UNIVERSIDADE FEDERAL DE PELOTAS
INSTITUTO DE FÍSICA E MATEMÁTICA
DEPARTAMENTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

USO DE PADRÕES DE PROJETO NO DESENVOLVIMENTO DE APLICAÇÕES BASEADAS EM MOBILIDADE DE CÓDIGO NA COMPUTAÇÃO PERVASIVA

GIULIAN GONÇALVES VIVAN

PELOTAS, 2007

GIULIAN GONÇALVES VIVAN

USO DE PADRÕES DE PROJETO NO DESENVOLVIMENTO DE APLICAÇÕES BASEADAS EM MOBILIDADE DE CÓDIGO NA COMPUTAÇÃO PERVASIVA

Trabalho acadêmico apresentado ao Curso de Bacharelado de Ciência da Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Prof[a]. Ana Marilza Pernas Fleischmann, MSc.
Co-orientador: Prof. Adenauer Correa Yamin, Dr.

PELOTAS – 2007

Dados de catalogação na fonte:
Ubirajara Buddin Cruz – CRB-10/901
Biblioteca de Ciência & Tecnologia - UFPel

V855u Vivan, Giulian Gonçalves
Uso de padrões de projeto no desenvolvimento de aplicações baseadas em mobilidade de código na computação pervasiva / Giulian Gonçalves Vivan ; orientador Ana Marilza Pernas Fleischmann ; Adenauer Corrêa Yamin. – Pelotas, 2007. – 101f. -Monografia (Conclusão de curso). Curso de Bacharelado em Ciência da Computação. Departamento de Informática. Instituto de Física e Matemática. Universidade Federal de Pelotas. Pelotas, 2007.

1.Informática. 2.Computação pervasiva. 3.Mobilidade de código. 4.Padrões de projeto. 5.Desenvolvimento de aplicações. I.Fleischmann, Ana Marilza Pernas. II.Yamin, Adenauer Corrêa. III.Título.

CDD: 004.678

BANCA EXAMINADORA

Prof. MSc. Ana Marilza Pernas Fleischmann (Orientador)

Prof. Dr. Adenauer Corrêa Yamin (Co-orientador)

Prof. Dr. Gerson Geraldo Homrich Cavalheiro

Prof. MSc. Marcello da Rocha Macarthy

Prof. João Ladislau Barbará Lopes

AGRADECIMENTOS

Primeiramente gostaria de agradecer à minha família, que me apoiou, incentivou e deu todo o suporte durante minha trajetória nesta conquista. Sem eles, eu jamais teria chegado até aqui.

Agradeço também aos meus mais que orientadores, professora Ana Marilza Pernas Fleischmann e professor Adenauer Correa Yamin, que estiveram sempre dispostos a me ajudar, e por me proporcionarem crescimento tanto pessoal quanto profissional.

Agradeço também a todos os meus colegas, pela contribuição que de alguma forma me deixaram, e que ao longo destes quatro anos e meio de curso trouxeram muitos momentos de diversão, os quais com certeza deixarão saudades.

A todos vocês meu muito obrigado.

RESUMO

VIVAN, Giulian Gonçalves. **Uso de Padrões de Projeto no Desenvolvimento de Aplicações Baseadas em Mobilidade de Código na Computação Pervasiva**. 2007. 101f. Trabalho acadêmico – Curso de Bacharelado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

O avanço da tecnologia nos últimos anos permitiu o surgimento de um novo paradigma computacional: a computação pervasiva; que visa fornecer conectividade aos usuários em todo lugar a todo o momento. Neste escopo, desenvolver aplicações torna-se mais difícil, e consequentemente demanda maior tempo, devido aos novos requisitos impostos pela computação pervasiva. Tais aplicações caracterizam-se por serem distribuídas, móveis e sensíveis ao contexto em que executam. Assim, para viabilizar um estilo de programação apropriado, torna-se necessário o desenvolvimento de uma infra-estrutura de suporte para modelagem, implementação e execução do software a ser desenvolvido. O projeto GRADEp traz uma arquitetura de software que fornece esse suporte, através da implementação de um *middleware* que atende os requisitos das aplicações pervasivas. Sobre a estrutura provida pelo GRADEp, este trabalho propõe uma otimização para o desenvolvimento de aplicações pervasivas com foco em mobilidade de código. Para tanto, foi desenvolvido um *framework* que implementa padrões de projeto direcionados à mobilidade. Os padrões de projeto contidos no *framework* expressam diferentes comportamentos, disponibilizando ao desenvolvedor de aplicações pervasivas uma ferramenta alternativa, onde possa usar uma estrutura pré-definida em sua aplicação. As aplicações construídas dessa forma terão o comportamento descrito no padrão, tendo a comunicação, sincronismo e distribuição, gerenciados pelo mesmo. A expectativa é de que, com a redução do número de aspectos a serem gerenciados durante a codificação, o ciclo de desenvolvimento destas aplicações torne-se otimizado, com o reuso tanto de modelagem como de código. Para os testes iniciais, foram desenvolvidas aplicações utilizando-se dos recursos do *framework*. Os resultados obtidos foram satisfatórios, demonstrando a viabilidade do mesmo.

Palavras-chave: Computação Pervasiva. Mobilidade de Código. Padrões de Projeto. Desenvolvimento de aplicações.

ABSTRACT

VIVAN, Giulian Gonçalves. **Uso de Padrões de Projeto no Desenvolvimento de Aplicações Baseadas em Mobilidade de Código na Computação Pervasiva**. 2007. 101f. Trabalho acadêmico – Curso de Bacharelado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

The advance of technology in the last years allowed the rising of a new computational paradigm: the pervasive computing; which aims to provide connectivity for its users everywhere, every time. In this scope, application development becomes harder, and subsequently requires higher time, due to the new demands imposed by pervasive computing. These applications characterize themselves by being distributed, mobile and context aware. Thus, to turn viable an appropriate way of programming, it is needed a support infra-structure to design, implement and execute software applications. The GRADEp project brings an architecture that offers this support, through an implementation of a middleware which fulfill the pervasive applications demands. Above the GRADEp structure, this work proposes an optimization for the development of mobility code-based pervasive applications. For that, it was developed a framework which implements mobility design patterns. The framework design patterns describe different behaviors, making available to the pervasive application developer an alternative tool, where he can uses a pre-defined structure in his application. Applications built in this way will have the design pattern behavior, having related communication, synchronism and distribution managed by the framework. It is expected that, with reduced issues to manage in codification, the development of this applications becomes optimized, with both design and code reuse. In the initial tests, it was developed some applications using the framework resources. The results achieved were satisfying, arguing viability of the framework.

Keywords: Pervasive Computing. Code Mobility. Design Patterns. Application development.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AVA – Ambiente Virtual da Aplicação

AVU – Ambiente Virtual do Usuário

BDA – Base de Dados pervasiva das Aplicações

CIB – *Cell Information Base*

EXEHDA – *EXecution Environment for Highly Distributed Applications*

GRADEp – GRADE Pervasiva

ISAM – Infra-estrutura de Suporte às Aplicações Móveis

ISAMpe – ISAM *Pervasive Environment*

JME – *Java Micro Edition*

JSE – *Java Standard Edition*

JVM – *Java Virtual Machine*

MIT – *Massachussetts Institute of Technology*

OX – Objeto eXehda

PDA – *Personal Digital Assistant*

RMI – *Remote Method Invocation*

RPC – *Remote Procedure Call*

SGBD – Sistema de Gerência de Banco de Dados

UCPEL – Universidade Católica de Pelotas

UFPEL – Universidade Federal de Pelotas

UFRGS – Universidade Federal do Rio Grande do Sul

UFSM – Universidade Federal de Santa Maria

UML – *Unified Modeling Language*

URL – *Uniform Resource Locator*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A computação pervasiva segundo Satyanarayanan (2001), é um passo evolucionário cujos precedentes foram a computação distribuída e a computação móvel. Por sua vez, de acordo com Saha e Mukherjee (2003), computação pervasiva é a proposta de um novo paradigma computacional, que permite aos usuários acesso aos seus respectivos ambientes computacionais de todo o lugar, o tempo todo. Essa forma de computação, que está tornando-se realidade nos dias de hoje, já havia sido prevista por Mark Weiser, cujas idéias foram pioneiras nessa área.

No final da década de 80, Mark Weiser descreveu sua visão da computação do século 21, prevendo a ubiqüidade dos computadores pessoais. Segundo ele, a computação estaria perfeitamente integrada com o ambiente onde se encontra, concentrando-se nas interfaces humano-humano e reduzindo as interfaces humano-computador (WEISER; GOLD; BROWN, 1999). Ainda segundo ele, as tecnologias mais profundas são aquelas que desaparecem, elas são misturadas com os objetos do dia-a-dia a ponto de serem indistinguíveis deles (WEISER, 1991). Esta visão de computação inteligente e invisível ao usuário foi batizada sob o nome de computação ubíqua.

A computação pervasiva é um termo para a visão de Weiser, que representa uma realidade mais próxima à que vivemos. Se refere à parte prática, a implementação viável econômica e tecnicamente dessa visão. Computação pervasiva é um campo de pesquisa relativamente novo e ainda carece de consenso para alguns conceitos.

Nos últimos anos pode-se perceber um considerável esforço de pesquisa em direção à computação pervasiva. O surgimento de periódicos específicos nessa área,

como *Pervasive Computing, Mobile and Ubiquitous Computing* da IEEE (primeiro lançamento em 2002), atestam esse esforço (AUGUSTIN, 2004).

A infra-estrutura física disponível que pode ser explorada pela computação pervasiva é cada vez mais ampla. A disseminação natural de dispositivos de processamento, tais como: PDAs (*Personal Digital Assistants*), celulares, computadores pessoais de mesa e portáteis, até mesmo máquinas fotográficas e fornos microondas; caracterizam essa estrutura, além da conectividade extra provida por pontos de acesso, permitindo maior mobilidade física.

Em (GRIMM et al., 2001) é argumentado que essa estrutura de hardware necessita de aplicações que saibam explorar estes recursos, mas que essas aplicações são raras, pois são difíceis de desenvolver. Portanto, para facilitar a exploração desta estrutura, torna-se necessária uma arquitetura de software que atenda aos requisitos das aplicações, tanto em tempo de desenvolvimento quanto em tempo de execução.

Sob a arquitetura proposta pelo *middleware* GRADEp (YAMIN, 2004), pode ser construído um cenário pervasivo com suporte a distribuição em larga escala, mobilidade de dispositivos e de programas, e aplicações sensíveis ao contexto. Dentre estas características, este trabalho propõe uma otimização com relação ao desenvolvimento de um conjunto restrito de aplicações sobre esse ambiente, as quais são baseadas na mobilidade de fragmentos delas mesmas. Esta mobilidade é definida em (FUGGETTA; PICCO; VIGNA, 1998) como mobilidade de código.

1.1 Motivação

O GRADEp fornece subsídios para o desenvolvimento de aplicações móveis. Embora o *middleware* ofereça as primitivas necessárias, permitindo flexibilidade, as características inerentes à mobilidade de código introduzem esforços extras de programação, os quais se apresentam recorrentes entre aplicações. Em (FUGGETTA; PICCO; VIGNA, 1998) são discutidas algumas destas questões, como por exemplo:

a) mobilidade de código deve ser explorada em larga escala. Sistemas distribuídos tradicionais costumam ser projetados para um conjunto restrito de máquinas. Mobilidade de código deve prever escala em nível de Internet;

b) a programação é consciente de localização. Ou seja, localização passa a ser uma característica fortemente influente no desenvolvimento de aplicações. Ao contrário de sistemas distribuídos tradicionais onde se busca transparência de localização;
c) mobilidade de código está sobre o controle do programador. Devem existir mecanismos à disposição do programador para efetuar e controlar a mobilidade de código.

Características como estas demandam do desenvolvedor um tempo maior para a construção de aplicações. Sobre esse problema surge uma alternativa promissora, que consiste no emprego de padrões de projeto de software no desenvolvimento de aplicações baseadas em mobilidade de código. Os padrões de projeto sugerem uma solução para problemas recorrentes em um determinado contexto (GAMMA et al., 1995).

Ainda, segundo Gamma et al. (1995), o uso de padrões de projeto torna as modelagens de software mais flexíveis, elegantes e reusáveis. A intenção deste trabalho é contribuir na direção do problema citado, reduzindo questões recorrentes no que diz respeito ao desenvolvimento de aplicações móveis, através do emprego de padrões de projeto para mobilidade de código.

Em termos práticos, foi desenvolvido um *framework*, que implementa alguns padrões de projeto em mobilidade, permitindo ao programador optar por utilizar a estrutura pré-modelada em sua aplicação baseada em mobilidade de código, desde que a aplicação preencha os requisitos de abrangência do *framework*.

1.2 Objetivos

O objetivo central deste trabalho foi desenvolver um *framework* que implementasse padrões de projeto em mobilidade de código para o desenvolvimento de aplicações sobre o *middleware* GRADEp.

Para tanto, foi promovida uma revisão sobre os conceitos das áreas as quais este trabalho está inserido ou tangencia. Dentre elas, computação pervasiva, mobilidade de código e padrões de projeto.

Para a modelagem e implementação, também foram estudados conceitos e características de um *framework* bem como o ambiente GRADEp e suas particularidades, reconstruindo uma documentação mais específica sobre os aspectos utilizados e que influíram na modelagem e implementação.

Além disso, o *framework* proposto teve suas funcionalidades testadas e sua viabilidade analisada, através de aplicações desenvolvidas utilizando os recursos oferecidos pelo mesmo.

1.3 Contribuição esperada

Espera-se que ao final deste trabalho, a utilização do *framework* proposto possa contribuir para a otimização do tempo de desenvolvimento de algumas aplicações baseadas em mobilidade de código. A possibilidade do desenvolvedor de tais aplicações sobre o GRADEp, em optar por uma estrutura previamente modelada, desde que sua aplicação se encaixe nos requisitos do *framework*, pode economizar tempo, já que não apenas código será reutilizado mas também uma parte da modelagem.

Salienta-se também que o programador pode abstrair questões inerentes à mobilidade de código, de acordo com os padrões de projeto implementados, uma vez que o *framework* fica responsável por gerenciar todo o sincronismo, comunicação e distribuição nos casos os quais se propõe a fazê-lo, não havendo necessidade do programador se preocupar com a forma de emprego da mobilidade. Sendo assim, o desenvolvedor pode se dedicar a questões específicas de sua aplicação.

Outra contribuição é de que as aplicações desenvolvidas sobre esse *framework* seguirão a mesma estrutura. Isso pode acabar por facilitar a manutenção das mesmas.

Também se pode destacar a potencial contribuição deste trabalho em trabalhos futuros, uma vez que foi feita uma revisão de alguns serviços e aspectos operacionais particulares do GRADEp. Toda a documentação produzida, bem como códigos e configurações utilizadas estarão disponíveis no acervo do GRADEp, contribuindo para o aprimoramento do mesmo.

1.4 Organização do trabalho

Este documento foi dividido em sete capítulos, iniciando por capítulos introdutórios, de conceituação, até os de desenvolvimento e de conclusão.

O segundo capítulo faz uma breve introdução no paradigma escopo deste trabalho: a computação pervasiva; onde são explorados alguns conceitos bem como tendências e desafios nessa área.

O terceiro capítulo traz alguns conceitos em mobilidade de código, como mecanismos de mobilidade e de gerência de recursos, e também dos paradigmas em mobilidade.

No quarto capítulo são discutidos os padrões de projeto utilizados no *framework*.

No quinto são abordadas algumas características específicas do *middleware* GRADEp. Em seguida é apresentada a modelagem do *framework*, detalhando as estruturas dos padrões de projeto utilizados bem como aspectos relevantes de implementação do próprio *framework* e aspectos do GRADEp que influenciaram na modelagem.

No sexto capítulo são expostos o ambiente utilizado nos testes, as aplicações de testes desenvolvidas, os testes feitos sobre o *framework* e seus respectivos resultados.

Por fim, o capítulo sete apresenta as conclusões e trabalhos futuros.

2 FUNDAMENTOS EM COMPUTAÇÃO PERVASIVA

Neste capítulo será feita uma breve apresentação sobre o paradigma escopo deste trabalho: a Computação Pervasiva, bem como falar de suas tendências e desafios para o futuro. Serão também discutidas algumas características das aplicações pervasivas e apresentadas as iniciativas mais expressivas em termos de implementação.

2.1 Conceitos iniciais

No final da década de 80, Mark Weiser descreveu sua visão da computação do século 21, prevendo a ubiqüidade dos computadores pessoais. Segundo ele, as tecnologias mais profundas são aquelas que desaparecem, elas são misturadas com os objetos do dia-a-dia a ponto de serem indistinguíveis deles (WEISER, 1991).

O computador ainda é enxergado como uma ferramenta que executa programas em um mundo virtual, no qual os usuários entram para executar uma tarefa, e saem dele quando terminam (SAHA; MUKHERJEE, 2003). A computação ubíqua prevê uma visão bem diferente: ao invés do usuário ter de entrar no mundo virtual para usar a ferramenta, a ferramenta é que se integra ao mundo do usuário, fazendo com que o mesmo não precise de conhecimento específico para operá-la.

À medida que a tecnologia evolui, pode-se perceber que o computador cada vez mais faz parte do dia-a-dia, estando presente em: celulares, máquinas fotográficas, PDAs, fornos microondas, carros, sem contar os computadores pessoais e portáteis. Com a redução de custos com hardware, o crescimento da Internet e o

aperfeiçoamento de redes sem fio, aos poucos se torna realidade a proposta da **computação pervasiva** (YAMIN, 2004).

A computação pervasiva é um termo para a visão que Weiser, que representa uma realidade mais próxima a que vivemos. Se refere à parte prática, a implementação viável econômica e tecnicamente dessa visão. Computação pervasiva é um campo de pesquisa relativamente novo e ainda carece de consenso para alguns conceitos.

Segundo Saha e Mukherjee (2003), computação pervasiva é a proposta de um novo paradigma computacional, que permite aos usuários acesso aos seus respectivos ambientes computacionais de todo o lugar, o tempo todo.

Percebe-se que uma realidade como esta exige uma infra-estrutura adequada tanto de hardware quanto de software para execução e desenvolvimento de programas.

Neste cenário, a infra-estrutura deve suportar mobilidade do usuário e das aplicações e serviços. Entende-se por **mobilidade física** a mudança de localização física do usuário ao longo do sistema, seja portando um dispositivo móvel, seja acessando o sistema por dispositivos fixos, mas em localidades diferentes (ISAM, 2007). Entende-se por **mobilidade lógica** a movimentação de fragmentos de software entre equipamentos, também chamada de **mobilidade de código** (FUGGETTA; PICCO; VIGNA, 1998).

Ainda, a infra-estrutura intrinsecamente deve ser largamente distribuída, seguindo as premissas da computação em grade. O estado atual da tecnologia já oferece uma infra-estrutura fixa como esta: a Internet. A adição de pontos-de-acesso nessa estrutura tem permitido mobilidade para os usuários.

Segundo estas características, as aplicações devem ser desenvolvidas seguindo a **semântica siga-me**. A semântica siga-me prevê que as aplicações sigam seus usuários fisicamente, ou seja, a aplicação acompanha seu usuário esteja ele portando seu dispositivo ou não. Para que as aplicações tenham esse comportamento, elas **devem ser distribuídas, móveis e adaptativas ao contexto** (YAMIN, 2004).

A idéia de adaptação ao contexto refere-se à capacidade da aplicação de se adaptar às mudanças de condições de execução. De acordo com Yamin (2004), **adaptação** consiste de alterações funcionais ou não-funcionais, para atender novas condições do meio onde está se realizando o processamento. Entende-se por

adaptação funcional aquela disparada pela aplicação; e por **adaptação não-funcional** aquela disparada pelo ambiente de execução.

2.2 Tendências e desafios na computação pervasiva

Satyanarayanan (2001) fez um estudo na área de computação pervasiva, abordando em seu documento aspectos sobre o surgimento e futuros desafios na computação pervasiva.

Segundo ele, computação pervasiva representa um passo evolucionário de um trabalho iniciado em meados da década de 70. Os dois passos anteriores a esta evolução foram a computação distribuída e a computação móvel. Alguns dos problemas encontrados hoje na computação pervasiva já foram identificados e estudados nos passos anteriores. Alguns outros requerem soluções novas e uma outra classe foi introduzida, cujos estudos ainda estão em desenvolvimento.

O campo de sistemas distribuídos surgiu da intersecção de computadores pessoais com as redes locais. As pesquisas neste campo levaram ao um conjunto de conceitos e uma base científica fundamentais no estudo da computação pervasiva. Entre as áreas de estudo que emergiram a partir da computação distribuída e que possuem forte influência na computação pervasiva pode-se salientar: comunicação remota, incluindo protocolos de abstração (RPC – *Remote Procedure Call*, por exemplo); tolerância à falhas; alta disponibilidade; acesso à informação remota; e segurança.

O surgimento de *laptops* e redes sem fio levou pesquisadores a dedicarem-se aos problemas relativos a um sistema distribuído com clientes móveis. Assim nasceu a computação móvel. Por mais que muitos dos princípios básicos dos sistemas distribuídos sejam aplicáveis à computação móvel, quatro restrições inerentes à mobilidade forçaram o desenvolvimento de técnicas específicas: variação imprevisível da qualidade das conexões em rede; baixa confiabilidade dos dispositivos móveis; limitações de recursos impostas pelo peso e tamanho dos dispositivos; e a preocupação com consumo de energia das baterias. A busca por soluções para estas restrições constitui ainda um forte e ativo campo de pesquisa na computação móvel, com trabalhos ainda em desenvolvimento.

Como próximo passo da evolução, pesquisadores já começaram a pensar em uma maneira de abstrair do usuário as tecnologias do dia-a-dia. Neste ponto, surgem quatro desafios para pesquisa na computação pervasiva: o uso efetivo de espaços inteligentes; invisibilidade; escalabilidade localizada; e o mascaramento de condições adversas.

O uso efetivo de espaços inteligentes possibilitará uma integração de dois mundos, o real e o computacional, permitindo o controle do primeiro através do segundo, automatizando tarefas que antes exigiam atuação explícita do usuário.

A invisibilidade refere-se à tecnologia usada em espaços inteligentes estar totalmente fora da consciência do usuário. O usuário não deve se preocupar em saber como fazer o sistema funcionar, e sim operá-lo de forma subconsciente.

À medida que espaços inteligentes crescem em sofisticação, cresce também a necessidade de interações com o usuário. Isso implica em maior quantidade de largura de banda e energia. Escalabilidade torna-se um ponto crítico no projeto de espaços inteligentes.

A heterogeneidade dos dispositivos demandará das aplicações adaptabilidade às condições fornecidas. Essa adaptabilidade é provida por técnicas de mascaramento de condições adversas, permitindo ao usuário operações em ambientes com condições adversas sem que o mesmo precise se preocupar com essa questão.

2.3 Execução de aplicações pervasivas

O projeto de aplicações pervasivas é dirigido por dois conjuntos de restrições. Primeiro, os recursos locais nos dispositivos móveis como peso, tamanho, consumo de energia, poder de processamento e capacidade de armazenamento, são limitados, em comparação com os sistemas fixos de custo similar. Segundo, a infra-estrutura de suporte é altamente variável (conectividade, recursos remotos), especialmente em comparação com o suporte mais estático utilizado nos sistemas distribuídos tradicionais.

Neste ambiente, desenvolver aplicações móveis é difícil porque o projetista deve conhecer o domínio da aplicação e o da rede. Por essas razões, pesquisadores concordam que as aplicações móveis devem ser adaptativas: dependendo das

características do ambiente de execução, aplicações alteram seu comportamento visível e/ou o modo como utilizam os escassos recursos. Linguagens, bibliotecas ou *middlewares* podem auxiliar na tarefa de adaptação (YAMIN, 2004).

Hoje, as aplicações direcionadas para dispositivos de elevada portabilidade como PDAs e assemelhados, ainda são predominantemente caracterizadas por um perfil nômade. O usuário opera em uma perspectiva onde os programas e os dados estão disponíveis localmente, no próprio equipamento. Neste caso, quando necessário e por iniciativa do usuário, é executada uma operação de sincronização com um equipamento externo, a qual é utilizada para envio e recebimento de programas e dados, e para realização de cópias de segurança.

Apesar de bastante disseminado, o modelo nômade não atende à demanda de serviços introduzida pela computação pervasiva. A dinamicidade da mobilidade física do usuário, portando ou não seu dispositivo móvel, traz novos requisitos para a aplicação. Entende-se que os recursos empregados no atendimento ao usuário estão geograficamente distribuídos em uma rede de abrangência global.

Nesta perspectiva, os componentes que integram o meio de execução, dentre estes, dados, códigos, equipamentos e serviços, deverão ser gerenciados por um ambiente de execução, que viabilize um acesso independente do equipamento utilizado, do local e do momento de acesso aos mesmos (acesso pervasivo).

Além disso, as aplicações são desenvolvidas e executadas segundo a semântica siga-me, a qual define que o usuário executa suas aplicações no local em que se encontra, mesmo em deslocamento. Para isto, neste ambiente de execução, os usuários dispõem de um ambiente virtual, que poderá ser acessado independentemente da localização onde este se encontrar, ou do dispositivo que dispuser no momento, e o código das aplicações é enviado sob demanda. Logo, este código deve estar adaptado ao contexto corrente de uso da aplicação.

Desta forma, **as aplicações pervasivas são distribuídas, tem mobilidade lógica e física e precisam ser adaptativas ao contexto em que serão processadas**. A plataforma pervasiva utilizada neste trabalho é direcionada para estas aplicações.

2.4 Projetos em computação pervasiva

Na tentativa de colocar em prática um ambiente pervasivo, surgiram várias iniciativas, todas elas ainda em desenvolvimento. A seguir, algumas das mais expressivas.

2.4.1 Aura

Segundo o projeto Aura (AURA, 2007) o mais precioso recurso em um sistema de computador não é processador, memória, disco ou rede: é o usuário. Os sistemas de hoje distraem o usuário de várias maneiras, assim reduzindo sua efetividade.

O projeto Aura fundamentalmente repensa o sistema na tentativa de solucionar esse problema. O objetivo do projeto é prover a cada usuário uma auréola invisível de computação e serviços de informação que persistem independentemente de localização.

O projeto Aura é uma proposta da Universidade de *Carnegie Mellon*, e visa projetar, implementar e avaliar sistemas de larga escala para demonstrarem o conceito de “aura de informação pessoal”. Seu foco é no usuário, suas tarefas e preferências (GARLAN; STEENKISTE; SCHMERL, 2002).

2.4.2 Gaia

No projeto Gaia (GAIA, 2007) é defendida uma visão de futuro onde o espaço habitado pelas pessoas é interativo e programável, chamado de espaços ativos (*Active Spaces*). Os usuários interagem com seus escritórios, casa, carros, etc... para requisitar informações, beneficiar-se dos recursos disponíveis e configurar o comportamento de seu habitat. Dados e tarefas estão sempre acessíveis e são mapeados dinamicamente para os recursos convenientes e presentes na localização corrente. Gaia coordena as entidades de software e dispositivos em rede dentro dos espaços ativos, exporta serviços para pesquisar e utilizar recursos, acessar e usar o contexto corrente, e também fornece um *framework* para desenvolver aplicações.

2.4.3 Oxygen

O projeto Oxygen (OXYGEN, 2007) é uma iniciativa do MIT (*Massachussetts Institute of Technology*) que prevê um futuro onde a computação será livremente disponível em qualquer lugar, como o oxigênio do ar que respiramos. O projeto se baseia em uma infra-estrutura de dispositivos móveis e estacionários, conectados por uma rede que se auto-configura. Essa infra-estrutura fornece computação e comunicação em larga escala, intrínsecas ao sistema, e tecnologias de software que satisfazem as necessidades dos usuários.

3 MOBILIDADE DE CÓDIGO: CONCEITOS E TECNOLOGIAS

Como parte da computação pervasiva, a mobilidade lógica (de componentes de software) apresenta características particulares relevantes para o desenvolvimento deste trabalho. Este capítulo apresentará os principais fundamentos pertinentes ao entendimento de mobilidade de código, bem como os mecanismos, os paradigmas de programação em mobilidade.

3.1 Definição de mobilidade de código

As tecnologias, arquiteturas e metodologias tradicionalmente usadas para desenvolvimento de aplicações distribuídas possuem várias limitações e normalmente falham quando aplicadas em sistemas de larga escala. Para suprir essa necessidade, pesquisadores começaram a investir em novas abordagens. As mais promissoras são aquelas baseadas na idéia de mover código pelos nodos de uma rede, explorando a noção de **código móvel** (*Mobile Code*) (FUGGETTA; PICCO; VIGNA, 1998).

Na área de **mobilidade de código**, mais recentemente pesquisada, ainda há falta de consenso entre os termos e os conceitos envolvidos, sendo assim, a expressão código móvel também é utilizada para referenciar essa área.

Segundo Carzaniga, Picco e Vigna (1997), mobilidade de código pode ser definida informalmente como a capacidade de, dinamicamente, mudar as ligações entre fragmentos de código e o local onde eles são executados.

Em outras palavras, mobilidade de código é a habilidade de um componente de software (código) em mudar o seu local de execução em um sistema distribuído, em tempo de execução.

3.2 Conceitos preliminares

Segundo Tanenbaum (1995), um sistema distribuído é uma coleção de computadores independentes que aparecem para o usuário do sistema como um único computador. Para tanto é necessária uma estrutura de software; uma camada de software responsável em fazer essa abstração. Essa camada executa acima do sistema operacional, e deve ser comum a todos os nodos do sistema, assim provendo a transparência de localização à aplicação, ou seja, as máquinas estão distribuídas, mas a aplicação vê apenas um sistema local.

Esta definição de sistema distribuído tradicional não é apropriada em **sistemas baseados em mobilidade de código**. Mobilidade de código parte de uma perspectiva diferente.

Nos sistemas baseados em mobilidade de código, a estrutura subjacente de rede não é mais transparente ao programador, pois a camada de software que provia a transparência não é comum a todos os nodos, sendo substituída pela camada **ambiente computacional** (*Computational Environment*). Um ambiente computacional representa um nodo do sistema, conservando a identidade do nodo onde está localizado. O propósito dos ambientes computacionais é fornecer às aplicações a capacidade de alocar dinamicamente seus componentes em nodos diferentes, com consciência de localização, consequentemente fazendo com que o sistema operacional delegue parte da gerência de comunicação e da gerência de recursos para o ambiente computacional (FUGGETTA; PICCO; VIGNA, 1998).

A Fig. 1 ilustra a idéia de ambiente computacional, fazendo uma comparação superficial com um sistema distribuído tradicional, onde o sistema operacional está representado pelas camadas *network operating system* e *core operating system.*

Os componentes hospedados em um ambiente computacional são divididos em **unidades de execução** (*Execution Units*) e **recursos** (*Resources*). Unidades de execução representam fluxos seqüenciais de computação, como uma *thread* de um processo. Recursos representam entidades que podem ser compartilhadas entre várias unidades de execução, como um arquivo ou um objeto.

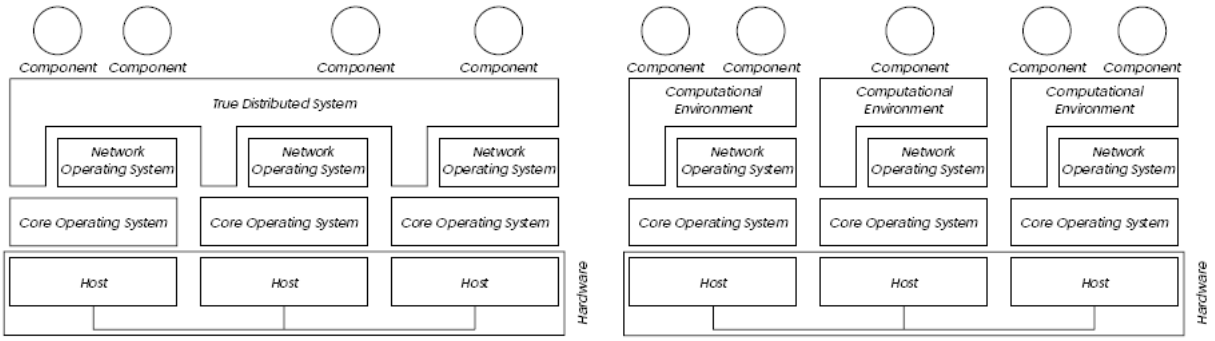

Figura 1 – Sistema distribuído tradicional (esquerda) vs. sistema baseado em mobilidade de código (direita)

Fonte: FUGGETTA; PICCO; VIGNA, 1998.

Uma unidade de execução pode ser mais especificamente modelada como mostra a Fig. 2. Um **segmento de código** (*code segment*) descreve o comportamento da execução da computação, e um **espaço de dados** (*data space*) somado a um **estado de execução** (*execution state*) descrevem o estado da computação. O espaço de dados é um conjunto de referências a recursos que a unidade de execução pode acessar (não necessariamente localizados no mesmo nodo). O estado de execução é composto de dados privados bem como de dados de controle, como ponteiro para pilha.

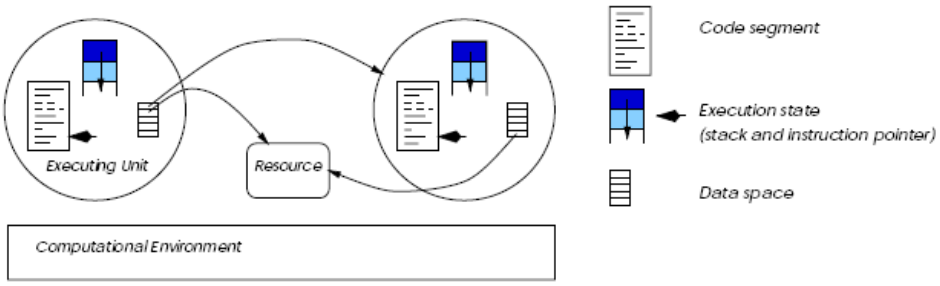

Figura 2 – Estrutura das unidades de execução

Fonte: FUGGETTA; PICCO; VIGNA, 1998.

3.3 Mecanismos para mobilidade

Em sistemas distribuídos convencionais, cada unidade de execução está associada a um único ambiente computacional, durante toda sua existência. A

associação entre uma unidade de execução e seu segmento de código é geralmente estática. Em sistemas baseados em mobilidade de código, o segmento de código, o estado de execução e o espaço de dados podem ser alocados para diferentes ambientes computacionais. A princípio cada parte de uma unidade de execução pode ser movida independentemente da outra. Entretanto, existem limitações em conseqüência das tecnologias utilizadas.

Em (FUGGETTA; PICCO; VIGNA, 1998) é discutida uma classificação para mecanismos de mobilidade, ilustrada na Fig. 3, que será apresentada a seguir.

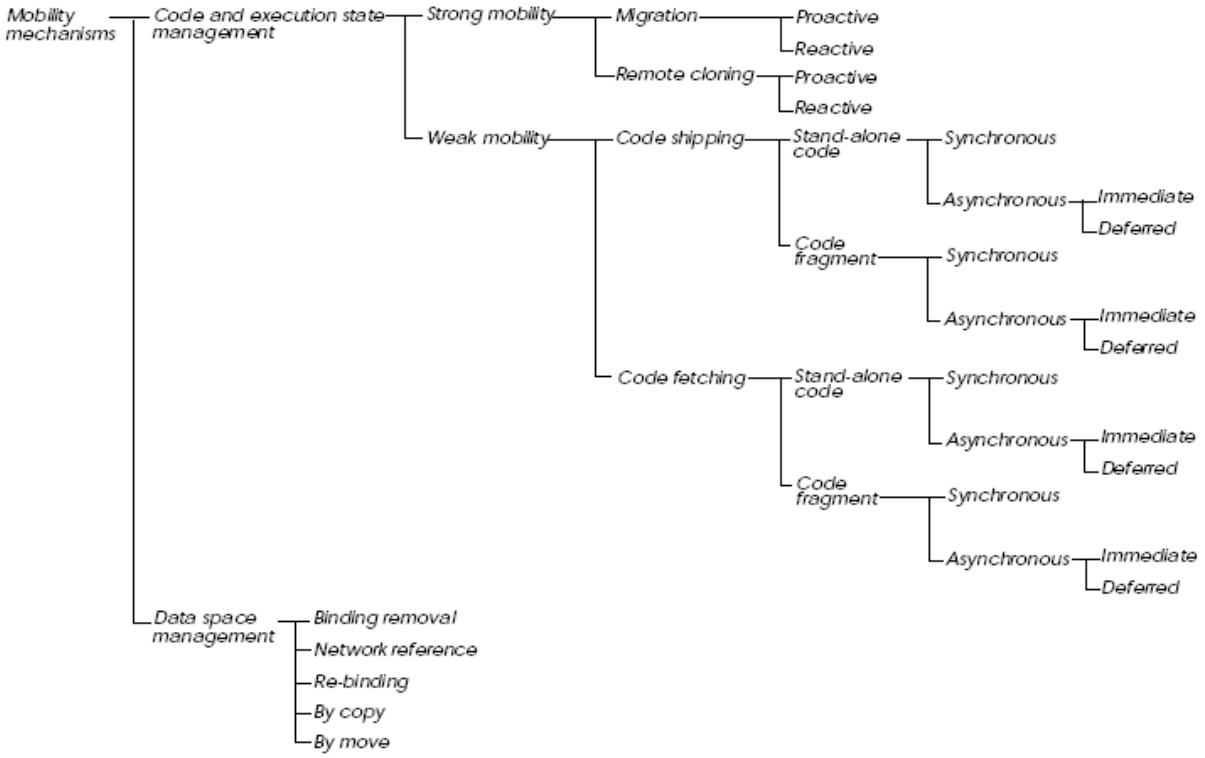

Figura 3 – Classificação dos mecanismos de mobilidade

Fonte: FUGGETTA; PICCO; VIGNA, 1998.

3.3.1 Mobilidade do estado de execução e do segmento de código

Sistemas baseados em mobilidade de código podem oferecer duas formas de mobilidade, caracterizadas pelas partes da unidade de execução que podem ser movidas (migradas):

a) **Mobilidade forte** (*Strong Mobility*) - a capacidade de um sistema de migrar tanto o segmento de código quanto o estado da computação de uma unidade de execução para um ambiente computacional diferente;
b) **Mobilidade fraca** (*Weak Mobility*), que é a capacidade de um sistema de migrar o código de uma computação para um ambiente computacional diferente, que pode ser acompanhado de alguns dados iniciais, mas não do estado de execução.

Mobilidade forte é suportada por dois mecanismos: **migração** (*Migration*) e **clonagem remota** (*Remote Cloning*). Migração suspende uma unidade de execução, transmite para o ambiente computacional destino, e então retoma a execução. Clonagem remota cria uma cópia da unidade de execução no ambiente computacional remoto. A unidade de execução original não é separada de seu ambiente atual.

Tanto a migração como a clonagem remota podem ser: **pró ativa** (*Proactive*) ou **reativa** (*Reactive*). Quando é pró ativa, a hora e o destino da migração ou clonagem são decididos autonomamente pela unidade de execução origem/original. Quando é reativa, a migração ou clonagem é disparada por uma outra unidade de execução que possui algum tipo de relação com a unidade a ser migrada/clonada.

Mecanismos que suportam mobilidade fraca fornecem a capacidade de mover segmentos de código pelos nodos bem como ligá-los dinamicamente a uma unidade de execução ou usá-la como segmento de código para uma nova unidade. Estes mecanismos podem ser classificados de acordo com a direção da transferência, a natureza do código a ser transferido, a sincronização envolvida e a hora em que o código deve ser executado no local de destino.

De acordo com a direção da transferência, uma unidade de execução pode tanto **buscar** (*Code Fetching*) o código para ser ligado ou executado, como **levar** (*Code Shipping*) o código para um outro ambiente computacional.

Quanto à natureza, o código tanto levado quanto buscado pode ser **autônomo** (*Stand-alone Code*) ou **fragmento** (*Code Fragment*). O código autônomo é auto-contido e será usado para instanciar uma nova unidade de execução no destino. O fragmento

de código deve ser associado ao contexto de um código que já iniciou sua execução no ambiente destino.

Quanto à sincronização, tanto um fragmento quanto um código autônomo pode ser **síncrono** (*Synchronous*) ou **assíncrono** (*Asynchronous*). Se for síncrono, a unidade de execução de origem é suspensa até que a execução do código transferido seja terminada; se for assíncrono, a unidade de origem não é suspensa.

Quanto à hora em que o código deve ser executado, nos mecanismos assíncronos pode ser executado de duas maneiras: **imediata** (*Immediate*), o código é executado assim que recebido; ou **postergada** (*Deferred*), o código é executado após uma determinada condição ser satisfeita.

3.3.2 Gerência do espaço de dados

Após a migração de uma unidade de execução para um novo ambiente computacional, as ligações com os recursos devem ser reorganizadas. Isto pode envolver ligações nulas, novas ligações ou até migração de recursos. A escolha dependerá da natureza do recurso, do tipo de ligação e dos requerimentos impostos pela aplicação.

Recursos podem ser modelados como uma tripla: *recurso = (I,V,T)*, onde *I* é um identificador único, *V* é o valor do recurso e *T* o tipo do recurso, que determina o tipo de informação que ele retém bem como sua interface. O tipo determina também se o recurso é **transferível** ou **não transferível**, ou seja, se ele pode ser migrado de um ambiente computacional para outro. Por exemplo, um arquivo pode ser transferido, mas uma impressora provavelmente não. Recursos transferíveis podem ser associados aos valores **livre** ou **fixo**. O recurso livre pode ser migrado de um ambiente a outro, enquanto que o fixo está permanentemente associado a um ambiente computacional.

Recursos podem ser ligados a uma unidade de execução através de três formas: **por identificador**, a unidade de execução deve estar ligada a um recurso único específico, que não pode ser substituído por outro equivalente (mesmo tipo); **por valor**, a unidade de execução deve estar ligada a um recurso de um determinado tipo e que seu valor obedeça a uma condição estabelecida pela unidade de execução, neste caso o que interessa é o conteúdo do recurso e o identificador é irrelevante; e **por tipo**, a

unidade de execução deve estar ligada com um recurso que deve obedecer apenas a uma condição de tipo, não interessando seu valor ou sua identidade. É possível ter diferentes tipos de ligação no mesmo recurso.

Na migração de uma unidade de execução, duas classes de problemas devem ser analisadas na estratégia de gerenciamento do espaço de dados: realocação de recursos e reconfiguração das ligações. Os mecanismos mais usados para o gerenciamento são cinco, discutidos a seguir e ilustrados na Fig. 4.

O primeiro mecanismo e mais simples é independente do tipo de ligação, sendo chamado de **remoção da ligação** (*Binding Removal*). Neste caso, quando uma unidade de execução ligada a um recurso migra, a ligação é simplesmente descartada.

Quando um recurso é intransferível, depois que a unidade de execução alcançou o ambiente computacional destino, a ligação na unidade de execução pode ser feita por **referência na rede** (*Network Reference*), onde a ligação é modificada para referenciar o recurso no ambiente computacional fonte.

Quando no ambiente computacional destino há um recurso do mesmo tipo que o recurso referenciado pela unidade de execução no ambiente fonte, pode ser feita uma **re-ligação** (*Re-Binding*). Neste mecanismo, a ligação com o recurso no ambiente computacional fonte é desfeita, e uma nova ligação com o recurso no ambiente computacional destino é criada.

Quando o recurso pode ser copiado, a ligação pode ser feita **por cópia** (*By Copy*), onde uma cópia do recurso é criada, a ligação é modificada para referenciar a cópia e a mesma é transferida junto com a unidade de execução para o ambiente computacional destino.

No último mecanismo, **por movimento** (*By Move*), o recurso é transferido junto com a unidade de execução para o ambiente computacional destino, mas a ligação com o ambiente computacional fonte não é desfeita.

Os modos em que estes mecanismos são aplicados para reconfigurar as ligações são limitados tanto pela natureza do recurso envolvido quanto pelo tipo de ligação. Estas relações são apresentadas na tab. 1.

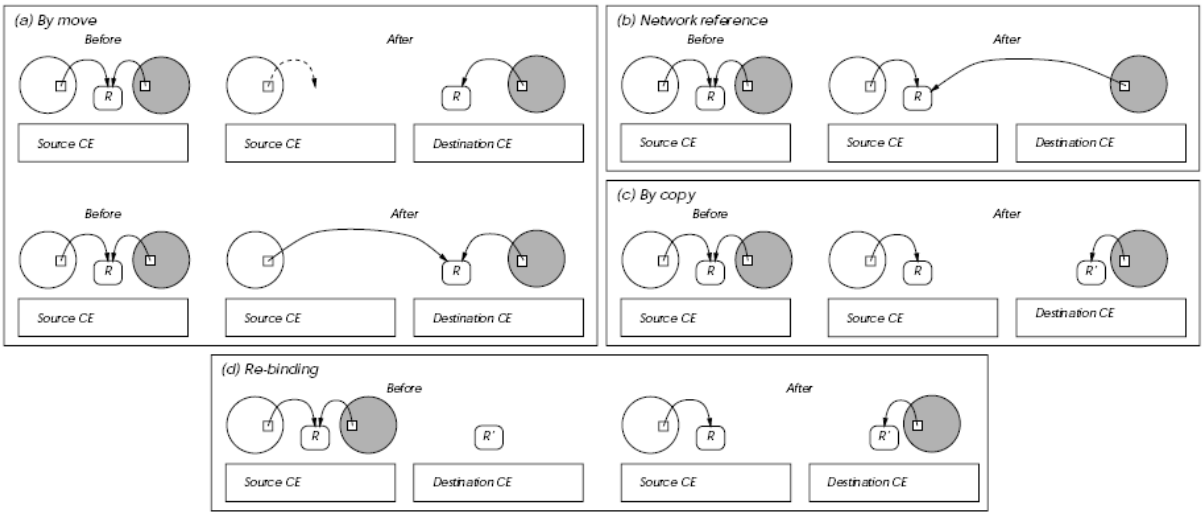

Figura 4 – Mecanismos de gerenciamento do espaço de dados

Fonte: FUGGETTA; PICCO; VIGNA, 1998.

Tabela 1 – Ligações, recursos e mecanismos de gerenciamento do espaço de dados

	Transferível livre	**Transferível fixo**	**Intransferível fixo**
Por identificador	*Por movimento (referência na rede)*	*Referência na rede*	*Referência na rede*
Por valor	*Por cópia (por movimento, referência na rede)*	*Por cópia (referência na rede)*	*(referência na rede)*
Por tipo	*Re-ligação (referência na rede, por cópia, por movimento)*	*Re-ligação (referência na rede, por cópia)*	*Re-ligação (referência na rede)*

Fonte: FUGGETTA; PICCO; VIGNA, 1998.

3.4 Paradigmas para mobilidade

Abordagens tradicionais para modelagem de software não são suficientes para modelagem de sistemas largamente distribuídos, que exploram mobilidade de código e reconfiguração dinâmica de componentes de software. Nestes casos, os conceitos de localização, distribuição dos componentes pelas localidades e migração de componentes entre localidades devem ser levadas em conta explicitamente durante a fase de modelagem.

Também se devem considerar os aspectos que diferenciam a interação entre componentes em ambientes remotos da interação entre componentes locais. Dentre esses aspectos cita-se uma maior latência de comunicação, maior tempo para acesso à memória não local, mais pontos de falha e necessidade de utilização de mecanismos para controle de concorrência e sincronização. A não atenção a aspectos como estes, pode resultar em problemas de desempenho e confiabilidade após a fase de implementação.

Mencionada a importância de se identificar paradigmas para projeto de sistemas distribuídos explorando código móvel, deve-se também identificar a tecnologia mais adequada para a implementação do paradigma.

A fim de conceituação, algumas concepções básicas são apresentadas, que abstraem as entidades envolvidas no sistema, como arquivos, variáveis, código executável ou processos. As abstrações básicas são as seguintes: Componentes, Interações e *Sites* (CARZANIGA; PICCO; VIGNA, 1997). Pode-se perceber a semelhança com os conceitos apresentados na seção 3.2. A nova nomenclatura é usada de acordo com a referência citada.

Entende-se por **componentes** os elementos que compõem a arquitetura. Podem estar divididos em: **componente recurso**, que é o componente embutido na arquitetura que representa dados passivos ou dispositivos físicos. Um tipo particular é o **componente código**, que representa um algoritmo ou programa (*know-how*); e **componente computacional**, que representa o fluxo de controle da computação. É caracterizado por um estado e ligações com outros componentes.

Interações são eventos que envolvem dois ou mais componentes.

A noção de localização é dada pelos ***Sites***, que são os ambientes de execução.

Os paradigmas são descritos em termos de padrões de interação que definem a recolocação e a coordenação entre componentes necessários para realizar um serviço (computação). A computação em si só pode ser efetuada quando o *know-how* descrevendo a computação, os recursos necessários e o componente computacional responsável pela computação estão no mesmo *site*.

Em (FUGGETTA; PICCO; VIGNA, 1998), são identificados os três principais paradigmas que exploram código móvel: **Avaliação Remota**, **Código em Demanda** e

Agentes Móveis. Estes paradigmas são caracterizados pela localização dos componentes antes e depois da execução da computação, pelo componente computacional que é responsável para execução do código e onde a computação realmente acontece.

Paradigmas para código móvel modelam explicitamente o conceito de localização. Assim é possível determinar, em fase de modelagem, custos para interações entre componentes. Em geral, conforme (FUGGETTA; PICCO; VIGNA, 1998), interações entre componentes que compartilham a mesma localização tem um custo considerado negligenciável quando comparado a interações entre *sites* através da rede. A escolha do paradigma a ser utilizado deve ser pensada caso a caso, de acordo com o tipo de aplicação.

Grande parte dos paradigmas mais conhecidos são estáticos quanto à localização e código. Através dos paradigmas descritos a seguir, as aplicações ganham uma flexibilidade provida pela mobilidade de componentes. Em conseqüência dessa capacidade, os componentes podem mudar dinamicamente a qualidade das interações, diminuindo custos.

3.4.1 Avaliação remota

No paradigma de avaliação remota, um componente tem o *know-how* da computação que deseja realizar, mas não possui os recursos necessários. Este componente, a fim de que sua tarefa seja efetuada, envia o *know-how* de sua tarefa para um *site* remoto que possua os recursos necessários. Em uma interação adicional, o componente remoto que realizou a tarefa envia os resultados para o componente original.

3.4.2 Código em demanda

Na abordagem de código em demanda, um componente deseja acessar um serviço, já possui os recursos necessários para tal, mas não possui nenhuma informação de como manipulá-los (*know-how*) para obter os resultados. Neste ponto, o componente faz uma requisição do *know-how* do serviço em um *site* remoto que o

possua. Assim que o mesmo é entregue, o componente executa localmente a computação.

3.4.3 Agentes móveis

No paradigma de agentes móveis, o *know-how* é conhecido por um componente que deseja executar uma computação. Este componente possui parte dos recursos, e o restante dos recursos está em um outro *site.* Então o componente tenta executar as partes da computação em que são possíveis com os recursos locais. Após, o componente migra para o *site* que possui o restante dos recursos para completar sua computação. A migração inclui o estado da computação, o código e possivelmente alguns recursos.

O paradigma de agentes móveis se difere fundamentalmente dos outros, pois neste o foco é a migração do código e do estado da computação para o *site* remoto, enquanto que nos outros, o foco é a transferência de código entre componentes.

4 PADRÕES DE PROJETO PARA COMPUTAÇÃO MÓVEL

Seguindo a pauta de conceitos necessários para o desenvolvimento do trabalho, este capítulo introduz alguns fundamentos básicos sobre padrões de projeto e contextualiza-os em computação móvel, bem como descreve o comportamento dos padrões utilizados neste trabalho.

4.1 Fundamentos

Padrões de projeto foram identificados pela primeira vez no final da década de 70, pelo arquiteto Christopher Alexander. Apesar dos documentos escritos por Alexander serem direcionados à Arquitetura e Urbanismo, seus conceitos puderam ser aplicados em outras disciplinas, incluindo desenvolvimento de software (APPLETON, 2007). Na área da computação, padrões de projeto se popularizaram através do livro (GAMMA et al., 1995).

Um padrão de projeto sugere uma determinada solução para um problema recorrente em um determinado contexto (GAMMA et al., 1995). O uso de padrões proporciona um vocabulário comum para a comunicação entre projetistas, criando abstrações num nível superior ao de classes e garantindo uniformidade na estrutura do software (GALL; KLOSCH; MITTERMEIR, 1996).

De acordo com Gamma et al. (1995), padrões de projeto variam em granularidade e grau de abstração, e são classificados por dois critérios: **propósito** e **escopo** (tab. 2).

A classificação por propósito reflete o que o padrão faz. Segundo essa classificação, os padrões podem ser:

a) de criação: tratam do processo de criação de objetos;
b) estruturais: tratam da composição entre classes e objetos;
c) comportamentais: caracterizam as maneiras em que classes e objetos interagem e distribuem responsabilidades.

A classificação por escopo define se o padrão deve ser primariamente aplicado sobre classes ou sobre objetos. Dentro dessa classificação os padrões podem ser:

a) de classes: tratam das relações entre classes e suas subclasses. Essas relações são estabelecidas através de herança, ou seja, são fixadas em tempo de compilação.
b) de objetos: tratam das relações dinâmicas, que podem mudar em tempo de execução.

Tabela 2 – Classificação de padrões

		Propósito		
		De criação	**Estrutural**	**Comportamental**
Escopo	**De classes**	*Factory Method*	*Adapter*	*Interpreter* *Template Method*
	De objetos	*Abstract Factory* *Builder* *Prototype* *Singleton*	*Adapter* *Bridge* *Composite* *Decorator* *Facade* *Proxy*	*Chain of responsibility* *Command* *Iterator* *Mediator* *Memento* *Flyweight* *Observer* *State* *Strategy* *Visitor*

Fonte: GAMMA et al., 1995.

Os padrões para computação móvel utilizados no desenvolvimento do *framework* descrito neste trabalho foram modelados como ***Template Method***, ou seja, são classificados como **comportamentais** e **de classe**. Um padrão ser modelado como *Template Method* significa ter um conjunto de classes abstratas, que definem uma estrutura, um esqueleto de um algoritmo para uma tarefa, deixando alguns passos desse algoritmo para serem definidos pelo programador através de subclasses, sem mudar a estrutura do algoritmo.

4.2 Padrões Recorrentes de Computação Móvel

O principal critério de seleção dos padrões a serem utilizados no *framework* foi sua recorrência em sistemas de aquisição de informação distribuída (BOIS; TRINDER; LOIDL, 2005), que compõem uma classe de aplicações típicas na mobilidade de código (FUGGETTA; PICCO; VIGNA, 1998).

Na comunidade científica pode-se encontrar uma grande variedade de padrões de projeto direcionados especialmente ao paradigma de agentes móveis. Particularmente, as referências (ARIDOR; LANGE, 1998) e (LIMA et al., 2004) não tiveram seus padrões incluídos neste trabalho, mas algumas questões discutidas nelas, como por exemplo metodologia de modelagem, ajudaram na compreensão de como os padrões podem e devem ser modelados, e tiveram influência no desenvolvimento deste trabalho.

Na referência de onde foram retirados os padrões, eles foram implementados na linguagem *mHaskell* (BOIS; TRINDER; LOIDL, 2004) através de **Esqueletos de Mobilidade** (*Mobility Skeletons*). Esqueletos de mobilidade são funções polimórficas de alto nível na linguagem *mHaskell* que encapsulam padrões recorrentes de computação móvel. O objetivo dos esqueletos de mobilidade é facilitar a codificação de aplicações móveis, abstraindo os fatores de baixo nível da mobilidade de código, fazendo com que o programador preocupe-se apenas com a computação realizada localmente nos nodos.

Pode-se perceber a similaridade dos esqueletos de mobilidade com este trabalho, onde ambos visam facilitar o desenvolvimento de aplicações móveis. A diferença fundamental é de que este trabalho tenta aplicar o mesmo tipo de abstração dos esqueletos de mobilidade em um ambiente pervasivo, aproveitando-se dos recursos que o mesmo oferece, em uma linguagem de maior aceitação na comunidade de desenvolvedores, neste caso, Java.

Os padrões escolhidos são três: **Mmap**, **Mfold** e **Mzipper**. Os nomes foram inspirados em funções típicas de linguagens funcionais, onde os comportamentos assemelham-se aos padrões propostos (STORCH; BOIS; YAMIN, 2007). A seguir serão

descritos brevemente os comportamentos dos padrões, que serão detalhados no capítulo de desenvolvimento do *framework* (cap. 5).

4.2.1 O Padrão Mmap – Broadcast

O primeiro e o mais simples padrão de computação móvel selecionado é o Mmap. O padrão Mmap define um comportamento de *broadcast* (difusão). Esse padrão estrutura uma aplicação que envia uma tarefa a vários nodos, contidos em uma lista que deve ser passada como parâmetro além de um valor de entrada que pode ser processado. O resultado retornado pela execução do padrão também deve ser uma lista, do mesmo tamanho da lista de nodos, contendo o resultado da execução de cada tarefa, como ilustrado na Fig. 5.

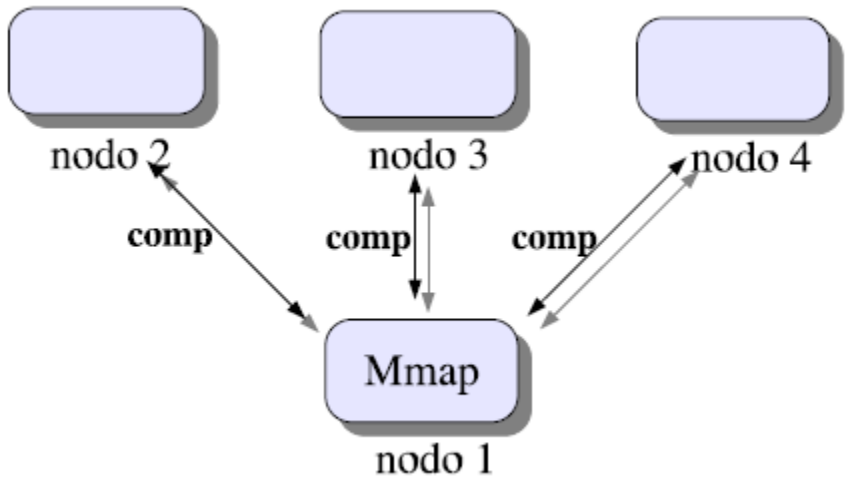

Figura 5 – O comportamento do Mmap

Fonte: STORCH; BOIS; YAMIN, 2007.

4.2.2 O Padrão Mfold – Aquisição de Informação Distribuída

O segundo padrão selecionado é o Mfold. Este padrão descreve uma tarefa que visita uma lista de nodos, executando uma computação em cada um deles combinando os resultados encontrados em um acumulador, utilizando uma operação apropriada. Quando atinge o último nodo da lista, o Mfold retorna o resultado da tarefa para a máquina inicial que disparou o padrão. A Fig. 6 ilustra o funcionamento deste padrão.

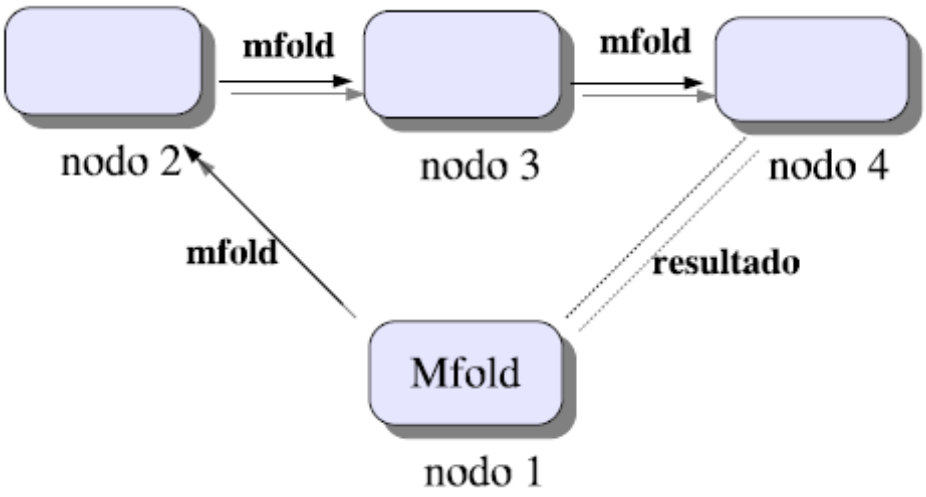

Figura 6 – O comportamento do Mfold

Fonte: STORCH; BOIS; YAMIN, 2007.

Teoricamente, as aplicações implementadas com o padrão Mfold também podem ser implementadas usando-se o padrão Mmap, porém os programas terão um comportamento operacional completamente diferente, como pode ser visto na Fig. 5 e na Fig. 6. Utilizando o Mmap, a máquina inicial ficaria encarregada de fazer a operação de combinação após o retorno dos resultados de cada execução nos nodos. Utilizando Mfold, a operação de combinação seria executada localmente em cada nodo, logo após a computação da tarefa.

4.2.3 O Padrão Mzipper – Iteração

O terceiro padrão é o Mzipper. Este padrão descreve uma tarefa móvel que tenta encontrar um valor que seja satisfatório a uma lista de nodos em uma rede. O valor é gerado no primeiro nodo e é testado através de um predicado em cada nodo restante. Caso cada teste seja satisfatório, a computação é feita no próximo nodo da lista. Caso o predicado falhe, a computação é movida para o nodo inicial da lista, onde um novo valor é gerado e a busca recomeçada. A Fig. 7 ilustra o comportamento do padrão.

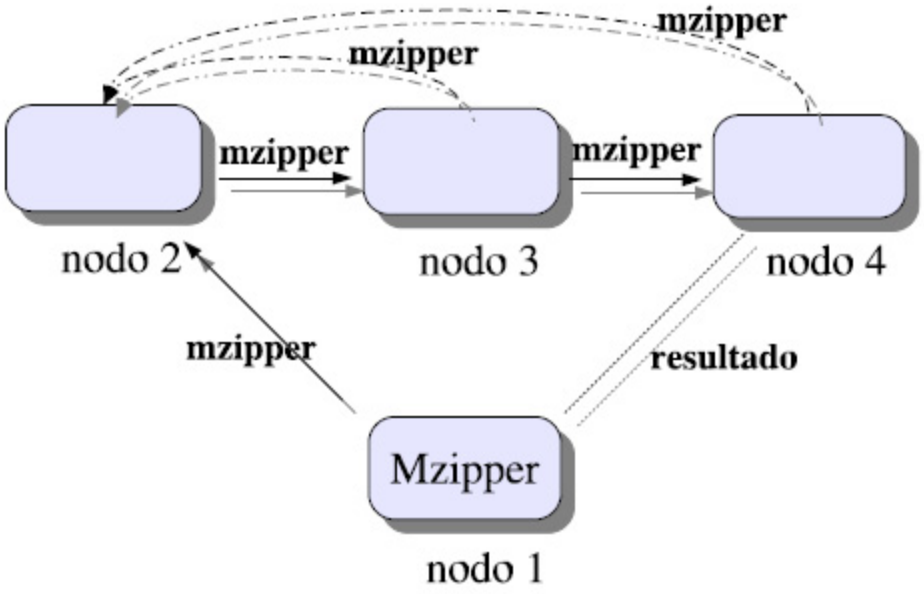

Figura 7 – O comportamento do Mzipper

Fonte: STORCH; BOIS; YAMIN, 2007.

5 MODELAGEM DO FRAMEWORK

Neste capítulo será apresentado o projeto GRADEp, plataforma base escolhida para este trabalho, que implementa um ambiente de execução de aplicações para a computação pervasiva. Também serão detalhados os aspectos mais relevantes do projeto, levados em consideração para o desenvolvimento deste trabalho.

Após, será descrita a forma de modelagem do *framework* proposto. Serão apresentados alguns conceitos iniciais bem como o argumento que caracteriza este trabalho como um *framework*. Em seguida, será apresentada a modelagem final, com a estrutura geral dos padrões e depois as estruturas específicas de cada um. Por fim, serão abordadas questões relevantes levadas em consideração durante a implementação.

5.1 GRADEp revisitado

O objetivo desta seção é sumarizar os principais aspectos da plataforma de execução pervasiva utilizada, o GRADEp, que foram considerados para o desenvolvimento deste trabalho.

5.1.1 O projeto ISAM e o GRADEp

O projeto ISAM (Infra-estrutura de Suporte às Aplicações Móveis) é uma iniciativa que começou seus primeiros trabalhos em 2001 na Universidade Federal do Rio Grande do Sul (UFRGS), com o objetivo de definir uma arquitetura de software para a computação pervasiva (YAMIN et al., 2004). O foco do projeto ISAM é a infra-estrutura de suporte necessária para o desenvolvimento e execução das aplicações

móveis distribuídas com comportamento adaptativo em um ambiente da computação pervasiva (ISAM, 2007). Inserido no projeto ISAM, está o projeto GRADEp (GRADE pervasiva).

O projeto GRADEp define um ambiente de execução para aplicações pervasivas, implementado na forma de um *middleware*. Este *middleware* fica responsável por prover funcionalidades para gerenciamento de uma grade pervasiva em tempo de execução, e fornecer subsídios para desenvolvimento das aplicações móveis, distribuídas e conscientes de contexto, da computação pervasiva.

O *middleware* foi inicialmente proposto sob o nome de EXEHDA (*EXecution Environment for Highly Distributed Applications*), que mais tarde começou a ser distribuído com o nome GRADEp. Hoje, além da UFRGS, o projeto conta com o apoio de outras instituições como a Universidade Católica de Pelotas (UCPEL), Universidade Federal de Santa Maria (UFSM) e, mais recentemente, da Universidade Federal de Pelotas (UFPEL). O projeto ainda está em desenvolvimento, mas já oferece um conjunto considerável de funcionalidades.

5.1.2 Objetivos do GRADEp

Considerando o propósito do GRADEp de atender a perspectiva do ambiente pervasivo definido no ISAM, de trabalhar com:

a) equipamentos de alta heterogeneidade, de *PDAs*, *desktops* até *workstations* e servidores, sendo significativamente diferentes quanto ao perfil computacional;
b) suporte à mobilidade física e, assim, viabilizar uma possível troca de ambiente onde acontece o processamento;
c) um meio físico de execução de abrangência global, instável quanto à disponibilidade de recursos.

Faz-se necessário que o *middleware* proposto expresse consciência do seu próprio estado e do ambiente computacional, tendo a capacidade de adaptar-se às condições dos recursos disponibilizados pelo meio físico de execução.

De acordo com Yamin (2004), para atender adequadamente os requisitos da arquitetura ISAM foi concebido o GRADEp: **um ambiente para execução de**

aplicações móveis e distribuídas, que oferece suporte à sensibilidade ao contexto (consciência de contexto, que pode gerar uma adaptação), à semântica siga-me, e a uma estratégia colaborativa entre a aplicação e o *middleware* nas decisões de adaptação.

5.1.3 A arquitetura de software do GRADEp

A Fig. 8 apresenta uma visão geral da arquitetura de software para o GRADEp. A concepção original apresenta sua arquitetura em uma organização lógica de três camadas, que serão discutidas a seguir: (superior) camada de aplicação; (intermediária) camada de suporte e ambiente de execução; e (inferior) camada de sistema básico.

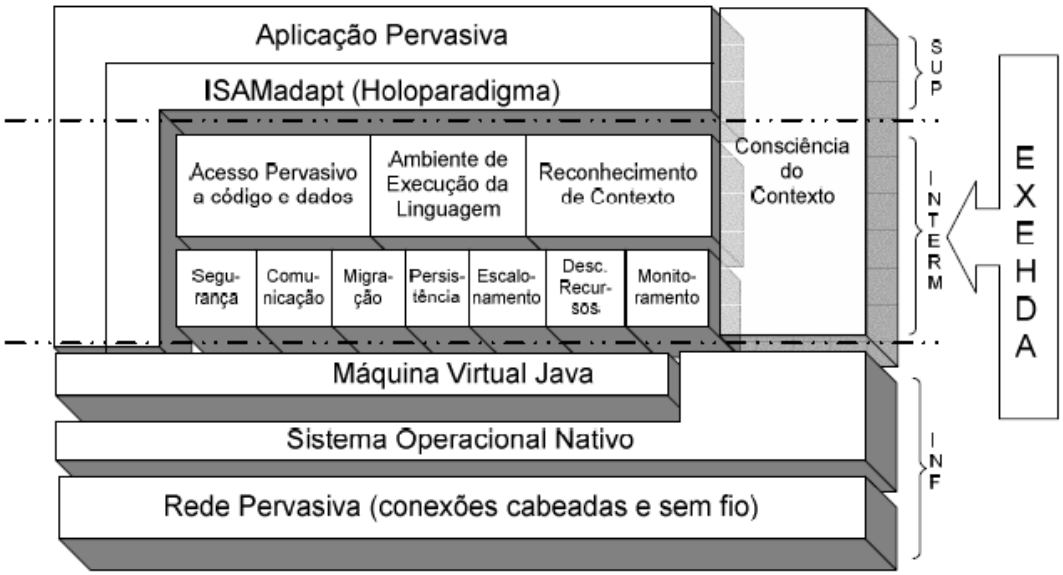

Figura 8 – Visão geral da arquitetura GRADEp

Fonte: YAMIN, 2004.

Na camada superior (**aplicação**) encontra-se a linguagem de programação ISAMadapt (AUGUSTIN, 2004), a qual disponibiliza abstrações para programação de aplicações distribuídas, móveis e conscientes do contexto direcionadas à computação pervasiva. O uso dessa linguagem específica para o desenvolvimento de aplicações pervasivas poderia melhor atender à demanda de requisitos imposta pelas mesmas. Entretanto, por ser uma linguagem diferente, também exigiria uma curva maior de

aprendizado. Por esse motivo, na maioria das vezes, a comunidade de desenvolvedores prefere usar a linguagem nativa do GRADEp, o Java.

Na camada intermediária (EXEHDA), que representa a implementação do GRADEp, estão os **mecanismos de suporte à execução das aplicações pervasivas e às estratégias de adaptação**. Esta camada é formada por dois níveis:

O primeiro nível é composto por três módulos de serviços às aplicações: acesso pervasivo a código e dados, reconhecimento de contexto e ambiente de execução da aplicação.

O acesso pervasivo a dados e código compreende os componentes que disponibilizam o ambiente pervasivo (ISAMpe), incluindo Ambiente Virtual do Usuário (AVU), Ambiente Virtual da Aplicação (AVA) e Base de Dados pervasiva das Aplicações (BDA). O AVU compõe-se dos elementos que integram a interface de interação do usuário com o sistema. Este módulo é responsável por implementar o suporte para que a aplicação que o usuário está executando em uma localização possa ser instanciada e continuada em outra localização sem descontinuidade, permitindo o estilo de aplicações siga-me num ambiente pervasivo. Por sua vez, entende-se como AVA o conjunto de atributos que identifica uma execução específica de uma aplicação, enquanto a BDA constitui o repositório de códigos das aplicações em geral.

O ambiente de execução da linguagem é o encarregado pelo gerenciamento da execução da aplicação durante seu tempo de vida.

O serviço de reconhecimento de contexto é responsável por informar o estado dos elementos de contexto de interesse da aplicação e os de interesse do próprio ambiente de execução.

No segundo nível da camada intermediária estão os serviços básicos do *middleware*, que provêm as funcionalidades necessárias para o primeiro nível e cobrem vários aspectos, tais como migração – mecanismos para deslocar um componente de software de um equipamento para outro; persistência – mecanismo para aumentar a disponibilidade e o desempenho do acesso a dados; descoberta de recursos – para dar suporte à mobilidade dos dispositivos e dos componentes entre diferentes células; comunicação – com possibilidade de ser anônima e assíncrona; escalonamento – permite decidir o melhor nodo para criar ou migrar os componentes da aplicação;

monitoramento – sensores que fornecem informações sobre o ambiente de execução e a aplicação.

A camada inferior da arquitetura é composta pelos **sistemas e linguagens nativas que integram o meio físico de execução**. Devido à característica de portabilidade, nesta camada a plataforma base de implementação é a Máquina Virtual Java (JVM) em suas diferentes abordagens, tornando a linguagem Java (JAVA 2007a) a linguagem nativa do GRADEp. Duas plataformas são utilizadas prioritariamente: JSE (Java Standard Edition) e JME (Java Micro Edition). A arquitetura supõe a existência de uma rede global com suporte à operação sem fio, conectada a outra rede cabeada, que disponibilize uma infra-estrutura de equipamentos e serviços em escala global (rede pervasiva).

Como visto, a arquitetura GRADEp está organizada em camadas lógicas, com níveis diferenciados de abstração, e está direcionada para a busca da manutenibilidade da qualidade de serviços oferecida ao usuário móvel através do conceito de adaptação. Nesta, o sistema se adapta para fornecer qualidade dos serviços prestados, enquanto que a aplicação se adapta para atender a expectativa do usuário móvel, mantendo a funcionalidade da aplicação (YAMIN, 2004).

5.1.4 Ambiente físico pervasivo do GRADEp

A estrutura física gerenciada pelo GRADEp inclui recursos e serviços distribuídos, compostos tanto por equipamentos dos usuários do sistema como pelos equipamentos de infra-estrutura de suporte, configurados com seus respectivos perfis de execução. Essa estrutura foi denominada de ISAMpe (ISAM *pervasive environment*). O ISAMpe caracteriza o escopo de gerenciamento do GRADEp.

A organização do ISAMpe pode ser observada na Fig. 9. O cenário proposto pelo GRADEp para suportar as premissas da computação em grade, computação móvel e computação sensível ao contexto, é mapeada em uma organização composta pela agregação de células de execução – EXEHDAcels. Conforme já mencionado, o GRADEp foi inicialmente proposto sobre o nome de EXEHDA, o que reflete na nomenclatura usada para as entidades que compões o ambiente.

Os recursos da infra-estrutura física são classificados em três abstrações básicas, as quais são utilizadas na composição do ISAMpe.

A primeira é a **EXEHDAcel**. Uma EXEHDAcel delimita o escopo de atuação de uma EXEHDAbase. É composta por uma EXEHDAbase e por vários EXEHDAnodos. Os principais aspectos considerados na definição da abrangência de uma célula são: o escopo institucional, a proximidade geográfica e o custo de comunicação.

A **EXEHDAbase** é o ponto de contato para os EXEHDAnodos. É responsável por todos os serviços básicos de gerenciamento do ISAMpe e, apesar de constituir uma referência lógica única, seus serviços poderão estar distribuídos entre vários equipamentos, refletindo escalabilidade.

A última é o **EXEHDAnodo**. EXEHDAnodos são os equipamentos de processamento disponíveis no ISAMpe, sendo responsáveis pela execução das aplicações. São gerenciados pela EXEHDAbase e normalmente são equipamentos de usuários. Um caso específico deste tipo é o **EXEHDAnodo móvel**. São os nodos do sistema com elevada portabilidade, tipicamente dotados de interface de rede para operação sem fio e, neste caso, fazem parte da célula a qual seu ponto-de-acesso está subordinado. São funcionalmente análogos aos EXEHDAnodos, porém eventualmente com uma capacidade mais restrita de processamento.

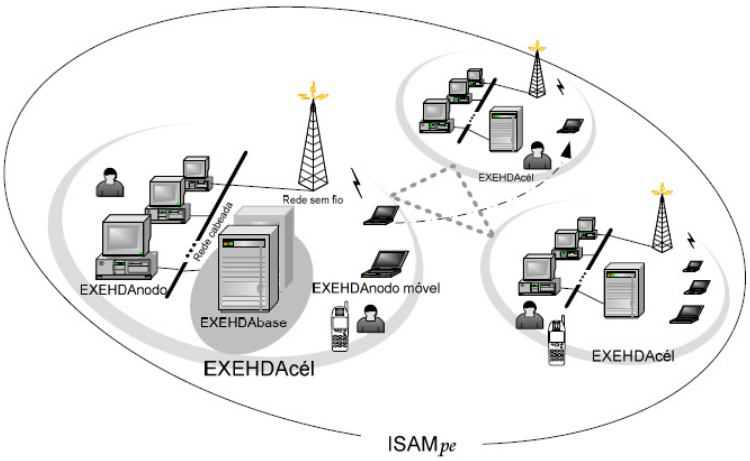

Figura 9 – ISAM *Pervasive Environment*

Fonte: YAMIN, 2004.

A estrutura física que compõe o ISAMpe constitui-se por uma rede de infra-estrutura, cuja composição final pode ser alterada pela agregação dinâmica de nodos móveis.

A perspectiva intercelular do ISAMpe implica em uma composição multi-institucional de equipamentos, o que conduz a um procedimento de gerência análoga ao praticado nos ambientes de grade computacional (YAMIN et al., 2003). A forma como a organização celular do ISAMpe é gerenciada tem por objetivo central conservar a autonomia das instituições envolvidas.

O GRADEp não oferece mecanismos específicos de gerência para recursos especializados como impressoras por exemplo. Mas permite a catalogação destes dentro de uma célula, tornando-os passíveis de serem localizados dinamicamente e, portanto, podem ser utilizados pelas aplicações pervasivas.

5.1.5 Organização em serviços do GRADEp

O GRADEp é um *middleware* adaptativo organizado em serviços. A premissa de operação em um ambiente heterogêneo, onde a capacidade de processamento e memória dos dispositivos é muito variável, fez do GRADEp um *middleware* que segue a estratégia de *micro-kernel.* O GRADEp é constituído de um conjunto mínimo de

serviços básicos necessários, com funcionalidades estendidas por outros serviços carregados sob demanda, facilitando a adaptação. O GRADEp implementa diferentes versões dos serviços, permitindo que seja carregada a versão que se encaixe melhor no dispositivo. Isto é possível, pois na modelagem os serviços foram definidos por sua interface e não por sua implementação (YAMIN, 2004).

Prevendo mobilidade no ambiente pervasivo gerenciado pelo GRADEp, o *middleware* foi projetado para manter-se funcional mesmo em períodos de desconexão (**desconexão planejada**). Para tanto, além de fornecer primitivas específicas de comunicação para esta situação, o GRADEp separa a implementação dos serviços em instância nodal (serviço que roda em um EXEHDAnodo) e instância celular (serviço que roda em uma EXEHDAbase). Assim, o *middleware* consegue gerenciar os recursos do ambiente a fim de permitir que instâncias nodais permaneçam funcionais mesmo desconectadas. Percebe-se aqui que para permitir um comportamento móvel dos EXEHDAnodos, a EXEHDAbase assume o papel de referência em uma EXEHDAcel, sendo uma entidade estável na célula, por definição.

Desta forma, o relacionamento entre os serviços de instância nodal e os de instância celular (relacionamento **intracelular**) assume um caráter ***superpeer***, enquanto que o relacionamento entre serviços de instância celular (**intercelular**) assume um caráter ***peer-to-peer***. A abordagem *peer-to-peer* entre células vai ao encontro do fator escalabilidade (YAMIN 2004).

O GRADEp tem a carga dinâmica de serviços orientada através de **perfis de execução**. Um perfil de execução define um conjunto de serviços a ser ativados, bem como suas respectivas implementações: a princípio, nodal ou celular; e suas respectivas políticas de carga: **boot** ou **sob demanda**. A primeira política faz com que o serviço seja iniciado junto com o *middleware*, a segunda faz com que o serviço seja carregado quando for necessário. Observa-se que esta personalização do perfil de execução é flexível para os EXEHDAnodos, para as EXEHDAbases existem algumas restrições devido à responsabilidade das mesmas no ambiente.

A carga dos serviços é controlada pelo núcleo mínimo, que é composto por dois componentes: um **gerenciador de perfis** (*ProfileManager*) e um **gerenciador de serviços** (*ServiceManager*).

O gerenciador de perfis interpreta a informação descrita no perfil de execução e a disponibiliza para os outros serviços do *middleware*.

O gerenciador de serviços realiza a ativação de serviços conforme a informação disponibilizada pelo gerenciador de perfis.

Percebe-se que para ter a funcionalidade de instalação de código sob demanda é necessário que o mesmo esteja disponível em algum lugar. Em termos de computação pervasiva, é inviável que este código esteja disponível em todos os dispositivos do sistema. Deste modo, o *middleware* oferece um serviço de armazenamento de código na EXEHDAbase, que pode ser acessado pelos EXEHDAnodos para a carga sob demanda, remotamente.

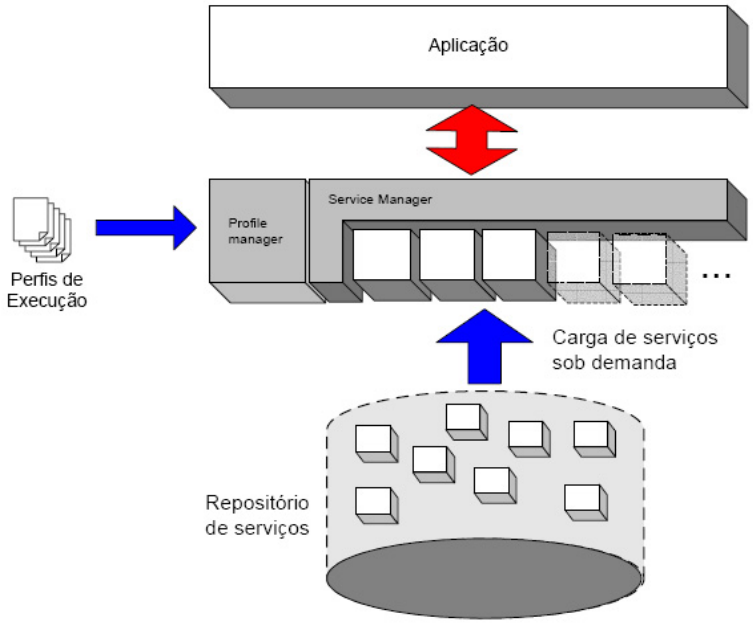

Figura 10 – Organização do núcleo do GRADEp

Fonte: YAMIN, 2004.

5.2 Mobilidade em Java

Como já mencionado anteriormente, a linguagem nativa do GRADEp é o Java. A linguagem Java foi desenvolvida pela Sun Microsystems e lançada em meados de 1995. Desde então, se popularizou rapidamente, chegando a quatro milhões de desenvolvedores em todo o mundo, antes de 2005 (JAVA, 2007b).

O objetivo inicial dos projetistas da linguagem era prover uma linguagem orientada a objetos, que fosse portável, limpa, fácil de aprender e de propósito geral. Mas esse objetivo ganhou outro rumo com o crescimento da Internet, despertando bastante a atenção na área de mobilidade de código.

A linguagem Java, é baseada na idéia de que os programas desenvolvidos com ela, só podem ser executados sobre uma máquina virtual projetada para isso: a Máquina Virtual Java (JVM), que é a implementação de um ambiente computacional. O compilador Java traduz os códigos-fonte escritos nesta linguagem em um código intermediário chamado *Java Byte Code*, que é interpretado pela JVM. Já que a JVM faz um papel de ambiente computacional e todo código-fonte Java é traduzido para *Java Byte Code*, pode-se concluir que programas Java são independentes de plataforma.

Java fornece um mecanismo programável, o carregador de classes (*class loader*), que recupera e liga classes dinamicamente em uma JVM. O carregador de classes é acionado pela JVM quando o código que está rodando contém uma classe não resolvida. Então, o *class loader* busca a classe, que pode estar em um outro ambiente computacional, e carrega esta classe na JVM. Nesse ponto, o código correspondente é executado. Ainda, a busca e a ligação da classe podem ser disparadas explicitamente pela aplicação, independente da necessidade de executar o código da classe.

Desta maneira, Java suporta mobilidade fraca, usando mecanismos de busca de fragmentos de código. Tais mecanismos são assíncronos e suportam tanto execução imediata quanto postergada. Em ambos os casos, o código carregado é sempre executado do início e não possui nem estado de execução, nem ligações a recursos remotos, não sendo necessário gerenciamento do espaço de dados (FUGGETTA; PICCO; VIGNA, 1998).

5.2.1 Mobilidade no GRADEp

Nesta subseção, serão abordados os aspectos de mobilidade no GRADEp, conforme documentado em (YAMIN, 2004).

O GRADEp foi construído sobre a plataforma Java e, portanto, baseia-se nos mesmos mecanismos citados na seção anterior. Adicionalmente, o GRADEp não

suporta o mecanismo de mobilidade forte, no entanto simula esse suporte parcialmente, através de uma primitiva para migração de objetos. A migração provida pelo GRADEp permite que o espaço de dados seja migrado com o objeto, mas não permite que o estado da execução seja migrado. Dessa forma, valores de atributos e referências a objetos associados ao objeto em questão não são perdidos durante a migração, mas o ponteiro da pilha e o ponteiro de instruções são perdidos.

A primitiva mencionada é implementada por um método no serviço Executor responsável pela migração: *moveObject* (o qual será tratado na subseção 5.6.3). Antes da migração, o *middleware* captura o estado (apenas o espaço de dados) do objeto a ser migrado, guarda em uma estrutura específica (*MarshaledOX*), e o envia juntamente com o objeto. No nodo destino, este estado é restaurado e sua execução é reiniciada a partir de um objeto específico de ativação.

Ainda, existe uma primitiva responsável pela instanciação de um novo objeto no ambiente pervasivo. Ela permite que a instanciação do objeto seja feita em um nodo remoto, caracterizando mobilidade fraca. A primitiva é implementada através do método *createObject* (subseção 5.6.3), que instancia um objeto de uma classe passada por parâmetro.

O gerenciamento do espaço de dados no GRADEp é implementado através de um esquema integrado de manipulação de recursos, possuindo serviços específicos para catalogação, descoberta, alocação, monitoração e acesso inter-celular de recursos.

Os recursos são catalogados da mesma forma discorrida na seção 3.3.2, onde são representados pela tripla: *recurso = (I,V,T)*. A cada recurso, também podem ser associados atributos, com o propósito de detalhamento da descrição do mesmo. A catalogação e o registro de recursos serão mais bem abordados na subseção 5.6.3.

Os recursos ainda podem ser tratados através de descrições XML (*eXtensible Markup Language*), armazenadas de forma persistente e tratadas pelo *middleware* em tempo de execução conforme a necessidade. O programador pode optar por, ao invés de escrever código relativo ao registro de um recurso específico ou de um conjunto de recursos, descrever este recurso ou conjunto de recursos sob uma sintaxe XML, de

forma a fazer o registro com apenas uma primitiva, onde qualquer recurso que se encaixe na descrição XML será incluído no processamento de catalogação.

A gerência do espaço de dados também leva em consideração o escopo de administração (recurso local, celular ou inter-celular). Como já mencionado, existem vários serviços com responsabilidades diferentes para a manipulação de recursos. Estes serviços interagem entre si para a tomada de decisão final.

5.3 Conceito e características de um *framework*

Framework é um conjunto de classes que cooperam entre si para tornar modelagens reusáveis para uma classe específica de software (JOHNSON; FOOTE, 1988). Um *framework* deve ditar a arquitetura da aplicação, definindo classes e objetos, suas responsabilidades e como vão se relacionar, prevendo decisões comuns de modelagem do domínio de aplicações o qual irá cobrir, permitindo que o programador se concentre na modelagem específica de sua aplicação. Assim, *frameworks* dão mais ênfase para reuso de modelagem do que reuso de código (GAMMA et al., 1995).

Frameworks seguem uma lógica inversa a de bibliotecas em relação ao reuso. Com bibliotecas, o programador escreve a estrutura principal da sua aplicação, fazendo chamadas ao código que deseja reusar. Com um *framework*, a estrutura principal é reusada e o programador deve escrever o código das funções chamadas.

Com a estrutura da aplicação pré-definida, as decisões de modelagem a serem tomadas pelo programador são reduzidas, devido a elas terem sido tomadas na concepção do *framework*. Como resultado, aplicações são construídas mais rapidamente e possuem a mesma estrutura, tornando-se mais fáceis de manter. Em contrapartida, é perdida certa liberdade e flexibilidade nas aplicações, já que a estrutura foi definida previamente e decisões já foram tomadas.

Frameworks costumam ter a missão de cobrir todas as aplicações do domínio ao qual se propõem a fazê-lo. As aplicações construídas sobre um *framework* são dependentes do mesmo. Este fato torna o baixo acoplamento uma das características mais importantes para ser levadas em consideração na concepção de um *framework*, caso contrário qualquer mudança neste pode gerar maiores mudanças na estrutura da aplicação.

Padrões de projeto e *frameworks* são conceitos para reuso de modelagem e de código que possuem algumas similaridades. Segundo Gamma et al. (1995), eles se diferem basicamente de três maneiras:

a) padrões de projeto são mais abstratos que *frameworks*. *Frameworks* podem ser transformados em código, mas apenas exemplos de padrões de projeto podem ser transformados em código;
b) padrões de projeto são elementos arquiteturais menores que *frameworks*. Um *framework* pode conter vários padrões de projeto, mas o inverso não é verdadeiro;
c) padrões de projeto são menos específicos que *frameworks*. *Frameworks* sempre têm um domínio específico. Padrões de projeto podem ser utilizados em qualquer tipo de aplicação.

5.4 Domínio das aplicações cobertas pelo *framework*

Apesar de conceitualmente a principal contribuição de um *framework* ser definir uma arquitetura para uma classe de aplicações, este objetivo excede o escopo para caracterização de um trabalho de graduação. O *framework* proposto neste trabalho é uma tentativa de facilitar a modelagem e codificação de aplicações baseadas em padrões de projeto para mobilidade de código, mas sob uma perspectiva diferente.

O objetivo principal na construção do *framework* não foi cobrir um domínio inteiro de aplicações móveis, mas sim aquelas aplicações cujo comportamento se assemelha a pelo menos um dos comportamentos descritos nos padrões de projeto mostrados na seção 4.2. Dessa forma, o foco do *framework* está em definir o comportamento da aplicação que o usará, e não sua arquitetura.

Esta abordagem pode levar a interpretação de que este trabalho caracterize-se como uma biblioteca ao invés de um *framework*. O argumento a favor do *framework* está no fato de que para expressar o comportamento desejado, foi necessário desenvolver uma micro-arquitetura, que ficará subjacente à aplicação, ditando apenas o comportamento, sem ditar a arquitetura da mesma. Mesmo o *framework* desenvolvido neste trabalho tendo foco no comportamento, a demanda de alguns requisitos na aplicação por fim acabam definindo parte da arquitetura da mesma, assim

caracterizando definitivamente um *framework*, e ainda permitindo flexibilidade no restante da arquitetura da aplicação.

5.5 Modelagem no GRADEp

A modelagem do *framework* foi fundamentada em alguns princípios:

a) foco no comportamento das aplicações. Como já mencionado, a preocupação maior no desenvolvimento foi manter o comportamento previsto pelo padrão de projeto;
b) baixo acoplamento. A preocupação em abstrair do programador particularidades do *framework* e até mesmo do *middleware* baseou muitas das decisões tomadas;
c) encapsulamento. Houve um cuidado especial para delegação das responsabilidades de cada elemento, de modo a expressar da forma mais correta possível “quem pode acessar o que”.

A modelagem partiu de (FIELD et al., 2006), onde dois dos padrões deste trabalho estão modelados independentemente de plataforma e no *framework* Voyager (VOYAGER, 2007). Nesta referência são discutidas questões como: o que motivou o desenvolvimento deste padrão, quando usá-lo, que paradigma é o mais apropriado para a solução, quais as vantagens e desvantagens, entre outras. A modelagem final no GRADEp procurou equilibrar tais questões com características particulares do *middleware*, como aquelas referentes à mobilidade, discutidas na seção 5.2.1, bem como aquelas específicas de implementação, que serão discutidas na seção 5.6. Durante o desenvolvimento, a modelagem sofreu eventuais modificações de acordo com necessidade em respeitar tais características.

5.5.1 Estrutura geral dos padrões de projeto

Os padrões foram modelados e implementados como *template methods* na linguagem Java e todos eles seguem uma estrutura semelhante. A idéia empregada do *template method* foi de definir o algoritmo relativo ao comportamento geral do padrão, e de deixar para o programador o código a ser executado remotamente em cada nodo da lista. Devido à característica intrínseca de programação distribuída, de haver uma

thread local de controle e demais *threads* para execução remota, o *template method* foi modelado separando o algoritmo relativo à *thread* de controle do padrão em uma classe diferente do algoritmo das *threads* remotas, que conterão o código do programador.

Cada padrão possui uma **classe principal**, com o mesmo nome do padrão, que encapsula o algoritmo relativo ao controle do comportamento do respectivo padrão. Esta classe possui um método que dispara a execução do padrão sob um contexto estático (método com o modificador *static*), e gerencia a mesma até o retorno dos resultados. Todos os padrões foram construídos sob uma natureza síncrona, com a semântica de que a execução da *thread* que disparou o padrão é interrompida para a execução do código do mesmo, até que este retorne os resultados. Dessa forma, através desse método de disparo, a aplicação do programador terá toda a comunicação, distribuição e sincronismo referentes ao comportamento gerenciados por este mesmo método.

Em adição a esta classe, cada padrão possui uma classe correspondente ao código da aplicação do programador, que se refere à computação a ser executada remotamente em cada nodo da rede. O algoritmo do programador deve ser colocado dentro de métodos específicos de acordo com o padrão escolhido. Estes métodos foram declarados como abstratos no padrão, conforme sugere o *template method.* Sendo assim, a implementação deles é delegada de forma obrigatória para o programador que irá utilizar o *framework*. Esta classe é que encapsulará o **código móvel,** sendo movida para os nodos remotos de acordo com o respectivo padrão, caracterizando a mobilidade de código.

Além destas duas classes referentes aos padrões de projeto, foi necessário incluir outras duas inerentes ao funcionamento no GRADEp. A primeira é uma **interface** que descreve os métodos de acesso para comunicação remota entre os objetos móveis e a classe principal de controle. A interface definida é utilizada por um serviço de comunicação GRADEp que é baseado no Java RMI (*Remote Method Invocation*): o WORB; mas com a vantagem de não requerer manutenção da conexão. Este serviço será abordado na subseção 5.6.3.

A segunda é uma **classe de ativação** que é utilizada pelo *middleware* quando da instanciação de um novo objeto ou da migração de um objeto já existente. O

funcionamento dessa estratégia de ativação está detalhado na subseção 5.6.2. Mas, basicamente, é uma classe que implementa algoritmos de ativação e desativação de um objeto instanciado/migrado no *middleware*, que são chamados na forma de *callback*. Além de seu uso obrigatório ela foi útil para complementar os comportamentos dos padrões, no sentido de que o código a ser executado remotamente pelo objeto móvel é chamado e gerenciado através destes métodos, conforme o comportamento do padrão.

Os padrões de projeto Mfold e Mzipper contêm uma classe adicional devido ao fato de terem sido modelados segundo o paradigma de agentes móveis. Conforme já mencionado, o GRADEp é capaz de simular parcialmente mobilidade forte, de forma suficiente para a implementação deste paradigma. Sendo o "agente" a classe que conterá o código móvel, implementado pelo programador, a classe referida funciona como uma **classe "procuradora"** (*proxy*) do agente, criada para armazenar os resultados intermediários e outros dados que o agente carrega. Nessa classe também é guardada uma referência para o agente "procurado". Durante a migração de um agente, o objeto a ser migrado será o procurador do agente, e, como este possui uma referência para o agente, consequentemente o agente (na verdade, uma cópia dele) acompanhará a migração.

Essa estratégia de *proxy* é considerada um padrão de projeto por si própria (GAMMA et al., 1995). O emprego desta estratégia foi motivado pelo aumento do nível de encapsulamento que a mesma oferece. Assim, a instância *proxy* fica responsável pelos dados e a instância do agente fica responsável pelo código. Como a classe *proxy* possui acesso privado dentro do *framework*, foi ocultada da aplicação a estrutura de dados utilizada pelo agente.

5.5.2 Mmap

Conforme argumentado em (FIELD et al., 2006), a escolha do paradigma para a implementação manteve-se a mesma: Avaliação Remota (REV). Uma vez que esse é o paradigma subjacente à implementação de mobilidade de código no GRADEp, não haveria razão para mudar a escolha. Na Fig. 11 pode-se visualizar o diagrama UML (*Unified Modeling Language*) de classes do Mmap.

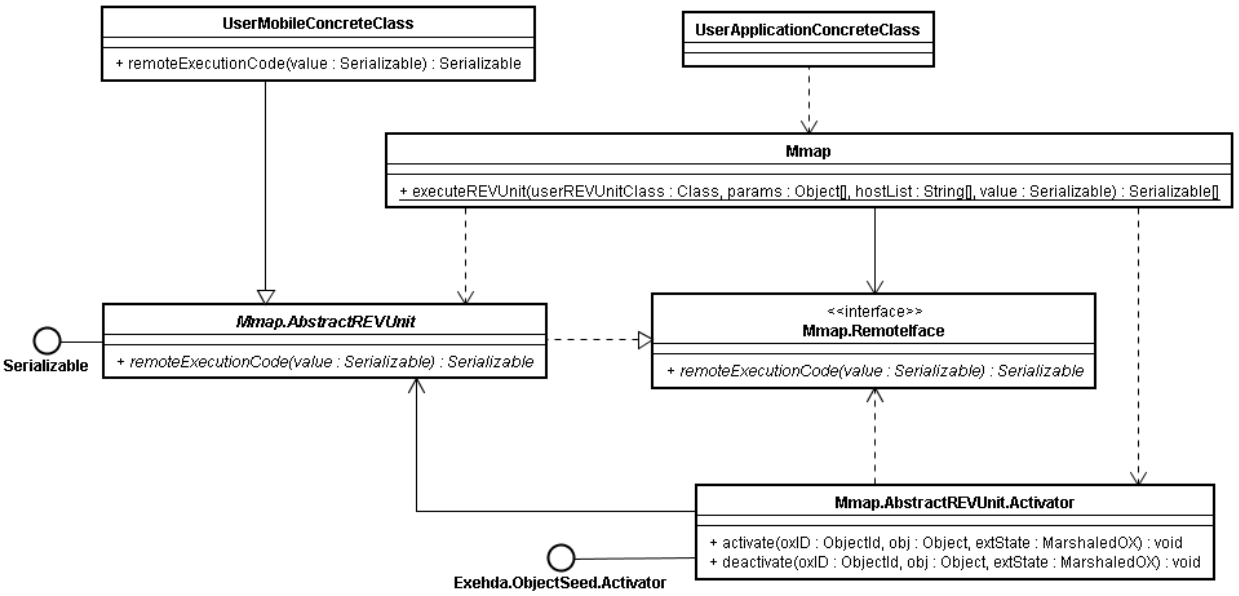

Figura 11 – Diagrama de classes do padrão Mmap

A classe *Mmap* faz o papel de **classe principal** do padrão. Nela, está contido apenas o método de disparo *executeREVUnit*, responsável pelo gerenciamento do comportamento do padrão. A *thread* de controle está relacionada a uma execução deste método.

O primeiro parâmetro do método *executeREVUnit* deve ser a classe do programador que implementa o método de execução nos nodos remotos, ou, no caso do diagrama da Fig. 11, a classe *UserMobileConcreteClass*. A escolha dessa forma de modelagem foi baseada no paradigma escolhido, com o objetivo de deixar intuitivo ao programador de que não haverá resultados intermediários, ou seja, o programador passa a classe e os parâmetros do seu método construtor, e a gerência da criação dos objetos é delegada para o padrão e para o *middleware*. Dessa forma, ainda é permitido ao programador executar um código de inicialização, através do construtor da classe.

O segundo parâmetro é a lista de argumentos a serem passados para o construtor da classe referida.

O terceiro parâmetro é a lista de nodos a serem usados na execução do Mmap. Em cada um dos nodos passados por esta lista será criada uma instância da classe passada por parâmetro, com os respectivos argumentos.

O quarto parâmetro é um valor qualquer de entrada, que pode ser utilizado na execução remota. Este valor é o mesmo valor passado por parâmetro no método implementado pelo programador. Assim, é possível enviar um valor de entrada para a execução remota, definido em tempo de execução.

A interface *RemoteIface* (aninhada dentro de *Mmap*) é a **interface** utilizada para comunicação interna do padrão; da *thread* de controle com as unidades REV remotas.

A classe *AbstractREVUnit* (aninhada dentro de *Mmap*), representa o **código móvel**. Esta classe apenas contém um método que implementaria a interface *RemoteIface*, mas já que esse método é abstrato, sua implementação é delegada ao programador. Este método corresponde ao código que será executado em cada um dos nodos da lista, e que deve retornar um valor de qualquer natureza. A razão para separar esta classe da interface acima é de abstrair do programador a estrutura interna do padrão, incluindo os mecanismos de comunicação. Por mais que funcionalmente pudesse ser retirada sem nenhum prejuízo, ela mantém o encapsulamento em um nível um pouco mais elevado.

Por fim, a classe *Activator* (aninhada dentro de *AbstractREVUnit*) faz o papel de **classe de ativação**. Os detalhes de seu uso podem ser vistos na subseção 5.6.2. É através dela que os objetos remotos ficarão acessíveis à *thread* de controle no nodo de disparo.

O funcionamento deste padrão é relativamente simples. A partir de uma chamada do método *executeREVUnit* dentro da classe *UserApplicationConcreteClass* é disparada uma execução do Mmap. A *thread* de controle faz alguns testes sobre os parâmetros passados, se são válidos, converte a lista de nodos em uma lista de identificadores no GRADEp, e inicializa algumas variáveis internas. Após, é criada em cada nodo remoto uma instância da classe *UserMobileConcreteClass* (passada como parâmetro) com os argumentos fornecidos no disparo do padrão e estabelecendo um objeto de ativação (subseção 5.6.2), instância da classe *Mmap.AbstractREVUnit.Activator*. Logo após a criação de um objeto no nodo destino, é recuperado o endereço para a chamada do método remoto, usado para recuperar a referência do objeto remoto através da interface *Mmap.RemoteIface*. Com a referência

para o objeto remoto, é criada localmente uma nova *thread* por nodo para fazer a chamada, de forma que cada *thread* fique responsável por chamar e esperar o resultado da execução remota. Após o retorno de todos os resultados, uma lista contendo todos eles é retornada como resultados da chamada de *executeREVUnit.*

A recuperação do endereço da referência do objeto remoto e da própria referência é feita com auxílio de serviços específicos do GRADEp, bem como a criação dos objetos nos nodos remotos e a disponibilização do acesso a estes objetos. A forma como eles foram utilizados será detalhada na subseção 5.6.3.

5.5.3 Mfold

O paradigma de Agentes Móveis foi escolhido para a implementação do Mfold por ser o mais apropriado considerando o comportamento do padrão, conforme sugerido em (FIELD et al., 2006). Mesmo o GRADEp não suportando mobilidade forte, a simulação de agentes provida pelo mesmo pode satisfazer as necessidades de implementação, com os devidos cuidados, descritos na seção 5.6. Na Fig. 12 pode-se visualizar o diagrama UML de classes representando a estrutura específica do Mfold.

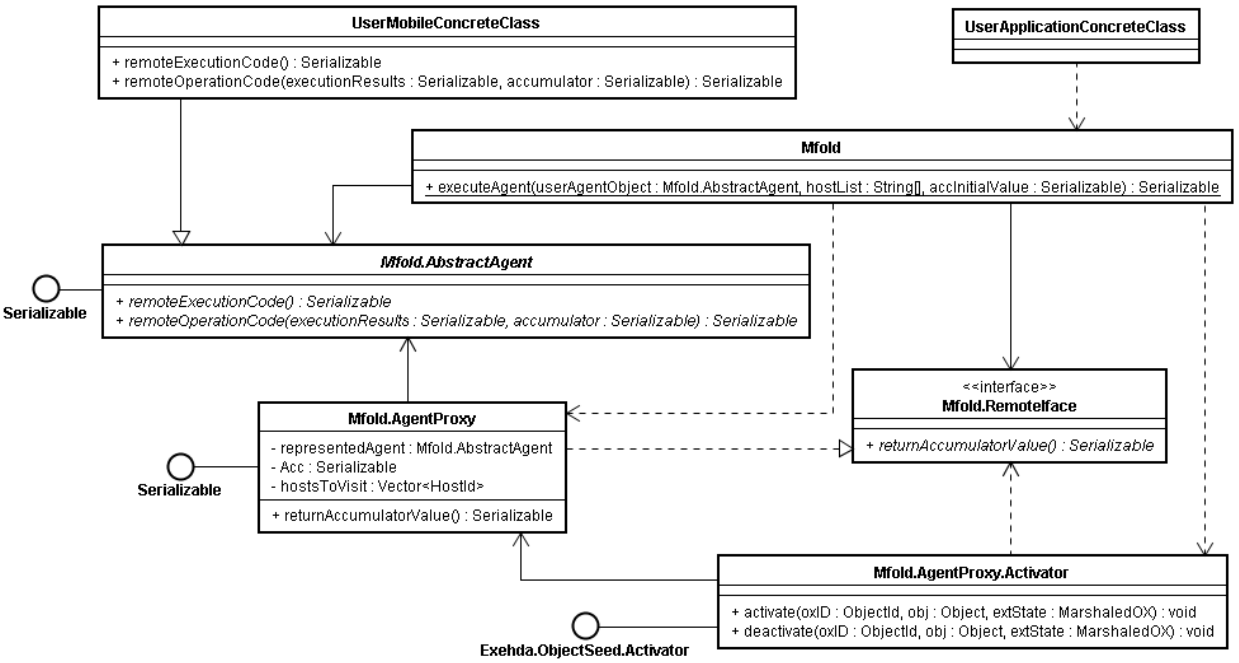

Figura 12 – Diagrama de classes do padrão Mfold

A classe *Mfold* faz o papel de **classe principal** do padrão. O método *executeAgent* é o método de disparo, responsável pelo gerenciamento do comportamento do padrão (*thread* de controle). Diferentemente do Mmap, onde é passada a própria classe como primeiro parâmetro, no Mfold o primeiro parâmetro deve ser uma instância da classe *UserMobileConcreteClass*. Como o paradigma escolhido é o de agentes móveis, essa forma de modelagem foi escolhida com o objetivo de deixar intuitivo ao programador o paradigma de agentes, fornecendo ao mesmo um mecanismo de armazenamento de dados que será “migrado” junto com o agente. Isso quer dizer que o programador pode criar uma instância da classe *UserMobileConcreteClass*, e configurá-la antes de fornecê-la ao Mfold, além de colocar um código de manipulação deste objeto dentro dos métodos remotos. Assim é feita a emulação do agente.

É necessário observar um cuidado especial que se deve ter com este mecanismo. Serão abordadas na subseção 5.6.4 as conseqüências de se optar por *createObject* ou *new*, mas pode-se adiantar que o objeto instanciado na máquina virtual Java, configurado e passado por parâmetro para o Mfold, não poderá mais deixar este

ambiente computacional. Ou seja, enquanto ele permanecer no ambiente computacional onde foi instanciado, as operações serão feitas diretamente sobre ele. No momento em que for feita a migração do procurador do agente, como o agente não pode deixar o ambiente computacional, será feita uma cópia dele no ambiente computacional destino. As operações subseqüentes à migração serão feitas sobre a cópia, ficando para a máquina virtual de origem uma referência obsoleta do agente.

O segundo parâmetro do *executeAgent* é a lista de nodos a serem visitados pelo agente do Mfold. O procurador do agente será instanciado no primeiro nodo da lista e migrará nodo a nodo da lista, na mesma ordem em que foi passada.

O terceiro parâmetro é um valor qualquer que será atribuído como valor inicial do acumulador do agente.

A interface *RemoteIface* (aninhada dentro de *Mfold*) é a **interface** utilizada para comunicação interna do padrão; da *thread* de controle com o procurador do agente.

Conforme já mencionado, os padrões modelados sob o paradigma de agentes móveis utilizaram uma estratégia de *proxy*. A classe que representa o **procurador do agente** é a classe *AgentProxy* (aninhada dentro de *Mfold*). Sob a responsabilidade dessa classe estão: o acumulador, utilizado para armazenar o resultado da operação de combinação feita nos nodos remotos; a lista de nodos a serem visitados; e uma referência para o agente representado. O único método provido por ela é o método de acesso ao valor do acumulador, que implementa a interface de comunicação, para que a *thread* de controle obtenha o resultado, quando for apropriado.

A classe *AbstractAgent* (aninhada dentro de *Mfold*), representa o **código móvel**, neste paradigma, o agente. Esta classe apenas contém os métodos que devem ser implementados pelo programador. O método *remoteExecutionCode* corresponde a uma execução genérica, que retorna um valor qualquer. O *remoteCombinationCode* corresponde à operação de combinação do resultado da execução feita pelo *remoteExecutionCode*, com o valor já contido no acumulador. O retorno da combinação será o novo valor a ser atribuído ao acumulador.

E por fim, a classe *Activator* (aninhada dentro de *AgentProxy*) faz o papel de **classe de ativação**. Os detalhes de seu uso podem ser vistos na subseção 5.6.2. É

através dela que o agente e seu procurador ficarão acessíveis à *thread* de controle no nodo de disparo.

A execução deste padrão é disparada a partir de uma chamada ao método *executeAgent* dentro da classe *UserApplicationConcreteClass*. A *thread* de controle faz alguns testes sobre os parâmetros passados, se são válidos, converte a lista de nodos em uma lista de identificadores no GRADEp, e inicializa algumas variáveis internas. Após, é criada no primeiro nodo da lista passada por parâmetro uma instância da classe *Mfold.AgentProxy*, com seus devidos valores iniciais: a lista de nodos a serem visitados; o valor inicial do acumulador, passado por parâmetro; e uma referência para o objeto agente passado por parâmetro (instância de *UserMobileConcreteClass*). Após a criação do procurador do agente, o mesmo começa automaticamente sua execução e a *thread* de controle passa a esperar a publicação dos resultados. Quando os resultados estiverem prontos, através de serviços do GRADEp, a *thread* recupera a referência remota do procurador do agente, através da interface *Mfold.RemoteIface* e em seguida recupera o valor do acumulador.

O comportamento do Mfold é complementado no objeto de ativação (subseção 5.6.2), instância da classe *Mfold.AgentProxy.Activator*, que é associada ao procurador no momento de sua criação. Através desse objeto, são chamadas as execuções dos métodos *remoteExecutionCode* e *remoteCombinationCode* implementados pelo programador.

A chamada ao método de migração e a atualização da lista de nodos a serem visitados também são realizados neste objeto. A recuperação da referência do procurador do agente bem como a migração do mesmo é feita com auxílio de serviços específicos do GRADEp. A forma como eles foram utilizados será detalhada na subseção 5.6.3.

5.5.4 Mzipper

Como este padrão não foi documentado em (FIELD et al., 2006), sua modelagem foi baseada em (BOIS; TRINDER; LOIDL, 2005). Devido à similaridade entre o comportamento do Mfold com o Mzipper, o paradigma escolhido para a

implementação deste padrão é o mesmo: Agentes Móveis. Na Fig. 13 pode-se visualizar o diagrama UML de classes, representando a estrutura específica do Mzipper.

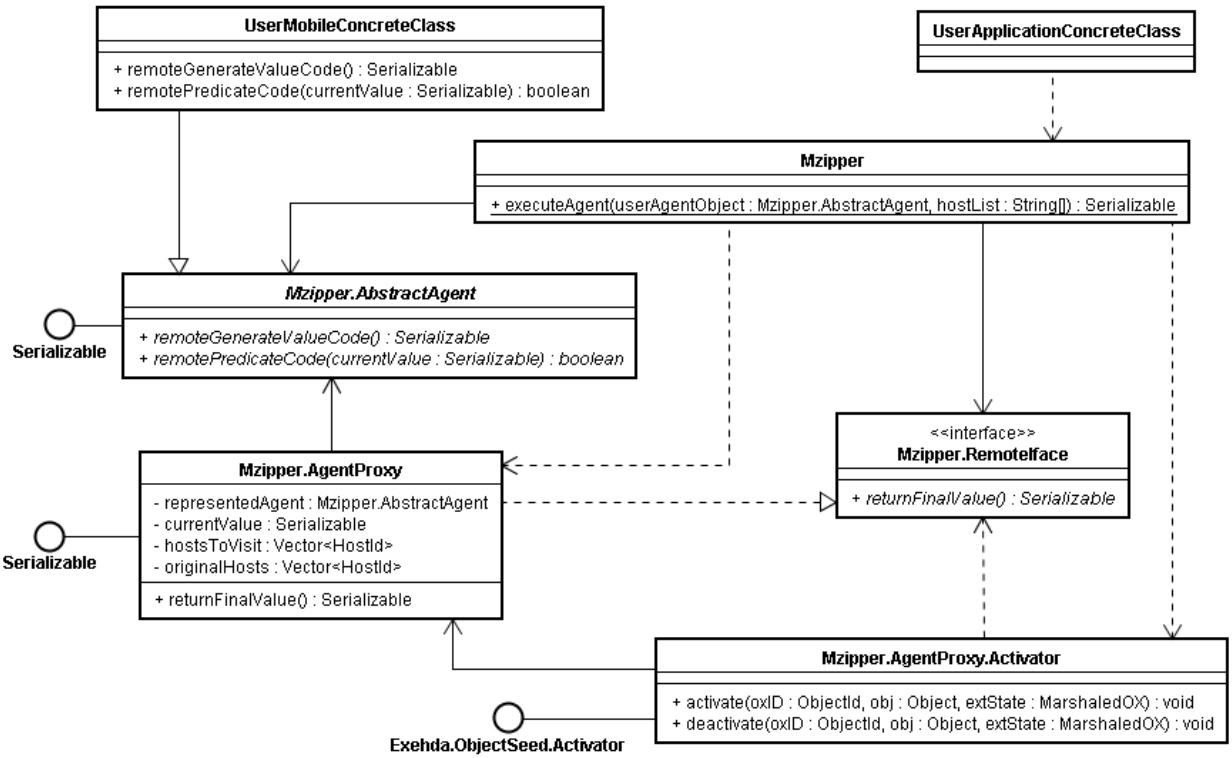

Figura 13 – Diagrama de classes do padrão Mzipper

Como pode ser visto, a estrutura do Mzipper é quase a mesma do Mfold. Basicamente o que mudou são as estruturas do agente e um parâmetro no método de disparo. A interface de comunicação apenas mudou de nome, para refletir a semântica associada ao Mzipper.

A classe *Mzipper* é a **classe principal**. O método *executeAgent* é o método de disparo (*thread* de controle). Assim como no Mfold o primeiro parâmetro deve ser uma instância da classe *UserMobileConcreteClass*. O segundo é a lista de nodos a serem visitados pelo agente. O procurador do agente será instanciado no primeiro nodo da lista e fará a migração nos nodos da lista, na mesma ordem em que foi passada. A observação feita no Mfold, relativa ao objeto agente não poder deixar o ambiente computacional, também é valida para o Mzipper.

A interface *RemoteIface* (aninhada dentro de *Mzipper*) é a **interface** utilizada para comunicação interna do padrão, mais especificamente, da *thread* de controle com o procurador do agente.

A classe *AgentProxy* (aninhada dentro de *Mzipper*) é a classe que representa o **procurador do agente**. Sob a responsabilidade dessa classe estão: o valor atual, que será testado nos nodos da lista; a lista de nodos a serem visitados e um *backup* dela, pois a lista vai sendo consumida durante a execução, então a original é mantida caso uma nova iteração seja iniciada; e uma referência para o agente representado. O único método provido por ela é o método de acesso ao valor corrente, que implementa a interface de comunicação, para que a *thread* de controle obtenha tal valor, quando for apropriado.

A classe que representa o **código móvel** é a *AbstractAgent* (aninhada dentro de *Mzipper*). Esta classe contém os métodos que devem ser implementados pelo programador. O *remoteGenerateValueCode* é o método que faz a geração de um novo valor a cada iteração do Mzipper. Ele é executado apenas no primeiro nodo da lista e o controle de quais valores já foram ou serão gerados deve ser feito pelo programador. O método *remotePredicateCode* é executado nos nodos restantes da lista (todos, com exceção do primeiro) a fim de testar se o valor atual (na iteração atual) é válido neste nodo ou não. Cabe salientar que o código escrito neste método pode estabelecer um comportamento diferente no Mzipper, mas igualmente funcional, caso o teste seja invertido (expressar o comportamento de achar um valor que seja satisfatório apenas no primeiro nodo), ou estático, o teste sempre retorna o mesmo valor booleano.

Da mesma forma que os outros padrões, a classe *Activator* (aninhada dentro de *AgentProxy*) faz o papel de **classe de ativação**.

O funcionamento do Mzipper é relativamente semelhante ao Mfold. A execução é disparada a partir de uma chamada ao método *executeAgent* dentro da classe *UserApplicationConcreteClass*. A *thread* de controle faz alguns testes sobre os parâmetros passados, se são válidos, converte a lista de nodos em uma lista de identificadores no GRADEp, e inicializa algumas variáveis internas. Em seguida, é criada no primeiro nodo da lista passada por parâmetro uma instância da classe *Mfold.AgentProxy*, com seus devidos valores iniciais: a lista de nodos a serem visitados;

e uma referência para o objeto agente passado por parâmetro (instância de *UserMobileConcreteClass*). Após a criação do procurador do agente, o mesmo começa automaticamente sua execução e a *thread* de controle passa a esperar a publicação dos resultados. Quando os resultados estiverem prontos, através de serviços do GRADEp, a *thread* recupera a referência remota do procurador do agente, através da interface *Mzipper.RemoteIface*, e, em seguida, recupera o valor final encontrado.

O comportamento do Mzipper também é complementado no objeto de ativação (subseção 5.6.2), instância da classe *Mzipper.AgentProxy.Activator*, que é associada ao procurador no momento de sua criação. Através desse objeto, são chamadas as execuções dos métodos *remoteGenerateValueCode* e *remotePredicateCode* implementados pelo programador.

No primeiro nodo da lista, é chamado o *remoteGenerateValueCode*. Nos nodos restantes, é chamado o *remotePredicateCode* e testado o seu resultado. Em caso de sucesso, o agente migra para o próximo nodo. Em caso de falha, recupera a lista de nodos através do *backup*. Após, se ainda existe um nodo a ser visitado na lista, é feita a migração para o primeiro da lista, se não, é publicado o resultado.

Cabe salientar aqui que o uso deste padrão exige um cuidado extra do programador. Pode-se perceber que o padrão não controla as iterações, ou seja, a menos que seja encontrado um valor bem sucedido (aceito em todos os nodos), o padrão executa um *loop* infinito. Em outras palavras, a condição de parada em caso de falha (de um, mais ou todos os valores) deve ser controlada pelo programador.

5.6 Aspectos relevantes de implementação

Conforme mencionado, a implementação do *framework* teve de se adaptar a algumas características intrínsecas ao estilo de programação do GRADEp, sendo que algumas delas influenciaram na modelagem também. As mais importantes serão discutidas nas próximas subseções.

5.6.1 A entidade OX

O GRADEp oferece às aplicações uma abstração para o tratamento de objetos no ambiente pervasivo. Essa abstração é o OX (Objeto eXehda). Um OX representa um

objeto no GRADEp, criado através da primitiva *createObject* do serviço Executor (subseção 5.6.3). Esta entidade guarda meta-informação referente ao objeto criado, na forma de pares (<atributo>,<valor>), onde <atributo> é uma cadeia de caracteres que representam um atributo e <valor> pode ter um conteúdo totalmente genérico, inclusive de natureza binária.

A gerência e manutenção das instâncias dessa abstração é feita por um serviço específico, o *OXManager*, o qual não foi utilizado explicitamente nesse trabalho, mas que interage com o serviço Executor. Através desse serviço é possível acessar e atualizar os atributos do OX em um caráter pervasivo, ou seja, de qualquer dispositivo e a qualquer hora, desde que se tenha o controle sobre aquele OX.

Na API do GRADEp, o OX é representado pela classe *ObjectId*. Um *ObjectId* armazena um identificador único composto de um inteiro de 16 bits acrescido também de um *hash* de 16 bits, juntamente com o identificador do nodo onde o OX foi instanciado.

5.6.2 Objeto de ativação

A modelagem do GRADEp previu para o serviço Executor (subseção 5.6.3) um modelo flexível, baseado no padrão de projeto *Strategy* (GAMMA et al.). Sob a perspectiva pervasiva, o serviço Executor implementa a idéia de que, ao instanciar remotamente ou migrar um OX, um algoritmo de ativação adequado àquele OX deve ser executado, como o disparo de *threads* associadas ou carregamento de bibliotecas por exemplo. O padrão *Strategy* materializa esse problema sob uma interface, permitindo que algoritmos diferentes sejam usados para ativar objetos diferentes.

Mais especificamente, existe uma interface na API do GRADEp que representa esses algoritmos de ativação: a interface *ObjectSeed.Activator*. Essa interface define dois métodos: *activate* e *deactivate*; que devem conter os códigos de ativação e desativação do objeto, respectivamente.

Quando um objeto é instanciado através do serviço Executor, é associada a ele uma instância de uma classe, escrita pelo programador, que implementa a referida interface. Esse objeto de ativação acompanhará o OX criado pelo resto do seu ciclo de vida, sendo utilizado tanto na instanciação quanto na migração.

Os métodos *activate* e *deactivate* são chamados pelo Executor na forma de *callback*. O método de ativação é chamado assim que o OX é instanciado através da primitiva *createObject* ou assim que o mesmo chega no nodo destino através da primitiva *moveObject*. O método de desativação é chamado antes de um objeto migrar para o nodo destino. Em ambos os métodos, são parâmetros o OX referente ao objeto a ser migrado; o objeto Java representado pelo OX e uma estrutura especial para armazenar/recuperar o estado do objeto.

5.6.3 Serviços do GRADEp utilizados explicitamente

O GRADEp é um *middleware* orientado a serviços, implementando portanto uma série destes, cada um com responsabilidades específicas que interagem entre si de forma a compor o ambiente pervasivo proposto. Eles foram modelados baseados em interfaces, fornecendo uma API acessível para o programador.

Embora o programador não faça uso de todos eles explicitamente, faz-se necessário entender o funcionamento dos serviços envolvidos em uma determinada situação. Para o desenvolvimento deste trabalho, foi feito um estudo superficial de todos os serviços do GRADEp, e, conforme necessário, foram aprofundados os estudos em três serviços específicos, cuja API foi utilizada explicitamente no trabalho. Estes serviços serão abordados a seguir.

O primeiro deles é o **WORB**. O WORB é um serviço que faz parte do subsistema de comunicação do GRADEp, que permite ao programador abstrair aspectos de baixo nível referentes ao tratamento das comunicações em rede. Este serviço é semelhante ao mecanismo RMI do Java, mas com a vantagem de não requerer manutenção da conexão durante a execução da chamada remota (GEYER, 2005). Esta característica vai ao encontro da premissa da desconexão planejada, permitindo uma melhor eficiência nas comunicações.

A interface do serviço WORB possui métodos relativos à manipulação de registro de objetos como serviços (perspectiva RMI). Neste trabalho, foram usados dois deles:

a) *exportService*: registra um objeto no nodo local que poderá atender chamadas remotas aos seus métodos, desde que eles estejam

definidos em uma interface. O próprio objeto, sua interface e um nome para o serviço são passados como parâmetros. O retorno dessa primitiva é o endereço completo do serviço, na forma de URL (*Uniform Resource Locator*);

b) *lookupService*: procura por um serviço exportado previamente pela primitiva anterior. A procura não precisa necessariamente ser no nodo local, podendo ser feita em um nodo remoto. O endereço de onde será procurado o serviço deve estar na forma de URL, da mesma forma provida pelo retorno da primitiva anterior. Tanto o endereço como a interface a ser procurada devem ser passados por parâmetro neste método. O retorno é uma referência, a princípio remota, do objeto que irá executar o serviço.

Este serviço foi empregado na implementação dos três padrões. No caso do Mmap, foi usado no disparo da execução remota, onde um objeto que contém o método implementado pelo programador é exportado através do *exportService*, com sua respectiva interface e com o nome sendo composto de uma constante associada ao padrão, concatenada com o identificador único do OX referente ao objeto exportado, na forma de string, garantindo desta forma que não haja conflitos de execuções simultâneas de aplicações utilizando o Mmap. A *thread* de controle, localizada no nodo de disparo, fica então encarregada de utilizar o *lookupService* para obter a referência para o objeto remoto e então fazer a chamada do serviço. Como a manutenção da conexão é provida pelo *middleware*, após a chamada basta esperar o retorno.

No caso do Mfold e do Mzipper, o serviço foi empregado na recuperação do resultado contido no procurador do agente. Quando um agente obtém os resultados da execução do padrão, após visitar todos os nodos da lista, o procurador é exportado como serviço, da mesma forma explicada acima, ou seja, com sua respectiva interface e com um nome associado ao padrão e ao identificador único do OX referente ao procurador. Da mesma forma, a *thread* de controle fica responsável em fazer a busca e a chamada ao serviço exportado.

Pode-se perceber que foram abstraídos alguns pontos relativos ao comportamento e à utilização deste serviço, como a forma que os elementos irão

compor a URL que conterá o endereço e o nome do serviço bem como permitir a *thread* de controle ter conhecimento sobre este endereço. Neste ponto, é introduzido o segundo serviço utilizado neste trabalho: o **CIB** (*Cell Information Base*).

O serviço CIB faz parte do subsistema de execução distribuída e possui a responsabilidade de armazenar informações referentes a recursos que se encontram na célula, de forma a facilitar o gerenciamento de sua infra-estrutura. O CIB utiliza uma abordagem similar à explicada na subseção 3.3.2, onde os recursos são modelados e catalogados na forma: *recurso = (I,V,T)*, com um adicional de terem atributos associados, o que enriquece a descrição de um recurso.

O registro de um recurso é implementado por mais de uma estrutura. O identificador único está associado à classe *ResourceName* na API do GRADEp. Uma instância dessa classe possui uma referência a uma instância da classe *NameComponent*, que representa o par (<tipo>,<valor>) associado ao recurso. A estrutura de atributos é implementada em pares de *strings* na forma (<nome>,<valor>).

A interface deste serviço provê métodos para registro e remoção de recursos na célula, bem como para fazer pesquisa de recursos e definir atributos ou resgatar atributos já definidos. Neste trabalho, foram utilizados quatro métodos deste serviço:

a) *addResource*: registra a existência de um recurso na célula. O recurso a ser adicionado é passado por parâmetro na chamada, que deve ser uma instância da classe *ResourceName*;
b) *removeResource*: cancela o registro de um recurso feito através da primitiva anterior, cancelando também todos atributos associados a ele. O parâmetro a ser passado é o recurso a ter o registro cancelado, que deve ser uma instância da classe *ResourceName*;
c) *setAttribute*: associa um novo atributo e seu respectivo valor a um recurso, ou, caso o atributo já exista, apenas atribui um novo valor. O recurso a ser alterado, o nome do atributo e seu valor devem ser passados por parâmetro;
d) *getAttribute*: recupera o valor de um atributo associado a um recurso através da primitiva anterior. O recurso e o nome do atributo devem ser passados por parâmetro. O retorno é o valor do atributo.

Este serviço foi útil de forma a complementar a funcionalidade do WORB. Como mencionado, havia ainda questões pendentes relativas à comunicação, que foram resolvidas com este serviço. Quanto à abstração da composição da URL, o CIB teve o papel de guardar o endereço da URL exportada como um atributo do nodo que fez a exportação. Dessa forma, logo após o serviço ser exportado, o nodo se registra como um recurso no CIB, através do *addResource*, e associa um atributo a si mesmo, através do *setAttribute*, cujo valor contém o endereço completo gerado na exportação. O nome deste atributo está relacionado com o identificador único do OX correspondente ao objeto exportado, sob o mesmo argumento de evitar conflitos de execuções simultâneas do mesmo padrão.

Dessa forma, não só a composição do endereço foi abstraída, mas também agora este foi publicado e está disponível para ser acessado por qualquer nodo da célula. Neste caso, está incluído o nodo que interessa, ou seja, o que está executando a *thread* de controle.

Como conseqüência, a *thread* de controle tem como obter o endereço do objeto remoto para fazer a chamada ao método. Como a lista de nodos foi passada por parâmetro na execução do padrão, a *thread* tem como converter qualquer nodo em recurso e sabe também sob qual nome o atributo foi registrado (definido em tempo de compilação). Então, basta consultar o CIB através do *getAttribute*, passando o nodo como recurso, e o nome do atributo. O retorno dessa chamada, o valor do atributo, conterá o endereço para ser feita a chamada do *lookupService*, eliminando assim os problemas restantes de comunicação. O CIB foi utilizado nos três padrões.

Por fim, o terceiro serviço utilizado é o serviço **Executor**. Este serviço faz parte do subsistema de execução distribuída e é responsável pelas funções de disparo de aplicações e criação e migração de objetos criados por estas. É sob o gerenciamento deste serviço que acontece a mobilidade de código provida pelo GRADEp.

O serviço Executor oferece uma API com métodos para manipulação de aplicações e de objetos no GRADEp. Objetos aqui se referem à abstração OX. Neste trabalho, foram usados apenas dois métodos de manipulação de objetos:

a) *createObject*: cria um objeto no GRADEp. A criação de um objeto através dessa primitiva resulta na criação de um OX que representará

um objeto Java dentro do GRADEp. São passados como parâmetros dessa primitiva a classe que se deseja instanciar e os argumentos a serem passados no construtor. Além destes, devem ser passados como parâmetros um objeto de ativação (subseção 5.6.2) e o nodo onde se deseja criar o objeto, que pode ser local ou remoto. O retorno é uma referência para a instância OX criada.

b) *moveObject*: muda a localização física de um OX. Esta primitiva permite que um objeto criado através do *createObject* seja migrado de um nodo para outro, mantendo referências e valores conforme discutido na seção 5.2.1. São parâmetros deste método o OX a ser migrado e o local de destino da migração.

O executor foi utilizado na criação de OXes nos três padrões deste *framework*. No Mmap, é criada remotamente em cada nodo da lista uma instância da classe que contém o método implementado pelo programador. Através do algoritmo de ativação, o objeto é exportado da forma explicada anteriormente, para ser posteriormente acessado. No Mfold e Mzipper, o *createObject* cria uma instância do procurador do agente, e através da classe de ativação, a execução local é feita e a primitiva *moveObject* é chamada, implementando a migração do agente segundo o comportamento do respectivo padrão.

Durante a fase de implementação, foi necessário um cuidado especial no uso da primitiva *moveObject*. A característica do GRADEp de não suportar mobilidade forte em sua totalidade, de não guardar o estado de execução, traz como conseqüência uma atenção especial por parte do programador com essa primitiva, de forma que ela só seja chamada após o término da execução do objeto a ser migrado, e que seja feita de uma *thread* independente daquele objeto. Foi observado nas aplicações um comportamento instável ao fazer a chamada de dentro do próprio objeto, ou quando a execução do mesmo ainda não estava completada. Como característica de programação do GRADEp, o programador fica responsável pela escolha adequada de quando o objeto será movido e qual *thread* fará a chamada, caso contrário, a chamada dessa primitiva pode resultar em um comportamento indesejado.

5.6.4 Criação de objetos no GRADEP: new X createObject

O fato de o *middleware* oferecer ao programador a abstração OX para a manipulação de objetos dentro do GRADEp se deve à necessidade de manipulação de objetos sob uma perspectiva pervasiva, ou seja, com suporte a mobilidade. Mas quando o programador não necessita de código móvel, o uso dessa abstração acaba por dificultar a manipulação dos objetos sob a perspectiva de acesso local. Isso se deve ao fato de que o OX funciona como meta-informação de um objeto Java, inicialmente não provendo acesso aos atributos e métodos do objeto referido, ficando ao encargo do programador de possibilitar esse acesso.

Como conseqüência, quando se quer acessar um método ou atributo do OX localmente, deve-se utilizar os serviços do GRADEp, da mesma forma como se fosse fazê-lo remotamente, pois os serviços foram construídos sob essa perspectiva. Por outro lado, o uso de OXes torna-se obrigatório se há o desejo de que o *middleware* tenha controle sobre os objetos, como por exemplo no caso de migração.

A criação de objetos Java com a cláusula *new*, permite que o acesso a métodos e atributos deste objeto seja feito da maneira comum, inerente à sintaxe da linguagem. Entretanto, tais objetos uma vez instanciados em um ambiente computacional serão ligados a ele pelo resto do seu ciclo de vida, permitindo apenas que cópias deste objeto sejam migradas no ambiente, e ainda assim não como OXes independentes, e sim como objetos referenciados por outro OX.

Estas questões implicaram em uma decisão à parte durante o desenvolvimento do *framework*, em relação à criação de objetos: a opção por *createObject* ou *new*.

6 TESTES E RESULTADOS

Este capítulo tem por objetivo descrever os testes feitos sobre o *framework* desenvolvido, bem como analisar seu funcionamento sobre o GRADEp. Serão abordados os requisitos necessários para que uma aplicação utilize o *framework*. Após, será descrito o ambiente onde os testes foram executados e então serão mostradas as aplicações implementadas. Por fim, será feita a demonstração dos resultados obtidos.

6.1 Observações e requisitos de uso

Durante o desenvolvimento do trabalho, foram observadas algumas características necessárias à aplicação que será desenvolvida sobre ele, para que o resultado obtido seja o esperado. Essas características serão expostas nesta seção sob a forma de observações e requisitos. Algumas delas foram herdadas do estilo de programação intrínseco ao GRADEp, outras dizem respeito o *framework* exclusivamente.

Conforme já discutido na seção 5.6, existe uma série de elementos que devem ser levados em consideração no desenvolvimento de aplicações sobre o GRADEp. Todas as observações discutidas naquela seção, como o funcionamento da entidade OX e a decisão por *createObject* ou *new*, por exemplo, também são válidas para o desenvolvimento de aplicações sobre o *framework*.

Além disso, foram identificados alguns requisitos extras. Detalhes pequenos, mas que se esquecidos podem gerar um esforço desnecessário.

O primeiro deles é que todas as classes necessárias à execução da aplicação desenvolvida devem estar no arquivo *jar* correspondente à mesma. Caso seja utilizada

uma ferramenta que inclua as classes do *framework* através de um arquivo diferente, este arquivo deve ser carregado em tempo de execução, e o código referente à carga é responsabilidade do programador.

Um outro requisito identificado é de que o programador deve estar atento ao uso de chamadas diretamente sobre a JVM. Um exemplo é o uso da chamada *System.exit()*. Como o GRADEp roda sobre a JVM, essa chamada não só termina a aplicação em execução como também mata o processo do GRADEp.

Em certos casos, houve um comportamento instável da primitiva *createObject,* quando, dentro do parâmetro relativo à lista de argumentos passados ao construtor, havia um ponteiro nulo (*null*). Portanto, na criação de objetos através do GRADEp, recomenda-se o uso de valores padrão ao invés de ponteiros nulos, quando localizados no parâmetro de argumentos. Essa questão reflete em pontos do *framework*, como por exemplo, o valor inicial do acumulador passado por parâmetro no disparo do Mfold. Se esse valor for nulo, o padrão não apresentará o resultado desejado, pois este mesmo valor é repassado como parâmetro do construtor do procurador do agente, resultando no problema recém descrito.

Basicamente, os requisitos específicos exigidos da aplicação para o uso do *framework*, é uma classe que herde a classe do código móvel do padrão de projeto utilizado e que seja feito o disparo do padrão. A classe que conterá o código móvel deve ser pública (modificador *public*), pois ela terá de ser carregada em um ambiente computacional remoto. Os demais detalhes foram abordados nos capítulos anteriores, sendo desnecessário repeti-los aqui.

6.2 Ambiente de testes

O GRADEp foi instalado e configurado em um ambiente virtual simulando quatro computadores ligados em rede entre si. Cada computador tem seu papel na arquitetura do *middleware*, definido nos arquivos de configuração. Cada um deles, independente de seu papel, ficou configurado com: processador turion64 (compartilhado), 96MB memória RAM, 300MB de disco rígido, sistema operacional Linux distribuição DSL (Damn Small Linux) (DSL, 2007) e Java JRE 1.5 (JAVA, 2007a).

Conforme explicado na seção 5.1.4, onde é abordado o ambiente físico do GRADEp, uma EXEHDAcel define um escopo de administração para uma EXEHDAbase, que é responsável por gerenciá-la. Os dispositivos de processamento comuns se dividem em EXEHDAnodos e EXEHDAnodos móveis, e pertencem a uma EXEHDAcel. O ambiente configurado para testes não preveu um EXEHDAnodo móvel, ficando então um computador configurado como EXEHDAbase e três configurados como EXEHDAnodos, compondo um EXEHDAcel. Visto que o desempenho não era um objetivo crítico do trabalho, este ambiente supriu as demandas necessárias.

Os arquivos de configuração utilizados, que definem os papéis das máquinas no ambiente, podem ser visualizados em (VIVAN, 2007).

6.3 Aplicações de teste

De acordo com Field et al. (2006), os domínios de aplicações que potencialmente podem se beneficiar de código móvel e padrões de mobilidade incluem documentos ativos, serviços de telecomunicações avançados, gerenciamento de rede, controle e configuração remota de dispositivos, gerenciamento e cooperação de *workflow*, redes ativas, computação em grade e global, e *e-commerce*. Para a execução de testes sobre o *framework* e aquisição de resultados, foi escolhido este último domínio, *e-commerce*.

Foram desenvolvidas três aplicações, de funcionalidades simples, com o propósito de extrair resultados apresentáveis sobre a viabilidade do *framework*.

6.3.1 Bases de dados utilizadas

As aplicações concebidas fazem parte da classe de aplicações de aquisição de informação distribuída. Ou seja, acessam bases de dados em nodos remotos adquirindo dados e ainda podendo processá-los remotamente. Para simular esse ambiente, foi colocada uma base de dados em cada nodo da rede.

As bases de dados foram modeladas com registros de três atributos: nome, que contém uma *string* com o nome completo do produto; valor, que contém o preço do produto, convertido para *string*; e a quantidade disponível no estoque, também convertida para *string*.

O motivo da conversão de todos os dados para *string* se deve ao fato de que eles estão sendo armazenados em arquivos texto. Como o objetivo das aplicações é demonstrar a viabilidade do *framework* proposto, foi desnecessário implementar uma base de dados sobre um SGBD (Sistema de Gerência de Banco de Dados). Essa abordagem não exclui a utilização de um código referente ao acesso a uma base de dados real. Uma vez que o comportamento dos padrões de projeto se apresente corretamente, a responsabilidade em gerenciar o acesso a essa base de dados fica implicitamente delegada ao *driver* correspondente àquela base.

6.3.2 Aplicação commerceMmap

Imaginando-se uma situação onde um usuário deseja fazer uma pesquisa de produtos, onde deseja obter uma relação completa de produtos, de várias lojas, que se encaixam em uma determinada descrição, a fim de examiná-los um a um antes de tomar a decisão final de compra. Para esta situação, o padrão Mmap atende os requisitos com naturalidade.

O *commerceMmap* faz uma consulta de produtos em vários nodos de uma rede de *e-commerce*, utilizando o padrão Mmap. Para isto, recebe como entrada a lista de nodos que devem ser examinados e uma *tag*, que representará qual produto deve ser procurado nos nodos. Através do Mmap, será feito um processamento nos nodos remotos a fim de obter a lista de produtos que satisfaz aquela *tag* em cada nodo. Após a obtenção dos resultados remotos, é feita a concatenação das listas, e como saída é mostrada ao usuário a relação dos produtos encontrados.

O critério de inclusão na lista de produtos encontrados é um teste se a *string* armazenada na *tag* está contida na *string* que representa o nome do produto. Dessa forma, se o usuário passar a *string* “forn”, serão incluídos nos resultados todos os produtos cujo nome contém a seqüência de caracteres “forn”, como por exemplo: “forno a gás rx4000”.

A lista de nodos de entrada deve seguir uma sintaxe definida no GRADEp, para a identificação dos mesmos. Cada nodo deve ser digitado com a constante “hostid:” como início da *string*, após, deve ser digitado o ID do nodo, seguido de um ponto, seguido do nome da célula. O ID e o nome da célula são os mesmos valores definidos

nas configurações dos perfis no GRADEp. Um nodo deve ser separado de outro por ponto e vírgula. A interface do programa já é inicializada com um exemplo de *tag* e de uma lista de nodos, como mostra a Fig 14. A Fig. 15 mostra a janela de resultado de uma consulta e na Fig. 16 pode-se visualizar o diagrama de classes desta aplicação.

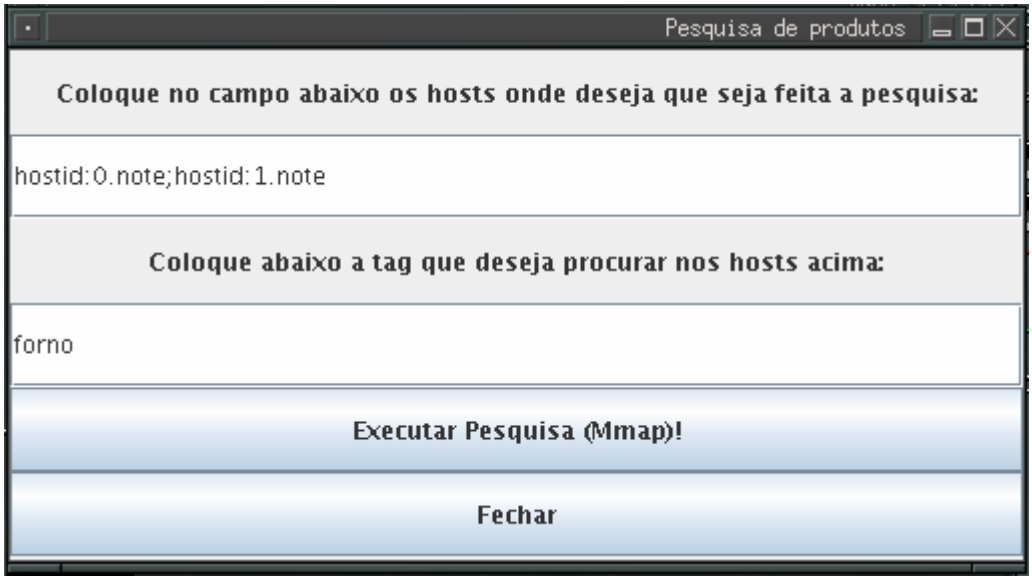

Figura 14 – Janela principal do *commerceMmap*

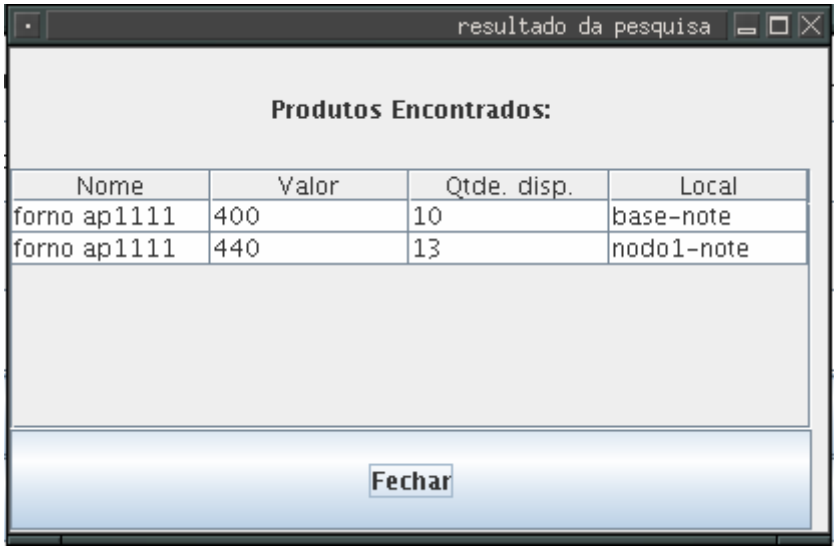

Figura 15 – Janela de resultado do *commerceMmap*

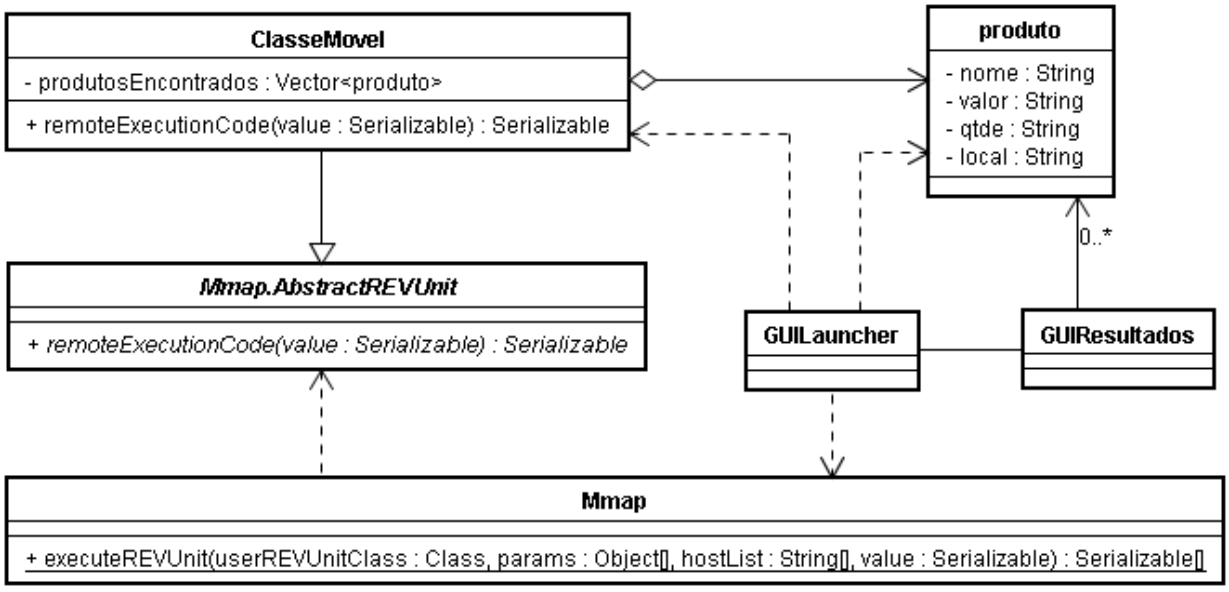

Figura 16 – Diagrama de classes do *commerceMmap*

Neste diagrama é possível observar a participação de duas classes do *framework*, *Mmap* e *Mmap.AbstractREVUnit*, caracterizando reuso de modelagem, objetivo parcial do *framework*. Estas duas classes cumprem o papel definido anteriormente no padrão de projeto Mmap.

A classe *produto* é a classe que representa o elemento chave da aplicação, sob a perspectiva do usuário. Instâncias dessa classe armazenam dados sobre produtos encontrados nas bases de dados da rede de *e-commerce*, modelados na subseção 6.3.1, com adição do local onde o produto se encontra.

A classe *ClasseMovel* representa a classe que contém o código móvel, incluindo o método que será executado nos nodos remotos. O código definido no *remoteExecutionCode* é uma busca na base de dados do nodo por elementos que se encaixam no critério de inclusão. Cada vez que um produto se encaixa, é inserido no atributo *produtosEncontrados*, e ao final da busca essa lista é retornada como resultado. A *tag* é passada como parâmetro neste método.

As classes *GUILauncher* e *GUIResultados* representam as interfaces gráficas para o usuário. A primeira é a janela principal do programa (Fig 14), e a segunda é a janela de demonstração do resultado da pesquisa (Fig. 15), sendo que a chamada ao método *Mmap.executeREVUnit* é feita na classe *GUILauncher*, como reação ao evento

de clique do botão de pesquisa. Após a execução, os resultados são concatenados em uma lista temporária, a qual é passada para a *GUIResultados*. Esta última recebe como entrada a lista final de produtos e exibe os itens dessa lista em uma tabela.

6.3.3 Aplicação commerceMfold

Imaginando-se uma situação onde um usuário não tem tempo para examinar produtos a fim de fazer a melhor escolha qualitativamente. Ele precisa de um processo automatizado sob um determinado critério quantitativo, que forneça para ele apenas o resultado final. Este usuário escolhe como critério o produto mais barato. O processo automatizado consiste em procurar o produto mais barato a partir de uma descrição. O padrão Mfold se demonstra o mais efetivo para atender aos requisitos desta situação. A procura pelo menor preço é uma aplicação típica de agentes na área de *e-commerce* (WAGNER; TURBAN, 2002; FIELD et al., 2006).

O *commerceMfold* faz uma busca do menor preço de um determinado produto, em uma lista de nodos de uma rede de *e-commerce*, utilizando o padrão Mfold. Assim como no *commerceMmap*, a entrada desta aplicação é a lista de nodos a serem verificados e uma *tag* especificando o produto. Conforme explicado no Mfold, o agente carrega consigo um acumulador (resultado intermediário), que será utilizado para armazenar o produto corrente de menor preço. Em cada nodo da lista, cada produto que satisfizer a *tag* terá seu preço comparado com o preço do produto contido no acumulador do agente, caso seja menor, o valor do acumulador é substituído pelo produto cujo preço é menor. Ao término do processamento no último nodo, o agente publica o resultado, e como saída é mostrado ao usuário o produto mais barato que satisfez a *tag* na lista de nodos passada por parâmetro. O critério de satisfação da *tag* e a sintaxe da lista de nodos são os mesmos descritos no *commerceMmap*. As Fig. 17 e Fig. 18 mostram as interfaces do usuário, principal e de resultados, respectivamente. Na Fig. 19 pode-se visualizar o diagrama de classes desta aplicação.

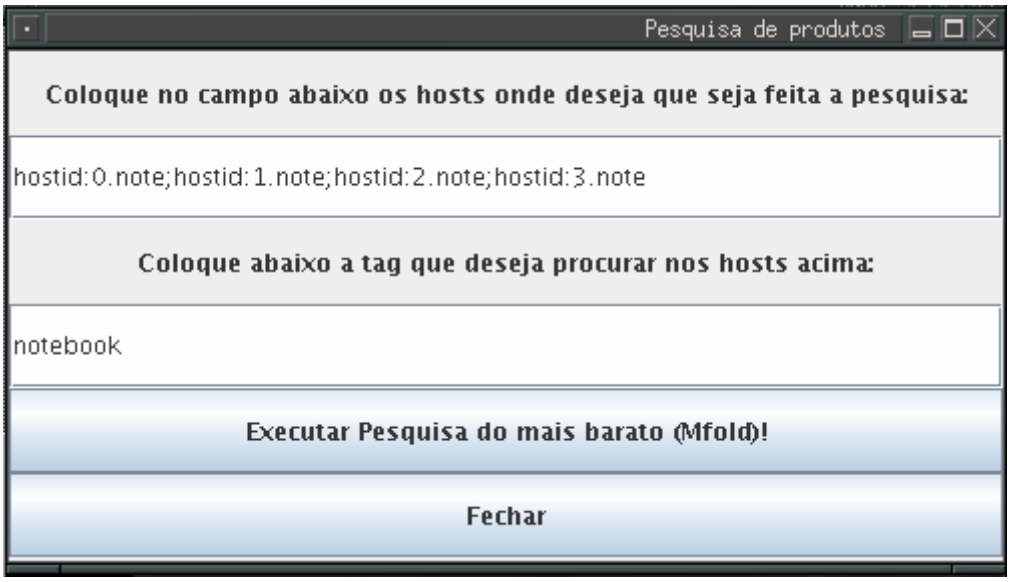

Figura 17 – Janela principal do *commerceMfold*

Figura 18 – Janela de resultado do *commerceMfold*

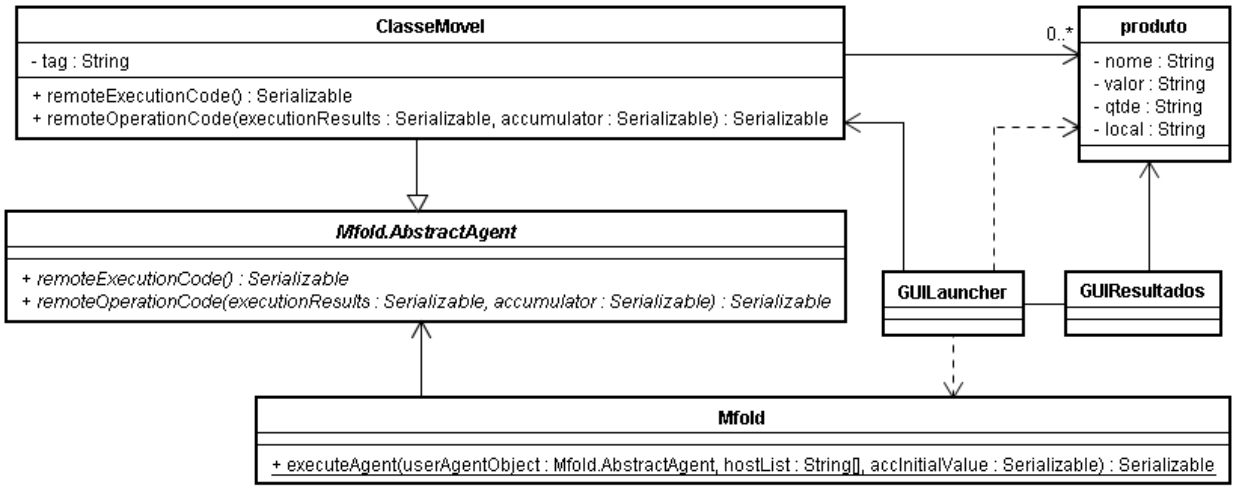

Figura 19 – Diagrama de classes do *commerceMfold*

Neste diagrama também é possível observar o reuso de modelagem provido pelo *framework*, com a participação das classes *Mfold* e *Mfold.AbstractAgent*. Ainda, a classe *produto* é exatamente a mesma usada no *commerceMmap* e as classes de interface gráfica foram baseadas nas classes usadas naquela aplicação.

As classes de interface gráfica, *GUILauncher* (Fig. 17) e *GUIResultados* (Fig. 18), foram devidamente alteradas para expressar a semântica correta do *commerceMfold*. Essas alterações incluem detalhes de interface, como os rótulos dos botões e mensagens. Ainda, na *GUILauncher* é feita a chamada do método *Mfold.executeAgent*, como reação ao evento de clique do botão de pesquisa. Não foi necessário fazer manipulação dos resultados, pois o processamento é feito localmente em cada nodo. Após a execução, o produto retornado do acumulador é passado como argumento para a *GUIResultados*. Esta última recebe como entrada apenas o produto mais barato encontrado e exibe-o em uma tabela.

A classe *ClasseMovel* representa a classe que contém o código móvel, incluindo os métodos que serão executados nos nodos remotos. O método *remoteExecutionCode* ficou responsável por fazer uma busca na base de dados do nodo por produtos que se encaixam no critério de satisfação. Igualmente ao método implementado no *commerceMmap*, ou seja, Cada vez que um produto se encaixa, é inserido em uma lista. Após a execução do método, essa lista é passada como argumento para o *remoteCombinationCode* juntamente com o acumulador do agente.

Este método percorre a lista testando se o preço do i-ésimo produto dessa lista é menor que o do acumulador, se sim, substitui o produto. Ao final da busca, o produto contido no acumulador será retornado como resultado.

6.3.4 Aplicação commerceMzipper

Esta terceira aplicação é uma versão mais sofisticada da *commerceMfold*. Sob o mesmo cenário proposto na aplicação anterior, pode-se criar um processo automatizado onde o agente não apenas buscará o menor preço, mas também tentará negociar esse preço, caso exista uma loja que faça uma oferta de venda melhor. Nessa situação, o comportamento de iteração do padrão Mzipper preenche os requisitos adequadamente. A negociação de compra através de agentes é uma aplicação típica em *e-commerce* (WAGNER; TURBAN, 2002).

O *commerceMzipper* faz uma busca do menor preço de um produto, tentando negociá-lo de acordo com os preços encontrados em outros nodos, utilizando o Mzipper. Assim como nas aplicações anteriores, a entrada desta é a lista de nodos e a *tag* de especificação do produto. Na primeira iteração, o Mzipper gerará um valor inicial no primeiro nodo da lista, neste caso, o valor é um produto que satisfaz a *tag*. A cada próximo nodo, o produto corrente contido no agente, tem seu preço comparado com produtos que satisfaçam a *tag* na base de dados do nodo. Caso o produto encontrado na base tenha preço inferior, ele é armazenado no agente para ser testado em uma nova iteração que é iniciada. Caso o preço seja maior, ainda ele é comparado com 10% de desconto. Caso o novo preço seja inferior, é armazenado e uma nova iteração é iniciada com o valor descontado. Se em nenhuma das oportunidades o preço foi menor, o nodo desiste de fazer oferta e passa o agente para o próximo da lista. Esse comportamento segue até que um preço gerado seja imbatível por todos os nodos, e então, o produto e seu respectivo preço e local onde o negócio foi fechado são as saídas mostradas ao usuário. O critério de satisfação da *tag* e a sintaxe da lista de nodos são os mesmos descritos nas aplicações anteriores.

Pode-se argumentar que o produto final retornado será sempre o mesmo que o retornado pelo *commerceMfold*. Isso é verdade, pois o critério de negociação é o mesmo para todos os nodos: ceder o preço em no máximo 10%. A diferença para o

usuário é que o preço retornado pode vir com 10% de desconto. Mas a intenção é acima de tudo exercitar o comportamento de iteração do Mzipper. Além disso, não se pode excluir a possibilidade de que este padrão seja usado para negociação entre agentes em um ambiente real, com critérios de negociação diferentes de loja para loja.

As Fig. 20 e Fig. 21 mostram as interfaces do usuário, principal e de resultados, respectivamente. Na Fig. 22 pode-se visualizar o diagrama de classes desta aplicação.

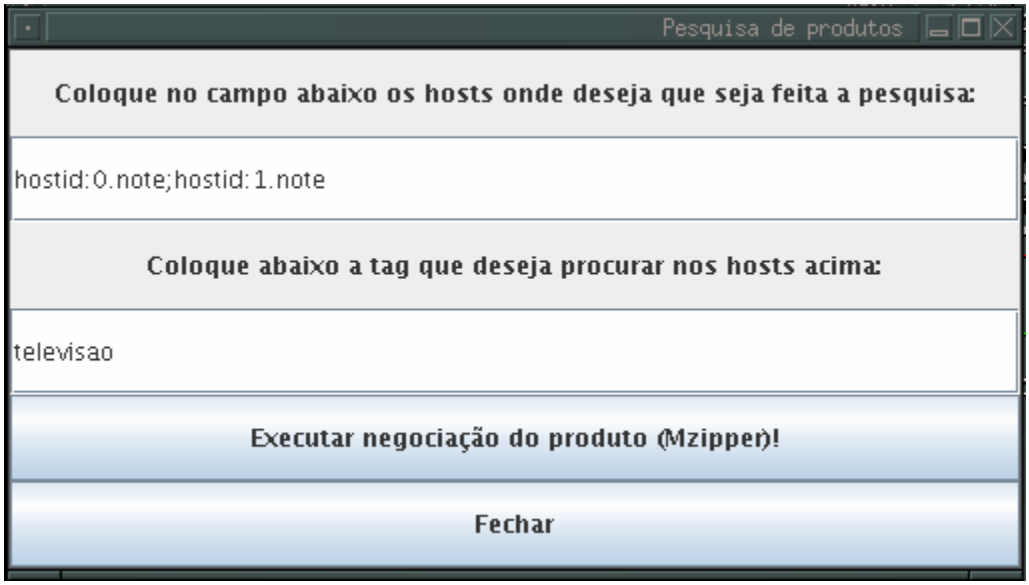

Figura 20 – Janela principal do *commerceMzipper*

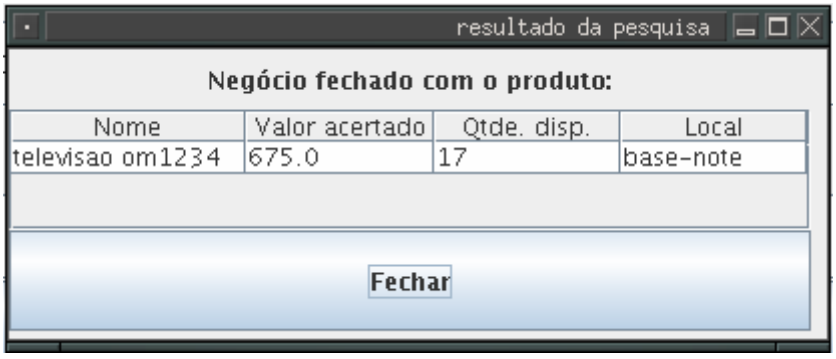

Figura 21 – Janela de resultado do *commerceMzipper*

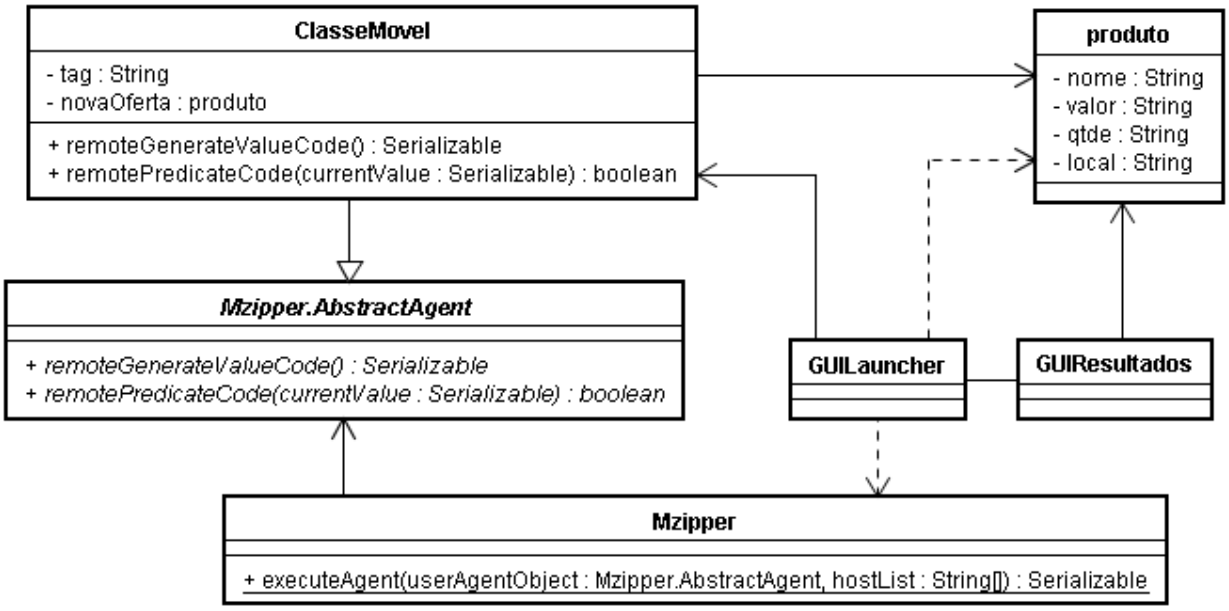

Figura 22 – Diagrama de classes do *commerceMzipper*

Devido à semelhança de comportamento entre o *commerceMfold* e o *commerceMzipper*, a modelagem do último foi baseada no primeiro. Neste diagrama é possível observar tanto o reuso de modelagem provido pelo *framework* quanto reuso em nível de aplicação. Ainda, a classe *produto* é exatamente a mesma usada no *commerceMmap* e no *commerceMfold*. As classes de interface gráfica foram baseadas nas classes usadas no *commerceMfold*.

As classes de interface gráfica, *GUILauncher* (Fig. 20) e *GUIResultados* (Fig. 21), tiveram apenas alterações em detalhes de interface, como os rótulos dos botões e mensagens, e na *GUILauncher*, alteração para o disparo do padrão correto, no caso, chamada para *Mzipper.executeAgent*.

A classe *ClasseMovel* representa a classe que contém o código móvel. O método *remotePredicateCode* (executado exceto no primeiro) ficou responsável por fazer uma busca na base de dados do nodo por produtos que se encaixam no critério de satisfação. Para cada produto assim encontrado ele compara o preço com o preço do produto corrente na iteração. Se possui o preço menor, o armazena em *novaOferta* e retorna *false*, que reinicia uma nova iteração. Se o preço for maior, ele faz o desconto de 10% para tentar ser menor. Se for, igualmente o armazena e reinicia uma iteração

com o preço descontado. Caso mesmo assim o preço não seja menor, retorna *true* e segue a iteração. O método *remoteGenerateValueCode* (executado apenas no primeiro nodo) testa se existe uma nova oferta. Se não, significa que é a primeira iteração, então acessa a base de dados do nodo pegando o primeiro produto que satisfaz a *tag*. Se existe uma nova oferta, ele usa o *remotePredicateCode* para saber se pode ou não cobrir essa oferta. Ao final da execução das iterações, o produto que tiver o preço mais baixo, descontado de 10% ou não, estará contido no valor corrente, sendo retornado como resultado.

6.4 Demonstração dos resultados

Os testes foram realizados de forma a executar as aplicações desenvolvidas com uma série de valores de entrada, e observando se os seus respectivos valores de saída são os esperados. Também foi possível acompanhar o comportamento dos padrões devido a mensagens de *log* colocadas no código.

6.4.1 População das bases de dados

As bases de dados de cada nodo no ambiente foram populadas com dados fictícios de produtos. As tab. 3, tab. 4, tab. 5 e tab. 6 representam os dados colocados nas bases de dados da EXEHDAbase, EXEHDAnodo 1, EXEHDAnodo 2 e EXEHDAnodo 3, respectivamente.

Tabela 3 – Base de dados da EXEHDAbase

Produto	**Valor**	**Quantidade disponível**
forno ap1111	*400*	*10*
geladeira rx2222	*900*	*10*
televisao om1234	*750*	*17*
aquecedor qr5800	*180*	*22*
celular re4444	*300*	*12*

Tabela 4 – Base de dados do EXEHDAnodo 1

Produto	**Valor**	**Quantidade disponível**
forno ap1111	*440*	*13*
televisao om1234	*800*	*20*
aquecedor xd3200	*230*	*30*

Produto	**Valor**	**Quantidade disponível**
notebook tw6666	*4500*	*3*

Tabela 5 – Base de dados do EXEHDAnodo 2

Produto	**Valor**	**Quantidade disponível**
aquecedor hf9800	*100*	*35*
notebook as7400	*2550*	*20*
televisao mw8800	*4000*	*5*
celular kf6446	*800*	*11*

Tabela 6 – Base de dados do EXEHDAnodo 3

Produto	**Valor**	**Quantidade disponível**
aquecedor xd3200	*280*	*26*
notebook aw5400	*2900*	*14*
forno wd1490	*670*	*13*
celular re4444	*330*	*9*
geladeira mx5555	*1200*	*7*

6.4.2 Testes executados e os respectivos resultados

Os testes foram executados na tentativa de cobrir o maior número de possibilidades possíveis. Dentre os casos de teste feitos, incluem-se: todos os nodos; alguns nodos; pesquisa local; pesquisa remota; repetição de nodos na lista; disparo de nodos diferentes (no caso dos baseados em agentes, é relevante); *tag* comum; *tag* em branco; *tag* desconhecida.

Quanto ao comportamento, os padrões se demonstraram estáveis em todos os casos. Quanto ao funcionamento, houve dois casos onde a execução da aplicação não foi bem sucedida.

O primeiro é quando uma das aplicações é disparada duas vezes de um mesmo nodo. Se uma aplicação é carregada e executa uma pesquisa, e logo em seguida a mesma aplicação é carregada e executa uma outra pesquisa, sem que a primeira tenha sido encerrada (duas instâncias da mesma aplicação carregadas na memória e com pesquisas executadas), no momento do disparo da segunda pesquisa, o *class loader* do GRADEp acusa conflito de classes (interface *RemoteIface* do padrão correspondente).

O segundo caso é quando duas aplicações diferentes são disparadas simultaneamente no mesmo nodo. Quando uma das aplicações é carregada e executa uma pesquisa, e uma segunda é carregada e executa outra pesquisa, sem que a primeira tenha sido finalizada, o *class loader* do GRADEp acusa que nenhuma definição da classe *RemoteIface* foi encontrada. Entre outras tentativas, também foi testado nomes diferentes para as interfaces, mas o problema não foi solucionado.

A título de ilustração, foram selecionados para este documento os casos mais simples, apresentados a seguir.

Teste 1: *commerceMmap* – busca em todos os nodos para uma *tag* qualquer. Os argumentos de entrada podem ser vistos na Fig. 23 e a saída na Fig. 24.

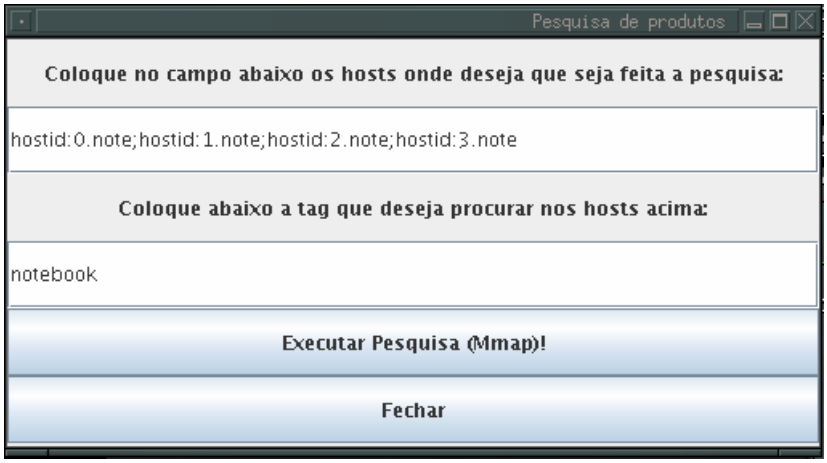

Figura 23 – Entrada do Teste 1

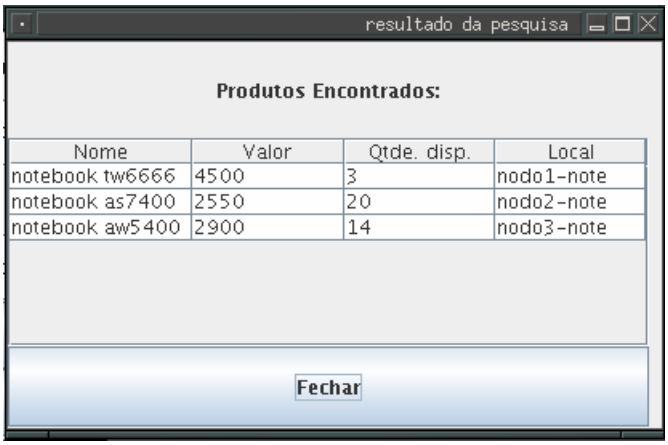

Figura 24 – Saída do Teste 1

Teste 2: *commerceMmap* – busca em um nodo para uma *tag* em branco (todos os produtos naquele nodo). Os argumentos de entrada podem ser vistos na Fig. 25 e a saída na Fig. 26.

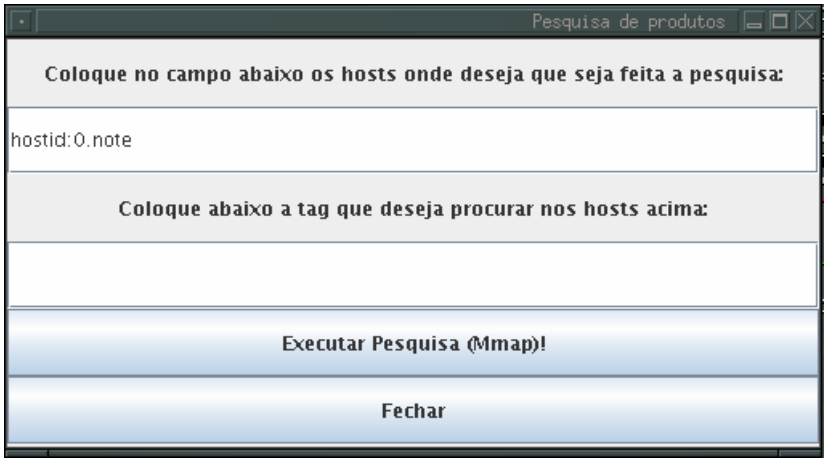

Figura 25 – Entrada do Teste 2

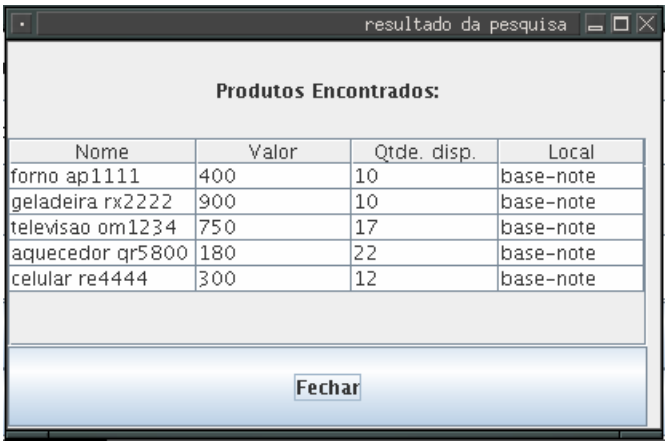

Figura 26 – Saída do Teste 2

Teste 3: *commerceMmap* – busca em uma quantidade qualquer de nodos para uma *tag* desconhecida. Os argumentos de entrada podem ser vistos na Fig. 27 e a saída na Fig. 28.

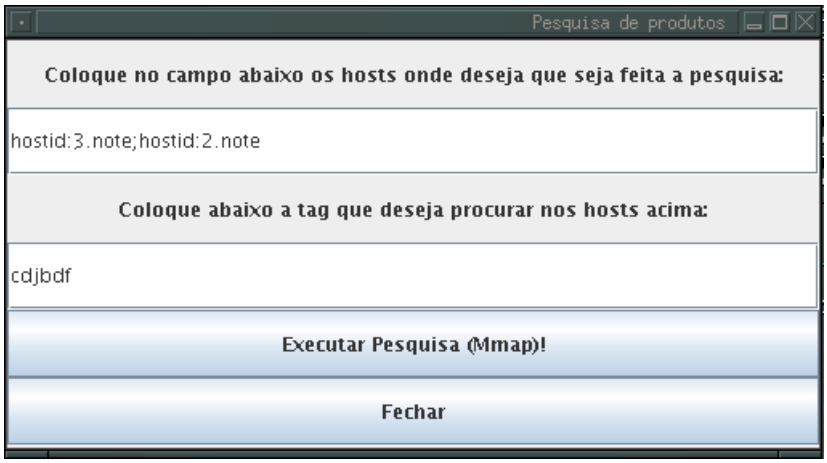

Figura 27 – Entrada do Teste 3

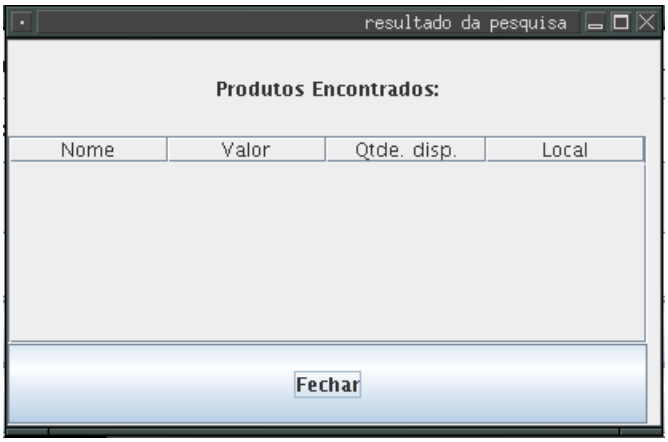

Figura 28 – Saída do Teste 3

Teste 4: *commerceMfold* – busca em todos os nodos para uma *tag* qualquer. Os argumentos de entrada podem ser vistos na Fig. 29 e a saída na Fig. 30.

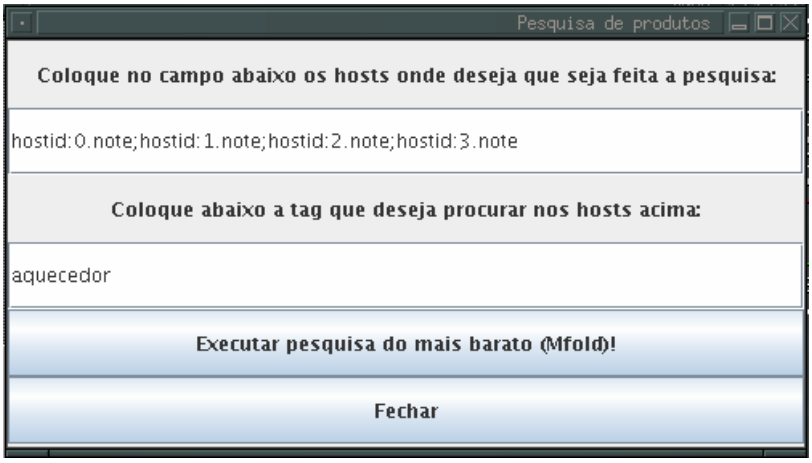

Figura 29 – Entrada do Teste 4

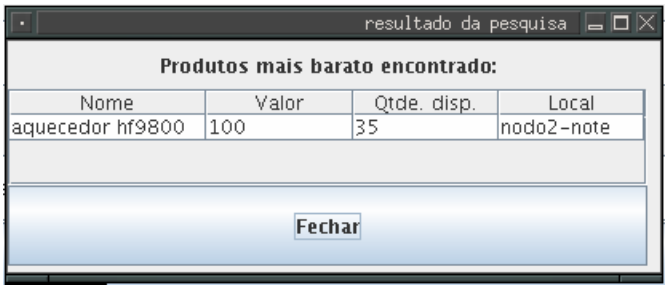

Figura 30 – Saída do Teste 4

Teste 5: *commerceMfold* – busca em um nodo para uma *tag* em branco (todos os produtos naquele nodo). Os argumentos de entrada podem ser vistos na Fig. 31 e a saída na Fig. 32.

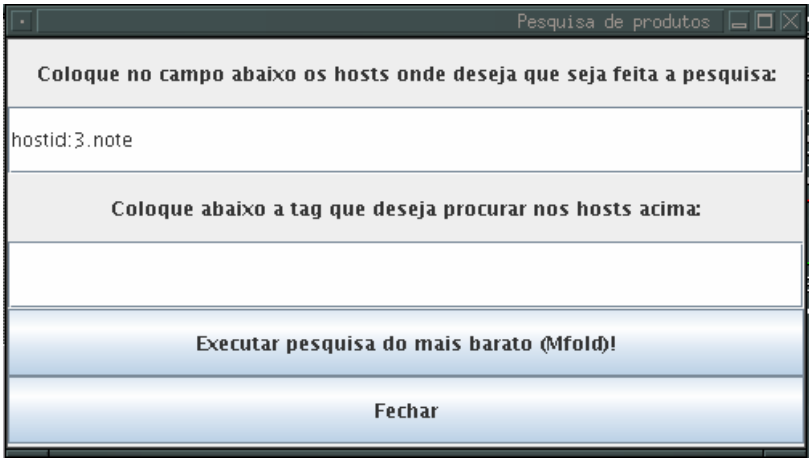

Figura 31 – Entrada do Teste 5

Figura 32 – Saída do Teste 5

Teste 6: *commerceMzipper* – busca em todos os nodos para uma *tag* qualquer (retorno de um produto sem desconto). Os argumentos de entrada podem ser vistos na Fig. 33 e a saída na Fig. 34.

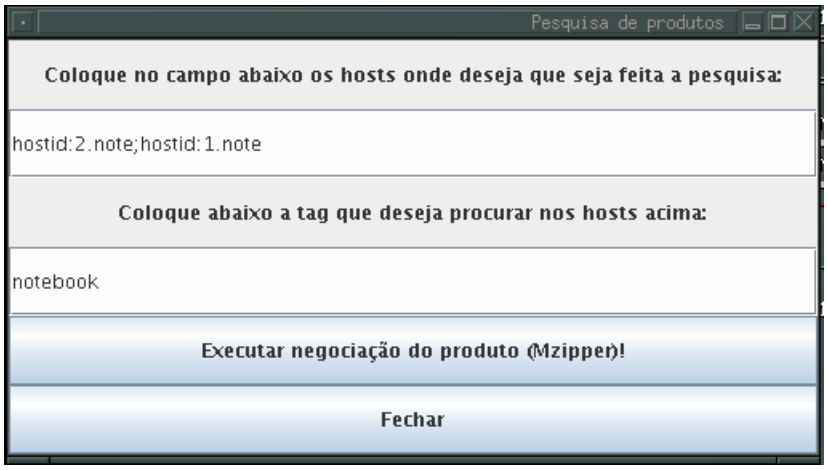

Figura 33 – Entrada do Teste 6

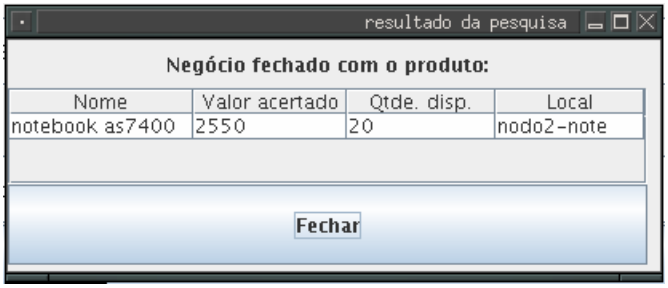

Figura 34 – Saída do Teste 6

Teste 7: *commerceMzipper* – busca em todos os nodos para uma *tag* qualquer (retorno de um produto com desconto). Os argumentos de entrada podem ser vistos na Fig. 35 e a saída na Fig. 36.

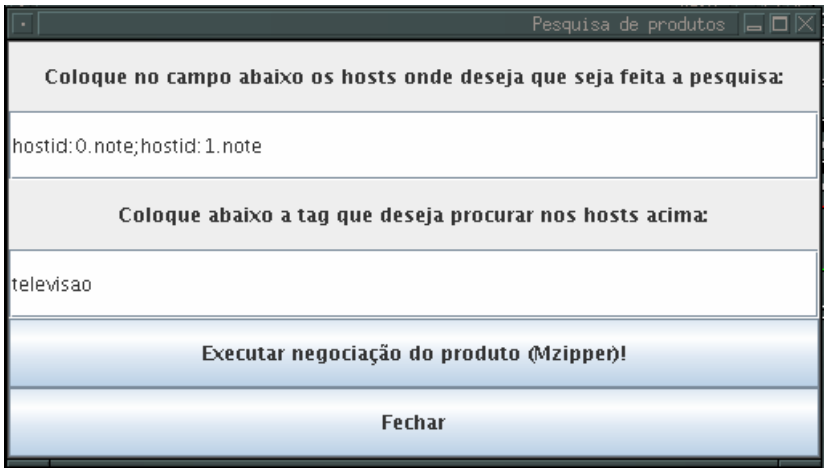

Figura 35 – Entrada do Teste 7

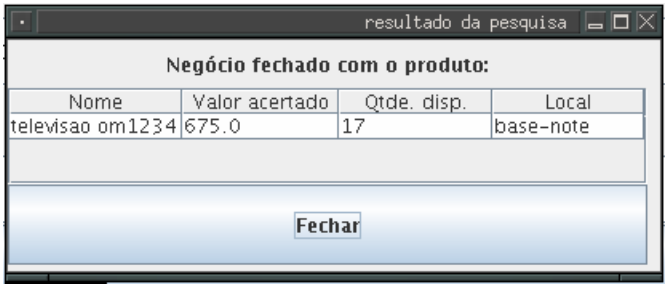

Figura 36 – Saída do Teste 7

7 CONCLUSÃO

Este trabalho propôs a utilização de padrões de projeto para o desenvolvimento de aplicações baseadas em mobilidade de código, no ambiente pervasivo GRADEp, através de um *framework* que implementa tais padrões.

Computação pervasiva é uma realidade iminente. Suas premissas voltadas para as necessidades dos usuários, na busca de abstraí-los do mundo computacional, favorecendo seu uso sem a necessidade de operação consciente, são campos de pesquisa férteis, cujos desafios valem a pena ser superados. Acredita-se que tal paradigma de computação definitivamente faça parte do futuro não apenas da própria computação, mas do cotidiano do usuário.

Dentre as características da computação pervasiva, a mobilidade de código destaca-se no sentido de ser uma funcionalidade de forte usabilidade. Os mecanismos providos por essa abordagem fornecem um controle maior sobre as aplicações, permitindo a implementação de comportamentos que na programação distribuída tradicional eram difíceis de expressar, comportamentos estes agora mais reincidentes na computação pervasiva. Entretanto, percebeu-se que, mesmo o *middleware* fornecendo os mecanismos de mobilidade, o tratamento da mesma requer alguns cuidados que na programação tradicional não são necessários, mas que, se gerenciados corretamente, trazem os benefícios desejados.

O emprego de padrões de projeto em mobilidade no desenvolvimento de tais aplicações tenta extrair os benefícios de ambas as abordagens, unindo a abstração provida pela programação distribuída tradicional com o controle provido pela mobilidade de código. Mas esse é um objetivo ideal. Em termos práticos, para se obter resultados

satisfatórios com essa união foi necessário que o escopo de utilização fosse reduzido, ou seja, apenas um determinado conjunto de aplicações podem se beneficiar de cada um dos padrões de projeto implementados.

Apesar da redução do conjunto de aplicações, dentro desse escopo as funcionalidades oferecidas demonstraram-se eficientes. Além disso, suas execuções apresentaram-se estáveis e os comportamentos propostos pelos padrões foram reproduzidos com sucesso.

Durante o desenvolvimento, o *middleware* GRADEp apresentou-se funcional e estável. Mas por ser um projeto recente e ainda em desenvolvimento, alguns serviços ainda não estão implementados em sua totalidade, o que limitou o leque de recursos utilizados para implementação. Ainda, a documentação disponível não é muito detalhada, o que refletiu em um esforço maior para descobrir certas particularidades do *middleware*. Em contraponto, destaca-se a convivência com a comunidade do GRADEp, o que atalhou o desenvolvimento em alguns momentos críticos.

Apesar das dificuldades encontradas, acredita-se que através dos resultados obtidos, os objetivos foram atingidos de forma satisfatória. Acredita-se também que este trabalho terá uma potencial contribuição não apenas como uma ferramenta alternativa para os desenvolvedores de aplicações sobre o GRADEp, mas também em trabalhos futuros, uma vez que além de código foi produzida também uma documentação relativamente detalhada sobre alguns aspectos relacionados ao desenvolvimento de aplicações sobre o GRADEp.

7.1 Trabalhos futuros

Como sugestão para trabalhos futuros pode-se propor a adição ao *framework* da capacidade de trabalhar com decisões adaptativas em função do contexto da aplicação distribuída. A exploração da adaptabilidade ao contexto pode incrementar significativamente o *framework* em funcionalidades e também desempenho. O suporte para isto pode ser construído utilizando os serviços já existentes do *middleware* GRADEp.

Outra oportunidade de trabalho futuro identificada diz respeito ao aprimoramento do *framework* proposto, utilizando-se um serviço diferente de

comunicação no GRADEp, como o CC-Manager. No pequeno conjunto de casos onde o funcionamento não foi bem sucedido, o problema estava relacionado com uma interface necessária para o serviço de comunicação utilizado, o WORB. O serviço CC-manager é baseado em uma abstração diferente, de espaço de tuplas, os quais não requerem declaração de uma interface explícita, podendo não apenas resolver o problema de funcionamento como também otimizar o desempenho do mesmo.

REFERÊNCIAS

APPLETON, Brad. **Patterns and Software: Essential Concepts and Terminology**. Disponível em: <http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html>. Acesso em: mai. 2007.

ARIDOR, Yariv; LANGE, Danny. Agent Design Patterns: Elements of agent application design. In: SECOND INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 1998. **Anais do...** Minneapolis, EUA, 1998. p.108–115.

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

AURA, projeto. Disponível em: <http://www.cs.cmu.edu/~aura/>. Acesso em: mai. 2007.

BOIS, André Du; TRINDER, Phil; LOIDL, Hans-Wolfgang. Towards Mobility Skeletons. **Parallel Processing Letters**, v.15, n.3, p.273–288, 2005.

BOIS, André Du; TRINDER, Phil; LOIDL, Hans-Wolfgang. mHaskell: Mobile Computation in a Purely Functional Language. In: Haskell Workshop, 2004, Snowbird, EUA. Enviado.

CARZANIGA, Antonio; PICCO, Gian; VIGNA, Giovanni. Designing Distributed Applications with Mobile Code Paradigms. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 19, 1997, Boston, EUA. **Anais do...** p.22–32.

DSL, Project. Disponível em: <www.damnsmalllinux.org>. Acesso em: jul. 2007.

FIELD, Zara; DEWAR, Rick; TRINDER, Phil; BOIS, André Du. Two Executable Mobility Design Patterns: mfold and mmap. Pattern Languages of Programming Conference, Portland, EUA, 2006. **Anais do...**

FUGGETTA, Alfonso; PICCO, Gian; VIGNA, Giovanni. Understanding Code Mobility. **IEEE Transactions on Software Engineering**, v.24, n.5, p.342–361, 1998.

GAIA, projeto. Disponível em: <http://gaia.cs.uiuc.edu/>. Acesso em: mai. 2007.

GALL, Harald; KLOSCH, René; MITTERMEIR, Roland. Application patterns in re-engineering: Identifying and using reusable concepts. In: INTERNATIONAL CONFERENCE ON INFORMATION PROCESSING AND MANAGEMENT OF UNCERTAINTY IN KNOWLEGDE-BASED SYSTEMS, 6, 1996. **Anais do...** p.1099–1106.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns**: Elements of Reusable Object-Oriented Software**.** Addison Wesley, 1995.

GARLAN, David; STEENKISTE, Peter; SCHMERL, Bradley. Project Aura: Toward Distraction-free Pervasive Computing. **IEEE Pervasive Computing,** v.1, n.3, 2002.

GEYER, Cláudio F.R. Computação em Grade Pervasiva: treinamento no GRADEp. RNP, 2005. 58p.

GRIMM, Robert et al. Systems Directions for Pervasive Computing. In: WORKSHOP ON HOT TOPICS IN OPERATING SYSTEMS, 8, 2001. **Anais do . . .** IEEE Computer Society, 2001. p.147–151.

ISAM, projeto. Disponível em: <http://www.inf.ufrgs.br/~isam/>. Acesso em: mai. 2007.

JAVA, Linguagem. Disponível em: <java.sun.com>. Acesso em: mai. 2007.

JAVA, Tecnologia. Disponível em: <http://www.java.com/pt_BR/about/>. Acesso em: mai. 2007.

JOHNSON, Ralph; FOOTE, Brian. Designing reusable classes. **Journal of Object-Oriented Programming**, v.1, n.2, 1988, p.22–35. Disponível em: <http://www.laputan.org/drc/drc.html>. Acesso em: jul. 2007.

LIMA, Emerson; MACHADO, Patrícia; SAMPAIO, Flávio; FIGUEIREDO, Jorge. An approach to modelling and applying mobile agent design patterns. **ACM Software Engineering Notes**, v.29, n.4, 2004.

OXYGEN, projeto. Disponível em: <http://oxygen.csail.mit.edu/>. Acesso em: mai. 2007.

SAHA, Debashis; MUKHERJEE, Amitava. Pervasive Computing: A Paradigm for the 21st Century. **IEEE Computer**, v.36, n.3, 2003, p.25–31.

SATYANARAYANAN, Mahadev. Pervasive Computing: Vision and Challenges. **IEEE Personal Communications**, 2001, p10-17.

STORCH, Mauro; BOIS, André Du; YAMIN, Adenauer. Padrões de Projeto para Computação Móvel. In: 6th LATIN AMERICA CONFERENCE ON PATTERN LANGUAGES OF PROGRAMMING, 2007, Porto de Galinhas. **Anais do...** 2007.

TANENBAUM, Andrew. **Distributed Operating Systems**. Prentice Hall, 1995. 648p.

VIVAN, Giulian, G. Uso de Padrões de Projeto no Desenvolvimento de Aplicações Baseadas em Mobilidade de Código na Computação Pervasiva. Disponível em: <http://percom.wkit.com.br/doku.php?id=percom:padroes>. Acesso em: ago. 2007.

VOYAGER, *framework*. Disponível em <www.recursionsw.com/Products/voyager.html>. Acesso em: jul. 2007.

WAGNER, Christian; TURBAN, Efraim. Are intelligent e-commerce agents partners or predators? **Communications of ACM**, v.45, n.5, 2002, p.84-90.

WEISER, Mark. The Computer for the 21st Century. Scientific Am., 1991, p.94-104. Disponível em: <http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html>. Acesso em: mai. 2007.

WEISER, Mark; GOLD, Rich; BROWN, John. The origins of ubiquitous computing research at PARC in the late 1980s. **IBM System Journal: Pervasive Computing**, v.38, n.4, 1999, p.693-696. Disponível em: <http://www.research.ibm.com/journal/sj/384/weiser.html>. Acesso em: mai. 2007.

YAMIN, Adenauer. **Arquitetura para um Ambiente de Grade Computacional Direcionado as Aplicações Distribuídas Móveis e Conscientes do Contexto da Computação Pervasiva**. 2004. 194f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

YAMIN, Adenauer; AUGUSTIN, Iara; SILVA, Luciano; REAL, Rodrigo; BARBOSA, Jorge; GEYER, Cláudio. ISAM: Uma Arquitetura de Software para Pervasive Computing. Conferencia Latinoamericano de Informatica. Arequipa, Peru. **Anais do...** 2004, p.347-358.

YAMIN, Adenauer; AUGUSTIN, Iara; BARBOSA, Jorge; SILVA, Luciano da; REAL, Rodrigo; CAVALHEIRO, Gerson; GEYER, Cláudio. Towards Merging Context-aware, Mobile and Grid Computing. In: INTERNATIONAL JOURNAL OF HIGH PERFORMANCE COMPUTING APPLICATIONS, 2003, Londres. **Anais do...** 2003. p.191–203.