Formato do registro . . . 59
5.3.2 Mensagens relacionadas ao funcionamento dos servidores . . . 60
5.4 Comunicação entre HoloVMs . . . 62
5.4.1 Acesso a história de um ente pai . . . 62
5.4.2 Executando ações de outros entes . . . 63
5.5 Integrando um sistema de descoberta de recursos ao modelo . . . 64
5.6 Detectando e contornando falhas nos componentes do sistema . . . 65
5.6.1 Contornando a falha de um servidor HNS . . . 66

5.6.2 Atualizando o sistema sobre a falha de uma HoloVM 67
5.7 Técnicas para otimização do sistema 67
5.7.1 Cache de consultas 67
5.7.2 Cache proativa 68
5.8 Sugestões para melhorar a implementação da Hololinguagem 68
5.8.1 Execução distribuída 68
5.8.2 Tratamento de exceções na linguagem 69
5.8.3 Segurança em nível de linguagem 69
5.9 Conclusão 70

6 Aspectos de implementação e resultados obtidos 71
6.1 Tecnologias utilizadas 71
6.1.1 GNUNet - Kademlia 72
6.1.2 Howl - Bonjour 73
6.2 Aspectos de implementação do sistema 73
6.2.1 Módulo HNS para o GNUnet 73
6.2.2 Comunicação entre HoloVM e HNS 75
6.2.3 Suporte à comunicação entre HoloVMs 75
6.2.4 Sistema de auto-configuração 77
6.2.5 Tarefas de implementação 78
6.3 Modelagem de aplicações Holo e o suporte a distribuição do HNS 78
6.4 Experimento previo: O History Server 80
6.4.1 Troca de mensagens 81
6.4.2 Testes: HS x HNS 82
6.5 Simulação para estudo da escalabilidade 84

7 Conclusão e trabalhos futuros 87
7.1 Conclusão 87
7.2 Contribuições 88
7.3 Trabalhos futuros 89

Bibliografia 91

Lista de Abreviaturas

API	Aplication Program Interface
CAN	Content-Addressable Network
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Tables
DNS	Domain Name System
DNS-SD	DNS Service Discovery
DSM	Distributed Shared Memory
DSO	Distributed Shared Objects
HNS	Holo Naming System
HOLO	Holoparadigma
HS	History Server
HVM	Holo Virtual Machine
INS	Intetional Name System
IP	Internet Protocol
ISAM	Infra-estrutura de Suporte as Aplicações Móveis Distribuídas
JVM	Java Virtual Machine
KS	Knownledge Source
MHOLO	Mobile Holoparadigm
NAT	Network Address Translation
P2P	Peer-to-peer
PDA	Personal Digital Assistent
PHOLO	Pervasive Holoparadigm

RMI Remote Method Invocation

RPC Remote Procedure Call

SD Service Discovery

SHA1 US Secure Hash Algorithm 1

SLP Service Location Protocol

SO Sistema Operacional

SSDP Simple Service Discovery Protocol

TDS True Distributed System

TTL Time To Live

UPnP Universal Plug and Play

VM Virtual Machine

XML Extended Modeling Language

mDNS Multicast DNS

Lista de Figuras

2.1 Evolução da computação em direção à pervasividade 22
2.2 Arquitetura do Aura 25
2.3 Arquitetura do Gaia 26
2.4 Arquitetura do one.world 28
2.5 Arquitetura do ISAM 29

3.1 Entes Elementar, Composto e Composição em 3 níveis 33
3.2 Mobilidade no Holoparadigma 34
3.3 Modelo de coordenação no Holo 35
3.4 Árvore de Entes (HoloTree) 35
3.5 Proposta para a arquitetura do PHolo. 38
3.6 Exemplo de mobilidade lógica. 40

4.1 Representação da topologia da aplicação em relação a topologia da rede 44
4.2 Representação da topologia do CAN 45
4.3 Visão de um nodo da tabela de roteamento do Tapestry 46
4.4 Anel de roteamento do Chord 47
4.5 Árvore binária do Kademlia 47
4.6 SLP atualizando um DA 49

5.1 Modelo HNS 56
5.2 Comparação da execução remota de código entre Holo e RPC. 58
5.3 Dados armazenados em cada registro nos servidores HNS 60
5.4 Troca de mensagens no gerenciamento da base de nomes 61
5.5 Formas de executar ações que são permitidas pela linguagem. 63
5.6 Processo de descoberta com o Bonjour. 65
5.7 Exemplo de falha em HNS. 66

6.1 Arquitetura do GNUNet 72
6.2 Diagrama de seqüência para a comunicação entre HoloVMs 76
6.3 Modelagem do XML utilizado para as mensagens 76
6.4 Exemplo dos possíveis campos de uma mensagem XML entre HoloVMs. . . . 77

6.5 Exemplo de uma aplicação Holo dividida em camadas 80
6.6 Figura ilustrando funcionamento do HS 81
6.7 Mensagem entre HoloVM e HS. 82
6.8 Teste de escrita de tuplas na história 83
6.9 Testes de clonagem de entes 84
6.10 Impacto do HNS na escalabilidade do Kademlia 85

Lista de Tabelas

4.1 Comparação da complexidade de operações entre sistemas DHT 52

6.1 Tarefas de implementação 78
6.2 Resultados do teste de inicialização da HoloVM (40 execuções) 83

Capítulo 1

Introdução

Na última década, os dispositivos computacionais vêm diminuindo consideravelmente em tamanho e custo. Um exemplo disso são equipamentos como PDAs, *laptops* e *tablet* PCs, cada vez mais populares em número de usuários e em opções de mercado. A evolução acelerada deste tipo de tecnologia conduz para um mundo onde o cotidiano é cada vez mais permeado por dispositivos computacionais.

A primeira menção a um mundo como o que começamos a ver hoje foi feita por Mark Weiser em 1991 [1]. De forma visionária Weiser descreveu um mundo onde os ambientes cheios de dispositivos computacionais e comunicação, interagiam naturalmente com as pessoas, de tal forma que passavam a fazer parte deste ambiente. Dentro desta visão surgiu o termo *computação ubíqua*, hoje também chamada de *computação pervasiva* [2, 3]. Contudo, em sua época, Mark e seus colegas não tinham a tecnologia para implementar esta visão.

Para que a computação pervasiva se torne uma realidade, ainda existem alguns desafios a serem vencidos. Na última década, desde que Weiser concebeu sua visão de pervasividade, evoluções importantes no hardware podem ser constatadas. Isto permitiu a criação de dispositivos menores e por tanto mais portáteis, bem como sensores e dispositivos de controle. É importante citar ainda as tecnologias para comunicação sem fio, como o Bluetooth [4] e o IEEE 802.11 [5], que criam a possibilidade de acesso a informação a qualquer hora em qualquer lugar, o que não era possível com as redes cabeadas.

Este cenário possibilita o surgimento de sistemas que tem por objetivo criar o suporte de execução para o desenvolvimento de ambientes que tiram proveito de aspectos da computação pervasiva. Como alguns exemplos disso, tem-se: Aura [6, 7, 8], Gaia [9, 10, 11], One-world [12, 13, 14, 15] e ISAM [16, 17, 18].

A existência de tantos projetos de grande porte mostra que existe um caminho promissor a ser seguido por todos que visam desenvolver sistemas para execução distribuída em larga escala. Como será mostrado posteriormente (capítulo 2.3), ambientes de suporte a pervasividade são por si só softwares complexos, que possuem vários componentes e que demandam um grande esforço de implementação. Neste sentido, o presente trabalho tem como um de seus objetivos fazer um estudo sobre pervasividade e sistemas pervasivos e como resultado propor um caminho

a ser seguido pelo projeto MHolo (seção 3), dentro do qual este trabalho foi desenvolvido. O projeto MHolo objetiva o estudo e a criação de um suporte à execução para ambientes móveis. O resultado do estudo sobre pervasividade consiste em fazer com que o ambiente de execução do MHolo passe a oferecer suporte à pervasividade para suas aplicações. Esta proposta consiste em uma arquitetura que segue os moldes dos sistemas pervasivos modernos, porém direcionada para os requisitos do projeto.

De posse desta arquitetura o passo seguinte é escolher um dos componentes da mesma, e sobre este fazer um levantamento mais detalhado de requisitos para criar então uma implementação do sistema. Neste sentido a escolha feita foi o módulo da arquitetura que oferece suporte à execução distribuída. Este sistema tem como objetivo prover um mecanismo para que entidades de *software* em execução em máquinas distribuídas em uma rede possam ser localizadas e para que possam se comunicar, preferencialmente da mesma forma que fariam se estivessem executando na mesma máquina.

1.1 Definição do Problema

O surgimento da computação pervasiva traz uma série de novos desafios. Alguns destes sendo problemas já abordados e solucionados anteriormente nas áreas de computação distribuída e computação móvel. Porém, a computação pervasiva é considerada uma revolução [19] na pesquisa feita na área de sistemas distribuídos, de forma que nem todas as soluções já conhecidas podem ser diretamente aplicadas e alguns problemas são completamente novos, não possuindo nenhuma abordagem já estudada.

Um ambiente pervasivo demanda que as aplicações sejam adaptativas por natureza [2, 3]. Neste sentido, torna-se interessante incluir novos elementos lógicos (mobilidade de código) em tempo de execução, ganhando assim funcionalidade para atender as demandas do ambiente. Elas devem também ser sensíveis ao contexto, ou seja, estar conscientes de modificações e adaptar o seu comportamento. A questão da adaptação é importante também quando se fala da necessidade de suportar a heterogeneidade de um ambiente pervasivo. Neste ambiente existe uma grande variedade de dispositivos e redes de comunicação, de forma que a aplicação deve utilizar os recursos disponíveis da melhor forma possível. Outra questão importante é a disponibilidade de dados e serviços, contornando desconexões e falhas em partes da aplicação.

A forma de programação e composição da aplicação é uma questão importante. A maioria das soluções existentes atualmente utilizam-se de paradigmas já consagrados para implementar suas soluções, contudo estes paradigmas foram concebidos para criar aplicações que executam em um único nodo. Desta forma, tais paradigmas são muito pouco expressivos para modelar aplicações para um ambiente tão dinâmico quanto o que é apresentado na computação pervasiva. Esse fator motiva o desenvolvimento de novas abstrações para programação, bem como a implementação do respectivo suporte.

O Holoparadigma [20] oferece uma abstração intuitiva para a modelagem de ambientes

móveis. É possível criar uma modelagem mais fiel do mundo real, e utilizando diretivas de mobilidade do paradigma, é possível manter o modelo coerente com aplicações reais. Esta modelagem aplica-se tanto a elementos fixos (prédios, salas, computadores, etc.) quando elementos móveis (*laptops*, PDAs, celulares, etc.). Contudo, o ambiente de execução que vinha sendo utilizado no projeto MHolo não possuía suporte para a execução distribuída de aplicações, suprimindo assim um grande potencial do Holoparadigma.

Dentro deste raciocínio, a criação de um suporte à distribuição passa a ter novos requisitos, criados principalmente pelo fato do trabalho estar situado nas áreas da computação móvel e pervasiva. Tais requisitos estão principalmente em questões como a tolerância a falhas do sistema, não só no sistema de suporte à execução distribuída, mas também os recursos que podem ser oferecidos para a aplicação em execução no sistema, para que ele tolere desconexões e falhas em partes da aplicação. Outra questão é escalabilidade do sistema, pois na pervasividade se assume um ambiente computacional massivo, repleto de aplicações. Um sistema pervasivo, que esteja sempre consciente do estado de todas as entidades de *software* em execução, precisa ser altamente escalável.

1.2 Objetivos

Este trabalho tem como objetivo especificar uma arquitetura que suporte o desenvolvimento de sistemas pervasivos utilizando o Holoparadigma e, usando os requisitos desta arquitetura como guia, especificar, implementar e testar um sistema de suporte à execução distribuída.

Os objetivos específicos deste trabalho são:

- Elencar sistemas pervasivos referenciados na bibliografia;
- Identificar os requisitos básicos para a arquitetura pervasiva;
- Propor, em linhas gerais, uma arquitetura que contemple serviços essenciais para a pervasividade e os requisitos existentes no projeto MHolo;
- Especificar e implementar a localização de entidades de programação, bem como o suporte à execução distribuída destas entidades, levando em conta os requisitos da pervasividade;
- Avaliar a solução proposta.

1.3 Metodologia

Para cumprir os objetivos propostos, foi definida uma metodologia para o trabalho. O primeiro passo foi estudar a computação pervasiva e os sistemas que têm por objetivo oferecer

suporte à execução de aplicações neste ambiente. O principal objetivo neste estudo é verificar como as abstrações de programação são suportadas por cada ambiente, bem como as estratégias de execução distribuída.

No passo seguinte foi feito um estudo sobre o Holoparadigma e suas abstrações. Neste estudo o objetivo é entender todos os recursos que o paradigma oferece, bem como quais tipos de aplicações podem ser desenvolvidas utilizando o mesmo.

Com o conhecimento adquirido sobre ambientes pervasivos e sobre o Holopararadigma, foi apresentada uma arquitetura que contempla aspectos da pervasividade para o ambiente de execução existente no MHolo. Esta arquitetura chama-se PHolo e não objetiva ser uma solução definitiva, mas um passo inicial rumo a pervasividade para o projeto MHolo.

A partir da proposta de arquitetura, foi feito o levantamento de requisitos para a criação de um dos módulos que fazem parte deste sistema. O módulo escolhido é responsável pelo suporte à distribuição e o estudo objetiva atender os requisitos da pervasividade para este módulo. A partir deste estudo é feita a especificação do modelo de distribuição para o PHolo.

A partir do modelo de distribuição, cria-se então um sistema de execução distribuída de aplicações, capaz de manter o controle sobre a estrutura da aplicação em execução e promover a comunicação entre partes da aplicação que se encontram executando em um ambiente distribuído.

Com o objetivo de diminuir a intervenção do usuário para a configuração do sistema, é agregado um serviço para a descoberta automática de recursos em uma rede. Isto permite que os elementos do ambiente de execução se encontrem em uma rede, sem que seja necessária a intervenção do usuário para isso.

1.4 Organização da Dissertação

O restante do trabalho está organizado como segue. No capítulo 2 é feita uma revisão histórica sobre a pesquisa na área de sistemas distribuídos, culminando com a discussão dos principais desafios encontrados hoje na computação pervasiva bem como uma revisão dos principais sistemas pervasivos existentes. No capítulo 3 é apresentado o projeto MHolo e de forma mais detalhada o Holoparadigma, os sistemas que compõem o ambiente de execução do projeto, concluindo com o primeiro resultado deste trabalho. Este resultado é a especificação de uma arquitetura pervasiva para o MHolo, chamada PHolo, e a escolha de um dos serviços desta arquitetura, o HNS, para uma especificação mais detalhada. O capítulo 4 faz uma revisão dos conceitos necessários para a implementação do HNS. No capítulo 5 é apresentado o modelo de um sistema de suporte à execução distribuída para o PHolo, o HNS. No capítulo 6 é mostrada a modelagem do sistema, as tecnologias utilizadas na implementação, um estudo de caso e experimentos realizados. Por fim, no capítulo 7 são feitas as considerações finais deste trabalho.

Capítulo 2

Computação Pervasiva

Em 1991, Mark Weiser, que na época era Chefe do Departamento de Tecnologia no Centro de Pesquisa da Xerox em Palo Alto, descreveu a sua visão da computação no Século 21. Em um artigo publicado na *Scientific American* [1] Weiser escreveu: *"As tecnologias mais significativas são aquelas que se tornam imperceptíveis. Elas se misturam em nosso dia-a-dia até que se tornam indistinguíveis"*. Desta forma ele criou o conceito de *computação ubíqua*, atualmente também conhecida como *computação pervasiva*.

Contudo a visão de Weiser foi um tanto a frente de seu tempo, e na época não se encontrava disponível a tecnologia necessária para criar o ambiente imaginado. Desde então as pesquisas na academia e na indústria focaram-se na evolução tecnológica que permitiria a criação de ambientes pervasivos. Atualmente podem ser constatados grandes avanços no *hardware* existente, culminando em dispositivos cada vez menores e mais poderosos no que diz respeito a processamento e armazenamento de dados. Avanços importantes também foram feitos no que diz respeito a sensoriamento [21, 22] e comunicação sem fio [4, 5]. O avanço tecnológico impulsionou o surgimento de várias pesquisas para a criação de sistemas que tirem proveito de ambientes pervasivos.

Este capítulo discorre sobre temas associados a computação pervasiva e apresenta ambientes que suportam a criação de aplicações pervasivas.

2.1 Áreas Relacionadas

No que diz respeito a pesquisa sobre computação pervasiva, esta área é um marco evolucionário em uma linha que tem suas origens na metade dos anos 70 [2, 3]. Daquela época até hoje, outros dois temas devem ser considerados, são eles: *sistemas distribuídos* (seção 2.1.1) e *computação móvel* (seção 2.1.2). Desta forma, alguns desafios da computação pervasiva já foram estudados previamente. Algumas das soluções encontradas se aplicam diretamente. Em outros casos, as demandas da computação pervasiva são diferentes a ponto de motivarem a busca por novas soluções. Obviamente existem ainda problemas que são inseridos pela computação

pervasiva, para os quais não existe nenhuma solução que possa ser retirada do que já foi desenvolvido.

2.1.1 Sistemas Distribuídos

Com a popularização dos computadores pessoais, surgiu a demanda por uma solução que fosse facilitadora da troca de informações entre computadores distribuídos geograficamente. Isto motiva então o surgimento das tecnologias de redes de computadores, e a computação pessoal, que era constituída por máquinas isoladas, passaram então a ter a possibilidade de usufruir de recursos remotos, surgindo assim a computação distribuída. Este avanço criou a necessidade de um esforço de pesquisa para desenvolver tecnologias que tratam as questões mais relevantes desta área. Pode-se dizer que os principais avanços para a área de sistemas distribuídos foram feitos entre a metade dos anos 70 e o final dos 90. Neste período foi desenvolvido um conjunto de algoritmos e técnicas que provaram-se de grande valor para qualquer trabalho que envolva dois ou mais computadores conectados por uma rede, sejam eles móveis ou estáticos, com ou sem fios, esparso ou pervasivo.

Este corpo de conhecimento inclui muitas áreas que são fundamentais para a computação pervasiva e que já se encontram bem descritas [23, 24, 25]. Dentre estas áreas, pode-se citar: comunicação remota, tolerância a falhas, alta disponibilidade, acesso a dados remotos e segurança.

2.1.2 Computação Móvel

No início dos anos 90 surgiram os computadores portáteis totalmente funcionais, mais conhecidos como *laptops*. Nesta mesma época começaram a ser disponibilizadas soluções para a comunicação sem fios. Estes acontecimentos colocaram os pesquisadores frente aos desafios existentes na criação de aplicações distribuídas para clientes móveis. Desta forma surge a computação móvel. Neste novo campo, muitas das tecnologias desenvolvidas para a computação distribuída continuam aplicáveis, contudo existem quatro aspectos chave que devem ser levados em consideração, são eles: impossibilidade de prever variações na qualidade da rede, menor confiança e robustez de elementos móveis, limitações nos recursos locais impostas por questões que envolvem tamanho e peso, e a necessidade de considerar o consumo de energia [26].

A computação móvel ainda é uma área de pesquisa bastante ativa, e por tanto seus conceitos se encontram em desenvolvimento. Os resultados alcançados até o presente momento podem ser agrupados nas seguintes áreas: redes móveis, acesso a informação móvel, suporte para aplicações adaptativas, técnicas para economia de energia e sensibilidade a localização.

2.2 Desafios

A chegada da computação pervasiva traz consigo novas possibilidades para o desenvolvimento de aplicações. Este aumento no potencial de desenvolvimento se reflete diretamente em um aumento na complexidade dos problemas relacionados a computação distribuída e móvel, bem como em novos desafios exclusivos desta nova área.

Na Figura 2.1 podem ser observados os principais desafios a serem vencidos para que possamos avançar rumo à pervasividade. Nas próximas seções estes tópicos serão abordados de forma mais detalhada.

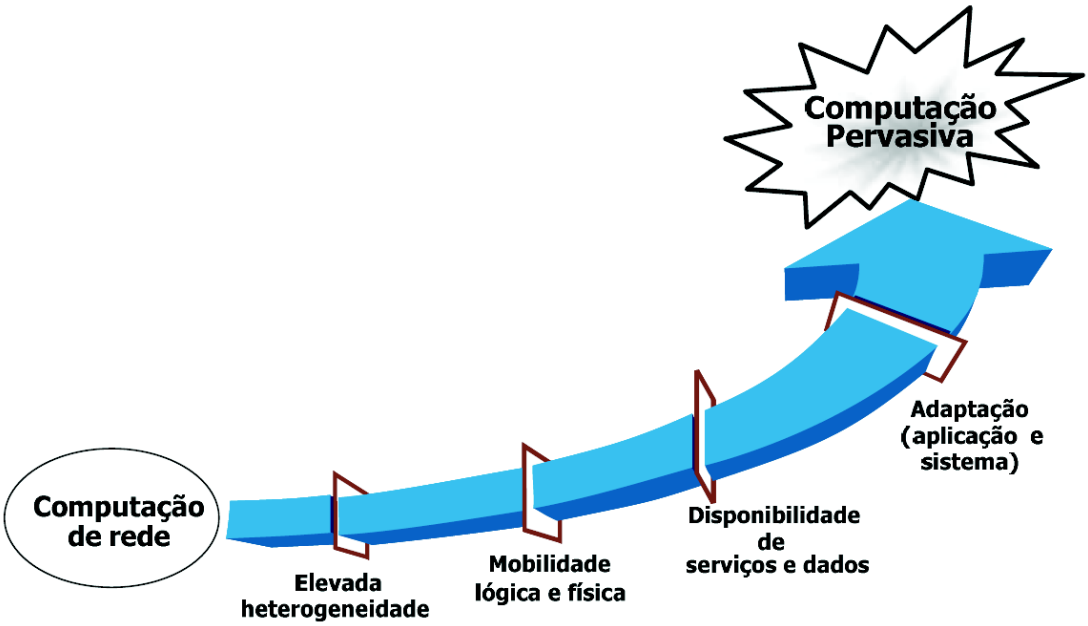


Figura 2.1 – Evolução da computação em direção à pervasividade [27]

2.2.1 Elevada Heterogeneidade

Um dos aspectos característicos da computação pervasiva é a mobilidade do usuário. Em um cenário como este, as aplicações podem ser executadas a partir de diferentes equipamentos, conectados a diferentes tipos de redes. Em um ambiente como este, a disponibilidade de recursos computacionais pode ser bastante variável.

Com relação as tecnologias de redes de computadores, a diferença na banda disponível para comunicação pode ter variações bastante consideráveis. Isto se torna ainda mais crítico quando a rede em questão é sem fio, onde a variação de sinal reflete diretamente no desempenho da rede. Para solucionar isso, alguns ambientes se preocupam com modificações no nível do *kernel* do SO. Outra possibilidade ainda é adaptar a especificação dos dados transmitidos (compactar, modificar resolução etc.), sem modificar porém o comportamento semântico da aplicação.

No que diz respeito aos equipamentos utilizados para executar aplicações, os casos são ainda mais extremos. Em um ambiente pervasivo, não devem existir limitações quanto ao equipamento usado, podendo este ser um computador de médio/grade porte (*desktop*) ou um PDA. Em equipamentos como estes, as diferenças no que diz respeito ao poder de processamento, memória, armazenamento, e mesmo periféricos, são consideráveis. Soluções para estas questões envolvem mapeamento de recursos, balanceamento de carga e seleção dinâmica de código a ser utilizado para o tratamento de entrada e saída na aplicação.

2.2.2 Mobilidade Lógica e Física

De forma genérica, um cenário que envolve computação móvel é aquele onde alguns dos nodos envolvidos na computação, são móveis [28]. Em um ambiente como este, é possível identificar dois perfis para usuários, separando estes em grupos bem definidos. No primeiro, os usuários são basicamente nômades, ou seja, suas conexões de rede ocorrem em locais e momentos totalmente arbitrários. No segundo grupo se encontram os usuários que estão sempre conectados, utilizando-se para isto de redes sem fio. Desta forma é possível identificar algumas propriedades da computação móvel, são elas: mobilidade, portabilidade e conectividade [29].

A natureza essencialmente dinâmica de hardware e software, em ambientes de computação móvel, cria a necessidade de existirem soluções para questões como a identificação física de nodos e a localização de componentes de software. Esta questão levanta a necessidade de mecanismos que conheçam a localização de componentes móveis de modo a permitir a interação com os mesmos. Assim, o ambiente que se propõe a oferecer suporte a computação pervasiva, deve levar em conta estas questões, disponibilizando serviços que permitam as aplicações não perderem consistência quando um nodo migrar na rede, ou mesmo quando um componente de software migrar para outro nodo para continuar sua execução. Neste ambiente a mobilidade lógica é movimentação de uma entidade de software em seu contexto, enquanto que a mobilidade física é a mobilidade do usuário, com ou sem o seu dispositivo computacional.

2.2.3 Disponibilidade de Serviços e Dados

Este é uma questão importante para qualquer sistema que pretenda dar suporte a aplicações pervasivas. A adaptação de uma aplicação é ligada não apenas a modificações relativas ao contexto, mas também a alterações em sua lógica, durante a execução. Uma das formas de modificar a lógica de uma aplicação pode ser através da mobilidade de código. Porém, para que uma aplicação mantenha a sua consistência, precisam existir mecanismos que permitam ao código que se moveu, continuar fazendo acesso aos mesmos dados e serviços que acessava antes de sua mudança de nodo.

Outra questão importante são as estratégias de tolerância a falhas. Ambientes pervasivos são bastante dinâmicos, sendo necessário oferecer mecanismos para que um sistema seja capaz de suportar este tipo de situação, onde entidades podem deixar de existir, levando consigo dados

e serviços que poderiam estar em uso no sistema.

2.2.4 Adaptação (Sistema e Aplicação)

A idéia de sistemas que se adaptam ao seu ambiente não é nova. Na verdade, mobilidade implica em adaptabilidade [30], ou seja, sistemas devem ter consciência do contexto onde estão inseridos e tirar proveito desta informação estruturando-se de modo distribuído e reconfigurando-se dinamicamente. Os desafios colocados pela mobilidade [29] ainda são foco de pesquisa e modelos, arquiteturas e tecnologias se encontram em desenvolvimento.

Em resposta a estes desafios, a comunidade científica vem desenvolvendo trabalhos que têm por objetivo resolver estas questões [13]. Desta forma surgiram algumas soluções que contemplam a gerência de aplicações com suporte à mobilidade lógica e/ou física (software e/ou hardware). Analisando estas soluções, algumas características em comum podem ser observadas [27]: (a) atuam gerenciando as larguras de banda utilizadas na comunicação; (b) se concentram em um domínio específico de aplicação; (c) normalmente implementam apenas uma estratégia de adaptação: alteração no formato dos dados, replicação de dados ou migração de tarefas.

2.3 Sistemas para Suporte à Computação Pervasiva

Nesta seção é feita uma revisão sobre os sistemas existentes, que são direcionados para a criação de aplicações pervasivas. Esta revisão tem por objetivo identificar quais são as principais características dos ambientes em desenvolvimento atualmente e desta forma acumular o conhecimento necessário para propor uma nova solução, baseando-se no Holoparadigma [20].

2.3.1 Aura

O Aura [6, 7, 8] é um projeto que vem sendo desenvolvido na Universidade de Carnegie Mellon. O Aura se coloca como um sistema para a computação pervasiva que tem dois objetivos principais: (1) maximizar o uso dos recursos disponíveis, tanto de processamento quanto de comunicação; (2) minimizar a distração do usuário com fatores externos à aplicação.

O conceito é de que o usuário passa a ter uma "aura pessoal". Quando este entra em um novo ambiente, sua aura se encarrega de conseguir os recursos necessários para que o usuário execute a sua tarefa. Exemplos de tarefas: escrever um artigo, uma apresentação ou mesmo comprar uma casa, onde cada uma destas tarefas pode envolver uma grande quantidade de fontes de informação e aplicações.

Para que esta abordagem seja possível, a arquitetura do Aura inclui novas funcionalidades que precisam ser implementadas em nível de sistema e de aplicação. Outra questão importante é obter o máximo de informações possível sobre o tipo de aplicação que o usuário está executando, preferências pessoais e intenções. Para resolver esta questão é proposto um

"*placeholder*", que é então responsável por guardar estas informações. A Figura 2.2 da uma visão geral da arquitetura do *framework* proposto no projeto.

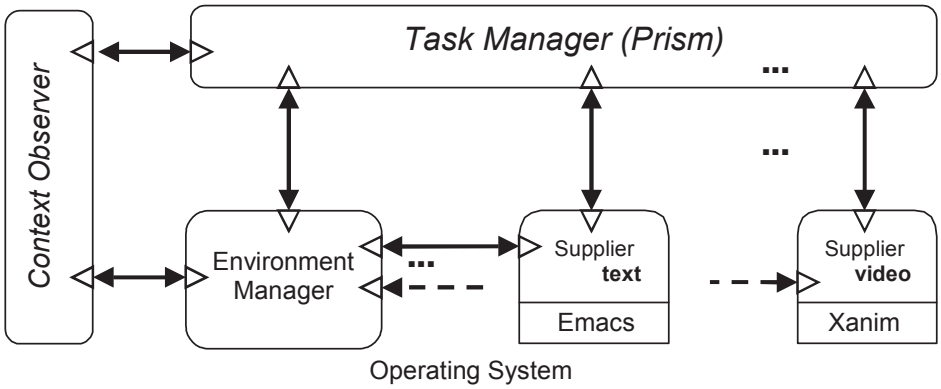


Figura 2.2 – Arquitetura do Aura [7]

Esta arquitetura é composta por quatro tipos de componentes:

- o *Task Manager*, também chamado de *Prism*, é responsável por personificar o conceito de "aura pessoal". Este componente é responsável pelo gerenciamento de tarefas como: mudança de ambiente por parte do usuário, mudança do próprio ambiente, mudança da tarefa e mudança no contexto;
- o *Context Observer* colhe informações relevantes sobre o contexto físico, providas por sensores, e reporta estas diretamente ao Prism e ao *Environment Manager*;
- o *Environment Manager* é a porta de entrada para o ambiente. Este componente sabe quais entidades estão disponíveis no sistema, que serviços elas oferecem e onde podem ser encontradas. Este serviço também possui mecanismos que implementam o acesso distribuído a arquivos;
- os *Supplier* provêm os serviços abstratos dos quais as tarefas são compostas: edição de texto, vídeo, etc.

A comunicação entre estes elementos é feita por um componente chamado *Connector*. Existem diferentes tipos de *Connectors* para comunicação entre os elementos da arquitetura, bem como implementações únicas de cada um deles para casos de uso específicos.

Com esta arquitetura o Aura pode implementar soluções que permitem ao usuário continuar o seu trabalho, mesmo quando se move de um ambiente para outro, de forma que a tarefa irá seguir o mesmo, utilizando os recursos que estiverem disponíveis. Por exemplo, se um indivíduo se encontra digitando um texto e resolve ir pegar um café. Ele então pega o seu PDA e sai da sala, o sistema detecta a sua saída e automaticamente migra a tarefa para o PDA, permitindo a continuidade do trabalho sem nenhuma ação adicional por parte do usuário.

2.3.2 Gaia

O Projeto Gaia [9, 10, 11], desenvolvido na Universidade Illinois, tem como objetivo criar uma infra-estrutura de *middleware* distribuído que coordena serviços de *software* e dispositivos, que utilizam-se de uma rede heterogênea, contidos em um espaço físico. O Gaia possibilita a consulta por serviços, utiliza-se dos recursos existentes para acessar e manipular o contexto atual e provê um *framework* para desenvolver aplicações: centradas no usuário, consciente do espaço físico e para múltiplos dispositivos.

O Gaia define um novo conceito chamado de *Active Space*. Enquanto um espaço físico tem fronteiras bem definidas (uma sala por exemplo), possui objetos, dispositivos conectados a redes e usuários desempenhando diferentes atividades, um *Active Space* é um espaço físico coordenado por uma infra-estrutura de software baseada em contexto que potencializa a habilidade do usuário para interagir e configurar seu ambiente virtual e físico de forma consistente.

A arquitetura proposta pelo Gaia (Figura 2.3) pode ser dividida em três peças principais: o *kernel*, o *framework* para aplicações e as próprias aplicações.

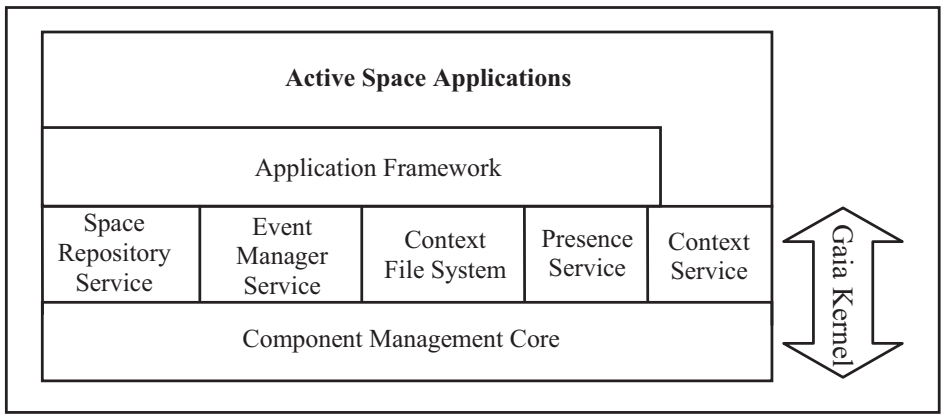


Figura 2.3 – Arquitetura do Gaia [10]

O *kernel* do Gaia é responsável por prover as funcionalidades mínimas requeridas para criar um *Active Space*, são elas:

- *Component Manager Core*, que permite ao Gaia manipular componentes de *software*, de forma que ele pode criar, destruir e transferir componentes. A outra parte são os serviços do *Kernel*, que se dividem em quatro serviços básicos: o *Event Manager Service* que é responsável pela distribuição de eventos no *Active Space*, ele implementa um modelo de comunicação desacoplado baseado em fornecedores, consumidores e canais;

- o *Context Service* é responsável por armazenar e fornecer informações sobre o contexto do *Active Space*, ele armazena tuplas de aridade 4 que consistem de: tipo do contexto, assunto, relator, objeto;

- o *Space Repository Service* é responsável por armazenar informações sobre todas as entidades ativas, bem como prover formas de consultar e recuperar entidades;

- e por último o *Context File System* que é um sistema de arquivos que tira proveito de informações de contexto para diminuir o número de tarefas que devem ser feitas pelo usuário.

O último elemento de arquitetura a ser abordado é o *framework* para aplicações. O objetivo deste componente é facilitar o desenvolvimento de aplicações que fazem uso de um *Active Space*. Este *framework* é composto por três elementos básicos: o *modelo*, que implementa a lógica da aplicação, *apresentação* que exporta o estado do modelo e o *controlador* que mapeia eventos de entrada em mensagens para o modelo. Programas que utilizam a infra-estrutura do Gaia podem ser feitos em Java, C++ ou Lua [31].

2.3.3 one.world

O *one.world* [12, 13, 14, 15] é um projeto desenvolvido na Universidade de Washington, e seu principal objetivo é criar um *framework* para programar aplicações pervasivas. Ao contrário dos projetos abordados anteriormente, o one.world não apresenta mudanças fundamentais no SO ou novos serviços, ao invés disso o objetivo é criar um *framework* com um grande número de funcionalidades que facilitem o desenvolvimento de aplicações pervasivas.

Este projeto se baseia em três características de sistemas distribuídos tradicionais, onde estes falham em atender as necessidades de um sistema pervasivo, são eles:

- sistemas distribuídos tentam esconder a distribuição de dispositivos utilizando-se de sistemas de arquivos distribuídos ou RPC. Esta abstração simplifica o desenvolvimento de aplicações mas tem seu ponto fraco na disponibilidade de serviço e na resistência a falhas. Isto ocorre por que o modelo de programação utilizado nestes casos coloca a indisponibilidade de um recurso como um caso extremo, o que não é compatível com a dinamicidade de um ambiente pervasivo;

- RPC e sistemas baseados em objetos distribuídos compõem aplicações utilizando-se de interfaces criadas em nível de programação, o que faz com que os módulos da aplicação sejam fortemente acoplados. Como resultado, tem-se um sistema onde é desnecessariamente complicado adicionar novos módulos, e onde a modificação em um componente pode comprometer as interfaces existentes, criando um trabalho extra para o programador;

- por último, sistemas que utilizam objetos distribuídos encapsulam dados e funcionalidades em uma única abstração, chamada de objeto. Desta forma, utiliza-se em aplicações distribuídas uma abstração criada para aplicações que executam em um

único nodo. Esta abstração é limitada na forma como os dados podem ser utilizados, complicando o compartilhamento, busca e filtro dos dados.

Baseando-se nestas idéias foram identificados três princípios para o desenvolvimento de um *framework* pervasivo: **princípio 1**, o sistema deve expor suas modificações, mesmo que sejam falhas, de forma que as aplicações possam ser implementadas para tratar estes problemas; **princípio 2**, os sistemas devem permitir que uma aplicação seja composta e estendida em tempo de execução; **princípio 3**, os sistemas devem prover uma separação clara entre dados e funcionalidades, para que possam ser gerenciados separadamente.

Na arquitetura do one.world, cada dispositivo executa uma única instância do sistema. Desta forma cada nodo é independente e pode ser administrado separadamente. Uma instância da arquitetura pode executar várias aplicações. Esta arquitetura provê as mesmas abstrações básicas e serviços em todos os nodos e utiliza mobilidade de código para prover uma execução uniforme e segura.

Como é mostrado na Figura 2.4, a arquitetura fornece abstrações diferentes para os dados da aplicação e para as funcionalidades da mesma. Aplicações armazenam e transmitem dados na forma de *tuplas*, e são compostas por *componentes*. A principal vantagem das tuplas é que a aplicação pode determinar dinamicamente seus campos e o tipo dos dados a serem armazenados. Componentes implementam funcionalidade e interagem importando e exportando manipuladores de eventos. Outro elemento da arquitetura são os *ambientes*, que são responsáveis pela estrutura e controle da aplicação. Eles servem como um compartimento para tuplas, componentes e outros ambientes.

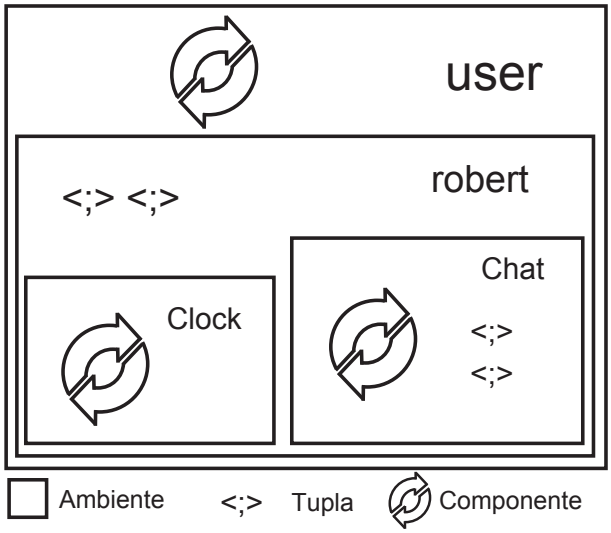


Figura 2.4 – Arquitetura do one.world [13]

O one.world fornece um conjunto de serviços que tem por objetivo ajudar o programador a implementar aplicações adaptativas. Entre estes serviços podem ser citados: *Migração*, que provê a habilidade de copiar ou mover um ambiente e seu conteúdo para outro nodo; *Passagem*

de Evento Remoto, que implementa a passagem de eventos para receptores remotos; *Replicação*, que torna tuplas acessíveis em vários nodos ao mesmo tempo; *checkpointing*, que permite a captura do estado de execução do ambiente e seu armazenamento em uma tupla, podendo mais tarde reverter o estado da árvore de execução do ambiente.

2.3.4 ISAM

O ISAM [16, 17, 18] é um projeto desenvolvido por pesquisadores em universidades do sul do Brasil, são elas: UFRGS, UFSM, UFPEL e UNISINOS. Um dos objetivos do ISAM é a adaptação colaborativa entre a aplicação (programa) e o ambiente de execução (*middleware*). Na arquitetura ISAM, o sistema se adapta para fornecer qualidade, enquanto que a aplicação se adapta para manter a qualidade dentro da expectativa do usuário móvel.

A arquitetura proposta é organizada em camadas com níveis diferenciados de abstração. O ISAM busca a manutenibilidade da qualidade de serviços oferecida ao usuário móvel, através do conceito de adaptação. Uma visão organizacional desta arquitetura é apresentada na Figura 2.5.

A camada superior da arquitetura é composta pela aplicação móvel distribuída. A construção desta aplicação baseia-se nas abstrações do Holoparadigma [20], as quais permitem expressar mobilidade, acrescidas de novas abstrações para expressar adaptabilidade providas pelo ISAMadapt [32]. Por sua vez, a camada inferior é composta pelas tecnologias empregadas nos sistemas distribuídos existentes, tais como sistemas operacionais nativos e a Máquina Virtual Java.

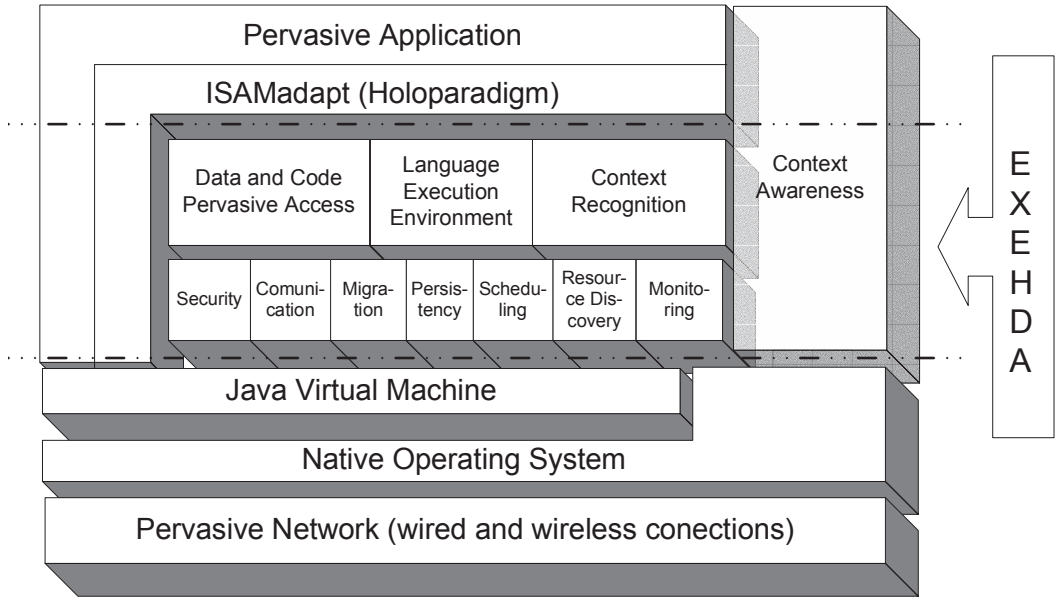


Figura 2.5 – Arquitetura do ISAM [17]

A camada intermediária é o núcleo funcional da arquitetura ISAM, sendo formada por três níveis de abstração. O primeiro nível é composto por dois módulos de serviço à aplicação:

Escalonamento e *Ambiente Virtual do Usuário*. O escalonamento, por sua vez, é o componente-chave da adaptação na arquitetura ISAM. O Ambiente Virtual do Usuário (AVU) compõe-se dos elementos que integram a interface de interação do usuário móvel com o sistema. Este módulo permite que uma aplicação sendo executada em uma localização possa ser instanciada e continuada em outra localização sem interrupção, suportando assim o estilo de aplicações *follow-me*.

O segundo nível é responsável pelo contexto da aplicação. O contexto é determinado através de informações de quem, onde, quando, o que está sendo realizado e com o que está sendo realizado. Obter essas informações é a tarefa do módulo de monitoramento, que atua tanto na parte móvel quanto na parte fixa da rede. As informações que dirigem as decisões do escalonador e dão suporte à aplicação para sua decisão de adaptação são advindas de quatro fontes: perfil da execução, perfil dos recursos, perfil do usuário e da aplicação (ISAMadapt).

No terceiro nível da camada intermediária estão os serviços básicos do ambiente de execução ISAM que provém as funcionalidades necessárias para o segundo nível e cobrem vários aspectos, tais como migração - mecanismos para deslocar um ente de uma localização física para outra; replicação otimista - mecanismo para aumentar a disponibilidade e o desempenho do acesso aos dados; localização e *naming* - para dar suporte ao movimento dos dispositivos móveis, mantendo a execução durante o deslocamento.

2.4 Conclusão

Neste capítulo foi apresentada uma visão de como surgiu a computação pervasiva, bem como dos principais sistemas pervasivos encontrados na literatura atualmente. Foram discutidas as características esperadas de um sistema que suporte aplicações pervasivas. Foram mostrados desafios que podem ser identificados, nesta área de pesquisa, que ainda é bastante recente [33]. O fato de a maioria dos artigos sobre computação pervasiva terem sido publicados nos últimos 5 anos, mostra que as pesquisas estão apenas começando a dar seus frutos, sendo assim ainda é complicado estabelecer claramente os requisitos de aplicações pervasivas.

Contudo é possível afirmar que a computação pervasiva cria um novo ambiente de execução que deve suportar: heterogeneidade, mobilidade, adaptação a modificações de contexto e oferecer ainda disponibilidade de serviços e dados a qualquer hora em qualquer lugar. Tal ambiente não se encontra disponível, principalmente considerando-se que é necessária pesquisa e desenvolvimento de uma série de novas tecnologias para suportar tais características.

Foi possível comprovar ainda que, com as devidas variações de estratégia, todos os ambientes tentam solucionar os requisitos identificados para a pervasividade, incluíndo em seus sistemas os seguintes suportes: adaptabilidade, contexto, mobilidade (tanto física quanto lógica) e heterogeneidade.

A adaptabilidade é tida como um fator importante em uma aplicação pervasiva, que permite modificações de comportamento e mesmo de lógica em tempo de execução. Esta é

uma questão que tem ligação direta com o suporte a mobilidade de código na aplicação e a sensibilidade ao contexto. A mobilidade de código permite que novas funcionalidades sejam agregadas na aplicação sem que ela tenha sido compilada previamente com aquele código. Isto também tem uma aplicação direta, por exemplo, no processamento da aplicação, que pode ser direcionado para um nodo com maior poder computacional e poupar os recursos (CPU e bateria) de um PDA.

O suporte ao contexto é uma questão tratada em todos os sistemas abordados. Informações de contexto podem incluir tanto informações do ambiente de execução quanto sobre o ambiente físico, pessoas, salas, etc. A abordagem mais comum é criar um serviço que fica responsável por coletar e informar sobre modificações no contexto. Uma abordagem interessante para o problema e que pode ser aproveitada no modelo proposto neste trabalho, é a solução utilizada no Gaia, onde as informações sobre alterações no contexto são armazenadas em tuplas, sendo este um formato natural para aplicações Holo.

Para resolver a questão da heterogeneidade, existe um problema principal, que está na cada vez mais comum diversificação dos SOs. Para resolver este problema existem duas abordagens possíveis: (1) utilizar uma máquina virtual como ambiente de execução, o que permite que o mesmo código seja executado em diferentes arquiteturas; (2) usar mais de uma linguagem e uma camada de comunicação comum entre elas. Esta última cria limitações para a utilização de migração de código na adaptação da aplicação, bem como em sua execução em diferentes plataformas. Assim, as soluções estudadas aqui optam pela primeira opção, sendo a Máquina Virtual Java (JVM) o ambiente de execução escolhido.

O fato de Java ser uma ferramenta de desenvolvimento popular em ambientes de computação pervasiva, traz uma questão interessante citada por Grimm [13], onde ele levanta os problemas de se utilizar um paradigma criado para aplicações que executam em um nodo, no contexto de aplicações pervasivas. Para resolver esta questão o one.world cria uma abstração do ambiente físico que pode ser utilizado para modelar a aplicação. No projeto Gaia, é oferecida uma linguagem script, chamada Lua, para criar as aplicações. Já o ISAM utilizou o Holoparadigma como a solução para uma modelagem mais intuitiva de programas para ambientes pervasivos. É interessante notar ainda que o one.world utiliza-se de um modelo bastante semelhante ao Holo, como poderá ser melhor observado no capítulo 3.

Capítulo 3

Projeto *Mobile Holoparadigm* (MHolo)

O objetivo principal do projeto MHolo é o desenvolvimento de uma plataforma que possibilite a modelagem e a implementação de sistemas para computação móvel, baseados no Holoparadigma [20]. O Holoparadigma (de forma resumida, Holo) é um modelo multiparadigma orientado ao desenvolvimento de sistemas distribuídos e de computação móvel. Seus aspectos básicos já foram definidos, porém ainda há muitos elementos a serem desenvolvidos e validados para a obtenção de uma plataforma completa para o desenvolvimento de sistemas móveis.

Os estudos relacionados com Holo envolvem os seguintes tópicos de pesquisa: multiparadigma, sistemas *blackboard*, redes de computadores, sistemas distribuídos e computação móvel. Uma nova linguagem baseada no modelo permite a criação de programas usando os conceitos propostos. A Hololinguagem suporta concorrência, mobilidade, *blackboards* hierárquicos e programação multiparadigma. Atualmente existem duas plataformas de execução: (1) os programas podem ser convertidos para Java usando uma ferramenta denominada HoloJava [34]; (2) os programas podem ser compilados para um *byte code* específico e executados usando uma máquina virtual criada no contexto do projeto (HoloVM [35]).

3.1 Holoparadigma

O Holoparadigma [20] propõe um modelo multiparadigma orientado ao desenvolvimento de software distribuído. Os conceitos apresentados pelo Holoparadigma são implementados na Hololinguagem [36]. Holo explora um mecanismo de coordenação baseado em *blackboards* [37]. Este modelo possui como unidade de modelagem o *ente* e como unidade de informação o *símbolo*. Um ente elementar (Figura 3.1a) é organizado em três partes: *interface*, *comportamento* e *história*. Um ente composto (Figura 3.1b) possui a mesma organização, no entanto, suporta a existência de outros entes na sua composição (entes componentes). Cada ente possui uma história. A história fica encapsulada no ente e, no caso dos entes

compostos, é compartilhada pelos entes componentes. Sendo assim, podem existir vários níveis de encapsulamento da história. Os entes acessam somente a história em um nível. A composição varia de acordo com a mobilidade dos entes em tempo de execução. A Figura 3.1c mostra um ente composto de três níveis e exemplifica a história encapsulada.

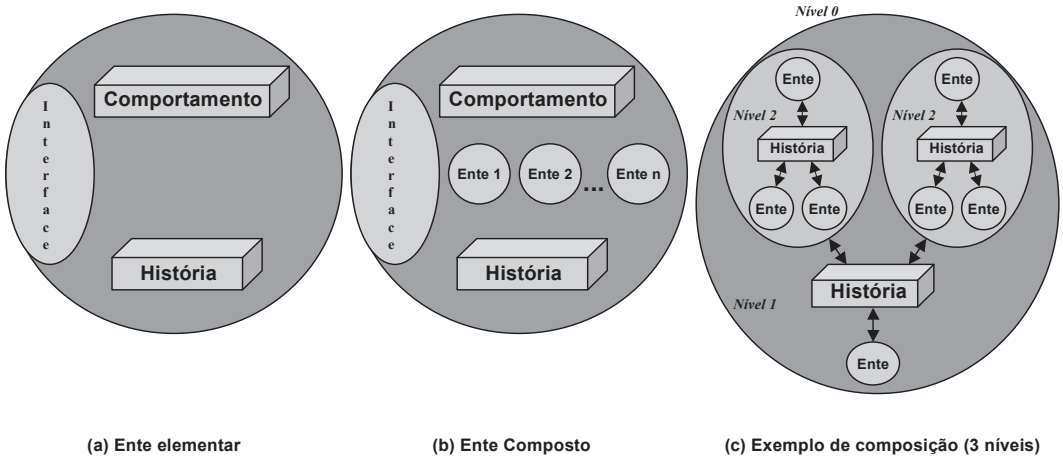


Figura 3.1 – Entes Elementar, Composto e Composição em 3 níveis

No escopo de sistemas distribuídos, um ente pode assumir dois estados de distribuição: centralizado ou distribuído. No estado centralizado um ente encontra-se fisicamente sediado em apenas uma máquina, já no estado distribuído um ente pode criar uma forma de agregar entes executando em máquinas diferentes. A Figura 3.2b mostra um possível estado de distribuição para o ente mostrado na Figura 3.1c. Neste caso, a história do ente composto atua como uma memória compartilhada distribuída (DSM) entre seus entes componentes.

No Holoparadigma, a mobilidade é a capacidade que permite o deslocamento de um ente. São utilizados dois tipos de mobilidade (lógica e física). A mobilidade lógica relaciona-se com o deslocamento em nível de modelagem, ou seja, sem considerações sobre a plataforma de execução. Neste contexto, um ente se move quando cruza uma ou mais fronteiras de entes. A mobilidade física relaciona-se com o deslocamento entre nodos de uma arquitetura distribuída. Desta forma, um ente se move quando se desloca de um nodo para outro. A Figura 3.2a exemplifica uma possível mobilidade lógica no ente apresentado na Figura 3.1c. Merece atenção o caso da mobilidade física não implicar obrigatoriamente a mobilidade lógica, ou seja, a ocorrência de um tipo de deslocamento não implica a ocorrência do outro. A Figura 3.2b mostra uma mobilidade física sem mobilidade lógica.

A Hololinguagem é uma linguagem de programação que surgiu para implementar os conceitos propostos pelo modelo. Esta linguagem suporta concorrência, mobilidade e *blackboards* hierárquicos.

O modelo de coordenação dos entes compostos (Figura 3.3b) assemelha-se a um *blackboard* [38] (Figura 3.3a). Neste caso, os *Knownledge Sources* (KSs) são entes e o *blackboard* é a história. Existem várias limitações estabelecidas pelo uso da invocação implícita

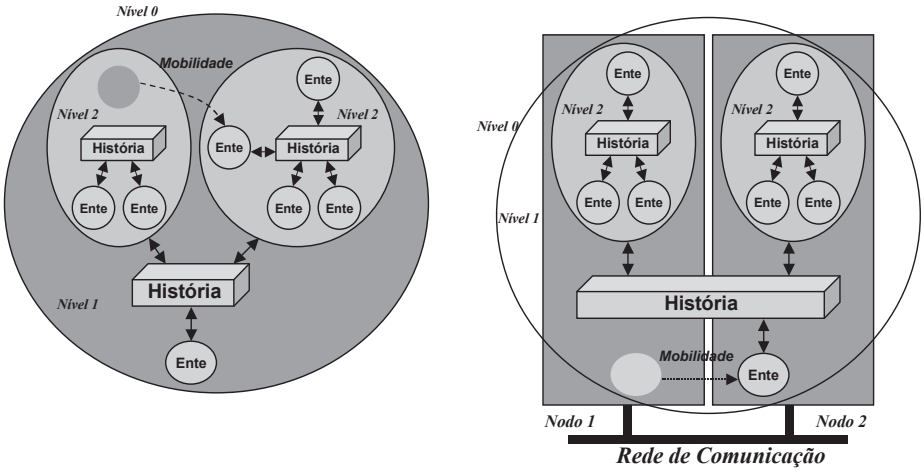


Figura 3.2 – Mobilidade no Holoparadigma

usada nos *blackboards*. O uso da invocação explícita conjuntamente com a invocação implícita é salientado como uma solução para essas limitações. Em Holo ambos os estilos de invocação são utilizados. Os entes influenciam outros entes através da história (invocação implícita), mas também podem trocar informações diretamente através de mensagens (veja Figura 3.3b). No caso da invocação implícita, ao utilizar a história, um ente pode utilizar cinco tipos de acesso, são eles:

- **Afirmação**: Escreve uma tupla na história;
- **Pergunta**: Recupera uma tupla sem bloquear a consulta nem destruir a tupla;
- **Pergunta Bloqueante**: Tenta recuperar tupla, caso ela não exista o programa fica bloqueado esperando;
- **Pergunta Destrutiva**: Recupera a tupla apagando a mesma da história;
- **Pergunta Bloqueante Destrutiva**: Tenta recuperar, se não existir fica em espera, quando ela for criada, lê e apaga a tupla.

A execução de um programa cria uma estrutura hierárquica de entes, denominada Árvore de Entes (HoloTree). A HoloTree implementa o encapsulamento de entes em níveis de composição, conforme proposto pelo Holoparadigma. Além disso, a HoloTree suporta o aspecto dinâmico da política de grupos, mudando continuamente durante a execução de um programa. As seguintes ações alteram a HoloTree: (1) clonagem: a clonagem cria um novo ente; (2) mobilidade: a mobilidade altera a HoloTree, deslocando um ente (elementar ou composto) na árvore. A Figura 3.4a exemplifica a HoloTree para a organização em níveis mostrada na Figura 3.1c. Um ente composto possui ligações com seus entes componentes, os quais ficam localizados no nível abaixo. Os entes componentes possuem acesso à história e

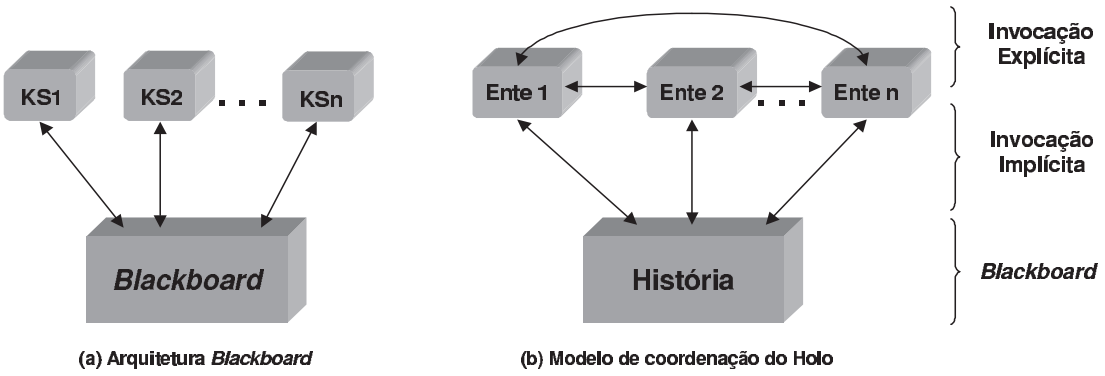


Figura 3.3 – Modelo de coordenação no Holo

ao comportamento do composto no qual estão inseridos. Por sua vez, o ente composto possui acesso aos comportamentos dos seus componentes. Além disso, um ente possui acesso ao comportamento dos demais entes no mesmo nível.

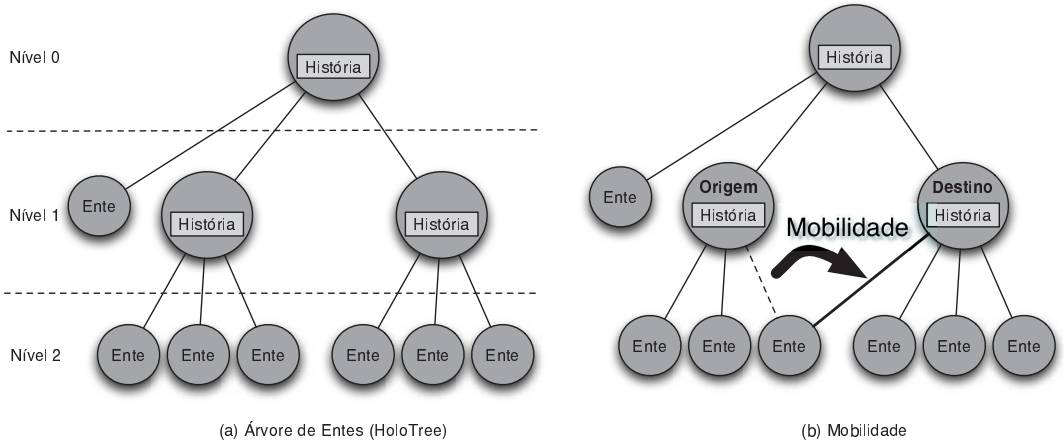


Figura 3.4 – Árvore de Entes (HoloTree)

Quando a mobilidade ocorre, torna-se necessária a mudança da visão do grupo dos entes envolvidos. O ente movido possui uma nova visão (novos entes no mesmo nível e um novo composto acima dele). Se o movido for um ente composto, a visão dos seus componentes não muda. A mobilidade implica a atualização da composição dos entes origem e destino. Além disso, os componentes de um ente podem ser entes compostos (grupos de grupos). A mobilidade de um ente elementar equivale a realocação de uma folha da árvore e a mobilidade de um composto transfere um ramo contendo vários entes. A Figura 3.4b apresenta a mudança que ocorreria na HoloTree para a mobilidade na Figura 3.2a.

3.2 Ambiente de Execução MHolo

O suporte à execução de programas desenvolvidos sobre o Holoparadigma no projeto MHolo é realizado por uma máquina virtual, a HoloVM [35]. Como é natural de uma máquina virtual, ela cria uma camada de abstração entre o programa e o hardware sobre o qual este será executado. Isto permite que programas Holo sejam executados em qualquer plataforma, de estações de trabalho até *handhelds*, desde que exista uma versão da HoloVM disponível para a mesma, facilitando a concepção do MHolo. Isto por que o MHolo visa o suporte para execução em ambientes móveis que são heterogêneos por natureza. A HoloVM oferece um conjunto de instruções específicas para dar suporte as funcionalidades propostas no Holoparadigma. Entre estas instruções, podem ser citadas duas que têm influência sobre a árvore de execução, e portanto com o controle da localização física dos entes. A primeira, chamada *clone* é responsável pela criação de um ente na aplicação. A segunda, chamada *move*, faz com que um ente se mova dentro da árvore de execução.

Quando este trabalho começou a ser desenvolvido a HoloVM suportava apenas a execução local de programas, ou seja, a modelagem de um sistema deve estar contida em um único programa, executando em uma única plataforma. Foi desenvolvido então um modelo de distribuição para o Holoparadigma, o *History Server* (HS), com o objetivo de permitir a execução dos programas que compõe uma modelagem em múltiplas plataformas.

O HS é basicamente um espaço de tuplas que pode ser compartilhado entre várias HoloVMs. Este servidor implementa ainda o suporte a algumas características que o Holoparadigma possui, sendo o respeito às restrições de acesso à história, a principal. Estas restrições são criadas pelo encapsulamento de entes, e consistem no fato de que um ente pode acessar apenas a sua própria história e a de seu ente pai. Ao executar uma aplicação utilizando este sistema, todo acesso a história é mapeado diretamente para o servidor, ou seja, quando um ente acessa a história de seu ente pai, ou a dele próprio, a HoloVM se encarrega de mapear este acesso para uma comunicação com um HS. Este então se encarrega de manter espaços de memória independentes para cada ente criado no ambiente de execução distribuído, permitindo assim que entes dispostos em máquinas diferentes sejam capazes de se comunicar. Maiores detalhes sobre a implementação deste sistema são apresentados na seção 6.4.

3.3 Pervasive Holoparadigm (PHolo)

Atualmente o projeto MHolo se propõe a criação de um ambiente de execução que suporte as particularidades de sistemas móveis. Durante a concepção deste trabalho ficou claro que os requisitos do sistema para execução distribuída necessários no contexto do MHolo, bem como a Hololinguagem e os trabalhos desenvolvidos no projeto, já possuíam uma ligação com os conceitos da computação pervasiva. Desta forma, o objetivo do PHolo é identificar os requisitos de ambientes que suportam a execução de aplicações pervasivas, bem como os requisitos do

MHolo para a pervasividade, e com a proposta de uma arquitetura de software dar o primeiro passo para levar a pervasividade ao projeto MHolo.

A primeira etapa desta pesquisa consiste na criação de uma arquitetura pervasiva para o ambiente de execução do projeto MHolo. O objetivo do PHolo é identificar os requisitos de sistemas pervasivos e propor uma arquitetura que contemple tais requisitos no contexto do projeto MHolo. A revisão feita sobre ambientes de execução que possuem propostas semelhantes (seção 2.3) permitiu o levantamento dos requisitos para este tipo de arquitetura. O intuito é propor uma arquitetura que adapte o atual suporte executivo (HoloVM), a um ambiente pervasivo, trazendo assim uma série de conceitos desta nova classe de aplicações para o MHolo. Na arquitetura proposta, um elemento foi escolhido para que um modelo mais refinado fosse criado, bem como uma implementação que prove seu conceito. Este elemento é chamado *Holo Naming System* (HNS) e será abordado com maior detalhamento no capítulo 5. Dois artigos apresentando este trabalho foram publicados [39, 40].

3.3.1 Arquitetura Proposta

O objetivo da arquitetura proposta é o uso do suporte de execução existente no projeto MHolo, e estende-lo agregando novos elementos e serviços. Com isto é criado um ambiente onde aplicações executam de forma distribuída sem que o programador tenha que se preocupar com: localização de uma entidade, forma de comunicação entre VMs, encontrar um HNS ou outra HoloVM ou como irá distribuir os elementos a serem processados. É importante ressaltar que esta proposta não contempla modificações de baixo nível no SO, por exemplo, para monitorar a rede e adaptar-se a modificações, como faz o Gaia.

Desta forma, a arquitetura do PHolo (Figura 3.5) é dividida basicamente em duas camadas. A primeira é responsável pela execução de programas, nela está a HoloVM. A HoloVM executa as aplicações, de forma distribuída (com auxilio do HNS), oferecendo todo o suporte à comunicação entre entes sem que isso acarrete em uma complexidade maior no desenvolvimento. Na segunda camada da arquitetura estão os serviços do ambiente. O HNS tem o controle de todos os entes em execução na aplicação. Este controle é feito através de uma estrutura chamada *DHolotree* (*Distributed HoloTree*), na qual estão contidas as árvores de execução de cada HoloVM. Com esta estrutura e os endereços físicos dos entes, o HNS pode informar para as HoloVMs sobre a localização de um ente, permitindo assim a modificação da organização lógica e/ou física da aplicação em tempo de execução. O modelo do HNS será apresentado em detalhes no capitulo 5, os demais serviços da segunda camada serão abordados nas seções a seguir.

3.3.2 Localização Física

Uma questão de bastante interesse em uma aplicação pervasiva são os sistemas de localização. No MHolo existe uma iniciativa que objetiva pesquisar o uso de sistemas de

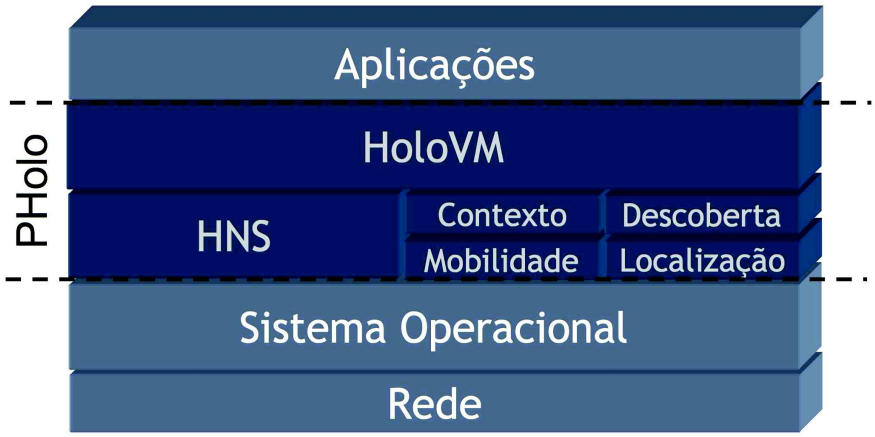


Figura 3.5 – Proposta para a arquitetura do PHolo.

localização no Holoparadigma [41]. Uma das principais características do Holoparadigma é sua habilidade para representar a movimentação de entes. Estes por sua vez podem modelar elementos físicos ou lógicos. Dada a estrutura hierárquica de uma aplicação Holo, é possível que, em tempo de execução, um ente passe a compor outro ente, neste processo também saindo da composição de um terceiro ente. É dito por um desenvolvedor que um ente "entra" ou "sai" de outro ente, e esta ação é implementada pelo comando *move* na Hololinguagem.

Esta abstração cria uma ferramenta poderosa para um desenvolvedor, especialmente quando se quer modelar elementos do mundo real. Estas abstrações facilitam a modelagem de ambientes reais, como salas e prédios, de dispositivos fixos dentro da sala, como computadores ou impressoras, bem como de dispositivos móveis, como *laptops* e PDAs.

Atualmente, em aplicações reais [42], a especificação da localização é feita através da interferência direta do usuário. Para garantir a modelagem fiel do mundo real, é necessário que as alterações reflitam diretamente na aplicação. Por exemplo, quando um PDA entra em uma sala, o ente da aplicação deve ser movido para dentro do ente que representa a sala na aplicação. Desta forma a aplicação que está executando no PDA terá acesso a informações de contexto bem como aos serviços providos pelo ente da sala.

Apesar de existirem comandos na Hololinguagem através dos quais o programador pode especificar ações de mobilidade, atualmente o programa não tem como saber a sua localização em um mundo real. A idéia é agregar uma extensão na camada de execução e assim criar o suporte à mobilidade automática de entes de forma a refletir o comportamento dos elementos que eles simbolizam.

3.3.3 Contexto

É essencial para aplicações pervasivas que estas tenham mecanismos para aquisição de informações sobre o mundo real. Desta forma é possível utilizar estas informações para adaptar

a aplicação as necessidades do usuário. O sistema de localização citado na seção anterior é um exemplo de adaptação guiada por informações de contexto. A modelagem de aplicações refletindo a organização de espaços físicos, juntamente com características do Holoparadigma como o encapsulamento de contextos e dados, criam um ambiente onde a disponibilidade de informações organizadas de acordo com os contextos aos quais elas pertencem parece bastante natural.

Para um primeiro protótipo, qualquer informação relevante para o contexto pode ser armazenada pelo ente em sua história. Esta abordagem pode seguir o modelo utilizado pelo Gaia [10], onde o seu Serviço de Contexto armazena os eventos do ambiente em tuplas aridade 4 que são compostas por: tipo do contexto, assunto, relator, objeto. A grande diferença é que para PHolo este seria um formato nativo, já que possui uma forma de armazenar e consultar dados como tuplas (história).

3.3.4 Tipos de Mobilidade

Quando se está falando de aplicações pervasivas, o termo mobilidade se torna bastante genérico. No Holo são definidos os termos apenas para mobilidade lógica, que é a mobilidade de um ente na estrutura da aplicação, e física que é, em última análise mobilidade de código. Porém, em ambientes pervasivos, quando se fala de mobilidade física isto pode dizer respeito mobilidade de um dispositivo computacional. Para facilitar o entendimento deste tópico, serão definidas aqui as nomenclaturas para os três tipos de mobilidade possíveis no contexto do PHolo:

- **Mobilidade Física**, que consiste na movimentação de um dispositivo, dentro de um espaço físico;
- **Mobilidade Lógica**, ocorre quando um ente se move, neste processo modificando a HoloTree, porém permanecendo na mesma HoloVM;
- **Mobilidade de Código**, que ocorre quando um ente é migrado, durante sua execução, de uma HoloVM para outra. Não necessariamente esta ação dispara uma mobilidade lógica.

O PHolo tem potencial para explorar estes três tipos de mobilidade. Com a solução baseada em sensibilidade a localização, o ambiente físico pode facilmente refletir em modificações de comportamento da aplicação. Na Figura 3.6 pode ser visto um exemplo de uma aplicação que utiliza mobilidade lógica. No exemplo duas HoloVMs dividem a execução de uma aplicação que tem por objetivo gerenciar um evento científico.

Neste exemplo o **nodo 1** tem a aplicação para o gerenciamento de uma sessão executando em sua HoloVM. O **nodo 2** é responsável pela execução dos entes que mapeiam o espaço físico no qual o evento está acontecendo. No lado esquerdo da figura pode ser vista a aplicação, já em execução, onde o **ente d** compõem o **ente b**. Como estes não se encontram na mesma máquina, é

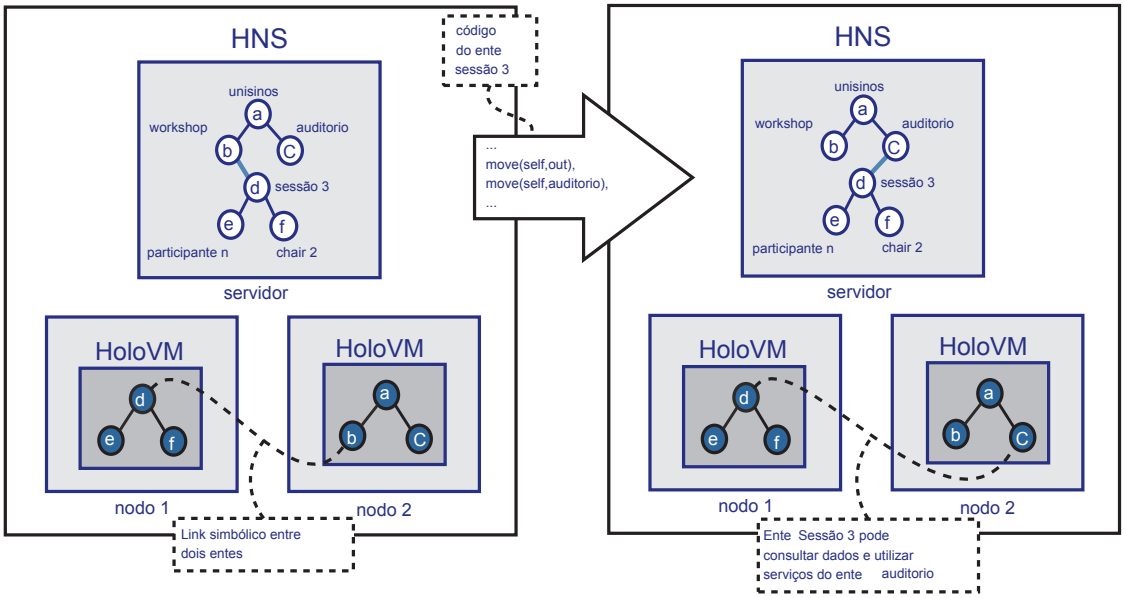


Figura 3.6 – Exemplo de mobilidade lógica.

criado uma ligação simbólica entre os dois entes. Desta forma o ambiente de execução mantém o controle desta situação. Caso os dois entes queiram executar uma ação, ou fazer acesso a história um do outro, a HoloVM consulta a localização do ente remoto no HNS e envia uma mensagem com a requisição do serviço para a outra HoloVM. No passo seguinte de execução, o **ente d** executa duas instruções *move* que resultam em uma mobilidade lógica deste, do **ente b** para o **ente c**. Com esta ação o **ente d** passa a compor o ente responsável pelo auditório, tendo por tanto acesso ao dados de contexto existentes nele, bem como os serviços. Desta forma pode-se dizer que o PHolo implementa o que é chamado de TDS (Sigla em inglês para, Sistema Verdadeiramente Distribuído). Um sistema TDS, é aquele onde um indivíduo pode utilizar os serviços do ambiente sem precisar considerar que nem todos os recursos estão locais.

A mobilidade de código, não é um tópico recente de pesquisa, e já existe uma boa classificação das soluções possíveis para a questão [43]. Atualmente o MHolo não possui suporte a este tipo de mobilidade. Nesta classificação, o tipo que se encaixa melhor nos requisitos do projeto é chamado de mobilidade forte. Esta se divide em duas possibilidades: *migração de código*, que consiste em ser capaz de parar um fluxo de execução, encapsular todos os dados do fluxo, enviar para outra máquina e colocá-lo novamente em execução. Outra possibilidade é a *clonagem remota*, onde uma instância do código é clonado e enviado para a máquina cliente pronto para executar.

3.3.5 Descoberta de Serviços

A descoberta de serviços é uma característica importante para aplicações pervasivas, pois permite a cooperação entre dispositivos e diminui o custo com configuração [44]. No PHolo a descoberta de serviços engloba duas funcionalidades importantes. A primeira é permitir que

HoloVMs descubram servidores HNS e mesmo outras HoloVMs na mesma rede, sem que seja necessário existir uma intervenção direta do usuário para isso. A segunda é o anúncio e localização de serviços. A auto configuração do ambiente é uma solução esperada, e para a qual já foi dedicada alguma atenção. Este assunto será abordado com maiores detalhes no contexto do HNS nos capítulos 4 e 5.

3.4 Conclusão

Nesta seção foram estudados o Holoparadigma e as tecnologias que criam o suporte de execução para este paradigma no projeto MHolo. Nos estudos feitos sobre sistemas que suportam a criação de aplicações pervasivas (seção 2.3) foi concluído que um dos principais problemas a serem resolvidos está na forma de representar os programas para estes ambientes. Os paradigmas existentes atualmente não foram pensados para a criação de aplicações distribuídas e por tanto o seu uso na criação de aplicações pervasivas apenas torna o processo mais complexo.

Por outro lado, o Holoparadigma apresenta uma proposta inovadora para a criação de aplicações que envolvem mobilidade. Suas abstrações implementam uma forma bastante intuitiva de representar os ambientes reais onde as aplicações pervasivas deverão executar. Este fato é validado, tanto pelo projeto ISAM, que utiliza o Holo para aplicações pervasivas, porém mais focado em arquiteturas para *Grid*, quanto pelo one.world, que possui uma abstração própria, muito semelhante ao Holo. Apesar de criar esta abstração, o one.world não oferece uma linguagem de programação que reflita os conceitos da abstração. No PHolo esta questão é solucionada utilizando como linguagem padrão, a Hololinguagem, que possui uma abstração mais próxima da realidade para a modelagem de aplicações pervasivas.

No MHolo foi implementada uma máquina virtual, chamada de HoloVM, que implementa os conceitos do Holoparadigma. A HoloVM oferece suporte nativo a todas as características do paradigma, e permite que os serviços do PHolo sejam implementados diretamente no ambiente que suporta execução, e não no *middleware*, como é feito em todos os projetos estudados. Esta característica permite uma maior flexibilidade no desenvolvimento do ambiente, bem como a garantia de que os programas serão mapeados diretamente para instruções na HoloVM, ao invés de serem passados para outra linguagem.

Capítulo 4

Trabalhos Relacionados

Neste capítulo será feita uma revisão sobre as tecnologias que são necessárias para a criação do modelo para o HNS. O objetivo é falar sobre as áreas de conhecimento que foram estudadas e os sistemas que representam o estado da arte nestas áreas. Na conclusão deste capítulo serão apontadas escolhas referentes à tecnologias que são utilizadas posteriormente no modelo.

4.1 Sistemas de Nomes

A primeira área estudada e mais antiga, é a de sistemas de nomes. Nesta área existe um sistema que é tido como ícone, principalmente por oferecer uma abordagem que resistiu a prova de ser um sistema amplamente utilizado na *internet*. O Domain Name System (DNS) [45] tem como objetivo fazer o mapeamento de nomes para endereços de rede, mantendo ainda uma estrutura de servidores distribuídos para gerenciar os dados do sistema.

4.1.1 Domain Name System (DNS)

Atualmente o DNS é responsável pelo mapeamento de todos os *hosts* da internet para IPs reais. O primeiro passo para a criação do DNS foi dado em 1982, quando foi observado que o sistema utilizado não era mais eficiente para a quantidade de *hosts* e clientes que faziam parte da rede. Naquela época a estratégia consistia em um arquivo chamado HOSTS.TXT que era armazenado em um servidor central e distribuído para todos os nodos na internet via transferência direta e indireta de arquivos. Naquela época a ARPANET passava por uma transformação que implicava no aumento do número de estações de trabalho conectadas na rede. Esta evolução refletiu em um aumento no custo de manter todos os arquivos HOSTS.TXT da rede atualizados. Além disso, era necessário prover uma estratégia para que cada organização tivesse autonomia para gerenciar seus próprios sub-domínios.

O *design* inicial do DNS foi especificado nas RFCs 882 [46] e 883 [47]. O sistema resultante destas especificações possui um espaço de nomes organizado hierarquicamente,

assim como a distribuição da base de dados que também é feita de forma hierárquica. Mesmo que ele tenha sido projetado para ser expandido com o surgimento de novas aplicações, as especificações atuais bem como o uso do sistema são bem similares aos definidos originalmente.

O DNS possuía alguns pré-requisitos básicos para o seu projeto, são eles:

- prover pelo menos toda a informação que o HOSTS.TXT era capaz;
- permitir que a base de dados seja mantida de forma distribuída;
- não possuir nenhum limite óbvio para os nomes, componentes dos nomes, dados associados com nomes, etc;
- interoperar com a internet da DARPA e em tantos sistemas quanto possível;
- prover um desempenho tolerável.

É interessante notar que, para ter esta solução mais facilmente aceita pela comunidade, algumas funcionalidades que poderiam constituir uma base de dados, em seu estado da arte, foram relevadas. Desta forma foi criada uma solução eficiente e relativamente simples.

4.2 Tabelas Hash Distribuídas (DHT)

Nas duas décadas que se passaram desde o desenvolvimento do DNS, os conceitos desenvolvidos nesta área começaram a ser utilizados para aplicações cada vez mais sofisticadas. Os sistemas ponto-a-ponto (p2p), por exemplo, motivaram o surgimento de uma maneira mais robusta e eficiente para encontrar recursos distribuídos em uma rede. Esta tecnologia, que é tida como um dos blocos básicos para a criação de sistemas p2p atuais, é chamada *Distributed Hash Tables* (DHT) [48, 49, 50, 51]. Um sistema DHT tem ênfase em três propriedades, que são:

- **Descentralização**, todos os nodos devem compor o sistema coletivamente, sem que seja necessária nenhuma coordenação central;
- **Escalabilidade**, o sistema deve ser eficiente mesmo que tenha milhões de nodos;
- **Tolerância a Falhas**, o sistema deve ser confiável, mesmo com nodos entrando, saindo e falhando.

Para atingir estes objetivos, o DHT utiliza uma tabela *hash* distribuída entre vários nodos. Nestas tabelas são armazenadas tuplas $< chave, valor >$, de forma bastante semelhante a um sistema de nomes como o DNS. A diferença é que o DHT cria um *hash* para cada chave a ser armazenada no sistema. Uma função *hash* bastante utilizada neste caso é a SHA1 [52], que gera chaves de 160 bits.

A técnica mais atual para a distribuição de dados é chamada *hash* consistente. Nesta abordagem cada servidor que entra no sistema ganha também uma *hash* gerada aleatoriamente. A partir deste ponto, a distribuição dos dados é feita utilizando uma função $\delta(\kappa_1, \kappa_2)$, que é capaz de dar uma noção abstrata entre a chave κ_1 e a chave κ_2, e cada servidor irá armazenar todas as chaves que forem mais próximas a sua, segundo a função δ.

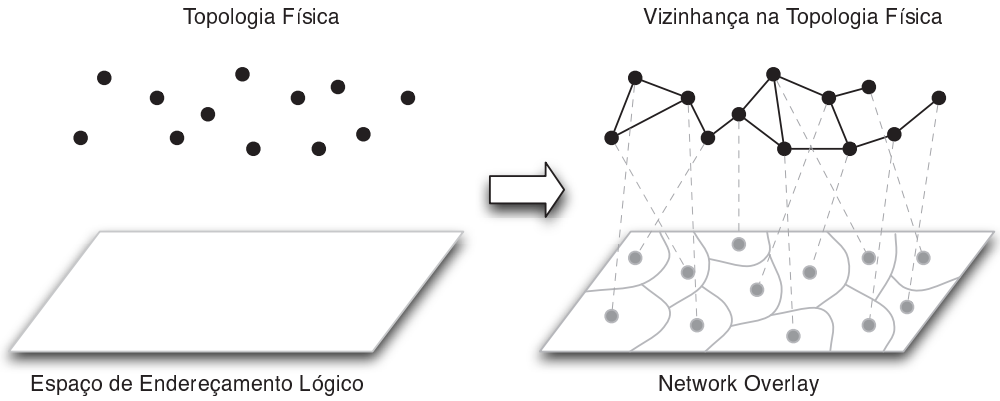


Figura 4.1 – Representação da topologia da aplicação em relação a topologia da rede [53]

Quanto a organização da rede de servidores, cada um deles mantém um conjunto de ligações com outros nodos, que é a sua tabela de roteamento, também chamados de vizinhos. Juntas estas ligações formam uma camada de abstração da topologia da aplicação sobre a topologia física da rede (Figura 4.1), que é chamada de *network overlay*. A principal diferença entre as propostas atuais para implementações de DHT está exatamente na topologia, que na verdade está intrinsecamente ligada com a estratégia de particionamento do espaço de *hashs*. Mudar estes dois aspectos pode demandar e/ou ser motivado pela criação de um algoritmo de roteamento mais eficiente para o sistema. A seguir serão abordados estes três aspectos (topologia, particionamento e roteamento) das soluções DHT estudadas.

4.2.1 Content-Addressable Network (CAN)

O CAN (Content-Addressable Network) [50] utiliza uma topologia de coordenadas cartesianas com d-dimensões. Este espaço de coordenadas é lógico, e não está necessariamente associado com um espaço de coordenadas físico. Cada registro no sistema possui uma chave única K e o CAN utiliza uma função *hash* que cria um vetor d-dimensional $P = hash(K)$ para cada chave, vetor este que corresponde a um ponto em seu espaço d-dimensional. Este ponto irá indica a posição virtual do dado a ser armazenado.

O espaço virtual de coordenadas do CAN é dividido em vários espaços menores, que são chamados de zonas. Cada máquina executando o sistema corresponde a uma zona (Figura 4.2) e armazena dados que são mapeados para esta zona pela função *hash*. Neste espaço d-dimensional, dois nodos são vizinhos se seus espaços de coordenadas se encontram ao longo do eixo $d - 1$. No caso da Figura 4.2, onde é demonstrado o particionamento de um espaço com

duas dimensões, as zonas de nodos vizinhos precisam se encontrar ao longo de um eixo apenas. Esta noção de vizinhança é que irá construir a tabela de roteamento de cada nodo, onde cada um sabe a zona e o endereço IP de seus vizinhos.

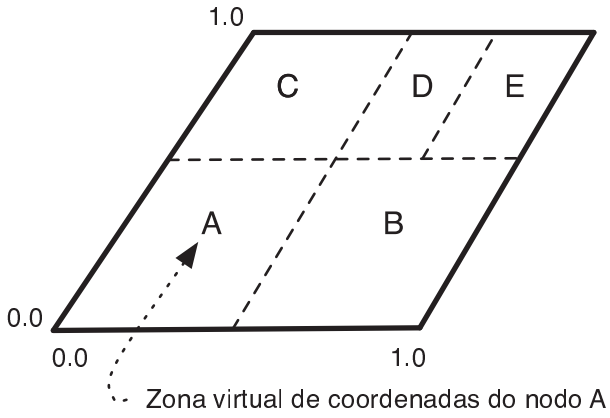


Figura 4.2 – Representação da topologia do CAN

A forma de organização dos servidores deixa bem clara a forma como ocorre o roteamento de uma mensagem no sistema. Ao receber uma chave o sistema calcula um vetor de posição no espaço virtual, partindo de um nodo, ele utiliza a informação que possui sobre as zonas que estão a sua volta para mandar a mensagem a um servidor que esteja mais próximo da coordenada calculada. Este processo se repete até que a mensagem chegue no nodo responsável pela chave. Em um espaço de d-dimensões, cada nodo mantém 2d vizinhos (podendo chegar a 4d, na verdade) e o tamanho médio do caminho de roteamento é $(d/4)(n^{1/d})$ *hops*.

4.2.2 Tapestry

O Tapestry [51] foi originalmente concebido para ser o mecanismo de roteamento e localização do OceanStore [54]. O modelo do Tapestry é em grande parte baseado no esquema de Plaxton [55], incluindo a estratégia de nomes, estrutura e esquema de localização e roteamento. A contribuição do Tapestry está em sua adaptabilidade, tolerância contra múltiplas falhas e otimizações.

No Tapestry cada nodo possui um mapa de vizinhos que é organizado em níveis de roteamento, e cada nível possui registros que apontam para um grupo de nodos que são mais próximos em distância na rede e cujo sufixo se encaixa naquele nível (Figura 4.3). Cada nodo mantém ainda uma lista de ponteiros inversos para todos os nodos onde ele é referido como vizinho. Estas informações são utilizadas para montar mapas de vizinhos para um nodo, que depois é usado pelo algoritmo de integração do Tapestry.

Para localizar um objeto o Tapestry submete a ID do mesmo a um processo de *hashing* que tem como resultado um conjunto de nodos nos quais será feita a busca. O nodo então utiliza o mapa dos vizinhos para mandar a mensagem aos nodos desejados. Um nodo sempre repassa

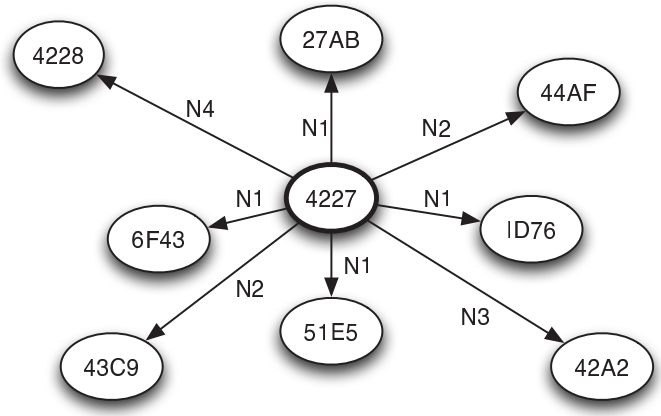


Figura 4.3 – Visão de um nodo da tabela de roteamento do Tapestry

a mensagem para aqueles nodos que ele julga estarem topologicamente mais próximos. Para facilitar esta busca o Tapestry mantém uma série de cópias dos dados armazenados, onde cada nodo tem conhecimento de todas as cópias que estejam próximas a ele, o que torna a localização de dados mais flexível.

4.2.3 Chord

O Chord [49] é um projeto desenvolvido pelo MIT e consiste em um sistema onde as chaves possuem m-bits e são criadas fazendo o *hash* do endereço IP de um nodo. Assim como nos outros sistemas, o Chord gera uma chave única para cada dado armazenado e esta é utilizada para indicar a sua localização. A característica que diferencia o chord dos demais sistemas é a sua topologia, nele todos os $N - 2^m$ IDs são ordenados em um círculo de uma dimensão, ao máquinas são mapeadas dependendo de suas respectivas IDs (Figura 4.4). Cada nodo possui uma maquina que é chamada de *nodo sucessor* e fica sempre ao seu lado no sentido horário.

Cada conjunto chave/valor (K,V) no sistema possui um identificador $P = hash(K)$, o qual indica uma posição virtual no círculo. Um registro (K,V) é armazenado no primeiro nodo no sentido horário, ou seja, no seu sucessor. Na Figura 4.4 é mostrada uma rede Chord que consiste de três nodos os quais armazenam os valores 6,1 e 2, seguindo a regra do sucessor.

Para que o roteamento no sistema seja eficiente cada nodo possui parte da informação de mapeamento. Na visão de cada máquina do sistema o circulo virtual é particionado em $1+logN$ segmentos, o que corresponde a ela mesma mais $logN$ segmentos. A tabela de roteamento de um nodo $logN$ entradas e para cada uma delas as seguintes informações de um segmento: os limites e o sucessor de seu primeiro nodo virtual. Para a busca por uma chave K, o sistema calcula o seu *hash* P e então usa a tabela de roteamento para encontrar o primeiro sucessor do segmento que contém P. Este processo se repete até que o nodo alvo seja encontrado, com a distância diminuindo pela metade com cada *hop*.

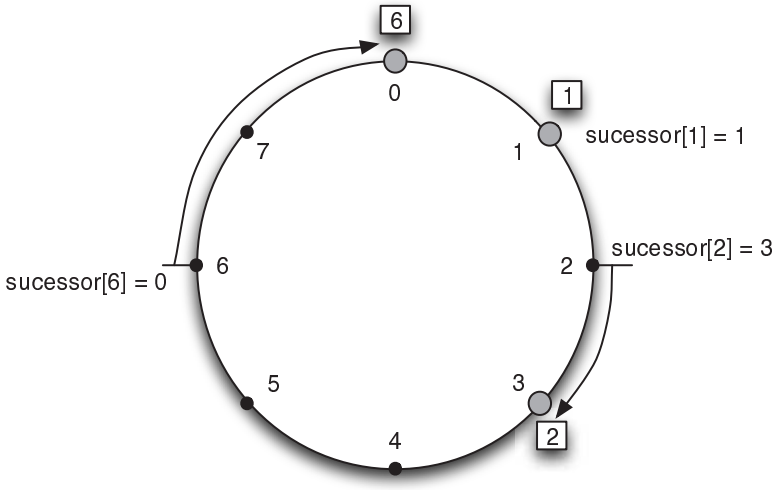


Figura 4.4 – Anel de roteamento do Chord

4.2.4 Kademlia

O Kademlia [56] é um protocolo cuja principal característica é o uso de uma função XOR para calcular a proximidade entre dois pontos no espaço de chaves existente. O Kademlia trata os nodos do sistema como folhas em uma árvore binária, onde a localização de cada nodo é determinada pelo prefixo único mais curto de sua chave. A Figura 4.5 mostra a posição de um nodo cujo prefixo único é 0011 em uma árvore exemplo. Para cada nodo a árvore binária é dividida em séries sucessivas de sub-árvores que não contém este nodo. A sub-árvore de maior nível consiste na metade da árvore binária que não contém o nodo, a próxima sub-árvore consistem na metade restante da árvore que não contém o nodo, e assim por diante. No exemplo do nodo 0011 (Figura 4.5), as sub-árvores estão destacadas com um círculo e consistem em todos os nodos de prefixos 1, 01, 00 e 0010 respectivamente.

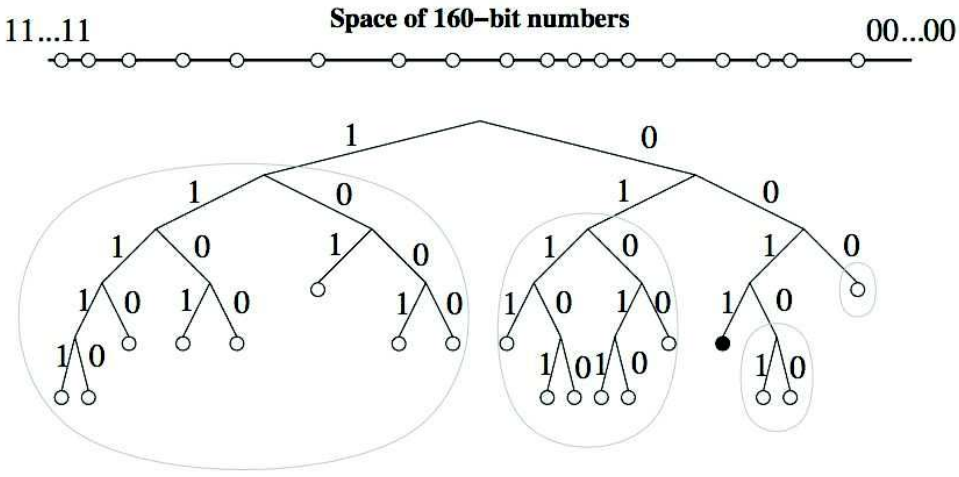


Figura 4.5 – Árvore binária do Kademlia

Uma propriedade importante do Kademlia é a de garantir que cada nodo conhece pelo menos um outro nodo em cada uma de suas sub-árvores, se esta conter um nodo. Com isso o Kademlia é capaz de disparar consultas sucessivas, adentrando as sub-árvores até que consiga encontrar o nodo que possui o dado desejado. Assim como no Chord, a busca do Kademlia é unidirecional, o que garante que todas as buscas venham a convergir por um mesmo caminho, independente de qual nodo originou a mesma. Esta propriedade permite que o Kademilia tenha um sistema de *cache* bastante eficiente, já que ele pode armazenar cópias ao longo do caminho percorrido para encontrar uma chave.

4.3 Descoberta de Serviços

Os sistemas de descoberta de serviços (*Service Discovery* (SD) em inglês) tem por objetivo permitir que componentes de software e hardware se encontrem em uma rede, bem como determinar se os serviços encontrados cumprem os requisitos. Protocolos de SD também podem incluir mecanismos para detectar a disponibilidade de componentes em uma rede e manter uma visão consistente dos serviços disponíveis. Desta forma, os protocolos existentes para descoberta de serviços possuem os seguintes objetivos [57]:

- **Descoberta -** que é a habilidade de encontrar o provedor de um serviço em uma rede que atenda as propriedades descritas na requisição. Para atender este requisito, de forma geral os protocolos implementam as seguintes funcionalidades: uma linguagem para descrever recursos, uma estratégia de armazenamento das informações sobre os serviços e a habilidade de fazer consultas sobre esta informação armazenada;

- **Auto-configuração -** o sistema deve organizar e prover informações sobre o seu conteúdo sem nenhuma interferência humana. Para este objetivo existem duas funcionalidades necessárias, que são: suporte a alteração nas propriedades dos serviços e suporte a modificações na topologia da rede, que podem implicar na falta de disponibilidade de um serviço ou mesmo no surgimento de um novo.

Neste trabalho o uso de sistemas de descoberta de serviços possui um requisito bastante específico, que é a auto-configuração do sistema, já que as tarefas de localização das entidades de programação do sistemas são delegadas ao HNS. No restante desta seção é feita uma revisão sobre quatro protocolos de descoberta de recursos.

4.3.1 SLP

O SLP (*Service Location Protocol*) [58, 59, 60] foi um padrão criado pelo IETF para prover configuração automática de aplicações e anúncio de serviços em uma rede. Apesar do grupo de trabalho que desenvolveu o protocolo ter atingido as suas metas e não existir mais, ainda existe um projeto que mantém o SLP ativo.

No modelo do SLP cada serviço disponível na rede deve possuir uma URL que é utilizada para localizar o mesmo. Adicionalmente o servidor pode armazenar uma série de pares nome/valor sobre um serviço, chamados atributos. Cada dispositivo deve sempre estar em um ou mais escopos, que são *strings* usadas para agrupar serviços. Não é permitido a um dispositivo o acesso a serviços de outros escopos. Uma URL descrevendo um serviço de impressão tem o seguinte formato: *service:printer:lpr://minhaimpressora/minhafila*. Nesta URL *minhaimpressora* corresponde a um *host* do serviço e *minhafila* é a fila de impressão utilizada. A *string* "service:printer:lpr" identifica o tipo do serviço e os dois primeiros componentes ("service:printer") servem para abstrair o tipo de serviço prestado. Com esta informação é possível consultar por qualquer tipo de impressora, e não apenas aquela que utiliza o protocolo *lpr*, como no exemplo dado.

Quanto a organização do sistema, o SLP possui diferentes funções que podem ser desempenhadas pelos dispositivos, são elas:

- *User Agents* (UA), são dispositivos que buscam por serviços;
- *Service Agents* (SA), são dispositivos que anunciam um ou mais serviços;
- *Directory Agents* (DA), são dispositivos que fazem *cache* de serviços.

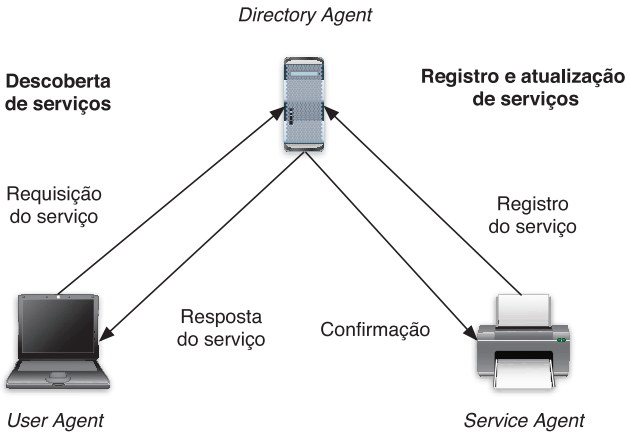


Figura 4.6 – SLP atualizando um DA [59]

Na Figura 4.6 é representada uma situação onde os três tipos de agentes interagem. Quando um novo serviço conecta a rede, o SA avisa o DA da disponibilidade do serviço. Caso um usuário precise de um serviço, o UA pergunta ao DA sobre os serviços disponíveis na rede O UA então recebe o endereço do serviço e o usuário poderá utilizar o serviço. Quanto a auto-configuração, o SLP consegue apenas anunciar os seus DAs utilizando um serviço de DHCP que ainda precisa sofrer algumas configurações para funcionar com o SLP.

4.3.2 Jini

O Jini [61, 62] é uma extensão da linguagem de programação Java que foi desenvolvida pela Sun. O Jini objetiva resolver o problema de como conectar dispositivos para formar uma rede *ad hoc* e fornecer uma maneira para que estes dispositivos sejam capazes de disponibilizar serviços para outros dispositivos na rede.

Quando um serviço entra em uma rede Jini de serviços e/ou dispositivos, ele divulga a si mesmo publicando o objeto Java que implementa a API do serviço. Um cliente por sua vez pode achar um serviço procurando pelo objeto que suporta a API desejada. Ao encontrar o objeto desejado o cliente pode receber todo código que seja necessário para o acesso ao serviço, aprendendo assim a se comunicar com aquela implementação do serviço através de sua API.

O modelo do Jini consiste em uma arquitetura e um modelo de programação. Os princípios da arquitetura são bastante semelhantes ao SLP, onde um serviço que queira se anunciar precisa se comunicar com um *lookup server*, que é semelhante a um DA no SLP. Além de ponteiros para serviços o Jini também pode armazenar código Java para estes serviços. Ao requisitar por um serviço, o cliente baixa o objeto Java desejado para a sua *Java Virtual Machine* local, sendo então capaz de utilizar o objeto que implementa o serviço desejado.

4.3.3 UPnP

O UPnP (*Universal Plug an Play*) [63] é a solução criada pela Microsoft e hoje desenvolvida por um consórcio. O objetivo do UPnP é utilizar estratégias p2p para permitir que a auto-configuração de dispositivos, descoberta e controle de serviços, e com isso simplificar a configuração de redes domesticas ou corporativas.

Para a comunicação o UPnP se propõe a utilizar apenas padrões abertos, baseando seus protocolos em SOAP/XML no caso de mensagens TCP/IP e em SSDP [64] quando a comunicação for UDP, porém sempre utilizando o HTTP como intermediário. Quanto a arquitetura, o UPnP estipula três componentes básicos, são eles: *dispositivos*, que são na verdade responsáveis por fornecer um ou mais serviços; *serviços*, que deve expor suas ações e modelar seu estado através de variáveis de estado; *pontos de controle*, que é um controlador de rede responsável pela descoberta de outros dispositivos. Na especificação do UPnP está definido que dispositivos devem agregar um ponto de controle em sua implementação, para garantir uma estratégia p2p para o sistema.

Ao inicializar, um ponto de controle pode enviar mensagens SSDP para descobrir se determinados tipos de dispositivos e serviços estão disponíveis na rede. Os dispositivos UPnP aguardam por tais requisições em portas *multicast* e ao receber a consulta é verificado se a mesma satisfaz os critérios dos serviços que estes possuem. Se o dispositivo possuir o serviço desejado ele responde enviando uma mensagem diretamente para o ponto de controle que enviou a consulta. Um dispositivo ao entrar na rede utiliza um processo semelhante ao ponto de controle, enviando mensagens na rede anunciando a existência de seu serviços.

4.3.4 Bonjour

O Bonjour [65] é a implementação da Apple para o protocolo Zeroconf da IETF. Até pouco tempo atrás a Apple ainda utilizava SLP, que foi deixado de lado para que fosse utilizada uma implementação própria e baseada em padrões abertos. O Bonjour tem como cerne de sua tecnologia o DNS, que é um serviço amplamente utilizado em qualquer ambiente de rede. Contudo, para implementar o bonjour duas tecnologias adicionais foram criadas: mDNS (Multicast DNS) e o DNS-SD (DNS Service Discover). O mDNS é responsável pelo padrão de mensagens DNS *multicast* e o DNS-SD especifica ma maneira como os recursos existentes no DNS podem ser utilizados para descoberta de serviços. Estas duas tecnologias não são diretamente dependentes, já que o DNS-SD poderia funcionar apenas com mensagens *unicast*, mas o melhor resultado é obtido combinando as duas.

O Bonjour utiliza o mesmo formato de anúncio do DNS, porém com uma adaptação para o seu novo propósito. Assim os serviços anunciados são chamados pelo nome e protocolo de comunicação, formatados segundo o padrão do DNS, que separa estas informações com um ponto e usa o sinal _ como prefixo (Ex.: _http._tcp). Um serviço pode utilizar ainda os campos TXT do DNS para dar mais detalhes sobre o que está sendo oferecido. Estes registros são compostos por nome/valor e no caso do HTTP por exemplo poderiam ser utilizados para indicar um diretório especifico no servidor (Ex.: homepage=/holo).

Uma implementação padrão de DNS não é capaz de suportar mensagens DNS-SD, de forma que um *daemon* foi criado para o propósito, a Apple o chama *mDNSResponder*, e ele já vem instalado por padrão em alguns sistemas operacionais. O papel deste sistema é escutar na rede por mensagens mDNS, registrar os serviços disponíveis e responder consultas. Além do Bonjour utilizar um padrão aberto, o código fonte de uma implementação completa e multi-plataforma é disponibilizada pela Apple.

4.4 Conclusões

O HNS em alguns aspectos pode ter seu propósito confundido com o DNS, já que seu objetivo também é mapear nomes para endereços de rede. Além disso ambos precisam ter uma estratégia eficiente de balanceamento de carga devido a grande quantidade de informações que devem armazenar. No caso do DNS, sua estrutura de nomes é praticamente estática, com poucas alterações ocorrendo nesta estrutura. Além disso um nodo na árvore de nomes não muda de lugar nunca, para fazer isso seria preciso mudar todo o nome do nodo e seu endereço. O nome no DNS é capaz de guiar uma busca através de toda a estrutura da árvore de nomes, o que demanda o conhecimento prévio do nome completo, sendo por outro lado uma técnica extremamente eficiente.

No caso do HNS também é utilizada uma estrutura em árvore como forma de organização. Esta árvore, a HoloTree, é altamente dinâmica sendo muito difícil saber a localização de um

ente na árvore com antecedência. Este fato impossibilita um programador de se referenciar aos entes com nomes compostos pela estrutura da aplicação, semelhante ao que é feito no DNS. Desta forma, as consultas no HNS são feitas usando como informação apenas o nome de um ente e o sistema é quem deve descobrir a localização da informação desejada em sua estrutura de servidores. Dada a complexidade das operações feitas no sistema, é preciso uma forma eficiente para armazenar um grande número de informações em um ambiente distribuído, para que o HNS seja capaz de responder o mais rápido possível às requisições.

Quanto as soluções de DHT, elas podem ser consideradas uma evolução dos sistemas de nomes, já que mantém a técnica de mapeamento <*chave,valor*> só que utilizam uma estrutura de dados mais eficiente para isso. Tal tecnologia é tida atualmente como uma parte fundamental de sistemas modernos, podendo ser utilizada ainda em sistemas distribuídos em larga escala. Os projetos mais comumente vistos são para a criação de sistemas de arquivos distribuídos, sistemas de notificação de eventos, compartilhamento de arquivos e sistemas de *chat* [66]. Contudo, o uso de DHT em pesquisas relacionadas a sistemas de nomes ainda é inicial, com poucas iniciativas encontradas neste sentido e destas poucas, uma boa parte propõe uma abordagem de DHT específica para o DNS [67, 68, 69].

Quanto as estratégias de DHT apresentadas, não existe uma mais eficiente que as outras, pelo contrário, o desempenho das propostas mais difundidas, como as citadas neste capítulo, é considerado bastante semelhante [70]. Um exemplo na semelhança das propostas estudadas pode ser visto na Tabela 4.1, que mostra a complexidade relacionada a busca e manutenção do estado nas diferentes abordagens, e a única que apresenta alguma diferença é o CAN, onde a complexidade está diretamente relacionada ao número de dimensões utilizado.

Tabela 4.1 – Complexidade da busca e manutenção de estado nos principais sistemas de DHT, onde N é o número de nodos da rede e d é o número de dimensões do CAN

Algoritmo	Busca	Estado
CAN	$O(dN\frac{1}{d})$	$O(d)$
Chord	$O(logN)$	$O(logN)$
Kademlia	$O(logN)$	$O(logN)$
Tapestry	$O(logN)$	$O(logN)$

Existem caraterísticas de cada protocolo que podem ser analisadas para que seja possível optar pela mais adequada. O algoritmo XOR do Kademlia por exemplo garante uma propriedade de assimetria que permite a um nodo receber consultas precisamente da mesma distribuição de nodos contidos em sua tabela de roteamento. Isso cria uma vantagem sobre protocolos como o Chord que sem esta propriedade não conseguem aprender informações de roteamento que poderiam ser úteis a partir das consultas recebidas. Outra vantagem do Kademilia é que ele utiliza um único protocolo de roteamento do início ao fim, ao contrário de outros que utilizam um no início e outro algoritmo para os últimos *hops*. Quanto a organização

da topologia o CAN possui uma vantagem aparente, que pode ser vista na Tabela 4.1, onde a complexidade de suas operações está vinculada ao número de dimensões (d) utilizado. Contudo, esta vantagem será real apenas se $d = logN$, isto por que o CAN não é feito para ter um número de dimensões variável, que acompanhe as alterações constantes no sistema.

O estudo sobre DHT objetivou a escolha de uma abordagem que se mostrasse mais apropriada para os requisitos de distribuição dos servidores HNS. Como pode ser visto, as diferentes abordagens existentes são bastantes semelhantes nos mais diversos aspectos analisados. Este fato incentivou o uso de outro aspecto para avaliação da abordagem a ser usada. Para que se tenham resultados reais sobre os impactos dos requisitos do HNS sobre uma abordagem DHT é preciso criar um protótipo do sistema. Este fato motivou a busca por implementações dos diferentes protocolos que pudessem auxiliar na prototipação. Neste aspecto, o único protocolo que possui uma implementação estável, documentada e que cumprisse os requisitos da linguagem e do ambiente de programação foi o Kademlia, com uma implementação chamada GNUNet [71].

Quanto aos protocolos para descoberta de serviços, estes tinham um objetivo bastante específico dentro do trabalho e este fator é o decisivo na escolha da solução a ser utilizada, que é o auxílio na configuração automática do sistema, ou seja, permitir que os diferentes módulos do software se encontrem automaticamente. O Jini é uma solução bastante interessante, bem documentada e madura em seu desenvolvimento, mas possui um problema básico, que é a compatibilidade da solução, já que ele é implementado em Java. O SLP por sua vez, para que seus diferentes módulos possam se encontrar em uma rede, precisa ter o DHCP disponível e configurado com algumas opções específicas e que não são usadas por padrão.

Quanto a Bonjour e UPnP, ambos possuem implementações semelhantes, que se baseiam em padrões já conhecidos da internet, ambos possuem implantações disponíveis para uso, com apenas uma diferença essencial neste aspecto, o UPnP precisa de um ponto de controle para centralizar a descrição de seus serviços, e este *software* só é encontrado em algumas versões do Windows, enquanto que o mecanismo de centralização do Bonjour tem seu código distribuído junto com o resto da API, incluindo versões para as mais diferentes plataformas, podendo ainda ser integrado ao sistema. Outro fator que pesa a favor do Bonjour é que, apesar de a princípio sua estratégia de descoberta funcionar apenas para redes locais, já que é baseada em *multicast*, o fato de usar o DNS como forma de anúncio dos serviços possibilita que sejam configurados servidores DNS na rede aberta para que o sistema de configuração funcione de uma forma ainda mais ampla.

Capítulo 5

Holo Naming System (HNS)

Neste capítulo será apresentado o modelo do HNS (*Holo Naming System*), um sistema que visa o suporte para execução distribuída de programas Holo. Para isso, é necessário levar em consideração dois problemas principais. O primeiro é a localização de entes em um ambiente de execução distribuído. O segundo problema é, partindo da informação sobre a localização de um ente, prover uma camada de execução para que duas HoloVMs possam dar suporte ao uso de todos os recursos propostos pelo Holoparadigma, que envolvam a interação entre entes. Este modelo objetiva cobrir os requisitos de ambientes móveis (MHolo), mas com a sua visão voltada para a próxima geração de sistemas distribuídos pervasivos (PHolo).

5.1 Objetivos do Modelo

O suporte aos **conceitos do Holo** constitui o requisito mais básico deste modelo. Um dos objetivos do Holoparadigma é oferecer uma abstração mais poderosa para a criação de sistemas para a computação móvel. Desta forma, não só o modelo proposto deve oferecer total suporte a este tipo de ambiente, mas também o poder oferecido pela abstração na linguagem não deve ser prejudicado com um sistema que possua muitos passos de configuração, tanto através da própria linguagem quanto no ambiente de execução. A linguagem deve sofrer alterações mínimas, apenas as necessárias para que o programador seja capaz consultar informações sobre o ambiente de execução. O resto deve ser mantido sem modificações e todo suporte à distribuição deve ser usado sem que seja preciso conhecer nada mais do que a própria linguagem e mesmo programas já existentes possam ser facilmente adaptados à execução distribuída.

Este modelo deve também oferecer uma forma eficiente de manter o controle sobre a **localização de entes** executando em um ambiente distribuído. Este sistema de localização de entes deve se manter sempre coerente com o estado dos entes em execução, sendo este um dos desafios do modelo. Isto por que um ambiente de execução que depende da comunicação e que engloba dispositivos móveis é altamente dinâmico. Em um ambiente como este, modificações ocorrem freqüentemente, modificações estas que podem ser de criação/remoção de entes ou

de execução de comandos *move*, que modificam a estrutura da HoloTree e portanto devem ser propagados para o sistema. Este é um aspecto único do Holoparadigma e é necessário saber qual o impacto disto no sistema como um todo.

Para fazer uso direto deste sistema de localização é preciso criar um suporte à **execução distribuída**, que fará uso efetivo das consultas sobre localização de entes para promover a interação entre entes dispersos em diferentes nodos de uma rede. Este suporte deve ser integrado ao suporte de execução já existente no projeto, de forma a complementar as funcionalidades do mesmo.

Além destes requisitos, que podem ser considerados básicos para o sistema, é preciso considerar algumas questões que foram levantadas pelos estudos sobre computação móvel e pervasiva. Em ambientes como estes, é bastante comum que usuários estejam constantemente entrando e saindo da rede, tornando a disponibilidade de recursos incerta. Neste cenário a **tolerância a falhas** é uma característica importante. Assim, é preciso que o sistema esteja sempre consciente do comportamento dos nodos envolvidos no sistema, saber quando eles entram ou saem e informar isso a todos envolvidos no processo, além de ser capaz de se recuperar de falhas em certos pontos do sistema.

Em um sistema como o proposto, onde existem diversos entes executando simultaneamente e estes precisam constantemente fazer consultas a informações disponíveis em servidores, isto pode criar um problema de **escalabilidade** no sistema, demandando assim uma solução escalável, tanto para o armazenamento de dados quando para a resposta a requisições para estes dados.

Nas próximas seções deste capítulo serão abordados os diversos aspectos do modelo criado para o HNS.

5.2 Modelo HNS

O HNS não consiste apenas em um serviço de nomes, mas sim em uma infra-estrutura para suporte à execução de programas Holo em ambientes distribuídos, levando em consideração a consistência nos dados armazenados, abstração total do uso da rede, escalabilidade do serviço e atenuação de falhas em elementos do sistema, sejam HoloVMs ou servidores HNS. Na Figura 5.1 é mostrada a organização deste modelo.

Como ilustra a Figura 5.1, o modelo possui duas camadas. Na superior estão as HoloVMs, que são responsáveis pela execução de código Holo implementado pelas aplicações. Estas HoloVMs também têm o encargo de mapear as abstrações da linguagem para interações com outros elementos do modelo, utilizando conexões de rede para isso. As HoloVMs comunicam constantemente o estado de execução dos programas para os servidores HNS, para que esta informação possa ser consultada globalmente no sistema. Tais consultas são feitas pela HoloVM sempre que a execução de determinada instrução torna necessária a interação com um ente, o qual ela não possui executando localmente. A princípio cada HoloVM se comunica apenas

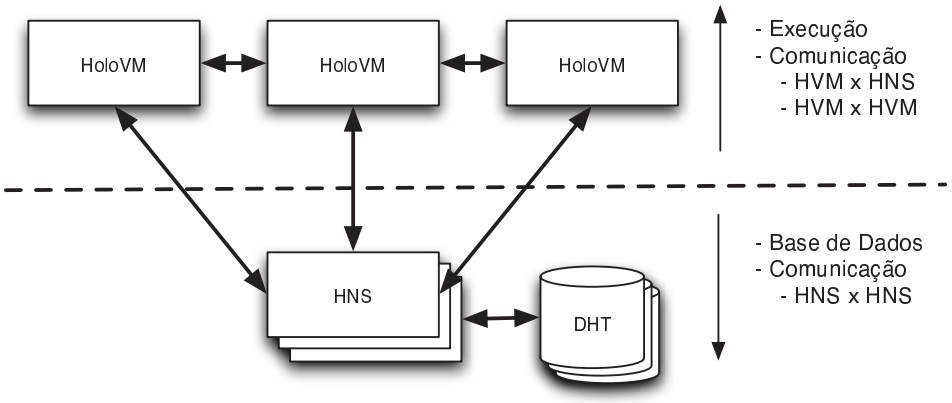


Figura 5.1 – Modelo HNS

com um servidor, mas por precaução ela conhece a localização de outros servidores, como uma forma de contornar uma possível falha do HNS em uso (seção 5.6.1).

Na camada inferior do modelo está a estrutura responsável pelo gerencimento dos dados relativos a execução dos programas Holo. Os servidores HNS devem possuir pleno conhecimento da estrutura global do programa executando no sistema, sendo capazes de responder consultas sobre a estrutura desta árvore de execução da forma mais ágil possível. Os servidores são responsáveis ainda pela abstração da organização de sua base de dados sobre os entes em execução. Por questões de tolerância a falhas e escalabilidade esta base de dados é distribuída entre uma rede de servidores. O uso de DHT (seção 5.2.1) como estratégia para organização dos dados facilita o gerenciamento dos mesmos em um ambiente distribuído, já que esta tecnologia oferece soluções para balanceamento de carga e roteamento de mensagens em uma estrutura de servidores distribuída.

5.2.1 Estratégia de distribuição dos servidores

O contexto no qual se encaixa o HNS, conforme abordado na seção 3.3, é o de suporte à execução distribuída para um ambiente pervasivo. Um ambiente como este representa um desafio considerável, pois traz com ele características que se tornam requisitos importantes para o sistema. Dentre os requisitos já citados para o HNS, dois deles possuem uma forte dependência na maneira como as informações são armazenadas pelos servidores, são elas: escalabilidade e tolerância a falhas.

Existem diferentes estratégias pelas quais se pode atingir estas características em uma estrutura de servidores distribuídos, contudo, a computação pervasiva demanda que a escalabilidade e tolerância a falhas do sistema seja elevada a níveis ainda mais altos que o convencional, pois neste ambiente poderiam existir milhares de HNSs e um número ainda maior de HoloVMs executando simultaneamente.

O DNS [45, 46, 47] é um bom exemplo de estratégia bem sucedida para uma estrutura

robusta de servidores, com sua organização de servidores e particionamento dos dados hierárquica. Para usar tal estrutura porém, o DNS se baseia no fato de que os registros em sua estrutura praticamente não se alteram e de que o nome de cada registro carrega em si todas as instruções para o roteamento da busca pela informação nos servidores. No caso do HNS, nenhuma destas propriedades pode ser assegurada, já que em um ambiente dinâmico como o pervasivo, as entidades que devem ser localizadas pelo sistema podem ser criadas, destruídas ou mudar de lugar a qualquer momento e sem aviso prévio. Desta forma torna-se necessário criar uma estrutura de servidores capazes de guardar uma grande quantidade de informações e de responder consultas por estas o mais rápido possível.

Atualmente existe uma tecnologia que vem despertando o interesse de pesquisadores e tem sido utilizada com sucesso em sistemas p2p. Esta técnica é chamada de *Distributed Hash Tables* [49, 51, 50, 72, 48]. O DHT surgiu para resolver um problema crítico de escalabilidade que existia nas primeiras propostas p2p, já que estas utilizavam estratégias centralizadas para armazenar dados sobre os recursos que estavam disponíveis na rede. O DHT é hoje umas das tecnologia mais robustas existentes para armazenamento de dados distribuídos e seu uso vem se multiplicando para os mais diversos tipos de sistemas.

O DHT foi a estratégia escolhida para solucionar o problema de escalabilidade e tolerância a falhas do HNS para que este seja capaz de atender a sistemas em larga escala. Um estudo sobre as estratégias de DHT mais conhecidas foi mostrado na seção 4.2. Dentre os sistemas estudados um foi escolhido para ser incorporado ao modelo aqui apresentado. Nas demais seções deste capítulo serão abordadas questões relativas ao funcionamento do modelo HNS integrado a estratégia de DHT escolhida.

5.2.2 Suporte à execução distribuída

Um dos requisitos do HNS é que a utilização do suporte para execução de programas distribuídos seja transparente para o programador. No Holoparadigma existem duas maneiras de dois entes em execução interagirem: (1) fazendo acesso à história do seu ente pai e (2) executando ações de outros entes. O processo necessário para realizar estas tarefas entre máquinas separadas por uma conexão de rede não é recente. No caso do acesso a história, o funcionamento pode ser visto como um sistema de *Distributed Shared Memory* (DSM) [73], tendo como regra o fato de que um único *blackboard* não será acessado em nodos diferentes. Para interagir com ele é necessário se comunicar com a HoloVM na qual o *blackboard* foi criado. Desta forma é possível diminuir significativamente o custo geral da solução, já que sistemas de DSM que utilizam replicação de dados podem ter um custo alto em processamento, uso de memória e de banda de rede, recursos estes que sistemas móveis e pervasivos nem sempre têm disponível.

No segundo tipo de interação, execução de ações, também existem tecnologias maduras que desempenham um papel semelhante para linguagens de programação estabelecidas no mercado. Este conceito permite a abstração da execução de pedaços de código que não se

encontram no mesmo espaço de memória do processo em execução. A idéia é abstrair a complexidade de uma conexão de rede na execução deste tipo de tarefa. Duas soluções bem conhecidas são o *Remote Procedure Call* (RPC) [74, 75, 76], usado em linguagens procedurais, em especial C/C++, e o *Remote Method Invocation* (RMI) [77] para a orientação a objetos, que foi criado para ser usado no Java. Em ambos os casos a idéia é tornar transparente o processo de comunicação necessário para executar o código na máquina remota. Ainda assim, existe uma série de passos necessários para fazer um programa funcionar com RPC/RMI, o que torna transparente o uso da rede mas coloca na linguagem e na compilação dos programas alguma complexidade adicional. Na Figura 5.2 é possível fazer uma comparação entre a complexidade das duas soluções, RPC e Holo.

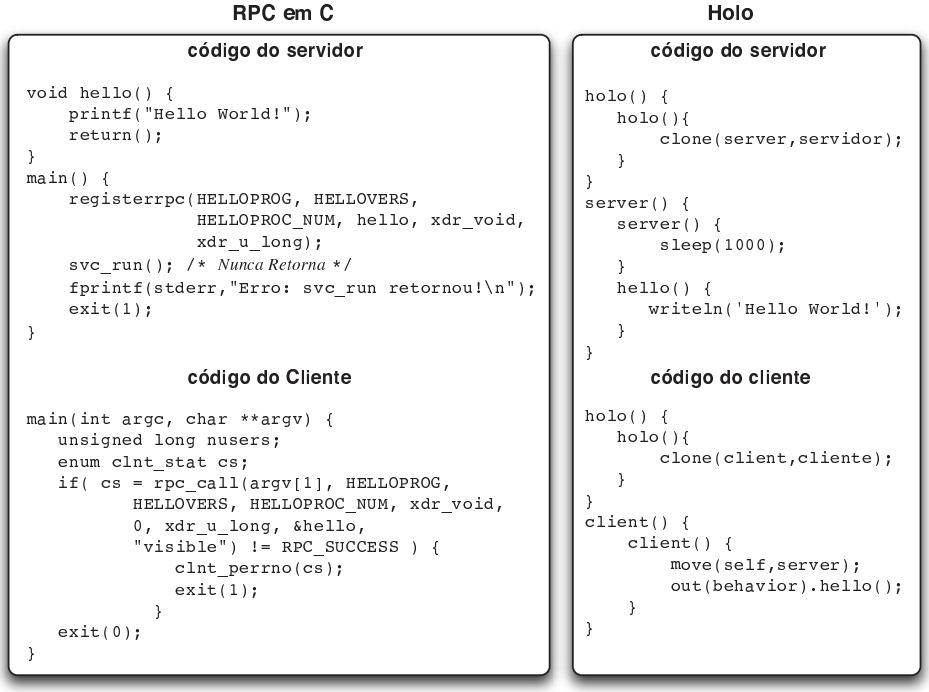


Figura 5.2 – Comparação da execução remota de código entre Holo e RPC.

No código do RPC são abstraídos os cabeçalhos do programa, mantendo apenas um exemplo simples de código, e mesmo neste nível pode ser notada a necessidade de usar funções com vários parâmetros para registrar e chamar o serviço desejado. E além de alguns detalhes de código que foram abstraídos, ainda existe todo o processo de compilação, que demanda vários passos para criar os binários, *stubs* e filtros para cliente e servidor. No caso do Holo, o sistema tem a vantagem da organização dinâmica na estrutura da aplicação para facilitar a execução remota da ação. No exemplo da Figura 5.2 é mostrado um programa em Holo que usa o paradigma cliente/servidor, assim como o RPC, onde no lado do servidor é preciso apenas criar os entes com as ações desejadas. No cliente está a lógica que permite a execução de outro ente, onde o ente *cliente* executa o comando *move*, com os parâmetros *self*, referindo

a mobilidade do próprio ente, e *servidor*, indicado que o ente cliente passa a compor o ente servidor. O comando seguinte, chamado *behavior*, permite o acesso ao comportamento de um ente. Juntamente com a opção *out*, que indica a execução do comando para o ente pai de *cliente*, no caso o *servidor*.

Esta funcionalidade é de grande importância para ambientes pervasivos, uma vez que permite a criação de aplicações que têm a capacidade de adquirir funcionalidades ao longo de sua execução, através da execução de ações que pertencem a entes executando em máquinas remotas. A execução remota de ações também permite que máquinas com menor poder computacional, como os dispositivos móveis por exemplo, utilizem os recursos de máquinas mais poderosas para executar suas tarefas. A HoloVM implementa ainda o comportamento dos entes usando a abordagem de *blackboards*, o que permite não só a execução, mas também permite que ações sejam copiadas de um ente para outro. Mapeando isto para um ambiente distribuído temos uma forma de mobilidade de código que é suportada pelo ambiente, que permite não só a execução de ações, mas possibilita também que um ente adquira novas funcionalidades permanentemente.

5.3 Adaptando o sistema de DHT à abordagem do HNS

A organização mencionada na seção anterior guia a modelagem de todos os aspectos deste sistema. Nesta seção é abordado o funcionamento da camada de comunicação do HNS, primeiramente mostrando como funciona a distribuição dos servidores e a consulta a informações armazenadas nos mesmos, para depois abordar a estrutura das mensagens utilizadas no suporte à distribuição.

5.3.1 Formato do registro

Sistemas de DHT são usados para mapear uma chave para os mais diversos tipos de dados, podendo ser desde arquivos em sistemas de compartilhamento até endereços em sistemas de nomes. No caso do HNS existe a necessidade de guardar não apenas a localização do ente, mas também toda a estrutura da HoloTree, ou seja, a partir dos registros distribuídos no sistema deve ser possível construir uma estrutura em árvore completa do programa. Para suprir esta demanda cada registro tem o formato de uma lista duplamente encadeada, com a diferença de que, para guardar uma estrutura de árvore, uma das pontas possui uma lista de chaves (filhos) enquanto a outra possui apenas uma chave (pai). A chave referenciada aqui é o número único gerado pela função *hash* para cada ente. Esta estrutura de dados pode ser vista mais claramente na Figura 5.3.

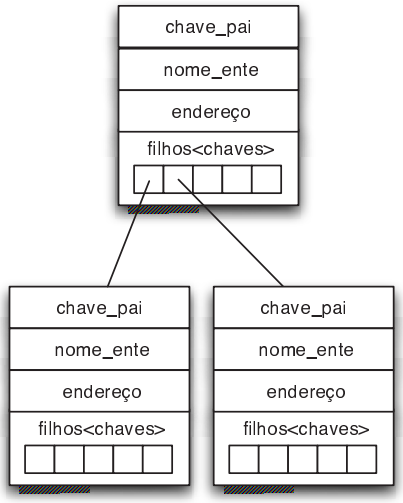


Figura 5.3 – Dados armazenados em cada registro nos servidores HNS

5.3.2 Mensagens relacionadas ao funcionamento dos servidores

Sobre este aspecto da comunicação no sistema vale destacar que não é dada ênfase a forma como a abordagem DHT distribui seus dados na arquitetura, ou como ela faz o roteamento das mensagens, já que isso é um aspecto de implementação. O aspecto a ser ressaltado aqui é a estratégia necessária para adaptar o comportamento de um sistema DHT para as consultas que devem ser suportadas pelo sistema. Existem basicamente dois grupos de instruções na HoloVM que disparam processos no HNS. No primeiro grupo estão as que servem para informar o servidor sobre uma alteração na árvore, ao segundo grupo pertencem as que consultam o servidor para saber a localização física de um determinado ente.

Na HoloVM existem duas instruções, que pertencem ao primeiro grupo, e que vão sempre reportar ao HNS quando forem desempenhadas, são elas: *clone* e *move*. O *clone* é responsável pela criação de novos entes na aplicação e o *move* faz a mobilidade de um ente na *HoloTree*. No segundo grupo estão todas as instruções que acionam, de forma indireta, uma consulta aos servidores buscando pelo ente com o qual precisam interagir para completar o processo. São exemplos destas instruções: *out(behavior)* e *out(history)*, que dizem respeito a uma ação no comportamento ou história de um ente no nível superior da composição.

A Figura 5.4 mostra em detalhes as mensagens necessárias no sistema para desempenhar as instruções citadas acima. Nela é mostrado uma configuração fictícia de servidores e HoloVMs, onde duas máquinas virtuais executam um mesmo programa, o qual sofre alterações que demandam interações com o servidor.

- **Inserção e Remoção** - Para desempenhar o processo de inserção de um ente na árvore de execução, a HoloTree, o ente executa uma instrução *clone*. No caso da Figura 5.4 esta ação é desempenhada pelo ente **004** para criar o ente **006**. A partir dai a **HoloVM2** envia uma mensagem ao **HNS4** informando a existência de um novo ente (mensagem **I1**), este

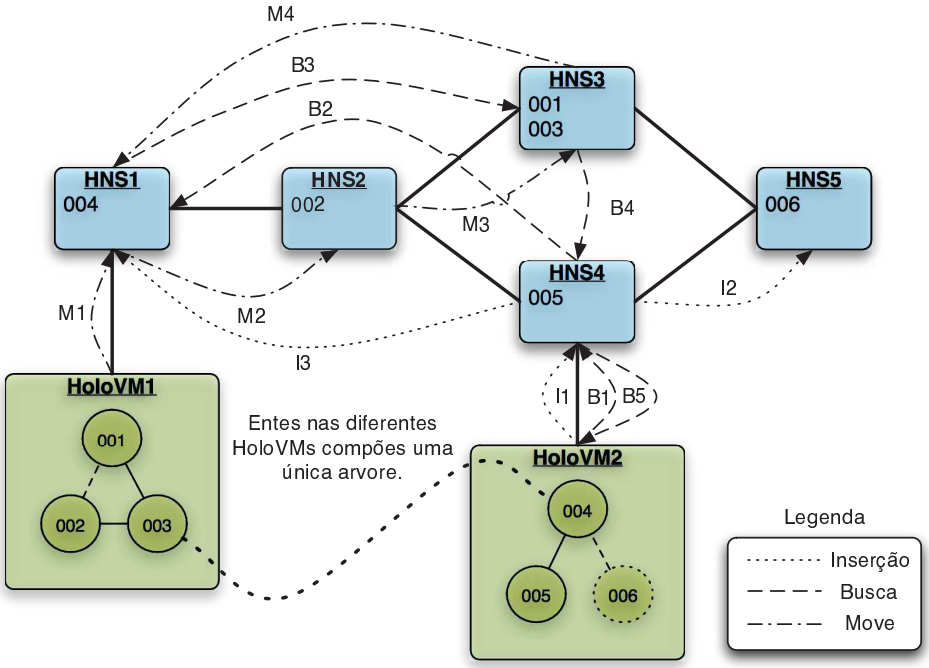


Figura 5.4 – Troca de mensagens no gerenciamento da base de nomes

HNS então cria uma chave para o novo registro e manda a mensagem de criação para o servidor que irá armazenar as informações do ente criado (mensagem **I2**). O **HNS4** também deve informar ao servidor **HNS1**, que é responsável pelas informações deste ente **004** (mensagem **I3**), para que ele atualize a lista de filhos deste ente. O processo de remoção do registro de um ente no sistema é bastante semelhante a inserção, é adicionado apenas mais um passo para que os registros relativos aos entes componentes deste ente removido sejam atualizados com a modificação;

- **Busca de informações** - No caso da busca, a informação desejada é o endereço de rede da HoloVM que executa determinado ente. Mas esta consulta pode ser feita de maneira bastante indireta, deixando a cargo dos HNSs descobrirem todas informações necessárias. No exemplo dado, a **HoloVM2** está consultando a localização do pai, mas a mensagem enviada (**B1**) vai pedir apenas pela localização do ente **004**, mesmo a **HoloVM** não sabendo nem quem ele é. O **HNS4** vai então localizar o registro de um ente **004** para descobrir quem é o seu pai, o **HNS1** recebe a mensagem (**B2**), recupera a informação e faz uma nova consulta no sistema procurando agora pelo ente **003**, que já se sabe ser o pai de **004**. O **HNS3** então recebe a requisição pelo endereço de **003** (**B3**) com uma indicação de quem é o requisitante, para que ele possa enviar a resposta. De posse de informação então ele envia esta para o **HNS4** (mensagem **B4**), que é quem possui contato com a **HoloVM2**, que por sua vez passa a mensagem para a máquina virtual (**B5**);

- **Mobilidade** - O último processo representado na Figura 5.4 diz respeito a atualização dos

registros disparada pelo uso do *move*. No caso o ente **002** executou uma mobilidade de **001** para **003** (*move(self,003)*), isto faz a **HoloVM1** enviar uma mensagem (**M1**) para o **HNS1** informando a modificação. Este então dispara uma mensagem (**M2**) procurando pelo ente **002**, que é recebida pelo **HNS2**, que então continua a atualização informando (mensagem **M3**) o ente **001** que **002** não é mais seu filho e informando **003** (mensagem **M4**) que **002** agora faz parte de sua composição.

5.4 Comunicação entre HoloVMs

A comunicação entre HoloVMs é mais simples do que entre HNSs, no sentido que envolve apenas dois elementos na comunicação, que interagem usando um paradigma cliente/servidor, onde cada HoloVM pode desempenhar qualquer um dos dois papéis, ou os dois simultaneamente. Os processos de comunicação remota são executados de forma transparente quando a HoloVM executa determinados comandos que envolvem a comunicação de um ente local com um ente remoto. Primeiramente a HoloVM consulta o HNS ao qual está associada em busca do endereço de um ente, depois ela tenta estabelecer uma conexão direta com a HoloVM desejada. É importante destacar que neste trabalho não são considerados os obstáculos de uma rede aberta, como *firewalls* e NATs, pois existem soluções que podem ser agregadas no sistema posteriormente.

Analisando a Hololinguagem foram identificados quatorze instruções que podem desencadear um processo de comunicação entre HoloVMs, sendo que destas quatorze, cinco são referentes a história e nove ao comportamento.

5.4.1 Acesso a história de um ente pai

Um detalhe no acesso a história é que apenas as instruções com a prerrogativa *out* possuem a propriedade de fazer o acesso a dados em um ente externo, no caso o seu ente pai. As instruções identificadas são as que seguem:

- **out(history)!** - Afirmação à história de um ente pai;
- **out(history).** - Lê uma tupla sem apagar a mesma, mas bloqueia a consulta caso a tupla não exista;
- **out(history)#** - Lê a tupla apagando a mesma, bloqueia a consulta se a tupla não existir;
- **out(history)?** - Lê uma tupla da história se existir uma que satisfaça os requisitos, não bloqueia caso ela não exista;
- **out(history)?#** - Lê uma tupla apagando a sua cópia na história, não bloqueia se não existir a tupla.

Dentre estas instruções existem duas que precisam de um tratamento diferenciado por parte da camada de comunicação, que são as leituras bloqueantes da história. Nestes casos o sistema deve simular um bloqueio no ente que fez o acesso remoto e ao detectar a escrita da tupla desejada deve executar a ação e informar o resultado a quem fez a solicitação. Esta questão exige atenção, pois um ente que fez um acesso remoto deve ter a mesma chance de ler a tupla que um ente local.

5.4.2 Executando ações de outros entes

No caso da execução de ações, a Hololinguagem oferece diferentes formas de acessar ações de outros entes. No acesso a história apenas a instrução *out* executa acessos externos, já na execução de ações existem três formas de acesso, são elas:

- **out(behavior).acao()** - Executa ação que está no comportamento de um ente pai;
- **out(ente).acao()** - Executa ação em um ente irmão;
- **ente.acao()** - Permite que um ente execute ações de seus filhos.

As possibilidades na execução de ações podem ser melhor vistas na Figura 5.5. Nela existem quatro formas de usar as ações, as três que foram listadas e mais uma que é referente a um ente executando uma ação própria.

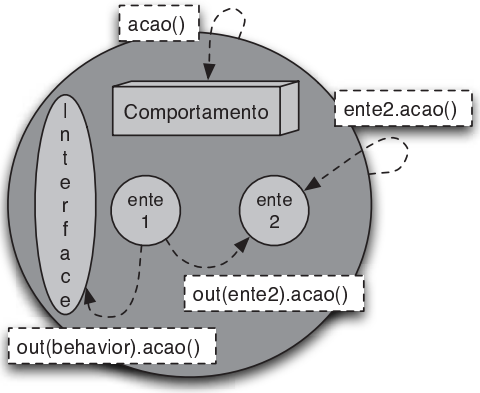


Figura 5.5 – Formas de executar ações que são permitidas pela linguagem.

Adicionalmente, cada uma destas formas de acesso possui variações de uso. Nos exemplos dados é usado o ponto (“.”), que indica a execução de uma ação, mas além deste ainda existem mais dois, que são: o sustenido (“#”) que é capaz de destruir a ação indicada e a exclamação (“!”) que copia a ação referenciada na instrução, do comportamento do ente que a executou para um ente externo, conforme as formas de acesso já citadas.

Uma questão que fica bastante explícita aqui é a segurança, ou seja, como um sistema que deixa entes se moverem livremente, apagar e escrever ações dos outros pode ser viável. Apesar

de não ser o foco deste trabalho o controle de acesso no nível de linguagem, uma solução que foi desenvolvida no contexto do projeto MHolo será abordada na seção 5.8.3.

5.5 Integrando um sistema de descoberta de recursos ao modelo

O modelo apresentado é capaz de armazenar a estrutura de uma aplicação Holo que se encontra executando em diversas HoloVMs e responder consultas sobre estes dados. Além disso é fornecida uma camada de abstração para execução distribuída. É sabido que, em um ambiente pervasivo, o usuário deve ser capaz de utilizar o sistema, e todos os recursos a ele associados, sem ter que se preocupar em como fazer isso. Assim surge o requisito para próxima funcionalidade do modelo a ser abordada, cujo objetivo é prover uma camada de auto-configuração para o sistema. Esta camada torna o sistema tão auto suficiente quanto possível, facilitando ainda mais o desenvolvimento da aplicação e reduzindo o envolvimento de usuários com tarefas administrativas.

Para resolver esta questão será utilizada uma solução baseada em um sistema de descoberta de serviços (em inglês SD). Sistemas de SD se propõem a localizar uma ou mais peças de *software* executando em um ambiente distribuído que ofereça o serviço desejado. O SLP (*Service Location Protocol*) [58] por exemplo define um protocolo e componentes de *software* que devem ser agregados a uma aplicação para que ela seja capaz de localizar serviços em uma rede. Além do SLP existem outras tecnologias que se propõem a resolver este problema com abordagens semelhantes, dentre elas podemos citar: UPnP (Universal Plug an Play) [63], Jini [61, 62] e Bonjour [65].

Além do Bonjour ser amplamente aceito e possuir diversas implementações, esta tecnologia implementa uma solução de baixo custo computacional, de banda de rede, administrativo e supre perfeitamente as funcionalidades desejadas para este modelo. No modelo do HNS o Bonjour é responsável por:

- Automatizar a entrada no sistema por parte das HoloVMs, que irão descobrir sozinhas o endereço dos HNSs disponíveis na rede na qual elas estão executando. Com a evolução do Bonjour, esta tecnologia vai permitir que dispositivos se encontrem inclusive em uma rede aberta, permitindo também que façam a sua inicialização sozinhos através do uso de um DNS público;

- Permitir que HoloVMs e HNSs tenham controle permanente de quem está participando do sistema. Uma vez que os serviços estão anunciados é mantido contato permanente para que a aplicação saiba quando um dos elementos deixou de existir, ou seja, quando uma HoloVM ou um HNS deixaram de executar ou saíram da rede. Esta consciência permite que o sistema execute suas estratégias para se recuperar da falha.

Na Figura 5.6 pode ser visto um exemplo simples de como o Bonjour se integra ao modelo. Neste exemplo, um usuário entra com seu computador na rede, com uma HoloVM

executando em sua máquina. Como o protocolo do Bonjour tenta constantemente se manter atualizado com os serviços disponíveis na rede, a **HoloVM1** envia uma mensagem em *broadcast* consultando a existência de um servidor HNS. Essa mensagem será recebida por todos os equipamentos que estejam na mesma sub-rede, incluindo a **HoloVM2**, que a princípio desconsidera a mesma. O **HNS** porém ao receber o contato, responde com uma mensagem **unicast** para a **HoloVM1**. Assim que a **HoloVM1** toma conhecimento da existência de um HNS em sua rede ela pode então dar início a um processo de *bootstrap* para começar a sua participação neste novo ambiente de execução. Para este foram criados dois serviços que são anunciados no Bonjour pelos componentes do sistema , são eles: _holovm._tcp e _hns._tcp (o formato dos anúncios é explicado na seção 5.5).

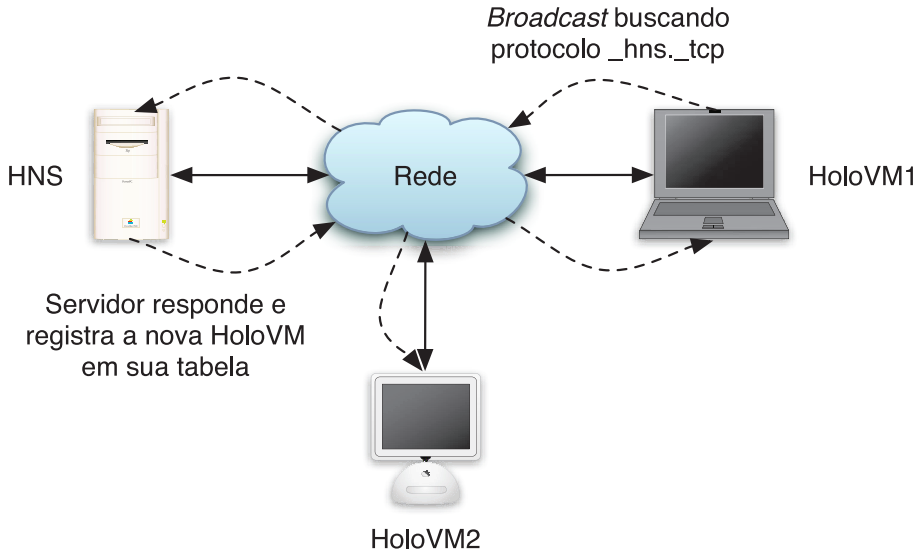


Figura 5.6 – Processo de descoberta com o Bonjour.

A próxima seção apresenta em detalhes como o Bonjour é capaz de auxiliar o sistema na detecção e recuperação de falhas.

5.6 Detectando e contornando falhas nos componentes do sistema

Em aplicações que executam em apenas uma máquina, o comportamento esperado é que as únicas falhas sejam as inseridas pelo programador e que o ambiente dificilmente irá falhar. Quando se fala de uma aplicação distribuída entre várias máquinas, o número de elementos que pode causar problemas aumentam consideravelmente, dentre eles o principal é a disponibilidade da rede que interliga as máquinas envolvidas na computação. No caso específico deste trabalho, onde se tem como base sistemas pervasivos, os ambientes de execução criados possuem diversos serviços, que refletem em um maior número de elementos de suporte à aplicação com possibilidade de falhas que poderiam refletir diretamente em seu funcionamento.

No caso deste modelo, existem duas peças de *software* (HNS e HoloVM) que são diretamente responsáveis pela execução e que podem apresentar falhas, seja na comunicação, no ambiente em que se encontram ou em seu próprio funcionamento. A idéia é que, se o sistema for consciente de todos elementos envolvidos e de seus estados, ele será capaz de se recuperar de falhas nos mesmos. Note que o objetivo aqui é tratar o problema no nível do ambiente de execução, visando manter o sistema funcionando mesmo na ocorrência de falhas.

5.6.1 Contornando a falha de um servidor HNS

No modelo apresentado cada HNS deve desempenhar dois papéis principais: (1) servir de interface entre um grupo de HoloVMs e os demais servidores em execução; (2) ser parte ativa do sistema de armazenamento de dados das aplicações. No caso de falha em um HNS o que se perde é uma parte da tabela DHT que está distribuída no sistema. A maneira de se recuperar de uma falha desta natureza é construíndo estratégias de replicação dos dados armazenados. Contudo, a replicação já é uma parte integrante do sistema, já que os dados armazenados nos HNSs são na verdade informações sobre uma aplicação que está executando nas HoloVMs do sistema.

Uma forma de contornar uma falha em um HNS é fazendo com que as HoloVMs saibam o endereço de rede de outros servidores que se encontram disponíveis. Uma HoloVM então é capaz de rapidamente notar a falha de um elemento da arquitetura, isso devido ao uso do Bonjour, e tendo consciência deste fato ela é capaz então de contatar outro HNS e enviar todas as informações que possui sobre a aplicação que está executando (Figura 5.7). Para agilizar o processo uma HoloVM pode armazenar inclusive informações como a chave que identifica o pai de algum ente que não se encontra na própria HoloVM, para que esta informação não precise ser localizada no sistema.

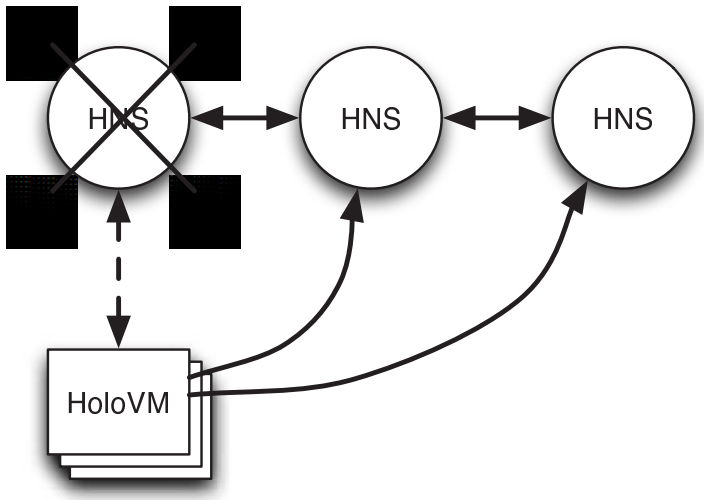


Figura 5.7 – Exemplo de falha em HNS.

5.6.2 Atualizando o sistema sobre a falha de uma HoloVM

Quanto as falhas em HoloVMs, existe pouco que o sistema possa fazer a respeito para minimizar a propagação dos efeitos causados pelo problema. Uma ação possível é atualizar o sistema o mais rápido possível, para que não seja feita nenhuma tentativa de acesso a uma HoloVM que já não se encontra executando. Novamente, a detecção da falha é feita com a ajuda do Bonjour, além disso, cada HNS deve manter uma lista das HoloVMs que conectam ao sistema através dele e as chaves dos entes que se encontram executando em cada uma delas. Assim que é perdido o contato com uma HoloVM, o HNS responsável pode dar início ao processo de remoção, para que todos os entes que se encontravam em execução na VM que falhou sejam retirados do sistema.

Ainda assim é possível que ocorram falhas em aplicações que estão em execução e que tentam o acesso a entes que não existem mais. Mesmo que o sistema seja consciente das maioria das falhas, a HoloVM ainda não oferece um mecanismo onde o ambiente possa enviar mensagens ao usuário, como um sistema de tratamento de exceções por exemplo. Tal sistema é sugerido na seção 5.8.2.

5.7 Técnicas para otimização do sistema

O modelo apresentado neste capítulo mostra uma abordagem para balanceamento de carga e tolerância a falhas no servidores HNS utilizando uma abordagem de DHT. Para que isto fosse possível foi necessário criar alterações no modelo original do DHT para que ele fosse capaz de armazenar os dados necessários bem como responder consultas por estes dados. Estes novos processos aumentam a complexidade original das operações desempenhadas pela rede de HNSs. Por este motivo, são sugeridas estratégias que visam melhorar o desempenho do sistema em alguns casos específicos.

5.7.1 Cache de consultas

Uma estratégia obrigatória para este tipo de sistema é o uso de *caches* no sistema. Em uma estratégia DHT como a do Kademlia, quando se dispara uma consulta, cada nodo utilizada uma função que calcula a proximidade da *hash* procurada em relação as *hashs* dos servidores que estão na tabela de roteamento e envia a busca para o nodo que for considerado mais próximo do que armazena o registro procurado. Este processo se repete até que o objetivo seja atingido. Isto faz com que, em uma topologia com diversos servidores, exista uma probabilidade de que a mensagem passe por alguns nodos antes de atingir o destino. O Kadamlia tira proveito desta característica do sistema para criar uma estratégia de *cache*, onde uma chave que comece a ser mais consultada será replicada em nodos próximos para que se atinja o objetivo da busca antes mesmo de chegar no nodo que originalmente armazenava a mensagem. Quanto mais popular a chave, maior o número de nodos onde ela poderá ser encontrada. No entanto é preciso evitar o

que é chamado de *over-caching*, ou seja, o excesso de cópias de um registro no sistema. Para isso é utilizada uma fórmula própria do sistema Kademlia que calcula o tempo de vida de cada registro em *cache*.

Devido a este método, que leva em conta a topologia da rede para criar a *cache*, o sistema tende a estabilizar em um estado onde a estrutura da HoloTree deve refletir na forma como as cópias das informações são distribuídas nos nodos da rede. Isto por que a estrutura das operações que o sistema desempenha se utilizam de consultas por entes que se relacionam diretamente na HoloTree. Diz-se que o sistema "tende a estabilizar" por que as aplicações em Holo são altamente dinâmicas e sua estrutura está sempre se modificando.

5.7.2 Cache proativa

A integração com um sistema DHT insere complexidade adicional nas operações do sistema, mas também traz benefícios. Um exemplo disso pode ser aplicado na política de *cache* do sistema, que pode tirar vantagem da forma de organização das aplicações em Holo. Estas aplicações, apesar de serem dinâmicas em sua composição, tem suas interações mapeadas, o que possibilita tentar prever a próxima consulta de uma HoloVM e antecipar a mesma, para que quando ela seja feita, o resultado já tenha sido encontrado.

Um exemplo disso é a comunicação de um ente com o seu pai. Em uma aplicação mais complexa, existe uma boa probabilidade de que alguns entes da aplicação tenham seus dados e ações acessados por seus entes filhos. Em estudo de caso apresentado na seção 6.3 por exemplo, é proposta uma abstração para aplicações pervasivas onde uma estrutura lógica de entes reflete uma estrutura física. Assim, aplicações de usuários e serviços podem se organizar de acordo com suas localizações físicas, usando entes como agregadores. Uma forma de aplicar a *cache* proativa seria fazendo a consulta da localização deste ente agregador assim que um ente se move para dentro dele. Desta forma, assim que o ente busca um serviço ou informações do ambiente, o sistema está pronto a passar a informação necessária.

5.8 Sugestões para melhorar a implementação da Hololinguagem

A Hololinguagem se encontra em constante evolução, com o objetivo de melhorar a sua abstração e principalmente as funcionalidades para que ela oferece. Ao longo da elaboração deste trabalho surgiram sugestões que podem ser úteis no sentido aumentar o controle do programador sobre a aplicação.

5.8.1 Execução distribuída

No estudo sobre soluções para execução distribuída, uma das soluções abordadas foi o RPC e dele surgiu uma constatação simples e que ainda assim pode ser útil em programas Holo. No programa servidor do RPC, após ser feita a declaração das funções disponíveis, o

programa chama uma instrução cujo nome é *rpc_run()*, a qual irá colocar o programa em espera por chamadas externas. Apesar de ser algo bastante simples, é também extremamente útil para programar aplicações que são apenas provedoras de serviços e/ou centralizadoras de dados para um sistema maior. Na Hololinguagem não existe uma forma simples de fazer isso. Desta forma é proposta a criação da função *wait()*. Ao ser executada esta função mantém o ente em espera por requisições, que serão atendidas pela própria HoloVM, sem que o ente saia deste estado de espera.

5.8.2 Tratamento de exceções na linguagem

Uma questão que já foi citada no trabalho é a falta de uma maneira de o sistema comunicar o programador de algum fato inesperado durante a execução de uma aplicação. A maneira mais usual de tratar este problema nas linguagens de programação atuais é através do uso de exceções. No Holo não existe tal mecanismo, porém a sua importância parece ser inegável. A criação de um mecanismo como este vai além do desafio técnico de implementar tal suporte na HoloVM, pois é necessário estudar e definir qual a melhor forma de criar este conceito na Hololinguagem. O Holoparadigma oferece uma forma diferente de ver as aplicações e qualquer conceito novo deve seguir as idéias originais. Desta maneira, este trabalho se reserva a constatar necessidade e sugerir a criação de tal sistema.

5.8.3 Segurança em nível de linguagem

Ao apresentar o modelo foram mostrados detalhes referentes ao funcionamento da Hololinguagem que deixaram bem claras questões relativas a segurança do sistema. O que acontece é que a princípio o Holo deve prover um ambiente de programação bastante flexível, de forma que o programador tenha total controle sobre o comportamento de sua aplicação. Porém, quando colocamos o Holo em um contexto pervasivo, com um ambiente de execução comum a várias aplicações, então se tem um cenário onde programas mal intencionados poderiam facilmente comprometer o funcionamento de todo sistema.

Quando se fala em segurança, na maioria das vezes se pensa na comunicação, torna-la segura ou restringir acesso ao sistema para evitar problemas. Criar canais de comunicação seguros é realmente indispensável e esta é uma tarefa que o próprio sistema pode gerenciar. Porém, a criação de restrições de acesso já é uma tarefa que aumenta a sobre-carga no sistema, exigindo a intervenção de pessoas para desempenhar a tarefa constantemente.

Neste contexto foi desenvolvido um trabalho dentro do projeto chamado HoloSafe [78], cujo objetivo é criar um mecanismo de segurança que englobe os principais elementos do Holoparadigma. Este mecanismo cria restrições à movimentação de entes e ao acesso às ações e interfaces dos mesmos, de forma a estender as estruturas já existentes no paradigma. Através do uso deste modelo, o projetista tem a possibilidade de criar as restrições e sua forma de atualização ainda na modelagem do sistema, provendo assim segurança no acesso a história e

comportamento de um ente.

5.9 Conclusão

Neste capítulo foi apresentado o modelo de distribuição chamado HNS, que é parte essencial da arquitetura PHolo, apresentada anteriormente neste trabalho. O HNS é responsável pelo controle do estado e localização de entes em um ambiente distribuído, bem como pelo fornecimento destes dados para as HoloVMs. Faz parte ainda deste sistema o suporte que possibilita a comunicação entre duas HoloVMs que possuem entes que precisam interagir, seja pela execução de ações ou pela utilização da história.

Além da localização e suporte à execução distribuída obedecendo os requisitos do Holoparadigma, o modelo do HNS ainda possui duas questões, que são: tolerância a falhas e escalabilidade. Para resolver estas questões o modelo utiliza uma abordagem para indexação de dados chamada DHT. O DHT permite que o sistema armazene dados naturalmente distribuídos entre vários servidores, e no caso da estratégia adotada ainda existe a replicação de dados. Esta abordagem proporciona tolerância a falhas no sistema, além de tratar a escalabilidade, dado que uma rede DHT é altamente escalável. A forma que o DHT utiliza para distribuição de dados proporciona ainda um balanceamento de carga entre os servidores, e estratégias de cache permitem a otimização do tempo de resposta para as consultas mais comuns.

Outra questão abordada no modelo é uma estratégia para diminuir a necessidade de interação do usuário na configuração do sistema. Para isso foi escolhido um protocolo de SD que apresentou a abordagem mais simples e eficiente para o caso específico do modelo HNS. Este protocolo permite que diferentes módulos do sistema (HoloVM e HNS por exemplo) sejam capazes de se encontrar em uma rede, sem que exista interação humana para isso, para então negociarem o funcionamento do sistema.

Foram apresentadas técnicas para contornar falhas no sistema que são baseadas especificamente no modelo criado, ou seja, tiram proveito da forma como os dados se organizam nas HoloVMs e HNSs para contornar problemas no sistema. De maneira semelhante foi abordada uma forma de estimular a criação de registros nas *caches* do sistema antes mesmo que as consultas por tais dados sejam feitas. Isto é feito com base nas características do Holoparadigma e da construção de aplicações em Holo.

Por fim foram apresentadas idéias para melhorar a usabilidade e segurança do sistema, partindo principalmente de pequenas modificações na Hololinguagem.

Capítulo 6

Aspectos de implementação e resultados obtidos

Neste capítulo serão abordadas questões relativas a implementação do modelo, bem como um estudo de caso mostrando possibilidades de utilização dos recursos fornecidos por este trabalho. Serão mostrados ainda resultados obtidos durante a implementação do modelo aqui apresentado.

6.1 Tecnologias utilizadas

Para agilizar a prototipação do modelo proposto neste trabalho, foram utilizadas implementações das principais tecnologias que integram o mesmo, que são: Kademlia e Bonjour. Os critérios utilizados para selecionar a solução que seria utilizada foram:

- **Código Aberto** - A solução deve estar disponível para uso livre e com preferência para aquelas que colocam a disposição o código fonte, pois isto facilita o entendimento da aplicação e torna possível que sejam implementadas modificações caso seja necessário;
- **Portabilidade** - Este é um requisito básico para todo código adicionado aos módulos do projeto, principalmente para a HoloVM e a libHolo. Atualmente estes dois componentes podem ser compilados para Windows XP e CE, Linux e Mac OS X, e novos componentes do sistema não devem comprometer esta portabilidade;
- **Documentação** - Uma boa documentação é essencial para agilizar o desenvolvimento.

Nas próximas seções serão apresentadas as duas soluções de *software* escolhidas para integrar o projeto.

6.1.1 GNUNet - Kademlia

O projeto GNUNet [71, 79] tem como objetivo principal desenvolver um *framework* para a criação de redes p2p seguras. O GNUNet é distribuído utilizando a GPL como licença, possui versões para Linux, Windows e Mac OS X e possui uma boa documentação tanto teórica quanto técnica. O GNUNet é chamado de *framework* pois utiliza uma arquitetura baseada em *plugins*, o que possibilita estender a aplicação e encoraja a reutilização de código. Isto permite que novos protocolos p2p sejam implementados utilizando a infra-estrutura do GNUNet. Os protocolos existentes atualmente neste projeto são: compartilhamento de arquivos anônimo, chat, teste de impacto de mensagens e visualização da topologia da rede.

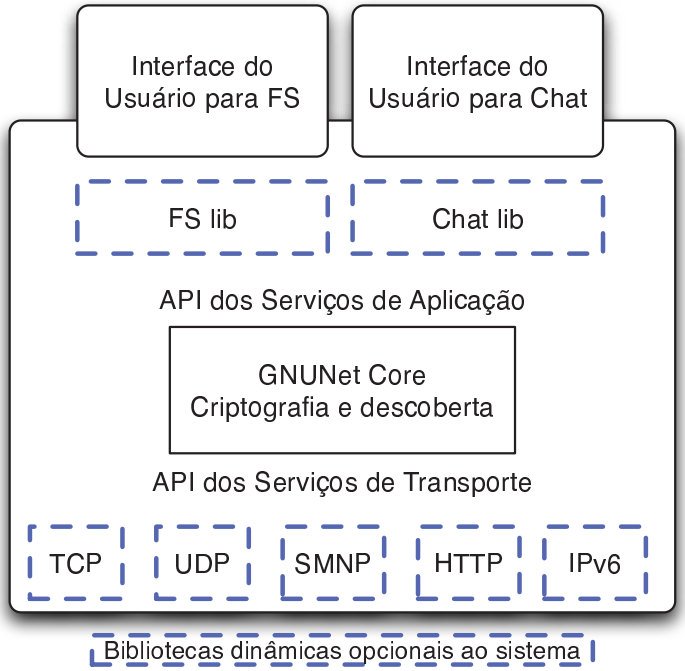


Figura 6.1 – Arquitetura do GNUNet

Na Figura 6.1 pode ser visto como o *framework* é implementado. No nível mais baixo da arquitetura estão os protocolos utilizados, que são TCP, UDP, HTTP, SNMP e IPv6. Logo acima existe uma camada de abstração, que visa facilitar a utilização de qualquer um dos protocolos de comunicação existentes. O *Core* do sistema é basicamente responsável pela comunicação criptografada e descoberta de recursos. Para executar estas tarefas o *Core* é dividido em vários módulos menores, responsáveis por funcionalidades específicas do sistema, como troca de chaves para sessão, divulgação de nodos, gerenciamento da topologia e transporte de mensagens. Acima do *Core* estão as APIs que abstraem os serviços do mesmo para que sejam utilizados pelas aplicações. Nesta arquitetura são colocadas duas aplicações que se encontram implementadas no sistema, *Chat* e FS (*File Sharing*). Estas aplicações são implementadas em duas partes, a primeira é uma biblioteca carregada juntamente do sistema, cuja função é executar

tarefas que exijam a utilização do *Core* do sistema. A segunda parte é a interface do usuário, que é capaz de se comunicar com a biblioteca para fazer uso das funcionalidades da aplicação.

6.1.2 Howl - Bonjour

No caso da busca por uma implementação do Bonjour existiam duas possibilidades. A primeira consiste na utilização do código fornecido pela Apple [80], que é composta por implementações separadas do Bonjour para diferentes arquiteturas e linguagens, mais algumas implementações que exemplificam a utilização dos códigos. A segunda opção chama-se *Howl* [81], que consiste em uma API criada por uma empresa chamada *Porchdog* e que implementa o protocolo do Bonjour e algumas aplicações de exemplo.

A implementação escolhida foi o Howl, principalmente por que ele foi implementado no formato de uma biblioteca e esta é compatível com diversos sistemas operacionais. A implementação da Apple por outro lado fornece um código fonte diferente para a implementação de cada arquitetura, o que é pouco prático do ponto de vista de desenvolvimento do sistema. A implementação do Howl é liberada sob a GPL e seu código pode ser compilado no Windows 2000/XP, Linux, FreeBSD e Mac OS X. Ele também possui uma documentação completa da API.

6.2 Aspectos de implementação do sistema

Nesta seção são abordados aspectos de implementação do sistema apresentado no capítulo 5. O suporte à distribuição proposto com o HNS pode ser dividido em quatro peças de *software* distintas, são elas: módulo HNS no GNUnet, sistema de consulta e atualização do HNS, suporte para distribuição entre HoloVMs e sistema de auto-configuração. Cada uma destas partes tem uma tarefa bem definida, que será abordada nas próximas seções, porém sua implementação precisa se adaptar a uma infra-estrutura de *software* já existente.

De forma geral o ambiente de execução apresentado aqui é composto por: HoloVM, libHolo e HNS. Um elemento de grande importância nesta organização é a libHolo, pois nela estão muitas das funcionalidades importantes do sistema, que podem ser utilizadas por diferentes componentes desta arquitetura, como o HoloCompiler, HNS, HS e HoloVM. A libHolo possibilita uma simplificação considerável na implementação e manutenção dos demais sistemas, permitindo por exemplo que HS e HoloVM utilizem exatamente a mesma implementação de *blackboards*.

6.2.1 Módulo HNS para o GNUnet

Na primeira versão do HNS, ele foi implementado como um servidor completo e independente. Contudo, na implementação de sua segunda versão, com suporte a DHT, não pode ser aproveitado quase nenhum código, já que para tirar proveito das APIs do GNUnet, o

HNS teve que ser implementado como um módulo para o mesmo. Além disso o módulo deve ser implementado em C e a implementação anterior foi feita em C++.

O GNUnet utiliza uma organização baseada na carga dinâmica de módulos. Isto permite que as funcionalidades de seu *daemon* sejam alteradas sem grande dificuldade, bastando para isso alterar os módulos carregados pelo sistema. Originalmente o GNUnet foi concebido para ser um sistema de compartilhamento de arquivos, contudo, para que ele deixe de fazer isso e passe a desempenhar as funcionalidades do HNS, basta informar no arquivo de configuração do mesmo quais módulos devem ser carregados.

A principal vantagem na utilização do GNUnet está no uso direto e indireto de suas APIs. Isto permite, por exemplo, que todo o sistema DHT seja resumido no uso de algumas funções como: *DHT_store*, *DHT_findValue* e *DHT_remove*. Ao executar estas chamadas o sistema se encarrega de descobrir quais nodos da rede possuem a tabela desejada e executa a ação requerida. Um exemplo de uso indireto destas APIs está no fato de que a comunicação do sistema de DHT é feita utilizando RPC, que também possui uma API dentro do GNUnet. O sistema oferece ainda algumas abstrações que estão no núcleo do sistema, que trata de toda a comunicação utilizada. Isto permite que sejam registrados *handlers* para mensagens de rede do sistema. Desta forma sempre que o núcleo receber uma mensagem cujo identificador tenha sido registrado para o HNS, o módulo será acionado para tratar da mesma.

O módulo de *bootstrap* do GNUnet por exemplo, que é responsável pela inicialização de uma instância do *daemon*, fazendo com que este baixe listas de *hosts* confiáveis para montar sua tabela de roteamento e para que o modulo de *advertising* possa anunciar este novo nodo para seus vizinhos na rede DHT. Este módulo não pode ser utilizado tal como em sua implementação original, pois suas funcionalidades são voltadas para questões específicas do compartilhamento de arquivos no GNUnet, e desta forma, segundo indicado pelos próprios desenvolvedores do sistema, a melhor abordagem é reimplementar o módulo para que este tenha o comportamento esperado.

O maior esforço de implementação para este módulo diz respeito a resposta do sistema com relação as requisições que são enviadas pelas HoloVMs. Conforme descrito na seção 5.3.2, cada requisição enviada por uma HoloVM pode desempenhar um processo com várias fases, do qual servidores diferentes são responsáveis por estágios distintos do processo. Este comportamento é anti-natural para o sistema, já que a API de DHT permite a busca de uma determinada chave na rede, retornando seu valor, e no caso do HNS, este valor é um conjunto de informações e que nem sempre precisam ser retornadas, mas sim utilizadas para desencadear a fase seguinte de um processo.

O grande número de APIs do sistema também trazem um revés, que está no aprendizado da utilização de suas interfaces e manipulação dos seus tipos de dados. Nem sempre estas interfaces apresentam o comportamento esperado e algumas vezes modificações mais profundas no sistema podem ser necessárias.

6.2.2 Comunicação entre HoloVM e HNS

Como foi citado anteriormente, o tratamento de mensagens no HNS é feito pelo núcleo do GNUnet, que recebe as mesmas e as repassa para o HNS. Desta forma, o sistema de comunicação entre HoloVM e HNS também é implementado utilizando as APIs. Isto por que o GNUnet utiliza mensagens com uma composição própria, criada por seus serviços e sua camada de transporte. Além disso o sistema tem registrado em seu código os identificadores de todos os tipos de mensagens utilizadas por serviços e aplicações dentro do GNUnet.

Por este motivo, o suporte à comunicação entre HoloVM e HNS é implementado na forma de uma biblioteca, compilada juntamente com o GNUnet e que é utilizada pela HoloVM. Esta biblioteca oferece uma interface simplificada, para que a HoloVM seja capaz de enviar mensagens ao HNS. Além disso ela é implementada em C++, abstraindo as APIs em C do GNUnet e facilitando seu uso no código da HoloVM.

As mensagens trocadas entre o servidor HNS e as HoloVMs podem ser separadas em dois grupos. No primeiro grupo estão as que servem para informar o servidor sobre uma alteração na árvore. Na HoloVM existem duas instruções, que pertencem ao primeiro grupo, e que vão sempre reportar ao HNS quando forem desempenhadas, são elas, *clone* e *move*. O *clone* é responsável pela criação de novos entes e o *move* faz a mobilidade de um ente na *HoloTree*. Desta forma o HNS será capaz de manter sempre a versão mais atualizada da árvore.

Pertencentes ao segundo grupo estão as instruções que disparam indiretamente consultas ao servidor para saber a localização física de um determinado ente. As instruções deste grupo são as que um ente pode utilizar para acessar a história de seu ente pai ou o comportamento e interface de outros entes (conforme descrito na seção 5.4).

6.2.3 Suporte à comunicação entre HoloVMs

Este módulo é integrado no núcleo de execução da HoloVM e seu objetivo é detectar qualquer tentativa de interação com um ente que não esteja em execução na mesma HoloVM. As instruções da HoloVM que tem sua execução monitorada são as mesmas citadas na seção anterior como pertencentes a um segundo grupo do instruções que disparam consultas ao HNS. O que ocorre é que, depois de consultar o HNS pela localização do ente, a informação obtida é o endereço de outra HoloVM. De posse desta informação a camada de comunicação entre HoloVMs pode entrar em ação executando as instruções desejadas na máquina remota. A Figura 6.2 utiliza um diagrama de seqüência para representar os passos executados pelo sistema quando é detectada esta necessidade. Estes passos vão desde a consulta a um HNS, que por sua vez consulta a rede DHT, até o retorno da localização da outra HoloVM e a comunicação entre as duas.

Em sua implementação na HoloVM, este suporte possui um módulo cliente e outros servidor. Na HoloVM cada ente eqüivale uma *thread*, e esta *thread* é que se torna o cliente quando um processo de comunicação é necessário. Já o módulo servidor possui uma *thread*

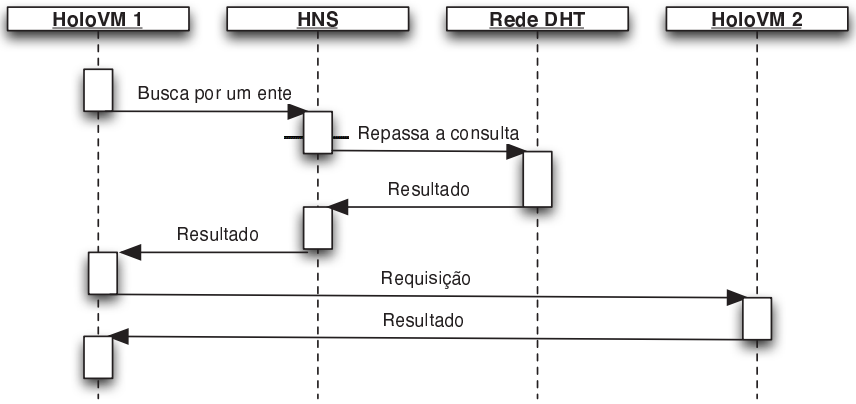


Figura 6.2 – Diagrama de seqüência para a comunicação entre HoloVMs

própria, pois este é responsável pelo tratamento das requisições que possam ser feitas a qualquer um dos entes em execução. Um caso especial desta implementação está no tratamento de acessos bloqueantes à história. O problema é que se pode bloquear uma *thread* do cliente, mas não do servidor, pois isto deixaria o serviço indisponível. A solução aqui é a requisição em uma fila e com intervalos de tempo testar se a tupla desejada já foi inserida. Quando isto ocorre a *thread* servidor restabelece a conexão com o cliente que requisitou a informação e finaliza transação.

Uma das melhorias do sistema que faz parte das metas deste trabalho é a melhoria do protocolo de comunicação através da utilização de XML. Desta forma é possível padronizar o formato das mensagens, facilitar o seu processamento e permitir que novos campos sejam inseridos sem que a compatibilidade entre versões seja comprometida, desde que os campos existentes não tenham seu formato modificado.

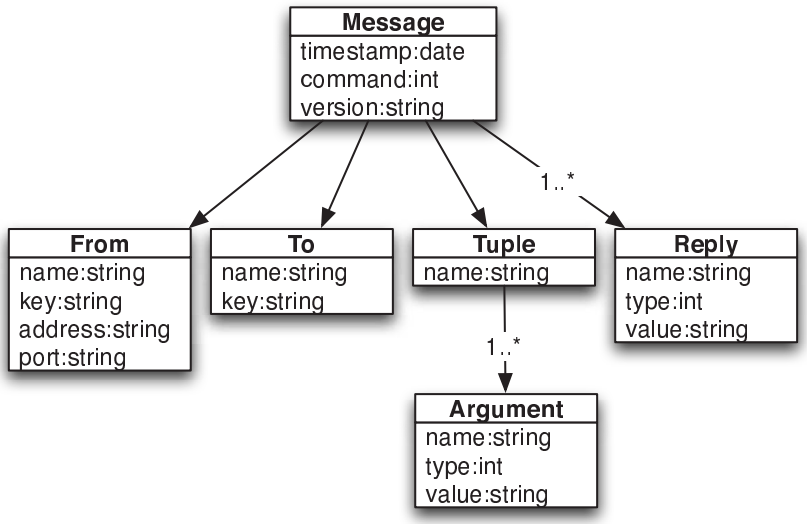


Figura 6.3 – Modelagem do XML utilizado para as mensagens

Na Figura 6.3 é apresentada a modelagem dos campos de uma mensagem entre HoloVMs. Com estes campos é possível codificar todos os dados necessários para executar interações entre dois entes remotos, conforme comentado na seção 5.4. Segue uma descrição das funcionalidades de cada campo da mensagem XML:

- **Message** - Neste campo vão informações relativas a mensagem, como um *timestamp*, que é utilizado apenas para registro, o comando a ser executado e a versão da HoloVM que enviou a mensagem;
- **From** - Este componente contém as informações relativas ao ente que está executando a ação, bem como o endereço da HoloVM que enviou a mensagem, caso seja necessário retomar o contato posteriormente;
- **To** - Para este campo são necessárias apenas as informações para identificar o ente que será alvo do comando enviado na mensagem;
- **Tuple** - Este campo tem usos diversos, mas sua função básica é codificar tuplas no formato utilizado na HoloVM, ou seja, para cada argumento é informado o tipo de dado e o valor é sempre colocado como uma *string*. Com este campo é possível enviar os dados necessários para consultas e afirmações as histórias, bem como a execução de ações e a codificação de seu retorno. Aqui é possível ainda codificar o *bytecode* de uma ação, para que seja feita a sua cópia para outra HoloVM;
- **Reply** - Este campo serve para respostas como *True* ou *False*, que podem ser retornadas por alguns tipos de acesso à história e ao comportamento.

Na Figura 6.4 é exemplificada a codificação de uma mensagem entre duas HoloVMs.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Message timestamp="01/12/2005 09:45:54AM" command="3" version="2.0">
  <From name="ente1" key="32384722304" address="10.1.1.1" port="60001"/>
  <To name="ente2" key="-1"/>
  <Tuple name="test">
    <Argument name="arg1" type="3" value="value1"/>
    <Argument name="arg2" type="2" value="34"/>
  </Tuple>
  <Reply name="arg2" type="3" value="resposta"/>
</Message>
```

Figura 6.4 – Exemplo dos possíveis campos de uma mensagem XML entre HoloVMs.

6.2.4 Sistema de auto-configuração

O sistema de auto-configuração é implementado em duas classes, que se organizam conforme o guia de implementação do *Bonjour* que é fornecido pela *Apple*. Nesta

documentação é dito que um sistema em *Bonjour* pode ter duas formas de implementação diferentes, a primeira com um núcleo mais funcional para o gerenciamento das informações e a segunda contendo apenas um cliente que anuncia seu serviço periodicamente. Segundo a empresa, não é necessário que todas as aplicações utilizando o Bonjour possuam este núcleo, de forma que muitas aplicações podem tirar proveito da implementação de um cliente mais simplificado. Aplicando este conceito ao trabalho, apenas os HNSs utilizam, enquanto que as HoloVMs precisam somente anunciar seu serviço.

As classes implementadas fazem parte da libHolo, para que HNS e HoloVM possam utilizar as mesmas. Outro detalhe é que ao serem instanciadas elas criam *threads* que ficam então responsáveis por anunciar o serviço continuamente, pois se pararem, mesmo que por poucos segundos, os outros nodos do sistema interpretam como se o serviço tenha deixado de existir.

6.2.5 Tarefas de implementação

A implementação do modelo proposto encontra-se em desenvolvimento. A Tabela 6.1 apresenta a porcentagem atual da implementação de cada um dos módulos que compõem o sistema.

Tabela 6.1 – Tarefas de implementação

Tarefas	Andamento
1.Módulo para o GNUnet	
1.1 Gerenciamento da tabela DHT	80%
1.2 Tratamento de mensagens HNS/HNS	70%
1.3 Tratamento de mensagens HoloVM/HNS	80%
1.4 Mecanismo de *bootstrap*	60%
2. Comunicação entre HoloVMs	
2.1 Acesso a história	100%
2.2 Executar ações	70%
2.3 Copiar ações	40%
2.4 Migração do protocolo para XML	50%
3. Sistema de auto-configuração - Bonjour	
3.1 Classes de abstração para o Bonjour na libHolo	70%
3.2 Integração das classes com a HoloVM	20%
3.3 Integração das classes com o HNS	20%

6.3 Modelagem de aplicações Holo e o suporte a distribuição do HNS

Nesta seção é feita uma análise das possibilidades no desenvolvimento de aplicações Holo utilizando os recursos criados pelo suporte do HNS, juntando a isto os requisitos de aplicações pervasivas. O modelo apresentado neste trabalho possui duas formas de criar

interações entre entes que pertencem a uma aplicação distribuída, que é o acesso a história e ao comportamento. Estas funcionalidades podem ser vistas comparativamente como dois recursos amplamente utilizados para a criação de aplicações distribuídas, sendo estes os sistemas de memória distribuída e compartilhada (DSM) e os sistemas para execução remota de código (RPC e RMI). Porém, estas funcionalidades são oferecidas na Hololinguagem de uma forma mais natural e sem a necessidade de configuração prévia para a utilização das mesmas.

Uma das vantagens da utilização do Holoparadigma é que, com a abstração oferecida por ele, um desenvolvedor é capaz de modelar mais facilmente aplicações, pois a organização do paradigma é condizente com a organização do mundo real. Na Figura 6.5 por exemplo, pode ser vista a modelagem de uma aplicação hipotética. Nesta figura a aplicação é dividida em camadas, para facilitar sua modelagem bem como para identificar melhor o papel de cada ente.

Uma abordagem bastante interessante para a modelagem de aplicações pervasivas em Holo é que se utilize uma representação do mundo real como centro do sistema. Toda **estrutura física** (Figura 6.5) possui essencialmente uma organização hierárquica, igual a de uma aplicação Holo, e modelar esta estrutura pode ser o primeiro passo para criar uma aplicação sensível ao contexto. Os entes criados para este propósito também serão chamados aqui de **entes físicos**. A criação destes entes físicos permite que seja feito o encapsulamento de recursos que são referentes aos ambientes do mundo real, onde um recurso pode ser um dado da história, uma ação do comportamento ou mesmo outro ente que oferece um serviço relativo aquele ambiente. No exemplo pode ser visto um fragmento da estrutura física da Unisinos mapeada para uma estrutura de entes. O programador então pode utilizar entes físicos como referência para organizar os entes que oferecem **serviços**. Assim pode ser observado que cada serviço fica encapsulado dentro do contexto no qual ele pode ser útil.

No momento que um usuário entra na Unisinos portando um dispositivo móvel, ele imediatamente passa a ter acesso a rede sem fio da universidade. Neste instante a HoloVM, que está executando uma **aplicação** (Figura 6.5) no dispositivo deste usuário, encontra um HNS disponível em sua rede e já negocia a sua entrada no contexto de execução da univesidade. O primeiro ente que a aplicação do usuário irá compor neste novo contexto é o *Unisinos*. Dentro deste ente podem estar disponíveis diversos serviços e dados que são úteis para a aplicação. A partir deste momento a aplicação do usuário pode refletir seus deslocamentos pela universidade com a execução de mobilidades lógicas na árvore de entes físicos.

Através do uso do acesso bloqueante à história de entes externos, um desenvolvedor pode criar mecanismos de sincronização da sua aplicação distribuída de uma maneira bastante simples, bastando para isso definir o ente cuja história será utilizada e a variável a ser usada como ponto de sincronização. Já os acessos bloqueantes e destrutivos podem ser usados para criar uma área de memória protegida que é lida por vários entes simultaneamente, só que cada ente ao ler a tupla da história, irá apagar a mesma, modificar os dados e escrever esta na história novamente. Outros entes que possam ter tentado fazer o acesso neste intervalo permanecem bloqueados em espera pelo dado, todos com a mesma possibilidade de acesso.

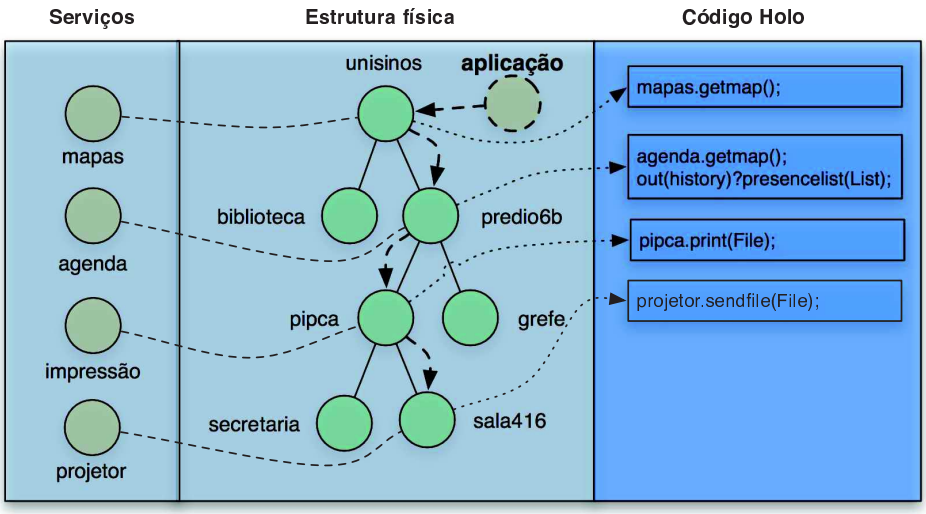


Figura 6.5 – Exemplo de uma aplicação Holo dividida em camadas

Na seção 5.4.2, foram descritas todas as formas de utilização de ações que são possíveis no sistema. Estas formas de acesso criam um ambiente bastante flexível para o programador, que pode executar ações existentes em diversos níveis da aplicação. Mais do que isso, uma aplicação tem a capacidade de adquirir novas ações durante a sua execução. Este processo quando feito por entes que estão em HoloVMs diferentes, implica em um tipo de mobilidade de código, o mais básico, que apenas copia o código estático para a máquina remota, que se apropria do mesmo e passa a utilizar como uma de suas funcionalidades. Apesar de aparentemente simples, esta funcionalidade permite que a aplicação seja dinâmica, e possa assim se adaptar a cada contexto que se insere.

As funcionalidades aqui descritas não são novas no Holoparadigma, porém a utilização destes recursos em ambientes distribuídos adquire um novo sentido e se tornam mais expressivas e importantes para o sistema.

6.4 Experimento previo: O History Server

No caminho para o desenvolvimento do suporte à execução de aplicações distribuídas em Holo foram constatadas duas alternativas possíveis. A primeira consiste em reduzir a complexidade de gerência a partir da centralização das informações de controle dos entes e de seus dados. A segunda alternativa tem uma abordagem distribuída, onde cada HoloVM é responsável pelas informações dos entes sendo executados em seu ambiente. Nesta segunda abordagem um mecanismo auxiliar deve oferecer suporte a localização de entes.

Quanto este protótipo foi desenvolvido o principal objetivo era iniciar os testes com aplicações distribuídas no projeto. Desta maneira a abordagem centralizada foi escolhida, e assim foi criado o *History Server* (HS). O HS é um servidor de histórias que respeita a estrutura

de encapsulamento de entes de aplicações Holo. O HS foi o primeiro passo para a posterior criação do conceito utilizado no HNS.

Na Figura 6.6 é mostrada a representação de um ambiente de execução utilizando o HS e as possíveis interações com uma ou mais HoloVMs. No **passo 1** a **HVM a** informa a criação de um ente para o servidor. Tendo feito isto, para que dois entes possam se comunicar basta que eles tenham acesso à história de um ente em comum. Os passos **2** e **3** exemplificam as **HVMs b** e **c**, interagindo com o servidor para acessar a história de um ente.

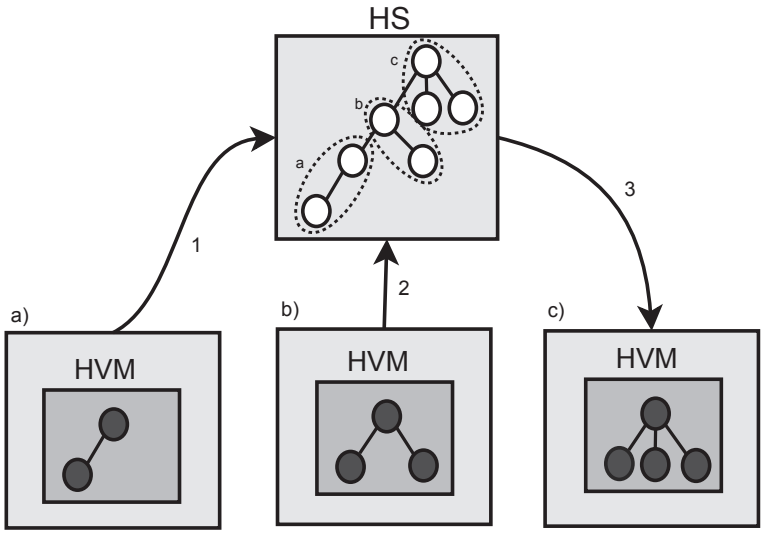


Figura 6.6 – Figura ilustrando funcionamento do HS

6.4.1 Troca de mensagens

Quando uma HoloVM é executada com o HS, imediatamente todo ente que é criado (comando *clone*) gera uma mensagem para o servidor e este então registra o novo ente. Esta é a primeira etapa de um processo que permite que o servidor mantenha a consistência com relação aos programas em execução. Para cada ente o servidor armazena os seguintes dados:

- Identificador de um ente - Nome único no sistema;
- Identificador de um ente que ele compõe - Localização lógica;
- Um *blackboard* que será utilizado durante a execução - Espaço de memória.

É importante perceber que nesta estratégia o servidor não registra a localização da HoloVM que enviou a mensagem, pois é ela (a HoloVM) apenas quem deve conhecer a localização do servidor. A partir deste momento, sempre que um ente faz uma mobilidade (comando *move*), o servidor é notificado, e o registro do mesmo é atualizado. Este controle consiste na segunda etapa do controle das aplicações em execução. Desta forma qualquer modificação na árvore de execução fica sendo conhecida pelo servidor, permitindo que este

execute os acessos às histórias de forma correta. A partir do momento em que o ente é registrado no servidor, este passa a ter controle sobre os dados que estiverem na história de um ente. Os comandos que acessam a história dos entes recebem tratamento semelhante aos comandos de *clone* e *move*. Sempre que um programa Holo executa um acesso a sua história ou a de um ente que ele compõe, este acesso é mapeado para uma mensagem que é enviada ao servidor. Na Figura 6.7 é mostrado o formato de mensagem utilizado.

```
From-ID: iPAQ
Father-ID: Holo
Command: 8             // Comando a ser executado
History-Name: teste    // Nome da tupla
Argument-Size: 2       // Número de argumentos da tupla
--
3 valor1               // Tipo e valor do argumento
3 valor2
```

Figura 6.7 – Mensagem entre HoloVM e HS.

Neste exemplo é criada uma tupla de nome "teste" na história do pai de um ente cujo identificador é iPAQ. Este formato de mensagem é utilizado tanto para afirmações quanto para consultas à história.

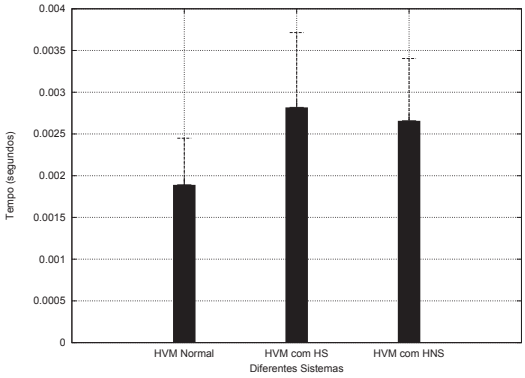
A existência de perguntas bloqueantes cria uma situação que precisa de um tratamento diferenciado no que diz respeito a comunicação. Estas perguntas são particularmente importantes pois permitem a criação de pontos de sincronismo para as aplicações. Para resolver isso, o servidor mantém a HoloVM que fez a consulta bloqueante travada, até que o dado esperado seja colocado na história.

6.4.2 Testes: HS x HNS

Para que fosse possível ter uma primeira avaliação da solução proposta com o HNS, foram feitos alguns testes comparativos. As aplicações foram executadas em três ambientes diferentes: uma HoloVM sem nenhum suporte à distribuição, uma com suporte a HS e outra com suporte ao HNS. Para a realização deste experimento foi utilizada a primeira versão do HNS, que não possui o suporte a distribuição de servidores que é proposto neste trabalho.

O primeiro teste apresentado teve por objetivo analisar o impacto que os diferentes suportes à distribuição têm na inicialização da HoloVM (Figura 6.2). Isso foi feito utilizando um programa em Holo que ao ser executado criava um ente, sem que nenhuma outra ação fosse executada. Esta análise se torna pertinente, pois considera o tempo de processamento gasto a cada nova conexão de usuário com o ambiente. Considerando que a versão distribuída da HoloVM conta com rotinas de inicialização, um sobrecusto de execução não existente na versão seqüencial é acrescentado.

Como pode ser visto, os dois suportes (HS e HNS) têm impactos semelhantes nos tempos de inicialização da HoloVM. Isso tem explicação principalmente no fato de que todo ente criado



HoloVM	Média	Desvio Padrão
Normal	0.001890	0.000559
Com HS	0.002818	0.000896
Com HNS	0.002657	0.000746

Tabela 6.2 – Resultados do teste de inicialização da HoloVM (40 execuções)

na HoloVM, gera uma mensagem que é enviada para o servidor informando sua existência. É importante ressaltar o desempenho obtido pela HoloVM com HNS, já que ela em particular possui um servidor *multi-thread* interno. Este servidor deve ser inicializado junto com a HoloVM. Mesmo assim o seu tempo médio de inicialização foi bastante próximo a HoloVM com HS.

Resultado semelhante foi obtido no teste com a escrita de tuplas na história (Figura 6.8). Este experimento reflete uma aplicação típica de MHolo, onde entes acessam informações (em leitura e/ou escrita) na história de entes remotos. Para este teste foi utilizado um programa que escrevendo 100 tuplas na história de um ente em execução em outra HoloVM. Foi feita uma variação na aridade das tuplas para verificar se a queda no desempenho seria significativa.

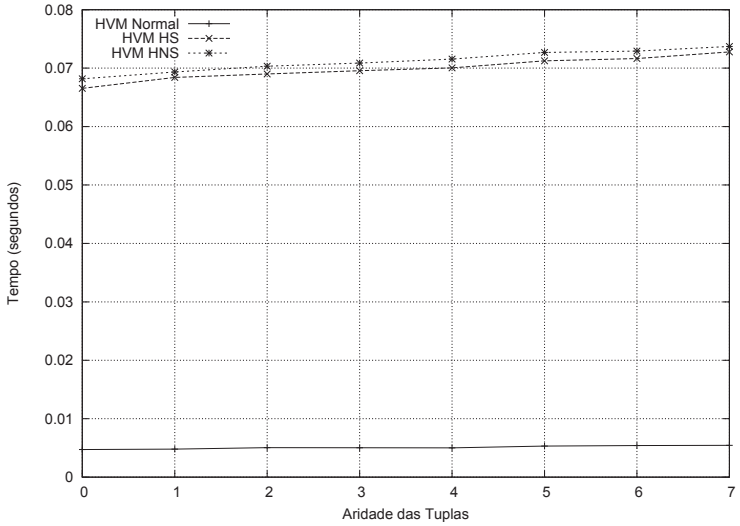


Figura 6.8 – Teste de escrita de tuplas na história

Este teste é particularmente importante pois o gerenciamento da história é uma especialidade do HS. No entanto pode ser notado que ambas soluções obtiveram tempos bastante semelhantes. Outro fato que torna este resultado relevante, vem da observação de que o HNS utiliza uma arquitetura mais complexa e que envolve um maior número de variáveis

no seu gerenciamento.

O último teste a ser mostrado aqui avaliou o desempenho das diferentes soluções ao executar criações massivas de entes (Figura 6.9). Para isso foi utilizado um programa em Holo que cria um determinado número de entes. Sobre este número foi feita a variação mostrada no gráfico.

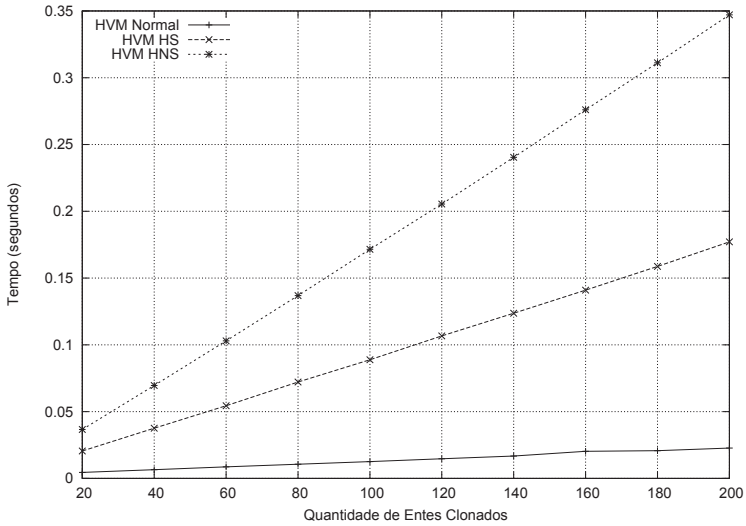


Figura 6.9 – Testes de clonagem de entes

Neste teste é avaliado o desempenho das diferentes estratégias de comunicação adotadas pelas duas soluções. Aqui a solução do HS levou vantagem em todas as configurações. Tal resultado levantou uma questão importante de desempenho em algumas partes da implementação do suporte ao HNS. No entanto, não foi possível precisar até o momento que aspectos causam esta diferença, já que esta é uma discussão onde muitas variáveis estão envolvidas.

6.5 Simulação para estudo da escalabilidade

Com o intuito de adaptar a estratégia DHT escolhida ao modelo criado para o HNS foram criadas novas operações. Tais operações objetivam refletir as modificações que ocorrem no ambiente, nos dados armazenados pelo DHT. Dentre elas, a que mais onera o sistema pela sua complexidade é o *move* (seção 5.3.2), já que é necessário modificar as informações contidas nos registros de vários entes para manter o sistema fiel ao estado da aplicação. Para estudar o impacto desta modificação foi criada uma simulação, a qual objetiva mostrar de forma comparativa o custo de uma operação básica no sistema, como uma busca, em relação a uma operação mais complexa, como o *move*, levando em conta ainda a escalabilidade.

Para implementar esta simulação foi utilizado um sistema que simula protocolos p2p chamado p2psim [82]. O p2psim já possui uma implementação do Kademlia, de forma que para criar a simulação foi feita uma modificação no código do Kademlia, criando assim a operação

move no protocolo. Durante esta modificação foi observado que a implementação Kademlia existente no p2psim não é completamente fiel a especificação do mesmo, mas ainda assim permite que boas estimativas sejam colhidas para o comportamento da aplicação.

O simulador foi configurado para criar redes começando com 50 nodos e com intervalos de 50 até 950 nodos, onde cada nodo corresponde a uma instância do HNS. Cada uma destas configurações foi repetida 20 vezes para garantir uma boa média nos resultados colhidos. Em cada repetição, durante todo o tempo de simulação cada nodo executou 720 buscas e 720 *moves*. Durante a simulação o p2psim se encarregava ainda de retirar e inserir nodos no sistema.

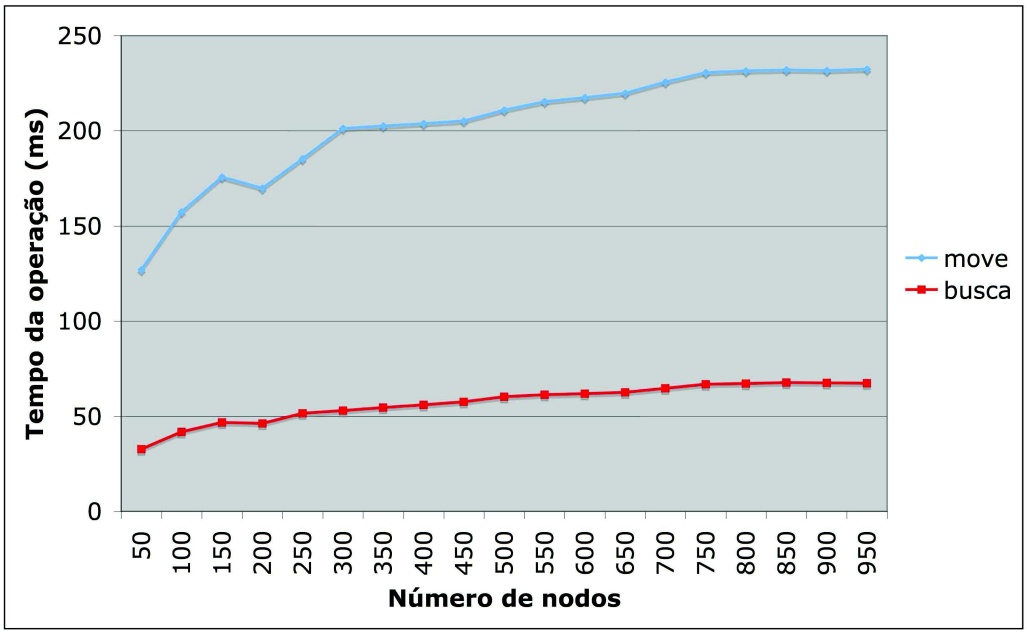


Figura 6.10 – Impacto do HNS na escalabilidade do Kademlia

A Figura 6.10 mostra os resultados obtidos com a simulação. No gráfico, de baixo para cima, a primeira curva equivale ao tempo médio de execução da busca de um registro no sistema e a segunda curva mostra o tempo médio para que o sistema execute um comando *move* nos servidores HNS. Estas curvas viriam de acordo com o aumento do número de servidores colocados na simulação (eixo X). A primeira observação que pode ser feita ao se visualizar o gráfico é referente ao tempo de resposta do DHT para uma operação executada na linguagem. Este custo adicional do *move* em relação a uma busca é esperado e natural, pois reflete a complexidade inserida no sistema. Contudo, os tempos em si podem ser diferentes quando este mesmo aspecto for observado em um sistema real, pois este é otimizado para realizar as operações, enquanto que um simulador pode apenas estimar custos.

Um dado interessante que é fornecido pela aplicação diz respeito a escalabilidade do sistema, onde é possível observar que enquanto o número de nodos foi aumentado dezoito vezes, o tempo de resposta do sistema não chegou a dobrar. Mais importante ainda é o comportamento observado a partir dos 750 nodos, quando o tempo de resposta do sistema estabiliza e se mantém

semelhante até o final. Isto mostra não só que o sistema é escalável, mas também que ele tende a estabilizar o tempo resposta a partir de um determinado número de nodos.

Se considerarmos que cada HNS é capaz de dar suporte a execução de várias HoloVMs, a simulação mostra que é possível criar um ambiente de execução consideravelmente grande, com mais de novecentos servidores, sem que se tenha um grande impacto na resposta do mesmo. Esta característica é de vital importância para um sistema que visa o suporte para aplicações pervasivas.

Capítulo 7

Conclusão e trabalhos futuros

Neste capítulo serão revisados os conceitos que motivaram o desenvolvimento deste trabalho, as contribuições que resultaram para o projeto MHolo e trabalhos futuros que foram vislumbrados no decorrer desta pesquisa.

7.1 Conclusão

A computação pervasiva vem sendo um foco de inúmeros projetos e já é vista, não apenas como um ambiente a ser suportado, mas como uma nova maneira de ver e criar aplicações em ambientes distribuídos. Dentre os projetos existentes, é possível notar que existe um ponto em que as soluções falham, que é na abstração utilizada para modelar o ambiente. Estes sistemas esbarram no fato de que as linguagens de programação existentes são limitadas na sua capacidade de abstração, quando o ambiente para o qual a aplicação será modelada é pervasivo. Neste sentido o Holoparadigma oferece uma abstração que foi concebida para modelar sistemas móveis, e que se adapta bem ao conceito existente de aplicações pervasivas.

Seguindo esta linha foram estudados os sistemas que oferecem suporte à pervasividade que se mostraram mais significativos. Com este estudo foram identificados os principais requisitos atendidos por estes sistemas, e assim foi possível adaptar esta idéia ao projeto já existente, o MHolo. O MHolo tem como objetivo principal criar o suporte para que o Holoparadigma funcione em um ambiente móvel. Com o domínio das tecnologias existentes no MHolo e nos principais ambientes pervasivos da atualidade foi criada a proposta de uma arquitetura pervasiva, que visa estender o suporte existente no projeto para atender os requisitos da pervasividade, esta arquitetura foi chamada PHolo.

O PHolo propõe uma série de funcionalidades identificadas como básicas para sistemas de suporte à pervasividade. Estas funcionalidades são organizadas em uma arquitetura em camadas. De posse desta arquitetura foi possível vislumbrar as possibilidades que uma visão pervasiva cria sobre o Holoparadigma. Entre os módulos definidos estão o suporte a mobilidade lógica e física, localização de aplicações em um espaço físico, informações de contexto e

suporte à execução distribuída. Este último foi definido como um serviço essencial, sendo assim foi o escolhido para que um modelo mais detalhado fosse feito.

O serviço que oferece o suporte a distribuição na arquitetura PHolo foi chamado de *Holo Name System* (HNS). O HNS é um serviço de nomes capaz de manter informações atualizadas sobre todos os entes em execução no sistema. Além disso, um serviço complementar ao HNS permite que HoloVMs se comuniquem para emular uma comunicação local entre entes distribuídos fisicamente. Para a especificação de um modelo para o HNS, alguns requisitos levados em consideração foram: escalabilidade, tolerância a falhas e transparêcia. Para atender tais requisitos foram estudadas tecnologias relacionadas, como sistemas de nomes, técnicas de indexação de recursos para sistemas P2P e protocolos de descoberta de recursos.

De posse destas informações foi definido um modelo para o HNS onde a estratégia de distribuição de servidores é baseada em uma tecnologia conhecida como DHT. O DHT oferece uma solução para indexar uma quantidade massiva de registros utilizando uma rede de servidores. Isto propicia balanceamento de carga, tanto no armazenamento de dados quanto na resposta a consultas, bem como um nível de tolerância a falhas, devido as estratégias de replicação de dados utilizadas. A transparência, que é citada com um dos requisitos, diz respeito o nível de interação que é exigido do usuário para que este tenha o sistema funcionando plenamente. Para otimizar este processo foram estudadas estratégias de descoberta de recursos, que permitem a diferentes módulos do sistema se encontrarem em uma rede sem que seja exigida nenhuma ação do usuário para isso.

A partir do modelo foi criada uma simulação que teve por objetivo avaliar o impacto das modificações necessária para realizar as consultas do HNS bem como avaliar a escalabilidade em um cenário onde existem até novecentos e cinqüenta servidores, que é muito mais do que seria possível testar com a aplicação real. Nestes testes foi concluído que o impacto da operação mais complexa criada, que implica em 3 buscas por informações no sistema, tem um custo adicional aceitável. Além disto o sistema demonstrou-se escalável em sua relação entre o aumento do número de servidores e o tempo para retornar um dado. Tão escalavel que chegou a demonstrar uma tendência a estabilização a partir de um determinado número de nodos.

7.2 Contribuições

No campo teórico, este trabalho tem como contribuição a especificação inicial de uma arquitetura de software que aplica os conceitos da computação pervasiva ao Holoparadiga (PHolo). O estudo sobre pervasividade contribui ainda com a identificação e revisão das principais pesquisas voltadas para o desenvolvimento de ambientes de suporte a aplicações pervasivas. A partir deste estudo foram definidas as funcionalidades consideradas mais importantes entre os sistemas estudados, para que estas fossem posteriormente especificadas no PHolo.

O segundo modelo desenvolvido, o qual é foco desta pesquisa, foi chamado de HNS.

Para o desenvolvimento deste trabalho foram identificados e estudados os principais sistemas nas áreas de sistemas de nomes, *Distributed Hash Tables* (DHT) e descoberta de recursos.

Quanto a materialização dos modelos, três sistemas foram desenvolvidos durante o curso da pesquisa. Todos eles oferecem suporte à distribuição utilizando como base o ambiente de execução original do projeto MHolo. O primeiro protótipo desenvolvido, chamado *History Server* (HS), suportava à execução distribuída através da centralização da história de todos entes registrados no servidor. Esta estratégia permitia uma comunicação indireta de entes. O segundo protótipo foi a primeira versão do HNS desenvolvida. Este HNS é bastante semelhante ao seu precursor, tendo como principal diferença o fato de que não existe distribuição entre servidores, já que apenas um pode ser usado. O terceiro *software* desenvolvido é a versão distribuída do HNS.

7.3 Trabalhos futuros

Durante a pesquisa para a criação dos modelos do PHolo e do HNS, foram identificadas novas oportunidades para desenvolvimento de trabalhos, alguns dos quais já se encontram em fase de pesquisa. A seguir serão listadas algumas destas oportunidades:

- Umas das principais funcionalidades de sistemas pervasivos é o suporte a **mobilidade de código**. Isto permite que aplicações se tornem altamente dinâmicas e adaptativas, já que assim elas podem adquirir novas funcionalidades em tempo de execução;

- Com a criação da mobilidade de código surge a necessidade de políticas e estratégias para o **escalonamento** das tarefas em execução no sistema. É preciso definir o que faz um ente migrar de uma máquina para outra, e por tanto quando o sistema deve fazer isso ou não;

- Outra funcionalidade utilizada constantemente em sistemas pervasivos é a **localização física** de dispositivos. Este recurso permite que aplicações se tornem conscientes de sua localização em um ambiente, podendo assim tomar decisões de acordo com o local que estão;

- No modelo do PHolo é proposta a utilização da história como forma armazenar **informações de contexto**, que ficariam por tanto naturalmente encapsuladas utilizando a estrutura da própria aplicação. Neste sentido, seria interessante um estudo mais aprofundado sobre o uso de informações de contexto no PHolo, avaliando assim as possibilidades e limitações do suporte oferecido;

- Uma funcionalidade que seria bastante útil no desenvolvimento de aplicações Holo é o **tratamento de exceções** na linguagem. Assim as aplicações poderiam fazer uso deste recurso, bem como o ambiente de execução. No caso do ambiente, ele seria capaz de

passar para o programador a responsabilidade pelo tratamento de algumas exceções, ou apenas informar sobre quaisquer eventos que ocorram no sistema;

- O ambiente de execução concebido para o PHolo permite a existência de um grande número de aplicações utilizando simultaneamente e muito provavelmente interagindo para executar tarefas. No Holoparadigma não existe uma forma de limitar acesso a dados ou funcionalidades dos entes, o que cria uma questão com a **segurança** do sistema. Já existe um trabalho nesta área no projeto, contudo ainda existem questões importantes a serem resolvidas envolvendo segurança e ambientes pervasivos em geral.

O PHolo é uma arquitetura de *software* que foi idealizado de forma a ser complementar ao ambiente de execução do projeto MHolo. Atualmente os tópicos de mobilidade de código e localização física já são objetos de estudo dentro do grupo. O trabalho desenvolvido aqui para a criação do HNS é de muitas maneiras uma dependência e/ou um facilitador de grande parte dos serviços que possam ser agregados na arquitetura.

Bibliografia

[1] Mark Weiser. The computer for the 21st century. *Scientific American*, Setembro 1991.

[2] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 8(4):10–17, 2001.

[3] Debashis Saha and Amitava Mukherjee. Pervasive computing: A paradigm for the 21st century. *IEEE Computer*, 36(3):25–31, 2003.

[4] The official bluetooth website. how it works. Disponível em: http://www.bluetooth.com/. Acesso em Abril de 2005.

[5] WiFi-802.11 Planet. Maximizing wireless lan performance. Disponível em: http://www.80211-planet.com/. Acesso em Abril de 2005.

[6] David Garlan, Dan Siewiorek, Asim Smailagic, and Peter Steenkiste. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(2):22–31, 2002.

[7] João Pedro Sousa and David Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *WICAS3: Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture*, pages 29–43, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V.

[8] Project aura - distraction-free ubiquitous computing. Disponível em: http://www-2.cs.cmu.edu/ aura/. Acesso em Junho de 2005.

[9] Manuel Roman and Roy H. Campbell. Gaia: enabling active spaces. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 229–234, New York, NY, USA, 2000. ACM Press.

[10] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces, 2002.

[11] Gaia - active spaces for ubiquitous computing. Disponível em: http://gaia.cs.uiuc.edu/. Acesso em Junho de 2005.

[12] Robert Grimm. one.world project home page, 2005. Disponível em: http://cs.nyu.edu/rgrimm/one.world/.

[13] Robert Grimm, Janet Davis, Ben Hendrickson, Eric Lemar, Adam MacBeth, Steven Swanson, Tom Anderson, Brian Bershad, Gaetano Borriello, Steven Gribble, and David Wetherall. Programming for pervasive computing environments. *In Proceedings of the 18th ACM Symposium on Operating Systems Principle*, Chateau Lake Louise, Banff, Canada., October 2001.

[14] Robert Grimm, Janet Davis, Ben Hendrickson, Eric Lemar, Adam MacBeth, Steven Swanson, Tom Anderson, Brian Bershad, Gaetano Borriello, Steven Gribble, and David Wetherall. System support for pervasive applications. *In Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Elmau, Germany.:147–151, 2001.

[15] one.world. Disponível em: http://www.cs.nyu.edu/rgrimm/one.world/. Acesso em Junho de 2005.

[16] Adenauer Correa Yamin, Iara Augustin, Jorge Luis Victória Barbosa, Luciano Cavalheiro da Silva, Rodrigo Araujo Real, Gerson Geraldo Homrich Cavalheiro, and Claudio Fernando Resin Geyer. A Framework for Exploiting Adaptation in High Heterogeneous Distributed Processing. In *Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2002)*, pages 125–132, Los Alamitos, 2002. IEEE Computer Society.

[17] Adenauer Correa Yamin, Jorge Victoria Barbosa, Iara Augustin, Luciano Cavalheiro da Silva, Rodrigo Real, Claudio Geyer, and Gerson Cavalheiro. Towards Merging Context-Aware, Mobile and Grid Computing. *International Journal of High Performance Computing Applications*, 17(2):191–203, 2003.

[18] Project isam - infra-estrutura de suporte às aplicações móveis. Disponível em: http://www.inf.ufrgs.br/ isam/. Acesso em Junho de 2005.

[19] W. S. Ark and T. Selker. A look at human interaction with pervasive computers. *IBM Syst. J.*, 38(4):504–507, 1999.

[20] Jorge L. V. Barbosa. *Holoparadigma: Um Modelo Multiparadigma Orientado ao Desenvolvimento de Software Distribuído*. Tese (doutorado em ciência da computação), Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002. 213p.

[21] H. G. Hegering, A. Küpper, C. Linhoff-Popien, and H. Reiser. Management challenges of context-aware services in ubiquitous environments. *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, 2003.

[22] Glenn Judd and Peter Steenkiste. Providing contextual information to pervasive computing applications. In *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, page 133, Washington, DC, USA, 2003. IEEE Computer Society.

[23] George F. Coulouris and Jean Dollimore. *Distributed systems: concepts and design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[24] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[25] Sape Mullender, editor. *Distributed systems*. ACM Press, New York, NY, USA, 1989.

[26] M. Satyanarayanan. Fundamental challenges in mobile computing. In *Symposium on Principles of Distributed Computing*, pages 1–7, 1996.

[27] Adenauer Corrêa Yamin. *Arquitetura para um Ambiente de Grade Computacional Direcionado às Aplicações Distribuídas, Móveis e Conscientes do Contexto da Computação Pervasiva*. Tese (doutorado em ciência da computação), Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2004. 194p.

[28] Aline Baggio. System support for transparency and network-aware adaptation in mobile environments. In *SAC '98: Proceedings of the 1998 ACM symposium on Applied Computing*, pages 405–408, New York, NY, USA, 1998. ACM Press.

[29] G. Roman, A. Murphy, and G. Picco. A software engineering perspective on mobility, 2000.

[30] R. H. Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, 1:6–17, 1994.

[31] Renato Cerqueira, Carlos Cassino, and Roberto Ierusalimschy. Dynamic component gluing across different componentware systems. In *DOA*, pages 362–371, 1999.

[32] I. Augustin, A . Yamin, and C. Geyer. Requisitos para o projeto de aplicações móveis distribuídas. 2001.

[33] Anand Ranganathan, Jalal Al-Muhtadi, Jacob Biehl, Brian Ziebart, Roy H. Campbell, and Brian Bailey. Towards a pervasive computing benchmark. In *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and*

Communications Workshops (PERCOMW'05), pages 194–198, Washington, DC, USA, 2005. IEEE Computer Society.

[34] Jorge L. V. Barbosa, André Rauber Du Bois, Laerte Kerber Franco, and Cláudio Fernando Resin Geyer. Java como linguagem intermediária para compiladores multiparadigma. *XXVIII Latin American Conference on Informatics (CLEI)*, 2003.

[35] Alex Sandro Garzão and Jorge L. V. Barbosa. Uma máquina virtual com suporte à concorrência, mobilidade e blackboards. In *XXIX Conferência Latinoamericana de Informática (CLEI)*, volume 24, La Paz, 2003. Universidad Mayor de San Andréas.

[36] Jorge Luis Victória Barbosa and Cláudio Fernando Resin Geyer. Uma linguagem multiparadigma orientada ao desenvolvimento de software distribuído. *In Proceedings of the V Simpósio Brasileiro de Linguagens de Programação (SBLP)*, maio 2001.

[37] Jorge Luis Victória Barbosa, Adenauer Corrêa Yamin, Patrícia Kayser Vargas, Débora Nice Ferrari, Alberto Egon Schaeffer, and Cláudio Fernando Resin Geyer. Using mobility and blackboards to support a multiparadigm model oriented to distributed processing. In *13th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2001)*, pages 187–194, Pirenópolis, GO Brasília, September 2001. UNB.

[38] H. Penny Nii. Blackboard systems: the blackboard model of problem solving and the evolution of blackboard architectures. pages 447–474, 1995.

[39] Daniel Torres Bonatto, Felipe Kellermann, Jorge Luis Victória Barbosa, José Dirceu G. Ramos, and Gerson Geraldo H. Cavalheiro. Estratégias para localização em um ambiente de computação móvel. In *XXIX Conferência Latinoamericana de Informática (SEMISH)*, São Leopoldo, 2005. Unisinos.

[40] Daniel Torres Bonatto, Jorge Luis Victória Barbosa, José Dirceu G. Ramos, and Gerson Geraldo H. Cavalheiro. Pholo: Uma arquitetura para a computação pervasiva utilizando o holoparadigma. In *VI Workshop em Sistemas de Alto Desempenho (WSCAD)*, Rio de Janeiro, 2005.

[41] José Dirceu G. Ramos. Integração de um sistema de localização no MHolo. Comunicação pessoal feita em Junho de 2005.

[42] Dario Fernandes Franz, Jorge Barbosa, and Gerson Geraldo H. Cavalheiro. Exploração do ambiente de computação móvel mholo no desenvolvimento de aplicações. In *Escola Regional de Alto Desempenho (ERAD) - Fórum de Pós-Graduação*, Ijuí - RS, 2006. Comissão Regional de Alto Desempenho (CRAD).

[43] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, 1998.

[44] III Richard, G.G. Service advertisement and discovery: Enabling universal device cooperation. *IEEE Internet Computing*, 4(5):18–26, Sep./Oct. 2000.

[45] Paul V. Mockapetris and Kevin J. Dunlap. Development of the domain name system. In *SIGCOMM*, pages 123–133, 1988.

[46] P.V. Mockapetris. Domain names: Concepts and facilities. RFC 882, November 1983. Obsoleted by RFCs 1034, 1035, updated by RFC 973.

[47] P.V. Mockapetris. Domain names: Implementation specification. RFC 883, November 1983. Obsoleted by RFCs 1034, 1035, updated by RFC 973.

[48] Gurmeet Singh Manku. Routing networks for distributed hash tables. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 133–142, New York, NY, USA, 2003. ACM Press.

[49] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[50] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[51] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[52] D. Eastlake 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174 (Informational), September 2001.

[53] A. C. Viana, M. D. Amorim, and S. Fdida J. F. Rezende. Self-organization in spontaneous networks: the approach of dht-based routing protocols. *Ad Hoc Networks Journal, special issue on Data Communications and Topology Control in Ad Hoc Networks*, 3(5):589–606, September 2005.

[54] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and

Ben Zhao. Oceanstore: an architecture for global-scale persistent storage. *SIGARCH Comput. Archit. News*, 28(5):190–201, 2000.

[55] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320, New York, NY, USA, 1997. ACM Press.

[56] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

[57] R. Marin-Perianu, P. H. Hartel, and J. Scholten. A classification of service discovery protocols. Technical report, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, June 2005. Imported from DIES.

[58] Erik Guttman. Service location protocol: Automatic discovery of ip network services. *IEEE Internet Computing*, 3(4):71–80, 1999.

[59] C. Bettstetter and C. Renner. A comparison of service discovery protocols and implementation of the service location protocol, 2000.

[60] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2 . RFC 2608 (Proposed Standard), June 1999. Updated by RFC 3224.

[61] Sun Microsystems Inc. Jini technology architectural overview. Technical report, Sun Microsystems, Inc., 1999. Disponível em: http://wwws.sun.com/software/jini/whitepapers/architecture.html.

[62] Damon Fenacci. The jini™lookup service: Introduction and discovery protocols. Technical report, Swiss Federal Institute of Technology, 2000. http://www.tik.ee.ethz.ch/ huang/teach/winter00-01/classes/JiniLookup/JiniDiscovery-report.pdf.

[63] Upnp device architecture 1.0. Disponível em: http://www.upnp.org/resources/documents.asp. Acesso em Junho de 2005.

[64] Yaron Y. Goland, Ting Cai, Paul Leach, Ye Gu, and Shivaun Albright. Simple Service Discovery Protocol/1.0 (SSDP). IETF draft-cai-ssdp-v1-03, october 1999.

[65] Bonjour - networking, simplified. Disponível em: http://www.apple.com/macosx/features/bonjour/. Acesso em Junho de 2005.

[66] Sylvia Ratnasamy, Ion Stoica, and Scott Shenker. Routing algorithms for dhts: Some open questions. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 45–52, London, UK, 2002. Springer-Verlag.

[67] Yusuke Doi. Dns meets dht: Treating massive id resolution using dns over dht. In *SAINT*, pages 9–15. IEEE Computer Society, 2005.

[68] Jeffrey Pang, James Hendricks, Aditya Akella, Roberto De Prisco, Bruce M. Maggs, and Srinivasan Seshan. Availability, usage, and deployment characteristics of the domain name system. In Alfio Lombardo and James F. Kurose, editors, *Internet Measurement Conference*, pages 1–14. ACM, 2004.

[69] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. Designing a DHT for low latency and high throughput. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI '04)*, San Francisco, California, March 2004.

[70] Fox Harrell, Yuanfang Hu, Guilian Wang, and Huaxia Xia. Survey of locating & routing in peer-to-peer systems, December 2001.

[71] GNUnet - GNU's decentralized anonymous and censorship-resistant P2P framework. Disponível em: http://gnunet.org/. Acesso em Janeiro de 2006.

[72] Yusuke Doi. Dns meets dht: Treating massive id resolution using dns over dht. In *SAINT '05: Proceedings of the The 2005 Symposium on Applications and the Internet (SAINT'05)*, pages 9–15, Washington, DC, USA, 2005. IEEE Computer Society.

[73] Jelica Protic, Milo Tomasevic, and Veljko Milutinovic. Distributed shared memory: Concepts and systems. *IEEE Parallel Distrib. Technol.*, 4(2):63–79, 1996.

[74] Sun Microsystems. RPC: Remote Procedure Call Protocol specification. RFC 1050 (Historic), April 1988. Obsoleted by RFC 1057.

[75] Sun Microsystems. RPC: Remote Procedure Call Protocol specification: Version 2. RFC 1057 (Informational), June 1988.

[76] R. Srinivasan. RPC: Remote Procedure Call Protocol Specification Version 2. RFC 1831 (Proposed Standard), August 1995.

[77] Govind Seshadri. Distributed Java computing using RMI. 2(10), November 1997.

[78] Guilherme Izidoro Lazzari. *HoloSafe: Um Modelo para Controle de Acesso a Recursos no Holoparadigma*. Trabalho de conclusão de curso. (graduação em ciência da computação), Universidade do Vale do Rio dos Sinos, São Leopoldo, 2004. 91p.

[79] Dennis Kügler. An Analysis of GNUnet and the Implications for Anonymous, Censorship-Resistant Networks. In Roger Dingledine, editor, *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*. Springer-Verlag, LNCS 2760, March 2003.

[80] Bonjour - source code. Disponível em: http://developer.apple.com/darwin/projects/bonjour/. Acesso em Janeiro de 2006.

[81] Howl. Disponível em: http://www.porchdogsoft.com/products/howl/. Acesso em Janeiro de 2006.

[82] p2psim: a simulator for peer-to-peer (p2p) protocols. Disponível em: http://pdos.csail.mit.edu/p2psim/. Acesso em Janeiro de 2006.