Universidade Federal de Goiás
Instituto de Informática

Alaor José da Silva Junior

ContextGrid: Uma infra-estrutura de suporte a contexto para integração de dispositivos móveis à computação em grade

Goiânia
2006

Alaor José da Silva Junior

ContextGrid: Uma infra-estrutura de suporte a contexto para integração de dispositivos móveis à computação em grade

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concetração: Redes e Sistemas Distribuídos

Orientador: Prof. Márcio Nunes de Miranda

Co–Orientador: Prof. Fábio Moreira Costa

Goiânia
2006

Alaor José da Silva Junior

ContextGrid: Uma infra-estrutura de suporte a contexto para integração de dispositivos móveis à computação em grade

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Ciência da Computação, aprovada em 17 de Julho de 2006, pela Banca Examinadora constituída pelos professores:

Prof. Márcio Nunes de Miranda
Instituto de Informática – UFG
Orientador

Prof. Fábio Moreira Costa
Instituto de Informática – UFG

Prof. Markus Endler
Instituto de Informática – PUC Rio

Prof. Sibélius Lellis Vieira
Departamento de Computação – UCG

Alaor José da Silva Junior

Graduou–se em Ciência da Computação na UFG - Universidade Federal de Goiás. Durante sua graduação, desenvolveu um trabalho sobre *cluster computing*. Mais recentemente no Mestrado, na UFG - Universidade Federal de Goiás, vem desenvolvendo trabalhos relacionados a grades computacionais e contexto. Atualmente desenvolve soluções para sistemas distribuídos, mais especificamente grades computacionais, e computação sensível ao contexto.

Ficha Catalográfica

da Silva Junior, Alaor José

ContextGrid: Uma infra-estrutura de suporte a contexto para integração de dispositivos móveis à computação em grade/ Alaor José da Silva Junior; orientador: Márcio Nunes de Miranda; co–orientador: Fábio Moreira Costa. — Goiânia : UFG, Instituto de Informática, 2006.

[13], 94f.: il. ; 29.7cm

Dissertação (mestrado) - Universidade Federal de Goiás, Instituto de Informática.

Inclui referências bibliográficas.

1. Grade Computacional. 2. Adaptação Dinâmica. 3. Computação Sensível ao Contexto. 4. Computação Móvel. I. de Miranda, Márcio Nudes. II. Costa, Fábio Moreira. III. Universidade Federal de Goiás. Instituto de Informática. IV. Título.

CDU: 004.75

Dedico esta dissertação a meus pais, cujo exemplo de honestidade e trabalho tem sido um norteador para a minha vida, e para minha futura esposa, que tem me dado apoio nos momentos mais difíceis e mostrado a simplicidade de ter esperança.

Agradecimentos

Em primeiro lugar a Deus, que me dá forças e me ilumina. Aos meus pais, Alaor e Maria, pelo apoio que foi fundamental não só durante este trabalho, mas durante toda a minha vida. A todos os meus familiares que sempre torcem por mim.

À minha noiva, Cristiane, em virtude de sua compreensão diante dos muitos momentos roubados da nossa convivência e por ter me ouvido nos momentos de desânimo e angústia. Muito obrigado.

Aos meus orientadores Fábio e Márcio, pela paciência e dedicação na orientação deste trabalho e por terem acreditado nele. Obrigado pelas experiências compartilhadas, elas serão de muita importância para toda minha vida.

Aos amigos de longa data e os conquistados durante o mestrado: Aníbal, André, Joelma, Karla, Fernando. Agradeço pelo incentivo, apoio, momentos de distração e auxílios prestados durante todo este trabalho.

Aos amigos do Laboratório de Internet e Sistemas Distribuídos: Marcelo, Jesus e Daniel. Pela ajuda na manutenção do laboratório e pelas instalações e configurações infindáveis.

À estimada Iara Barreto, amiga e grande orientadora, que teve papel decisivo para o inicio dessa jornada e durante os momentos mais críticos. Obrigado pelo apoio incondicional e por acreditar em mim.

Aos colegas do Instituto de Informática da UFG pela compreensão e presteza.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo financiamento do projeto FlexiGrid (Edital 031/2004, Processo 506689/2004-2) o que permitiu aquisição da infra-estrutura necessária a realização deste trabalho.

Algo só é impossível, até que alguém duvide e acabe provando o contrário.

Albert Einstein,
.

Resumo

da Silva Junior, Alaor José. **ContextGrid: Uma infra-estrutura de suporte a contexto para integração de dispositivos móveis à computação em grade**. Goiânia, 2006. 106p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

A integração de dispositivos móveis à computação em grade representa um novo cenário para o uso desses dispositivos. Esse cenário introduz uma série de novos conceitos e desafios que precisam ser investigados, detalhados e transpostos, visando uma integração transparente ao usuário e ampliando as aplicações das grades computacionais.

O ContextGrid é uma infra-estrutura que permite que o *middleware* de grade explore o uso de contexto para realizar adaptações de acordo com o estado atual da grade. Ele coleta, interpreta e notifica o contexto aos diversos componentes do *middleware*. O ContextGrid atua juntamente com o *middleware* de grade MAG/InteGrade, estendendo alguns conceitos deste. Essa atuação conjunta permite que o contexto seja considerado na execução de aplicações, diversificando as políticas de escalonamento de tarefas que podem ser adotadas. Fazendo uso da tecnologia de agentes móveis, disponível no MAG/InteGrade, o ContextGrid permite a realocação de tarefas cientes de contexto. Baseado no contexto e de forma dinâmica, o *middleware* pode fazer o remanejamento de tarefas que estão executando em um nó de grade com características desfavoráveis ou realizar um balanceamento de carga. Os componentes do *middleware* também podem fazer uso do contexto para se adaptarem. Nesse aspecto, tem especial destaque o componente de *proxy*, que explora o uso de contexto de forma a conseguir uma melhor integração dos dispositivos móveis à computação em grade.

O trabalho aqui desenvolvido consiste do estudo de contexto e adaptação para exploração em computação em grade, da elaboração de um modelo de contexto para este ambiente e da descrição da arquitetura do ContextGrid, passando pela definição de seus componentes, implementação e avaliação de desempenho em especial para dispositivos móveis.

Palavras–chave

Grade Computacional, Contexto, Adaptação, Computaçao Móvel.

Abstract

da Silva Junior, Alaor José. **ContextGrid – An Infrastructure for Context Support for Integration of Mobile Devices into the Computational Grid**. Goiânia, 2006. 106p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

The integration of mobile devices into grid computing represents a novel scenario for the use of such devices. This scenario introduces a series of new concepts and challenges that need to be investigated, detailed and overcomed, aiming transparent integration for the user and broadening the applications of grid computing.

ContextGrid is an infrastructure that allows grid middleware to exploit the use of context to perform adaptations according to the current state of the grid. It collects, interprets and notifies context information to the several components of the middleware. ContextGrid acts as an extension of the MAG/InteGrade grid middleware. This extension allows context to be considered in the execution of applications, contributing to the diversity of scheduling policies that can be adopted by the middleware. Using mobile agent technology, as proposed in the MAG/InteGrade middleware, ContextGrid allows the context-aware reallocation of tasks. In this way, based on context and in a dynamic fashion, the middleware can reallocate a group of tasks running in an unfavorable grid node. This feature can also be used as a means to perform load balancing. The middleware components can also use context for their own adaptation. In this regard, special emphasis has been given to proxy components, which explore the use of context in such a way as to promote better integration of the mobile devices into the grid.

This work presents a study of the concepts of context and adaptation and explores their use in grid computing. A model is proposed for the representation and communication of context information in grids that include mobile devices, along with the architecture for the infrastructure that handles context information in such an environment. The architecture and its components have been implemented and evaluated concerning its use in mobile devices.

Keywords

Computational Grid, Context, Adaptation, Mobile Computing.

Sumário

Lista de Figuras **11**

Lista de Tabelas **13**

1 Introdução **14**
- 1.1 Motivação 17
- 1.2 Objetivos 17

2 Fundamentos **19**
- 2.1 Contexto e Adaptação 19
  - 2.1.1 Interpretação do Contexto 23
  - 2.1.2 Modelagem de Contexto 24
- 2.2 Computação em Grades 27
  - 2.2.1 Globus 27
  - 2.2.2 Condor 31
  - 2.2.3 InteGrade 32
  - 2.2.4 MAG 36
- 2.3 Considerações 38

3 Trabalhos Relacionados **40**
- 3.1 EXEHDA 40
  - 3.1.1 Arquitetura 42
  - 3.1.2 Comparações 44
- 3.2 MoCA 44
  - 3.2.1 Arquitetura 45
  - 3.2.2 Comparações 47
- 3.3 Solar 47
  - 3.3.1 Arquitetura 49
  - 3.3.2 Comparações 50
- 3.4 Considerações 51

4 ContextGrid **52**
- 4.1 Visão Geral 52
- 4.2 Arquitetura 53
- 4.3 Componentes 54
- 4.4 Contexto no ContextGrid 56
  - 4.4.1 Características da Informação Contextual 57
  - 4.4.2 Modelagem do contexto para o ContextGrid 58
    - Notação 58

O Modelo 61
4.4.3 Interpretação do Contexto 66
4.5 Protocolos 68
4.5.1 Protocolo de Atualização de Contexto 68
4.5.2 Protocolo de Notificação de Contexto 69
Inscrição em um Canal de Eventos 70
Notificação de Mudanças no Contexto 70
4.6 Canais de Eventos 72
4.7 Implementação 73
4.7.1 Suporte a Contexto e Notificação 74
IcontextGridSubject 75
IcontextGridObserver 75
4.7.2 LCM 75
4.7.3 GCM 79
4.7.4 *Wrappers* 83
4.7.5 *Proxy* 84
4.7.6 Context Users 87
GRM - Escalonamento consciente de contexto 87
AgentHandler - Migração de tarefas consciente de contexto 88
4.8 Comentários 89

5 Resultados 90
5.1 Avaliação do consumo de CPU causado pelo LCM 90
5.1.1 Celular 91
5.1.2 PocketPC 91
5.1.3 Nó fixo 92
5.2 Cenário de Uso 92
5.3 Comentários 94

6 Conclusões e Trabalhos Futuros 95

Referências Bibliográficas 99

A Especificação dos dispositivos utilizados 106

Lista de Figuras

1.1 Taxonomia de Grades [44] 15

2.1 Modelo de aplicações sensíveis ao contexto 21
2.2 Localização de médicos no hospital [10] 22
2.3 Context Toolkit [21] 24
2.4 Evolução: Computação Distribuída - Grades em Computação Ubíqua 26
2.5 Componentes do GlobusToolkit 28
2.6 Globus Toolkit 3 29
2.7 OGSI para WSRF 30
2.8 Arquitetura de um *Condor Pool* 32
2.9 Arquitetura de um Aglomerado InteGrade 33
2.10 Nova Arquitetura do Integrade 35
2.11 Arquitetura MAG em camadas [47] 37
2.12 Dados manipulados pelo ExecutionManagementAgent 39

3.1 Visão geral do ISAMadapt e EXEHDA 41
3.2 EXEHDA no Projeto ISAM 42
3.3 ISAMpe 43
3.4 Interação entre os Serviços da Arquitetura MoCA 46
3.5 Exemplo de um Grafo de Operadores 49
3.6 Aquitetura do Solar 49

4.1 Visão geral do ContextGrid 53
4.2 Arquitetura em camadas do ContextGrid 54
4.3 Componentes do ContextGrid 55
4.4 Modelo de Contexto do ContextGrid 60
4.5 Protocolo de Atualização de Contexto 69
4.6 Protocolo de Disseminação de Contexto - Subscribe 70
4.7 Protocolo de Disseminação de Contexto - Notify 71
4.8 Interfaces de Suporte ao Contexto e Notificações 74
4.9 Diagrama de classes para o LCM 77
4.10 Informações de contexto transformadas pelo LCM 78
4.11 IDL do LCM 79
4.12 Contexto produzido pelo GCM 80
4.13 Diagrama de classes para o GCM 82
4.14 IDL do GCM 83
4.15 Proxy do ContextGrid 86
4.16 Context User GRM 87
4.17 Context User AgentHandler 88

5.1 Consumo de CPU do LCM - Celular 91
5.2 Consumo de CPU do LCM - Nó Fixo 2 92

Lista de Tabelas

2.1 Avaliação - Abordagens para Modelagem de Contexto 26

4.1 Associações inter-entidade 67
4.2 Hierarquia de canais de eventos 72

CAPÍTULO 1

Introdução

Desde o início da década de 90 observa-se que, ao contrário do passado, o custo de uma sofisticada máquina paralela com alto poder computacional é muito mais alto do que o de uma rede local de computadores de pequeno porte com poder computacional equivalente.

O desenvolvimento das tecnologias de redes de computadores tem permitido a exploração de diferentes soluções para obtenção de alto poder computacional. Trabalhos de destaque têm surgido desde então como, por exemplo, as arquiteturas em *cluster* [8], onde vários computadores dedicados são interconectados por alguma tecnologia de rede de computadores e trabalham juntos para resolver um problema.

Durante a *Supercomputing´95* foi estabelecida a I-WAY [27], que conecta supercomputadores de 17 laboratórios em um único sistema que pode ser utilizado por qualquer um dos 17 participantes. Esse é considerado o primeiro sistema de um novo paradigma, o da computação em grade [29]. Um sistema de computação em grade (*grid computing*) é uma infra-estrutura que explora as potencialidades das redes de computadores e *software* para interligar e gerenciar uma grande variedade de recursos computacionais, em geral distribuídos, e oferecê-los de forma transparente ao usuário. O nome grade (*grid*) é uma analogia às malhas de interligação do sistema de energia elétrica (*power grids*) e traduz a vontade de tornar o uso dos recursos computacionais tão transparente quanto o uso da eletricidade.

Com o crescente aumento do parque computacional instalado em instituições públicas e privadas, o uso de grades computacionais vem se tornando cada vez mais interessante como forma de racionalizar e aproveitar todo esse poder computacional, que na maior parte do tempo é subutilizado. As grades ainda possibilita que as instituições passem a dispor de uma ferramenta (a grade) para resolver problemas computacionais complexos.

Tradicionalmente, as grades são usadas para compartilhar apenas recursos de processamento. Porém, novas formas de recursos estão sendo exploradas e discutidas, como espaço de armazenamento e os chamados laboratórios virtuais [9], onde os recursos são instrumentos científicos especializados. Com o aumento na

diversificação dos recursos, Krauter *et al* [44] introduziram em 2002 uma proposta de taxonomia [44] que categoriza os sistemas de computação em grade conforme sua principal atividade. Sua classificação poder ser vista na Figura 1.1:

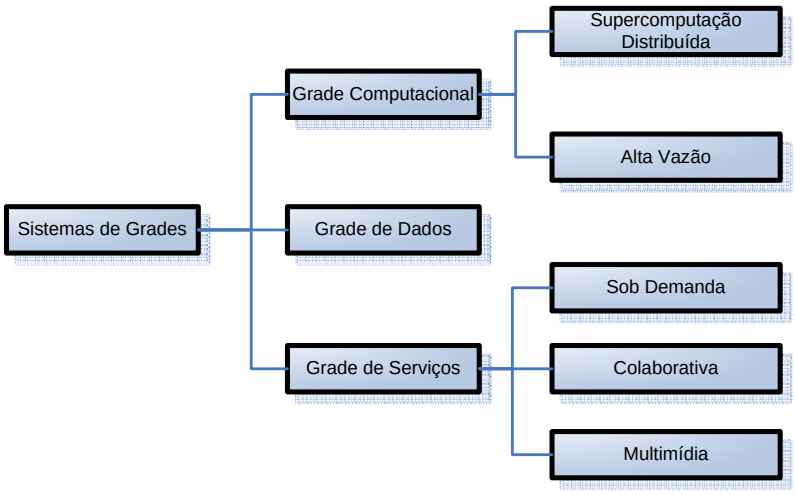

Figura 1.1: *Taxonomia de Grades [44]*

- **Grade Computacional** são sistemas de alto poder computacional agregado em um único sistema. Dependendo de como essa capacidade é utilizada, pode ser classificada como supercomputação distribuída ou como uma grade de alta vazão. Grade de supercomputação distribuída executam aplicações em paralelo em múltiplas máquinas. Um grade de alta vazão são aquelas onde o mais importante é a taxa de tarefas completadas.
- **Grade de dados** são sistemas que provêem uma infra-estrutura especializada em armazenamento gerenciado e acesso a dados, geralmente distribuídos em uma grande área geográfica, como por exemplo, bibliotecas digitais.
- **Grade de serviços** são sistemas que provêem serviços que não são fornecidos por nenhuma máquina isolada. Pode ser subdividido em sistemas de grade sob demanda, colaborativos e multimídia. As grades colaborativas integram usuários e aplicações em grupos de trabalho colaborativos. Um grade de serviços sob demanda agrega dinamicamente novos recursos para prover novos serviços. Por fim, uma grade multimídia disponibiliza uma infra-estrutura para aplicações multimídia em tempo real.

Apesar de não haver um consenso, dentre as características que definem um sistema em grade, pode-se destacar algumas que são mais citadas e que são se aplicam a diversos sistemas de grades atuais [34]:

Não substituem os sistemas operacionais: os sistemas de computação em grade podem utilizar os serviços do sistema operacional, porém são estruturados como um *middleware* que provê serviços ao usuário e aplicações da grade;

Podem integrar recursos distribuídos e descentralizados: a maioria dos sistemas de computação em grade é capaz de integrar recursos dispersos por múltiplos domínios administrativos e conectados por redes de longa distância;

Podem ser utilizados por diversas aplicações: a maior parte dos sistemas de grades provê serviços que podem ser utilizados por diversas aplicações;

Podem incluir várias plataformas de *hardware* e *software*: a maior parte dos sistemas de computação em grade pode integrar recursos heterogêneos, compostos por diversas plataformas de *hardware* e *software*;

Adaptabilidade às políticas locais: como integram recursos de diversos domínios administrativos, os sistemas de computação em grade devem ser capazes de se adaptar às políticas e restrições de uso de cada um deles.

As Grades computacionais podem ser formadas por diversos tipos de computadores, incluindo, mais recentemente, dispositivos móveis, onde tem-se dispositivos portáteis aliados a redes sem fio. Isto possibilita ao usuário estar conectado a uma rede em qualquer lugar e a qualquer momento com dispositivos como, por exemplo, *laptops*, *smartphones*, *palmtops* e *pocketPCs*.

A integração de dispositivos móveis [53] às grades, representa um novo cenário para o uso desses dispositivos. Além disso, possibilita explorar novas formas de recursos, onde as grades não são apenas fornecedoras de ciclos de CPU ou espaço de armazenamento, mas são também fornecedoras de diversos outros tipos de recursos como, por exemplo, sensores sem fio.

A exploração deste novo cenário depara-se com a natureza altamente dinâmica dos dispositivos móveis e suas diversas restrições [26, 55]. Entre essas restrições, pode-se citar a duração limitada da bateria, a conectividade intermitente e de qualidade muito instável e o limitado poder de processamento e de armazenamento. Todas estas características sugerem um ambiente altamente variável, com muitas restrições e com cenários diversos de disponibilidade de recursos e de conectividade, surgindo como pré-requisito a reconfiguração dinâmica dos componentes do ambiente. Os serviços de *software* devem se adaptar aos diferentes tipos de dispositivos, à largura de banda e às várias condições do ambiente computacional no qual estão inseridos.

A adaptabilidade permite que as aplicações executem eficientemente sobre uma grande variedade de condições. Através de adaptação, um sistema pode mudar seu comportamento ao invés de mantê-lo constante em todas as situações. Para tanto, é preciso monitorar os recursos e o ambiente computacional, além de notificar as aplicações sobre as mudanças. Portanto, obter informações sobre o contexto atual do sistema como um todo, torna-se um importante requisito para se construir sistemas efetivos de adaptação.

Ao aliar adaptação e monitoramento ou ciência de contexto, introduz-se um novo conceito, a computação sensível ao contexto [13], onde são utilizadas as informações monitoradas e processadas de modo a prover adaptações do comportamento das aplicações e serviços.

Esta dissertação descreve uma arquitetura de computação sensível ao contexto que provê a base para a integração de dispositivos móveis em ambientes de computação em grade, permitindo a adaptação dinâmica do suporte de *middleware* de acordo com as variações de contexto a que estes dispositivos estão sujeitos.

1.1 Motivação

A crescente demanda por poder computacional nas mais diversas áreas como: seqüenciamento genético, previsão do tempo, otimização de tráfego, prospecção de petróleo, mercado financeiro, dentre outras. Isso, aliado ao avanço nas tecnologias ligadas à computação e à redução de custos de *hardware*, tem atraído a atenção da comunidade científica para os sistemas de computação em grade.

As pesquisas sobre estes sistemas têm sido cada vez mais importantes, amadurecendo rumo a um sistema de grade que faça jus à idéia original quando da criação do termo grade. Entretanto, há uma vasta gama de assuntos a serem discutidos para que esse objetivo se torne real, dentre os quais é de especial interesse a integração de dispositivos móveis a computação em grade. A exploração desse tema tem como aspecto principal, definir formas de integrar estes dispositivos de características tão diversas e estado instável (conectividade, bateria) à computação em grade.

1.2 Objetivos

Os principais objetivos deste trabalho são:

- definição de uma arquitetura de suporte à computação sensível ao contexto para adaptação dinâmica integrada ao *middleware* de grade;

- uso de contexto e adaptação na execução de aplicações em grades; e
- uso de contexto na adaptação dinâmica dos componentes do *middleware*;

O restante dessa dissertação está organizado como se segue. O Capítulo 2 trata de conceitos fundamentais para o entendimento do trabalho. O Capítulo 3 aborda os trabalhos relacionados. O Capítulo 4 apresenta o ContextGrid, sua arquitetura e implementação. O Capítulo 5 descreve os testes e avaliações. Finalmente, o Capítulo 6 traz as conclusões e trabalhos futuros.

CAPÍTULO 2

Fundamentos

Neste capítulo são abordados alguns fundamentos cujo entendimento é de extrema importância para contextualização do trabalho. São apresentados os principais conceitos, sistemas e idéias sobre as quais este trabalho se baseia.

2.1 Contexto e Adaptação

Quando humanos se comunicam entre si, estão aptos a usar informações implícitas à situação, ou seja, o contexto, para complementar e favorecer uma correta interpretação do diálogo. Infelizmente, esta habilidade não é facilmente realizada na interação homem-máquina [13].

Para poder fazer com que essa informação possa ser passada na interação homem-computador e possa ser usada de forma a melhorá-la, deve-se primeiro compreender o que é contexto.

Contexto é definido pelo dicionário Aurélio como "Numa situação de comunicação, características extralingüísticas que determinam a produção lingüística, como, por exemplo, o grau de formalidade ou de intimidade entre os falantes". Essa definição, extremamente abrangente, é muito difícil de ser aplicada em ambientes computacionais, além de não ajudar muito na compreensão de seu significado para computação. Existem inúmeros desafios a uma definição de contexto que seja útil a um sistema computacional, dentre eles: (*i*) qualquer informação pode ser considerada contexto, ou seja, é difícil determinar qual informação é relevante para a tarefa corrente de uma entidade, e (*ii*) o contexto, em uma interação, pode variar dependendo da configuração e reconfiguração dinâmica dos recursos, do ambiente e das entidades envolvidas.

Não obstante, muitos autores têm pesquisado e tentado definir contexto, existindo uma variedade de definições na literatura científica. A discussão sobre o significado de contexto e sobre como fazer aplicações que detectem e lidem com essa idéia começou em 1994 com Schilit [56], o qual divide contexto em três categorias:

- Contexto computacional: conectividade da rede, custo de comunicação, largura de banda, recursos próximos, impressoras, *displays* e estações de trabalho.
- Contexto de usuário: perfil do usuário, localização, pessoas nas proximidades, detecção de situações sociais.
- Contexto físico: iluminação, nível de ruído, condições de tráfego e temperatura.

Em [13], além dessas três classes, é proposta uma quarta, como sendo muito importante e até mesmo natural para muitas aplicações: o contexto de tempo. Este tipo engloba: hora do dia, semana, mês, ano e até mesmo estações do ano.

Alguns pesquisadores tentam formalizar a definição de contexto. Dentre as primeiras tentativas, está a proposta por [56], que define contexto como "O conhecimento sobre o estado do usuário e dos dispositivos de tecnologia, incluindo sua localização e vizinhança". Contudo, uma das mais referenciadas e aceitas, e a que será adotada nesse trabalho devido à sua definição mais clara, é a apresentada por Dey [22], o qual define contexto como:

> "... qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar, ou objeto que é considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a própria aplicação."

Essa definição torna mais fácil para o desenvolvedor enumerar o contexto para o cenário da sua aplicação. Assim, se uma informação pode ser usada para caracterizar a situação de uma entidade em uma interação, então esta informação é contexto. À medida que são utilizados os diversos contextos, de modo a realizar adaptações no comportamento das aplicações e serviços, tem-se um sistema sensível ou ciente de contexto [57, 6].

A Figura 2.1 ilustra um modelo geral desse tipo de sistema. Nessa arquitetura, existe uma série de sensores de *software* e de *hardware* que monitoram o ambiente e os dispositivos de interesse. As informações colhidas por esses sensores são passadas a um conjunto de serviços de contexto, onde são processadas e modificadas para que possam ser entregues às aplicações e dispositivos. Será adotado neste trabalho informação de contexto como sendo os dados obtidos diretamente de algum sensor e que ainda não foram processados de nenhuma forma. Já a palavra contexto, será usada para designar o resultado do processamento das informações de contexto. Sendo que as aplicações e dispositivos só fazem uso de contexto.

Schilit [57] define computação sensível ao contexto classificando as funcionalidades das aplicações da seguinte forma:

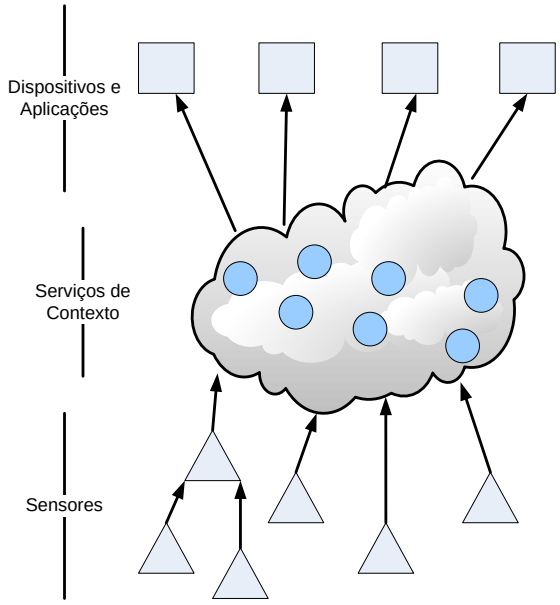

Figura 2.1: *Modelo de aplicações sensíveis ao contexto*

1. Seleção de proximidade, uma técnica de interface gráfica na qual os elementos da tala são posicionados de maneira a enfatizados uns em detrimento de outros ou, visando facilitar as escolhas do usuário.

2. Reconfiguração contextual automática, é o processo de adição ou remoção de componentes ou de alteração das conexões entre eles devido a uma mudança de contexto.

3. Informações e comandos contextuais, são aqueles que podem podem produzir diferentes resultados ou possuírem diferentes significados de acordo com o contexto no qual são usados.

4. Ações disparadas por contexto, compreendem simples regras do tipo se-então que são usadas para especificar como um sistema sensível ao contexto deve se adaptar.

Uma outra definição proposta por Dey [21], combina as idéias de Schilit com uma classificação baseada nas características de aplicações sensíveis ao contexto, proposta por Pascoe [52], e as mapeia dividindo suas características em:

- apresentação das informações e dos serviços ao usuário;
- execução automática de serviços; e
- marcas de contexto para uso posterior.

As aplicações sensíveis ao contexto vêm sendo exploradas em uma grande variedade de cenários como forma de prover aplicações que se adaptem às diversas variáveis do ambiente em que se encontram, visando um melhor aproveitamento dos recursos e uma interação mais adequada com o usuário.

Em [49] é destacada a aplicação de contexto em visualização de imagens médicas. Com o uso de contexto tenta-se obter um sistema mais intuitivo e que mostre as informações mais relevantes e se adapte a cada cenário. Em sistemas de informações médicas e, mais particularmente, na visualização de imagens médicas, o contexto é constituído de informações sobre a imagem (bytes por pixel, altura, largura, espaçamento entre planos, etc), sobre o usuário (médico, enfermeiro, preferência de visualização, etc), sobre o paciente (nome, endereço, histórico de doenças etc) e sobre o equipamento no qual o programa é executado (quantidade de memória livre, espaço em disco, largura de banda). O uso dessas informações e o correto processamento das mesmas traz inúmeras vantagens à aplicação como, por exemplo, fazer com que ela disponibilize informações correlacionadas ou esconda certas informações dependendo do usuário.

Em [10] o contexto é usado em um sistema de localização de médicos em um hospital, onde com base em informações de contexto, como as coordenadas geográficas de cada médico, são derivados os nomes das salas onde cada um se encontra e tenta-se inferir sua possível atividade. Com base nessas informações, o sistema decide qual médico será o responsável por uma emergência que eventualmente ocorra. Pode-se ainda, gerar um mapa, ilustrado na Figura 2.2, da localização e atividade de cada médico.

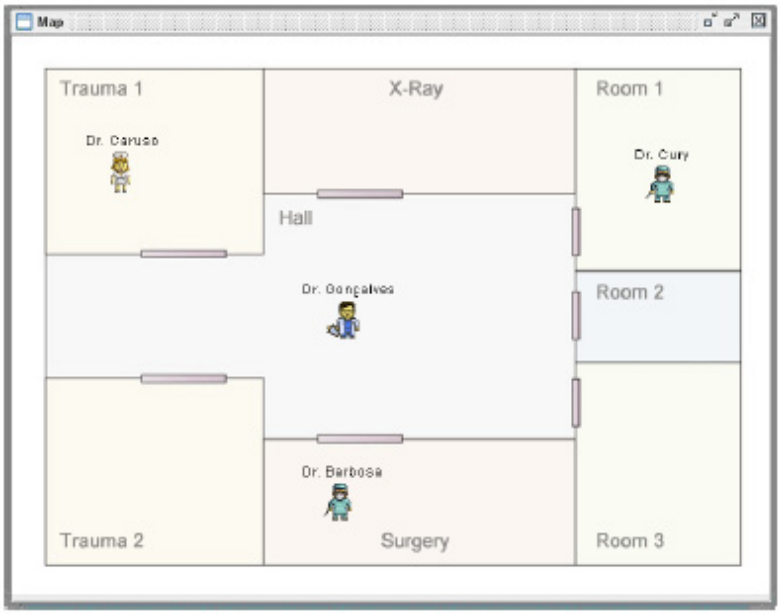

Figura 2.2: *Localização de médicos no hospital [10]*

2.1.1 Interpretação do Contexto

Um ponto que merece destaque quando se trata de aplicações sensíveis ao contexto é a interpretação do contexto. Dey [22] trata a interpretação do contexto como um processo onde se eleva o nível de abstração das informações contextuais, onde gera-se informações mais elaboradas a partir de informações primitivas. A interpretação do contexto usa a abstração, o refinamento, a agregação, a derivação e a inferência sobre as informações de contexto, tendo como objetivo torná-las úteis às aplicações. Como exemplo de elevação do nível de abstração, pode-se citar: dadas as informações de contexto obtidas de diversos sensores de localização de pessoas em uma sala, juntamente com dados de sensores de ruído, pode-se produzir um contexto que diz que há uma reunião em andamento. Ou ainda de forma mais direta, a conversão de coordenas fornecidas por um GPS (*Global Positioning System*) no nome do local apontado.

Dey propõe uma arquitetura para possibilitar que as aplicações obtenham os contextos requeridos sem que tenha que entender como os eles são obtidos. A Figura 2.3 apresenta a arquitetura proposta por ele, chamada de *Context Toolkit*. Ela trata de aspectos fundamentais para interpretação do contexto, onde pode-se destacar os seguintes componentes:

Context widgets - responsáveis por adquirir certos tipos de informações de contexto e torná-las disponíveis às aplicações de uma maneira padrão, não importando a forma como são obtidas. São independentes das aplicações que os utilizam e estão diretamente ligados ao tipo de sensor do qual coletam informações.

Context interpreter - responsável pela interpretação funcional de um contexto; aceita informações de contexto de um ou mais tipos como, por exemplo, interpretar o contexto de todos os *widgets* de uma sala de reunião para determinar se uma reunião está acontecendo.

Context aggregators - responsáveis por agrupar todos os contextos de uma entidade. Coletam os contextos sobre uma entidade a partir de um ou vários *context widgets* ou dos *context interpreters*, e agem como uma espécie de *proxy* de contexto para as aplicações. Esse componente tem como objetivo facilitar a definição de escopos para os diversos contextos. Seu uso não é obrigatório.

Outros aspectos importantes para a correta interpretação do contexto e que devem ser observados durante o processo são: (*i*) o armazenamento histórico das informações de contexto, o qual pode ser usado pelo *context interperter*, por exemplo, para predizer futuras ações, (*ii*) a distribuição transparente dos componentes e

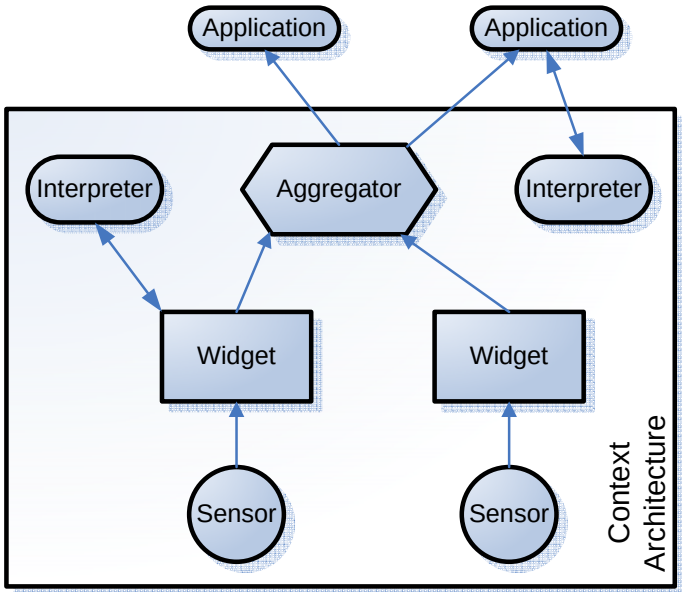

Figura 2.3: *Context Toolkit [21]*

sensores da arquitetura, mediando-se toda comunicação entre a aplicação e estes, ou seja, a localização dos componentes e sensores é abstraía para a aplicação, e (*iii*) uma correta modelagem das informações contextuais. Este último aspecto é um dos temas centrais desta dissertação e, portanto, é descrito em mais detalhes a seguir.

2.1.2 Modelagem de Contexto

Atualmente, o desenvolvimento de aplicações cientes de contexto é uma tarefa complexa, o que torna o uso de técnicas de modelagem extremamente úteis. Contudo, os atuais modelos para desenvolvimento de *software* não oferecem suporte para o projeto e a modelagem de tais aplicações, sobretudo, tais modelos não provêem suporte a modelagem das informações contextuais [38].

Existem inúmeras abordagens para modelar informações contextuais, dentre as quais pode-se ressaltar:

Modelos com métodos de marcação [65]: Seguem uma estrutura hierárquica de marcação com atributos e conteúdo. Em geral utilizam-se de linguagens de marcação derivadas da *Standard Generic Markup Language* (SGML)[60].

Modelos chave-valor [56]: Utilizam um modelo simples de atributo e valor, sendo fáceis de gerenciar. Contudo têm pouco poder de expressão.

Modelos gráficos [38]: Baseados em notações gráficas, em geral são derivados de adaptações e extensões de modelos gráficos já difundidos, como *Unified Modeling Language* (UML), *Object-Role Modeling* (ORM) ou o modelo Entidade Relacionamento (ER).

Modelos orientados a objetos [58]: Esta abordagem tem a intenção de aplicar os principais benefícios do modelo orientado a objetos, notadamente, encapsulamento e reusabilidade, à modelagem de contexto. O acesso às informações contextuais é feito somente através de interfaces bem definidas.

Modelos baseados em lógica [31]: Define-se o contexto de modo que se possa inferir expressões ou fatos a partir de um conjunto de outros fatos e expressões. Em geral, este modelo possui um alto grau de formalismo.

Modelos baseados em ontologias [72]: Uma ontologia é uma especificação de uma conceituação [35], isto é, uma descrição de conceitos e relações que existem entre eles, em um domínio de interesse. Nesse modelo o contexto é modelado em ontologias, construindo uma base de conhecimento do contexto.

Em [62] são avaliadas as abordagens citadas acima para modelagem de contexto, onde os seguintes critérios são considerados:

1. **Composição distribuída (cp):** A composição do modelo de contexto deve ser altamente dinâmica em termos de tempo, topologia de rede e origem. Podendo estar distribuído em diversas localidades ao longo do tempo.

2. **Validação parcial (vp):** Deve ser possível validar parcialmente o modelo, dado que nem todas as informações podem estar disponíveis ao mesmo tempo e que o conhecimento do contexto pode ser derivado da composição de outras informações distribuídas.

3. **Riqueza e qualidade da informação (rqi):** A qualidade e a riqueza das informações podem variar de acordo com o tempo e com o tipo de sensor, o modelo deve permitir anotações de qualidade e riqueza da informação de contexto representada.

4. **Incompletude e ambigüidade (ia):** as informações contextuais disponíveis em um dado momento podem ser incompletas ou ambíguas. Estas características devem ser cobertas pelo modelo.

5. **Nível de formalidade (nf):** A formalidade visa dar um visão única do modelo; é altamente desejável que todos os participantes tenham a mesma interpretação. A formalidade permite, também, o processamento automático das informações de contexto diretamente do modelo, por exemplo, para validação.

6. **Aplicabilidade nos ambientes existentes (aae):** para uma boa aceitação, é importante que o modelo seja aplicável às infra-estruturas de suporte a contexto já existentes.

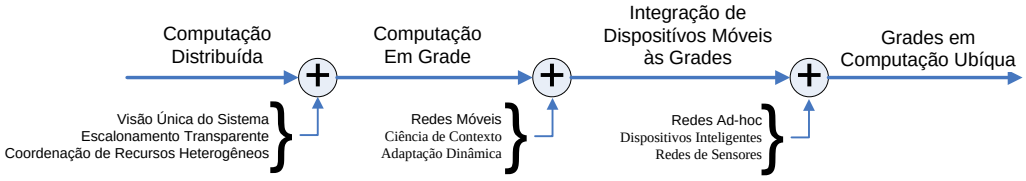

Figura 2.4: *Evolução: Computação Distribuída - Grades em Computação Ubíqua*

Os resultados da análise realizada por [62] podem ser observados na Tabela 2.1. A notação apresentada atribui o sinal "-" para o critério não satisfeito pelo modelo, e o sinal "+" para o critério satisfeito. Sendo "++" para os critérios que são fortemente atendidos.

Tabela 2.1: *Avaliação - Abordagens para Modelagem de Contexto*

	Critério					
Modelo	**cp**	**vp**	**rqi**	**ia**	**nf**	**aae**
Chave-Valor	-	-	-	-	-	+
Métodos de marcação	+	++	-	-	+	++
Gráficos	-	-	+	-	+	+
Orientados a objetos	++	+	+	+	+	+
Baseados em lógica	++	-	-	-	++	-
Baseados em Ontologia	++	++	+	+	++	+

Este trabalho adota o modelo proposto por [38, 37], o qual segue um modelo gráfico com suporte para validação parcial (vp) e incompletude e ambigüidade (ia), trazendo uma notação de fácil leitura e composição. Aspectos relevantes à escolha são discutidos com maior detalhe no Capítulo 4.

Contexto tem se apresentado como um requisito para sistemas que visam a computação ubíqua [66], além de ambientes altamente dinâmicos como os sistemas distribuídos [12]. O uso de contexto em ambientes de computação em grades pode trazer os mesmos benefícios já verificados em ambientes distribuídos e ainda permitir a inserção das grades em ambientes ubíquos. As grades assim como os ambientes ubíquos são altamente distribuídos, apresentando diversos pontos em comum como, por exemplo, abstração da localização dos recursos ou dispositivos. A inserção de grades nesse ambiente pode contribuir para a solução da falta de recursos em dispositivos móveis, que representam grande parte dos dispositivos presentes na

computação ubíqua. Este trabalho trata apenas da questão anterior, a inserção de contexto na computação em grade. Na Figura 2.4 é apresentada uma linha de evolução dos sistemas distribuídos até a inserção das grades em computação ubíqua, passando pela integração de dispositivos móveis às grades, que é discutida nesse trabalho.

2.2 Computação em Grades

Um sistema de computação em grade (*grid computing*) [29] é uma infra-estrutura que explora as potencialidades das redes de computadores e *software* para interligar e gerenciar uma grande variedade de recursos computacionais, em geral distribuídos, para oferecê-los de forma transparente ao usuário. Dentre as diversas iniciativas de sistemas de grades computacionais, pode-se destacar o Projeto Globus [33], por ser o mais conhecido e por ter influenciado inúmeros outros trabalhos, o Condor [17] pela característica oportunista na exploração de recurso e por fim projeto InteGrade [34] e o projeto MAG (*Mobile Agents Technology for Grid Computing Environments*)[47]. Esses dois últimos estão diretamente ligados ao ContextGrid.

2.2.1 Globus

O projeto Globus [33, 28, 29] foi iniciado em 1997, e é considerado um dos projetos de maior importância em computação em grade. Seus objetivos de baseiam no desenvolvimento e promoção de protocolos padrão para permitir interoperabilidade entre infra-estruturas.

O *Globus Toolkit* [33, 29], desenvolvido no projeto Globus, é um conjunto de ferramentas e bibliotecas de *software* que dão suporte à arquitetura e às aplicações em grade. É mantido por uma comunidade de programadores e é baseado no uso de padrões e componentes de *software* de código aberto. Com esta ferramenta é possível implementar segurança, busca de informações, gerenciamento de recursos e de dados, comunicação, detecção de falhas e alocação de recursos.

A primeira versão do *toolkit* foi lançada em 1999. Contudo, somente após dois anos, em 2001, o mercado demonstrou interesse e percebeu todas as possibilidades que poderiam ser alcançadas através dele. Após a versão 2 do *Globus Toolkit*, a arquitetura tornou-se um padrão *de facto* para a computação em grade.

Os componentes do conjunto de ferramentas Globus, representados na Figura 2.5, podem ser usados tanto independentemente quanto em conjunto no desenvolvimento de aplicações e de ferramentas para grades. Os principais serviços presentes na infra-estrutura são: *Globus Resource Allocation Manager* (GRAM),

Monitoring and Discovery Service (MDS), *Reliable File Transfer* (RFT) e *Grid Security Infrastructure* (GSI).

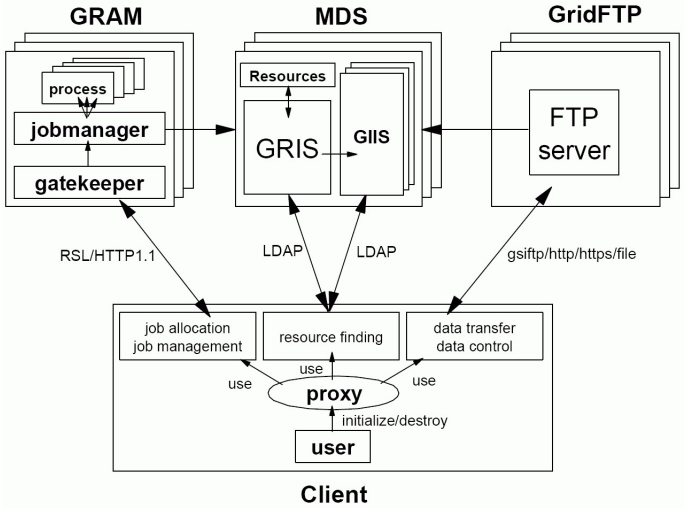

Figura 2.5: *Componentes do GlobusToolkit*

***Globus Resource Allocation Manager* (GRAM):** é responsável pelo serviço de alocação de recursos e gerenciamento de tarefas. Ele fornece uma interface única que permite submeter, monitorar e controlar tarefas de maneira independente da tarefa de escalonamento de recursos. Ele é o nível mais baixo da arquitetura de gerência de recursos do Globus. O GRAM é encarregado de administrar um conjunto de máquinas, sejam elas máquinas paralelas, aglomerados ou estações de trabalho.

***Monitoring and Discovery Service* (MDS):** grades são ambientes extremamente dinâmicos, onde os componentes podem falhar ou ser substituídos. Assim, é importante que haja um componente que disponibilize informações sobre a grade. No Globus, este componente é o serviço de descoberta e monitoramento. O MDS armazena informações sobre vários aspectos da grade, como sistema operacional, memória disponível, espaço em disco etc. Ele é constituído por dois serviços internos: GIIS e GRIS. O GRIS (*Grid Resource Information Service*) é responsável por obter as informações da máquina onde está sendo executado, isto é, as informações locais. Já o GIIS (*Grid Index Information Service*) reune em um único servidor as informações de vários GRISs.

Reliable File Transfer (RFT): O RFT é o serviço responsável pela transferência segura de arquivos entre as máquinas da grade.

Grid Security Infrastructure (GSI): [36] o principal desafio de um ambiente de computação em grade em relação à segurança é disponibilizar um sistema

seguro, capaz de se adaptar bem ao ambiente dinâmico e à distribuição dos usuários em domínios administrativos diferentes. A infraestrutura de segurança do Globus (*Grid Security Infrastructure*) trata da autenticação de entidades no sistema.

O objetivo da GSI é permitir que, uma vez autenticado, o usuário receba uma credencial que o permita acessar recursos distribuídos na grade sem a necessidade de se autenticar novamente em cada máquina. As credenciais são mapeadas para os mecanismos locais de autenticação de cada domínio administrativo, permitindo que diferentes ambientes participem da grade sem a necessidade de alterar suas políticas locais. O GSI provê mecanismos para autenticação e comunicação segura baseados em certificados X.509, infraestrutura de chave pública (PKI), protocolos SSL/TLS e certificados *proxy* X.509.

Em sua terceira versão, ou *Globus Toolkit* 3 (GT3) [33], ilustrado na Figura 2.6, houve importantes evoluções em relação às versões anteriores. Ele passou a ser baseado em uma arquitetura e padrões definidos pelo *Global Grid Forum*, [1] cujos principais conceitos são descritos a seguir.

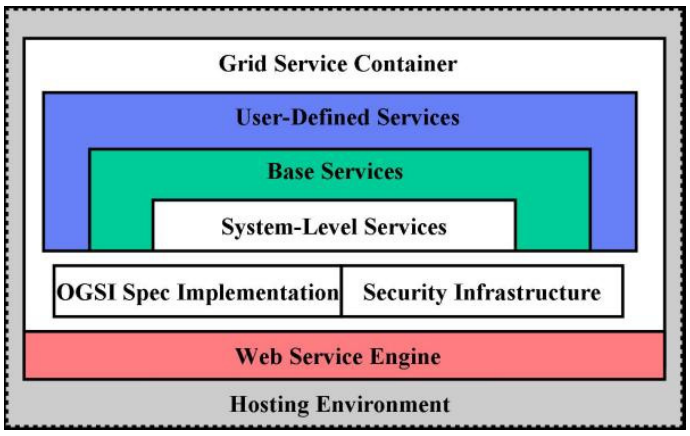

Figura 2.6: *Globus Toolkit 3*

***Open Grid Services Architecture* (OGSA):** a utilização de padrões é uma das principais necessidades para que os sistemas em grade sejam utilizados em larga escala. A arquitetura de computação em grade do Globus é definida pela *Open Grid Services Architeture* (OGSA) [50], desenvolvida pelo *Global Grid Forum*. A OGSA define os serviços e toda a estrutura que pode ser

[1] Comunidade de usuários, desenvolvedores e fornecedores que objetiva a padronização dos conceitos de computação em grade

provida em um ambiente de grade. É baseada nos padrões já definidos para *Web Services*[59] e considera um serviço em uma grade como um *Web Service* com algumas particularidades definidas através da linguagem padrão WSDL (*Web Services Definition Language*)[68], com pequenas extensões. Isto é importante porque fornece uma maneira de acessar um serviço em grade usando padrões abertos e consolidados, como SOAP, XML e WS-*Security*. Além disso, *Web Services* permitem padronizar a pesquisa, identificação e utilização de novos serviços conforme forem sendo adicionados a grade.

***Open Grid Services Infrastructure* (OGSI):** a OGSI [51] é a especificação da infraestrutura da OGSA. Baseada em grades e *Web Services*. OGSI é o *middleware* para os chamados *grid services*, ou serviços de grade. Ela define como construir, gerenciar e expandir um determinado serviço.

***Web Services*[59]:** são a base para o conceito de *Grid Services* e, por sua vez, a base para a OGSI, para a OGSA e para o GT3. *Web Services* é uma tecnologia de computação distribuída que permite a criação de aplicações cliente/servidor usando como plataforma de comunicação protocolos e linguagem abertos e amplamente conhecidos como o HTTP e o XML.

Na versão 4 do *toolkit* (GT4) [2] passou-se a usar a especificação WSRF (*Web Service Resource Framework*) [18]. Originada a partir do refinamento da especificação OGSI devido a necessidade de interoperabilidade*com web services* em geral. É uma especificação completamente baseada em serviços web. Nesse novo cenário o OGSI passa a fazer parte dos padrões de serviços web e o OGSA passa a ser baseado diretamente sobre eles, conforme ilustra a Figura 2.7.

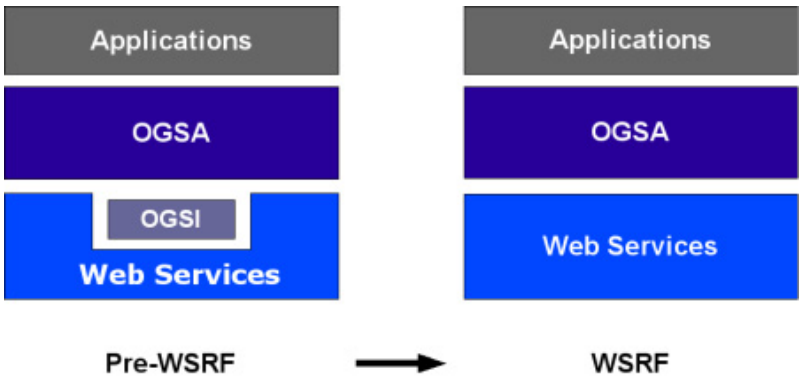

Figura 2.7: *OGSI para WSRF*

O fato do projeto Globus ser, atualmente, o que tem o maior impacto na prática, torna-o uma importe referência para estudo de grades computacionais, pois suas estrutura e seus padrões têm influenciado inúmeros trabalhos desde a sua afirmação como padrão *de facto*.

2.2.2 Condor

Condor [17] é um sistema desenvolvido na Universidade de Wisconsin-Madison. Seu principal objetivo é fornecer grande poder computacional a médio e longo prazo (dias ou semanas), utilizando processadores ociosos na rede [46]. Ou seja, Condor visa oferecer um desempenho sustentável, mesmo que o desempenho instantâneo do sistema possa variar consideravelmente. Esta ferramenta é especializada na execução de aplicações cujas tarefas são independentes, tais como aplicações do tipo *bag-of-tasks*[2].

Em geral, a utilização do Condor se dá da seguinte forma: o usuário submete as tarefas que devem ser executadas remotamente; o Condor localiza as máquinas ociosas na rede e envia a elas as tarefas para execução. O usuário tem a percepção de que as tarefas estão sendo executadas localmente. Neste cenário, nem sempre a máquina remota possui acesso ao mesmo sistema de arquivos utilizado na máquina do usuário. Para resolver este problema, a máquina remota Condor faz o redirecionamento das chamadas de sistema locais para a máquina onde a execução foi solicitada [5].

Atualmente Condor provê suporte para dois modos de operação: grade dedicada [3] e grade oportunista [4]. Uma grade Condor é organizada na forma de aglomerados de máquinas, chamados de *Condor Pool*. Cada aglomerado, pode pertencer a um domínio administrativo distinto e são independentes entre si. A arquitetura de um aglomerado Condor é ilustrado na Figura 2.8.

As máquinas que fazem parte de um *Condor Pool* podem desempenhar as seguintes funções, às vezes mais de uma ao mesmo tempo:

Central Manager : responsável pelas tarefas de gerenciamento do aglomerado; onde estão os serviços *Collector* e *Negotiator*, responsáveis respectivamente por manter as informações do aglomerado e escalonar as tarefas.

Submitter : representa um nó cliente da grade, que solicita a execução de computações. Possui o serviço *schedd* que permite ao usuário solicitar execuções; e

Executer : são os nós que cedem poder computacional ao aglomerado. Neles deve executar o *startd* que permite ao Condor executar aplicações na máquina em questão.

[2] Aplicação paralela onde cada tarefa é independente uma da outra

[3] Cada nó perticipante em um dado momento é totalmente dedicado à grade

[4] Cada nó participante cede apenas os recursos ociosos à grade, sem degradar o desempenho para o usuário legítimo do nó

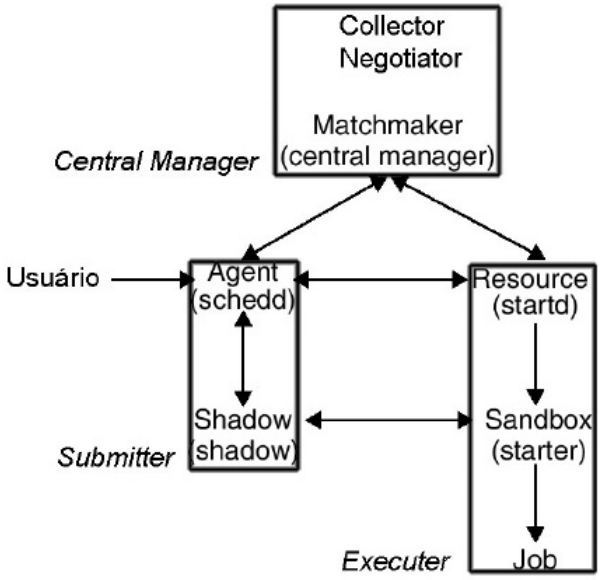

Figura 2.8: *Arquitetura de um Condor Pool*

Um dos aspectos que merece maior destaque no Condor, no contexto de computação em grade, é o de que ele suporta um modo de operação oportunista, possibilitando um melhor aproveitamento de parques computacionais de uso múltiplo, sem interferir nas tarefas do usuário legítimo da máquina. Esta característica dá uma maior aceitação, por parte dos usuários, da utilização de suas máquinas como integrantes da grade.

2.2.3 InteGrade

O InteGrade [34] é um ambiente de grade computacional que fornece uma plataforma para o desenvolvimento de aplicações que podem fazer uso da capacidade ociosa de computadores compartilhados, sendo considerado um sistema de grade oportunista. Os principais objetivos do projeto são:

- utilizar o poder computacional ocioso das máquinas em um parque computacional já instalado;
- utilizar extensivamente o paradigma de orientação a objetos;
- prover suporte para diversas categorias de aplicações paralelas;
- não degradar o desempenho das máquinas que fornecem recursos à grade; e
- analisar e monitorar os padrões de uso das máquinas para melhorar o desempenho do escalonador.

Desta maneira, a criação de uma infra-estrutura de *software* que permita a utilização efetiva de tais recursos que seriam desperdiçados possibilitaria uma economia financeira para as instituições que demandam grande poder de computação.

O InteGrade é organizado em uma estrutura hierárquica, ilustrada na Figura 2.9, onde as unidades estruturais são os aglomerados (*clusters*), que são conjuntos de máquinas agrupadas de acordo com um certo critério, como por exemplo, um mesmo domínio administrativo. As principais características do InteGrade incluem [34]:

- arquitetura orientada a objetos;
- infra-estrutura de comunicação baseada em CORBA;
- suporte a aplicações com paralelismo não trivial.

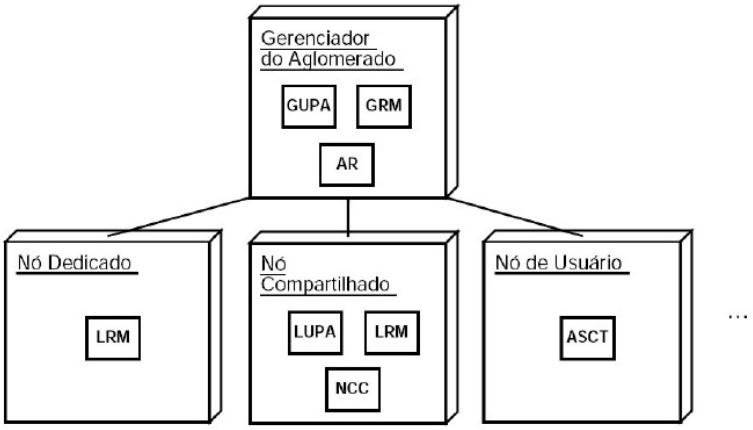

Figura 2.9: *Arquitetura de um Aglomerado InteGrade*

Cada uma das máquinas pertencentes ao aglomerado é chamada de nó, existindo vários tipos de nós conforme o papel desempenhado. O nó dedicado é uma máquina reservada à computação em grade. O nó compartilhado é aquele pertencente a um usuário que disponibiliza seus recursos ociosos à grade. Já o nó de usuário é aquele que tem capacidade de submeter aplicações para serem executadas na grade. Finalmente, o Gerenciador do Aglomerado é o nó onde são executados os componentes responsáveis pela coleta de informações e escalonamento, entre outros aspectos da gerência do aglomerado. Na Figura 2.9, pode-se observar que cada nó executa um conjunto de componentes de *software*, que são destinados a fornecer as funcionalidades exigidas para cada categoria apresentada. Estes componentes são descritos abaixo.

LRM (*Local Resource Manager*): componente executado em cada máquina que cede recursos para a grade. Responsável por informar ao GRM a quanti-

dade de recursos disponíveis na máquina, assim como receber requisições para execução de aplicações.

GRM (*Global Resource Manager*): Recebe informações sobre disponibilidade de recursos enviadas pelos diversos LRM. Também é responsável pelo escalonamento das aplicações submetidas à grade.

ASCT (*Application Submission and Control Tool*): Ferramenta que permite ao usuário da grade registrar aplicações no Repositório de Aplicações, solicitar execuções ao GRM, acompanhar o estado da execução e coletar os resultados das execuções.

NCC (*Node Control Center*): Executado nos nós compartilhados, permite que o proprietário da máquina controle o compartilhamento de seus recursos com a grade. Atuando conjuntamente com o LRM, permite que o usuário defina períodos em que os recursos podem ou não ser utilizados (independentemente de estarem disponíveis) e a fração dos recursos que pode ser utilizada.

LUPA (*Local Usage Pattern Analyzer*): Responsável pela análise e monitoramento dos padrões de uso. A partir das informações periodicamente coletadas pelo LRM, o LUPA armazena longas séries de dados e deriva categorias comportamentais do nó. Tais categorias serão utilizadas como subsídio para as decisões de escalonamento, fornecendo uma perspectiva probabilística de maior duração sobre a disponibilidade de recursos em cada nó.

GUPA (*Global Usage Pattern Analyzer*): Auxilia o GRM nas decisões de escalonamento, fornecendo as informações coletadas pelos diversos LUPAs. O GUPA pode funcionar de duas maneiras, de acordo com as políticas de privacidade do aglomerado. Se as informações fornecidas pelo LUPA não comprometem a privacidade de um usuário, como é o caso de uma estação de trabalho de um laboratório, o GUPA pode agir como aglomerador das informações disponibilizadas pelos LUPAs. Entretanto, caso o perfil de uso gerado pelo LUPA comprometa a privacidade do proprietário de um recurso (como uma estação de trabalho pessoal), os padrões de uso nunca deixam o LUPA e, nesse caso, o GUPA realiza consultas específicas ao LUPA, podendo assim funcionar como cache das respostas fornecidas sob demanda pelos diversos LUPAs.

AR (*Application Repository*): Armazena as aplicações a serem executadas na grade. Através do ASCT, o usuário registra sua aplicação no repositório para posteriormente requisitar sua execução.

BSPLib (*BSP Library*): Biblioteca que implementa uma API para aplicações BSP(*Bulk Synchronous Parallelism*). É baseada na Oxford BSPLib e permite a execução de aplicações BSP no Integrade.

Desde sua primeira versão, o InteGrade passou por pequenas reestruturações e adição de novos componentes, visando expandir e melhorar a arquitetura inicial apresentada por [34]. Atualmente, o InteGrade é dividido em apenas dois tipos de nós: o nó de gerenciamento de recursos do aglomerado (*Cluster Resource Manager node*), que representa um ou mais nós que são responsáveis pelo gerenciamento do aglomerado e pela comunicação com os gerenciadores de outros aglomerados, e o nó provedor de recursos (*Resource Provider node*) que é, em geral, uma estação de trabalho que exporta seus recursos ociosos para a grade. Esta nova arquitetura [40] e seus novos componentes são ilustrados na Figura 2.10 e descritos a seguir.

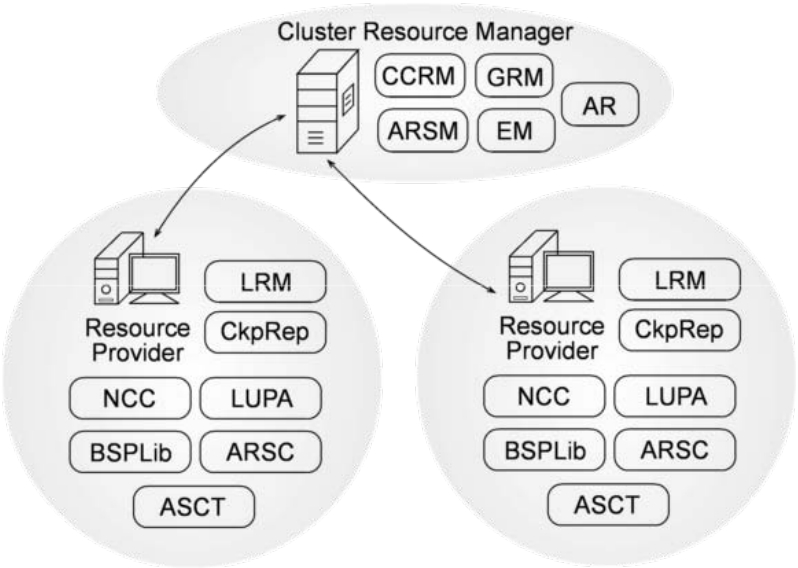

Figura 2.10: *Nova Arquitetura do Integrade*

ARSM (*Application Repository Security Manager*): gerencia a segurança em um aglomerado, provendo suporte para assinaturas digitais, encriptação de dados, autenticação e autorização.

CCRM (*Cluster Data Repository Manager*): gerencia dos repositórios de *checkpoints*[5] de um aglomerado. Distribui fragmentos codificados de *checkpoint* pelos repositórios disponíveis.

[5] Conjunto de informações sobre o estado de execução de uma aplicação, que permite seu reinício a partir do exato ponto em que foi salvo

EM (*Execution Manager*): Mantém informações de execução, como os *checkpoints* salvos e identificadores dos nós em que cada aplicação está executando para uma dada aplicação no Integrade. Também coordena a reinicialização e migração de aplicações.

ARSC (*Application Repository Security Client*): Auxilia os componentes locais na interação com componentes remotos de uma forma segura.

CkpRep (*Checkpoint Repository*): Armazena dados de *checkpoint* de forma distribuída. Cada provedor de recurso do aglomerado é um nó em potencial para hospedar uma CkpRep.

Essa arquitetura é a base para o MAG (*Mobile Agents Technology for Grid Computing Environments*) e, conseqüentemente, embora em um nível mais baixo, para o ContextGrid.

2.2.4 MAG

O uso de agentes móveis tem especial interesse devido à própria dinamicidade de um ambiente de grade, visando resolver problemas de alocação de recursos, balanceamento de carga e mobilidade de código [45].

O projeto MAG (*Mobile Agents Technology for Grid Computing Environments*) [47] tem como base o ambiente InteGrade. Ele explora a tecnologia de agentes móveis como forma de resolver alguns dos desafios relativos ao desenvolvimento de um middleware de grade computacional, com por exemplo, realocação dinâmica de tarefas. O uso de agentes é possível através do uso do arcabouço JADE (*Java Agent Development Framework*) [42], que provê várias funcionalidades, como comunicação, controle do ciclo de vida e monitoração dos agentes móveis.

A arquitetura do sistema MAG foi desenvolvida utilizando como fundação o middleware InteGrade, mantendo a compatibilidade com aplicações nativas do mesmo. O MAG adiciona um novo mecanismo de execução de aplicações, que faz uso da tecnologia de agentes móveis, cria um mecanismo de tolerância a falhas de aplicações, migração transparente e acesso à grade por clientes móveis. Na Figura 2.11 ilustra a arquitetura em camadas do MAG.

O MAG permite a execução de aplicações MAG escritas em Java, as quais executam sobre a camada MAG e fazem uso do paradigma de agentes móveis. Já as aplicações nativas do InteGrade são executadas diretamente pelo InteGrade. O MAG executa sua aplicação carregando dinamicamente o código da aplicação em um agente móvel. Estes agentes podem ser migrados dinamicamente entre os nós da grade em diversas situações como, por exemplo, para liberar um nó cujo usuário

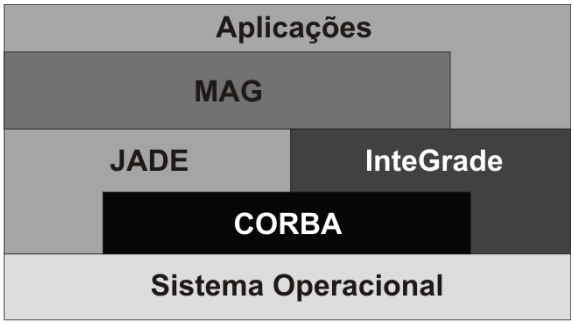

Figura 2.11: *Arquitetura MAG em camadas [47]*

solicitou exclusividade. Além disso, o MAG também explora o uso de agentes na implementação dos componentes de sua própria infra-estrutura.

O aspecto de tolerância a falhas é abordado no MAG através do conceito de migração forte [30], que é implementado por meio de uma modificação do *framework Brakes* [64], que permite a captura e a recuperação do estado de execução de aplicações em Java, o que abre caminho para o mecanismo de armazenamento de *checkpoints*.

Em sua versão atual, o MAG, permite a execução de dois tipos de aplicações: as regulares, que são aplicações seqüenciais que executam um programa binário sobre uma massa de dados e necessitam de uma única máquina, e as paramétricas ou *Bag-of-Tasks*, que executam várias cópias do mesmo binário sobre massas de dados distintas e, em seu melhor caso, cada cópia é executada em uma máquina e não há comunicação entre as tarefas.

O MAG incorpora uma série de componente e agentes ao InteGrade. Eles são responsáveis por iniciar a execução de uma aplicação, guardar informações e dados sobre as execuções em andamento, armazenar *checkpoints* de aplicações, realocar agentes na grade e promover recuperação em caso de falha. Estes componentes e agentes são detalhados a seguir.

ExecutionManagementAgent : agente que armazena dados relativos às aplicações em execução na grade como, por exemplo, o nó em que a aplicação executa, a lista de arquivos de entrada e saída, o nome do usuário que requisitou, a execução dentre outros. Um exemplo de arquivo XML, com os dados que são manipulados por este agente, é mostrado na Figura 2.12.

MagAgent : principal agente do MAG, ele é responsável por instanciar e executar uma aplicação MAG, assim como, monitorar sua execução, migrá-la (caso solicitado), salvar os *checkpoints* periodicamente e atualizar o *ExecutionManagementAgent*. Cada aplicação tem um *MagAgent* correspondente. A aplicação é encapsulada pelo agente, sendo executada como um *thread*, isso possibilita

com que ela passe a usufruir dos beneficios dos agentes móveis, por exemplo, migando um *MagAgent*, migra-se a aplicação encapsulada por ele.

AgentHandler : componente que executa em cada nó da grade. Ele mantém um contêiner de agentes que executam no nó e é responsável por criar um *MagAgent* em resposta a uma requisição de execução de aplicação. Além disso, é responsável pela notificação dos agentes para migração, pelo monitoramento dos agentes por ele criados em um dado nó e pela coleta dos resultados de um agente que finalizou sua tarefa. É o *AgentHandler* que interage com os componentes do InteGrade de forma a tornar o uso de agentes transparente ao mesmo. Ele se comunica com o repositório de aplicação, LRM e ASCT.

ClusterApplicationViewer : ferramenta destinada aos administradores da grade para a visualização de forma gráfica das aplicações que estão em execução e apresentação de informações detalhadas sobre cada uma. Em resumo, ele sintetiza as informações armazenadas no *ExecutionManagementAgent*.

StableStorage : repositório que armazena, de forma durável, os*checkpoints* das aplicações.

AgentRecover : agente com ciclo de vida breve e objetivos bem definidos. Ele é criado após a detecção de uma falha em algum nó da grade. Através de consulta ao *ExecutionManagementAgent* e solicitações ao GRM (componente do InteGrade), ele recria os agentes que falharam em outro nó e restaura os últimos *checkpoints* salvos no *StableStorage* para os mesmos.

mobileProxyAgent : agente responsável por agregar alguns dispositivos móveis ao MAG, atua de forma simples convertendo as requisições dos dispositivos, que se conectam a ele por *sockets*, em requisições válidas na grade. Atualmente realiza para grade fixa o papel de um ASCT.

Para este trabalho, o principais aspectos do MAG esão o uso de agentes móveis tanto para execução de aplicações quanto na implementação de elementos de sua arquitetura e a migração forte. Com as facilidades de migração dos agentes móveis e do estado da computação presentes no MAG, o ContextGrid explora a migração ciente.

2.3 Considerações

Neste capítulo foram apresentados e contextualizados os princípios gerais e principais conceitos adotados no desenvolvimento deste trabalho. Foram desta-

```
<application source="grid">  <!-- grid / web / mobile -->
        <appExecutionId>10000</appExecutionId>

        <applicationInfo>
                <appReposId>10000</appReposId>
                <binaryName>Fibonacci</binaryName>
                <platformType>Java</platformType> <!-- Linux_i686 / Linux_x86_64 / -->
        </applicationInfo>                        <!-- Macintosh / Java -->

        <executionInfo>
                <executionType>regular</executionType> <!-- regular / bsp / parametric -->
                <userName>rafaelf</userName>
                <executingHost>guaguin</executingHost>
                <appArgs>50</appArgs>
                <appMainRequestId>1</appMainRequestId>
                <appNodeRequestId>0</appNodeRequestId>
                <appConstraints>osName == 'Linux'</appConstraints>
                <appPreferences>max(freeRAM)</appPreferences>
                <executionState>running</executionState> <!-- running / finished -->

                <outputFiles>
                        <file>stdout</file>
                        <file>stderr</file>
                        <file>out.dat</file>
                </outputFiles>

                <inputFiles>
                        <file>in.dat</file>
                </inputFiles>

                <initialTimestamp>99999999</initialTimestamp>
                <finishTimestamp>99999999</finishTimestamp>
        </executionInfo>
</application>
```

Figura 2.12: *Dados manipulados pelo ExecutionManagementAgent*

cadas algumas das características de cada assunto abordado, assim como as suas interelações como a arquitetura proposta.

A base fundamental para o ContextGrid é o contexto, a exploração desse conceito em computação em grade pode resultar inúmeros beneficios, que vão desde a melhor utilização de recurso, passando pela integração de dispositivos móveis, até adaptação dinâmica.

Assim como o Condor, um dos aspectos que merece destaque no InteGrade é o fato de que ele provê suporte para operação oportunista, sem interferir nas tarefas do usuário de uma estação de trabalho que forneça seus recursos à grade. O MAG com exploração de agentes móveis introduz uma maior flexibilidade ao InteGrade, possibilitando tolerância a falhas, migração de tarefas e uma maior diversidade nos tipos de aplicações aceitos. O MAG não é um *middleware* completo de grades, muitos dos serviços básicos, como escalonador de tarefas, foram mantidos no InteGrade. Será usado o termo MAG/InteGrade para se referir a um *middleware* completo com as características dos dois. O ContextGrid é uma extensão do MAG/InteGrade, fazendo uso e provendo serviços a componentes dos dois.

A seguir serão apresentados trabalhos relacionados com a infra-estrutura desenvolvida.

CAPÍTULO 3

Trabalhos Relacionados

Considerável esforço de pesquisa tem sido dedicado à solução de problemas relacionados a dispositivos móveis, grades computacionais e integração entre os mesmos. Têm sido abordados vários aspectos relacionadas às grades móveis, como requisitos e modelos de infra-estrutura [48], além da investigação dos benefícios do uso dessa tecnologia e a influência desse novo modelo nas grades tradicionais [53].

Este capítulo descreve alguns projetos são compostos tanto por dispositivos fixos quanto móveis e fazem uso de contexto e adaptação dinâmica para lidar com essa diversidade. Para cada trabalho são feitos breves comentários sobre os pontos relacionados ao ContextGrid.

3.1 EXEHDA

O EXEHDA (*Execution Environment for Highly Distributed Applications*) [71, 70, 24], tem como aspecto central, a proposta de uma solução integrada para suporte à Computação Pervasiva, implementada na forma de um *middleware* reflexivo. São abordadas formas de criar e gerenciar um ambiente de computação pervasiva, bem como de permirtir a execução de aplicações do tipo "siga-me". Estas aplicações são, por natureza, distribuídas, móveis e adaptativas ao contexto em que seu processamento ocorre, estando disponíveis a partir de qualquer lugar, todo o tempo. Elas estão disponíveis a partir de qualquer lugar e em todo o tempo. O *middleware* proposto dá suporte à mobilidade física e lógica, ciência de contexto, adaptação dinâmica e execução de aplicações distribuídas. O EXEHDA faz uso de um ambiente integrado chamado ISAMpe [70], que provê uma linguagem de programação, um ambiente de execução, reconhecimento de contexto e carga dinâmica de componentes.

O EXEHDA faz parte do Projeto ISAM (Infra-Estrutura de Suporte às Aplicações Móveis Distribuídas) [41], cujo foco é adaptação ao contexto e computação pervasiva. Neste trabalho será dada ênfase ao EXEHDA, por ser o elemento do projeto ISAM similar ao ContextGrid.

O EXEHDA é estruturado em um núcleo mínimo e em serviços carregados sob demanda. Os principais serviços fornecidos são organizados na forma de subsistemas que gerenciam: execução distribuída, comunicação, reconhecimento do contexto, adaptação, acesso pervasivo aos recursos e serviços, descoberta de recursos e gerenciamento de recursos.

As condições de contexto e o suporte à execução devem permitir que tanto a aplicação quanto o *middleware* utilizem essas informações na gerência da adaptação de seus aspectos funcionais e não-funcionais.

No EXEHDA, as aplicações são distribuídas, adaptativas ao contexto em que executam e capazes de compreender a mobilidade lógica e física. Elas são baseadas no modelo de programação do projeto ISAM, o ISAMadapt [70]. Esse modelo surgiu das abstrações do Holoparadigma [3], ao qual foram adicionadas novas abstrações e construções de forma a adequá-lo ao ambiente de computação pervasiva. O ISAMadapt especifica abstrações para expressar, durante o desenvolvimento, a adaptação ao contexto. Dentre estas abstrações são definidos: adaptadores para o código das entidades da aplicação, políticas de adaptação e elementos de contexto. O código feito usando o modelo de programação ISAMadapt é compilado para Java, visando a portabilidade. As aplicações ISAMadapt são gerenciadas pelo EXEHDA. A Figura 3.1 ilustra o EXEHDA, o ISAMadapt e seu relacionamento dentro do contexto do projeto ISAM.

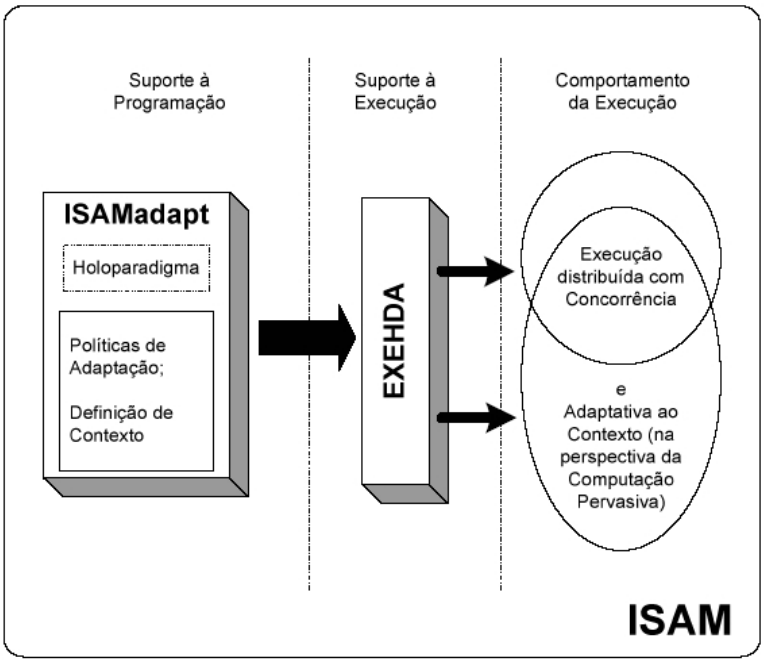

Figura 3.1: *Visão geral do ISAMadapt e EXEHDA*

3.1.1 Arquitetura

A arquitetura do ISAM, ilustrada na Figura 3.2, tem como objeto principal o EXAHDA. Ele apresenta uma organização lógica em três camadas, detalhadas a seguir.

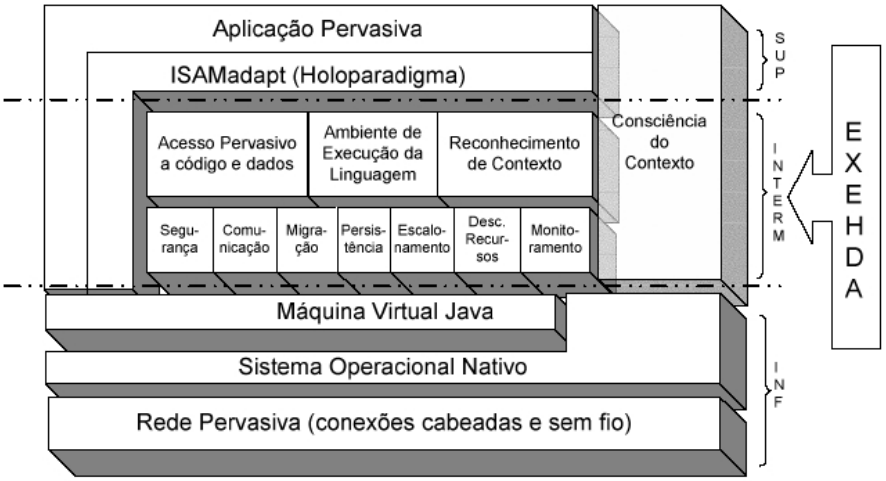

Figura 3.2: *EXEHDA no Projeto ISAM*

Na camada superior temos as aplicações propriamente ditas e a linguagem de programação ISAMadapt. Esta camada disponibiliza, como já discutido, abstrações para programação de aplicações distribuídas, móveis e cientes de contexto para computação pervasiva.

Na camada intermediária está o *middleware* EXEHDA, que disponibiliza os mecanismos de suporte à execução da aplicação pervasiva e cuida das estratégias de adaptação [70]. Ela é composta, em um primeiro nível, por:

Acesso Pervasivo a código e dados: é composta pelo Ambiente Virtual do Usuário, que é formado pelos elementos da interação do usuário com o sistema e pelo suporte às aplicações no estilo "siga-me"; pelo Ambiente Virtual da Aplicação, que é um conjunto de atributos o qual identificam uma execução específica de uma aplicação; e pela Base de Dados Pervasiva das Aplicações, que constitui o repositório de códigos das aplicações.

Ambiente de Execução da Linguagem: é responsável pelo gerenciamento da aplicação durante seu ciclo de vida.

Reconhecimento de Contexto: é encarregado de informar o estado dos elementos de contexto que são de interesse para a aplicação e para o ambiente de execução.

Em um segundo nível da camada intermediária estão localizados os serviços básicos do *middleware* EXAHDA. Eles provêem serviços ao primeiro nível dessa camada, a saber: segurança, comunicação, migração, persistência, escalonamento, descoberta de recursos e monitoramento.

A última camada da arquitetura representa a infra-estrutura básica de *software* e conectividade, como: Máquina Virtual Java, Sistema operacional e conexões de rede.

Na arquitetura do Projeto ISAM, o ambiente computacional onde recursos e serviços são gerenciados pelo EXEHDA, é denominado ISAMpe (ISAM pervasive environment). Esse ambiente é composto pelos dispositivos de usuários e equipamentos da infra-estrutura de suporte. O ISAMpe, ilustrado na Figura 3.3, pode ser composto por dispositivos multi-institucionais e é formado por: EXEHDAcel, que é a área de atuação de uma EXEHDAbase, geralmente relacionada a um escopo institucional, geográfico ou de comunicação; a EXEHDAbase, que é o ponto de comunicação com os EXEHDAnodos, é onde que estão todos os serviços básicos; os EXEHDAnodos, são os equipamentos de processamento disponíveis.

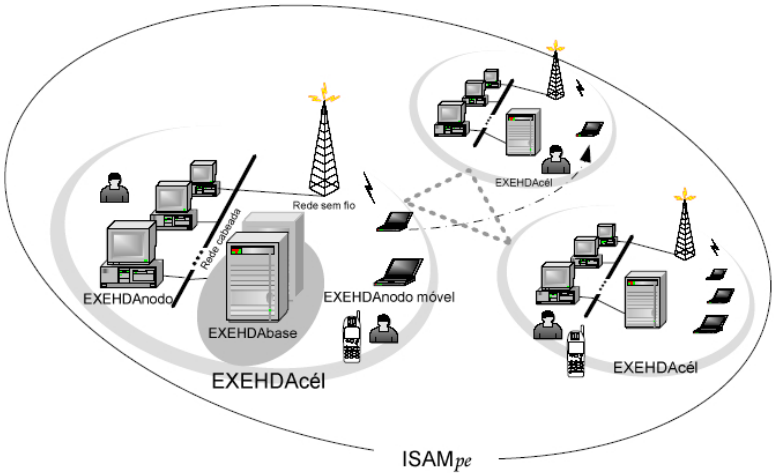

Figura 3.3: *ISAMpe*

O modelo de adaptação proposto para o EXEHDA emprega uma estratégia colaborativa entre aplicação e ambiente de execução. A adaptação não é uma propriedade funcional da aplicação; e sendo assim, seu tratamento pode ocorrer de forma autônoma. Considera-se que só ocorrera adaptação se o sistema dispor de recursos para o processamento da mesma. Ela é dividida em três etapas: detecção de alterações, escolha da ação e ativação da ação.

A adaptação colaborativa multinível [71], adotada no projeto, define que: (*i*) o *middleware* é responsável por adaptações relacionadas ao desempenho e gerência de serviços e recursos, (*ii*) a aplicação é responsável por adaptações dentro do seu

domínio de atuação e (*iii*) ambos (*middleware* e aplicações)negociam decisões de adaptação.

3.1.2 Comparações

O EXEHDA tem objetivos semelhantes ao ContextGrid quando se trata da integração de dispositivos móveis de forma transparente ao usuário e tendo o contexto como requisito para este ambiente pervasivo. Contudo, o EXEHDA está focado em ambientes distribuídos de forma genérica, sendo os objetos distribuídos a unidade de paralelismo. Apesar de serem empregados alguns conceitos de computação em grade, este, diferentemente do ContextGrid, não é o foco da arquitetura.

O ContextGrid e o EXEHDA oferecem diferentes modelos de aplicações. No entanto, no EXEHDA é preciso usar uma linguagem de programação própria, o ISAMadapt, causando uma dependência muito forte com a arquitetura e dificultando o porte ou a execução de aplicações legadas. O ContextGrid provê suporte para aplicações desenvolvidas com linguagens padrãos e amplamente aceitas por inúmeras arquiteturas de grades, como C, C++ e Java, segundo os modelos regulares, paramétricas e BSP.

O EXEHDA emprega a adaptação colaborativa, atendendo tanto o *middleware* quanto a aplicação de forma autônoma, desde que o programador utilize o modelo e linguagem próprios. Apesar da adaptação ser controlada por componentes externos, o programador deve incluir em seu código os fluxos de execuçãos alternativos a serem escolhidos diante de um contexto. Já no ContextGrid, a adaptação está focada no *middleware* de grade e é de responsabilidade do desenvolvedor de componentes, a escolha do que e de quando adaptar.

Outros aspectos importantes são a mobilidade de código e a tolerância a falhas. No EXEHDA apenas a mobilidade de código é tratada, tendo como itens móveis os objetos distribuídos de uma aplicação. O ContextGrid provê mobilidade de tarefas e componentes cientes de contexto, com migração forte do estado da computação e tolerância a falhas baseadas em *checkpoint*.

3.2 MoCA

A MoCA (Mobile Collaboration Architecture) [25] é uma arquitetura de *middleware* para o desenvolvimento de aplicações colaborativas e cientes de contexto para computação móvel. É constituída por APIs para implementação de clientes e servidores, serviços básicos para aplicações colaborativas e um *framework* para implementação de *proxies*.

A MoCA define que cada aplicação tem três partes: um servidor, um *proxy* e um ou vários clientes. O servidor e o *proxy* executam em nós fixos, enquanto a parte cliente executa em nós móveis.

O *proxy* realiza a intermediação entre toda a comunicação entre o servidor e o cliente. É nele que está a lógica de adaptação para a aplicação cliente-servidor a qual está vinculado.

Para a construção desses *proxies*, a MoCA disponibiliza um framework denominado *PoxyFramework*. Este *framework* provê mecanismos de acesso às informações de contexto relacionadas à interação cliente-servidor, e também define a programação de adaptações disparadas pelas mudanças de contexto. Ele implementa algumas das características mais comuns quando se usa uma abordagem baseada em *proxy*, como meios para lidar com a mobilidade dos clientes e conectividade intermitente.

3.2.1 Arquitetura

Os serviços que formam a arquitetura MoCA destinam-se a dar suporte ao desenvolvimento e execução de aplicações colaborativas cientes de contexto. São eles:

***Monitor*:** é um processo que executa em segundo plano em cada dispositivo móvel. Ele coleta informações sobre o ambiente e o estado de execução no dispositivo e os envia ao CIS (*Context Information Service*).

***Configuration Sevice (CS)*:** é responsável por armazenar e gerenciar as informações de configuração para todos os dispositivos móveis. Ele faz uso de tabelas *hash*, indexadas pelo endereço MAC, para armazenar o endereço do CIS e do DS (*Discovery Services*), aos quais um dado dispositivo está associado.

***Discovery Services (DS)*:** armazena informações das aplicações e serviços registrados no MoCA.

***Context Information Services (CIS)*:** recebe e processa as informações de estado enviadas por cada Monitor. Faz uso do modelo *Publisher/Subscribe* para receber inscrições de proxies e enviar eventos a cada um desses proxies, com informações sobre mudanças no estado dos dispositivos de interesse.

***Location Inference Service (LIS)*:** Este serviço fornece a localização aproximada dos dispositivos, tomando como base o padrão de sinais recebidos pelo dispositivo de diversos pontos de acesso 802.11. Este padrão é comparado com

pontos de referência pré-definidos, usando técnicas semelhantes às descritas em [11].

Na Figura 3.4 é ilustrada a arquitetura MoCA e a interação entre seus diversos componentes durante o registro e execução de uma aplicação.

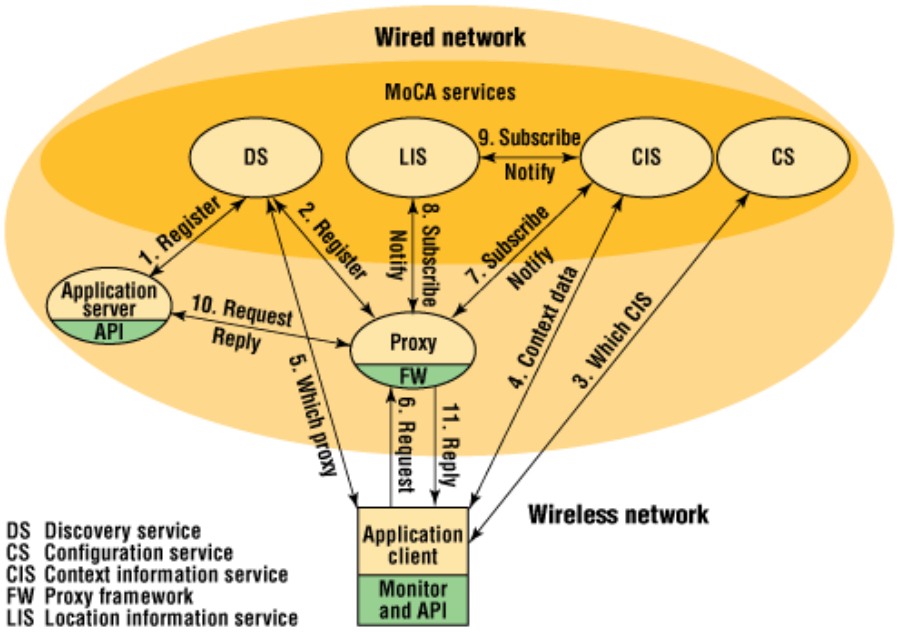

Figura 3.4: *Interação entre os Serviços da Arquitetura MoCA*

Primeiramente, a parte servidora da aplicação se registra no DS (1), informando os nomes e as propriedades dos serviços que implementa. Cada um dos proxies para esta aplicação realiza um processo semelhante (2). O Monitor, que executa em cada nó móvel, quando iniciado, obtém do CS o endereço do CIS (3), para o qual, periodicamente, enviará as informações de contexto (4).

A partir desse momento, a aplicação cliente entra em contato com o DS, com a finalidade de descobrir como acessar um serviço colaborativo na rede (5). O DS retorna os dados do *proxy* para a aplicação cliente. Este será o *proxy* que fará a ponte entre a aplicação e a parte servidor. A seguir, todas as requisições são encaminhadas ao *proxy* (6), o qual as processa e as adapta, caso necessário e as encaminha ao servidor (10). Nesse ponto o *proxy* se inscreve junto ao CIS para receber notificações de mudança no estado do cliente (7).

Para aplicações que requerem informações de localização, o*proxy* irá se inscrever também no LIS (8), o qual se inscreve no CIS (9) para receber informações sobre a potência dos sinais dos pontos de acesso ao cliente em questão.

Quando a parte servidora recebe a requisição, ela é processada e o resultado enviado ao *proxy* (10), o qual, de modo semelhante ao envio, processa e adapta os

resultados de acordo com o estado atual do cliente, obtido do CIS. Finalmente, a resposta é enviada ao dispositivo móvel (11).

3.2.2 Comparações

A MoCA, assim como o ContextGrid, faz uso da abordagem baseada em *proxy* para integração com dispositivos móveis, sendo que, na MoCA o *proxy* faz a intermediação da comunicação entre a parte servidora e a parte cliente de uma aplicação. Já no ContextGrid, o *proxy* tem um papel mais genérico, não sendo vinculado a uma aplicação. Ele realiza todas as comunicações entre os dispositivos móveis e a grade, além de possuir interfaces para comunicar-se com vários componentes do *middleware*.

O modelo de adaptação proposto pela arquitetura MoCA é baseado no *proxy*. É nesse elemento que estão situadas as políticas de adaptação, sendo ele que recebe as informações de contexto. As adaptações estão restritas às messagens trocadas entre a parte cliente e o servidor da aplicação. No ContextGrid, diferentemente, qualquer elemento da arquitetura pode usar as informações de contexto para se adaptar, incluindo o próprio *proxy*. As adaptações, como são de responsabilidade do desenvolvedor, podem ser as mais diversas possíveis como, por exemplo: de interfaces gráficas, protocolos e migração de tarefas e componentes.

Enquanto na MoCA o modelo de contexto reflete apenas os dispositivos móveis e o estado de execução das aplicações, no ContextGrid, o contexto reflete o estado de toda a grade computacional.

Uma outra diferença diz respeito ao foco das duas arquiteturas. Enquanto que a MoCA trata de aplicações colaborativas no modelo cliente servidor, o ContextGrid está focado em grades computacionais.

3.3 Solar

O projeto Solar [13, 14, 15, 16] visa permitir que aplicações executem mais eficientemente em dispositivos móveis. Ele está interessado em definir um plataforma aberta para suporte à coleta, agregação e disseminação de informações contextuais. O projeto provê um *framework*, denominado Solar, que possibilita a introdução dinâmica de componentes processadores de contexto. Estes componentes podem ser compartilhados por inúmeras aplicações. O Solar distribui, em tempo de execução, seus componentes, permitindo o paralelismo e balanceamento de carga.

Ele trata os sensores fornecedores de informações como Fontes de Informações, ou simplesmente Fontes. Eles são responsáveis tanto por prover informações

físicas, como temperatura, quanto informações computacionais, como latência da rede. Sua estrutura é baseada em eventos, os quais representam as mudanças no contexto. As Fontes são os publicadores de eventos, enquanto as aplicações e os operadores se inscrevem em canais ou fluxos de eventos para receber os contextos de seu interesse, sendo que os operadores também são publicadores de eventos. Um operador é um objeto que se inscreve em um ou mais fluxos de eventos, processando as entradas e publicando um outro fluxo de evento. Um operador pode se inscrever em um fluxo de eventos proveniente de outro operador. Eles podem ser conectados na forma de um grafo dirigido acíclico, visando a produção de um determinado contexto.

O operadores definidos inicialmente no projeto Solar são:

Filtro: tem por função filtrar os eventos recebidos. Sua saída é geralmente um subconjunto dos eventos recebidos; por exemplo, uma fonte que publica a localização de todas as pessoas em um edifício. Embora, deseje-se somente a localização de uma pessoa específica.

Transformador: este operador transforma o evento recebido em um evento de outro tipo; geralmente, o que ele faz é a elevação da abstração do contexto; por exemplo, ele recebe um evento contendo coordenadas GPS de algum local e publica um evento contendo o nome desse local.

Fusor: esse tipo de operador junta todos os eventos de entrada em um único evento de saída; por exemplo, ele se inscreve em todos os fluxos de eventos gerados pelos sensores de presença em cada sala e publica um fluxo com o status de todos eles.

Agregador: produz um fluxo de evento novo, baseado em um ou mais fluxos de eventos de entrada; por exemplo, produz um evento que diz se está ou não havendo uma reunião em uma determinada sala, com base nos sensores de presença daquela sala.

A Figura 3.5 ilustra um exemplo de grafo de operadores onde tem-se um conjunto de sensores de localização de pessoas (S) que passam por um operador do tipo transformador (T), o qual irá transformar as coordenadas obtidas nos nomes dos locais apontados. O agregador (A) *Building Locator* obtém a localização de todos os fluxos de eventos a partir do operador do tipo fusor (M) e só gera eventos quando ocorre mudança do local onde uma pessoa está. O evento gerado conterá somente as alterações. Já o agregador (A) 215 Monitor, que tem como entrada o fluxo do tranformador de uma fonte associada a uma determinada pessoa, produz eventos somente quando essa pessoa entra ou sai da sala 215.

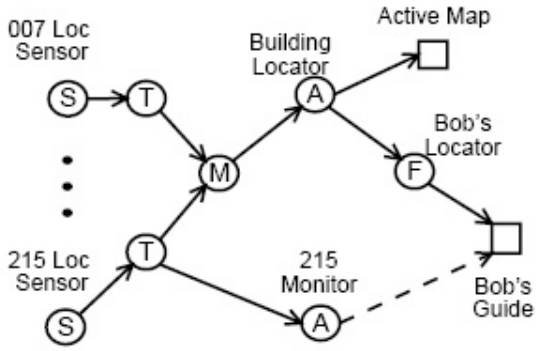

Figura 3.5: *Exemplo de um Grafo de Operadores*

3.3.1 Arquitetura

A arquitetura Solar é composta conforme ilustra a Figura 3.6, sendo formada por Estrelas, Planetas, Fontes e Operadores.

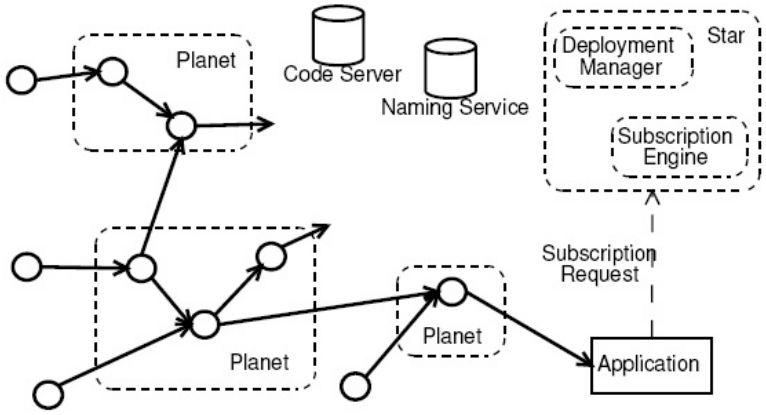

Figura 3.6: *Aquitetura do Solar*

Uma Estrela centraliza o processo de inscrição de uma aplicação em determinado contexto e é responsável por distribuir os Operadores pelos Planetas apropriados. A Estrela é responsável, também, por manter uma representação de todos os grafos de operadores. Essa representação é muito importante, pois quando uma aplicação requisita sua inscrição em um contexto, ela passa uma descrição de um grafo acíclico contendo as fontes e os operadores necessários para produzir o contexto de seu interesse. Como pode haver sobreposição de fontes e operadores entre aplicações, mesmo em um sub-grafo, a representação mantida pela Estrela permitirá o reuso dos componentes. Quando não há possibilidade de reuso de todos ou parte dos operadores, a Estrela deve escolher o Planeta onde irá executar o objeto operador necessário e inicializá-lo.

Nos Planetas são executados os objetos Operadores e alguns tipos de Fontes. Os planetas também são responsáveis por manter a lista de inscritos para cada operador local, incluindo outros operadores e aplicações. Quando um operador publica um evento, o Planeta o entrega a todos os inscritos para aquele operador. De modo prático, como o Solar foi implementado em Java, os operadores são objetos Java e os planetas são as máquinas virtuais Java. As aplicações fazem acesso a essa arquitetura usando uma biblioteca disponibilizada pelo Solar. Elas são consideradas externas ao sistema.

O Solar provê, ainda, formas para que os desenvolvedores possam estender o sistema, desenvolvendo novos tipos de Fontes e Operadores. Cada Fonte e Operador tem um nome identificador que é o mesmo usado pelas aplicações para a descrição de seu grafo no momento da inscrição. Essa flexibilidade se completa com o Servidor de Código, onde ficam as implementações dos Operadores e Fontes. Esse servidor permite a carga dinâmica desses elementos em qualquer um dos Planetas disponíveis.

3.3.2 Comparações

A mobilidade do código no Solar está concentrada nos componentes da arquitetura, mais especificamente nas Fontes e nos Operadores. Ela é explorada através do escalonamento e re-alocação dinâmica desses componentes. As implementações são armazenadas em um repositório central, de onde são copiadas para um dado Planeta e executadas. O ContextGrid explora a alocação dinâmica e ciente de contexto nos diversos nós da grade, fazendo uso, no caso das aplicações, de uma abordagem baseada em um repositório central. Entretanto, são exploradas outras formas de mobilidade de código ciente de contexto, através do uso de agentes móveis, sendo aplicada tanto para aplicações quanto para alguns componentes do próprio *middleware*.

De modo semelhante o Solar e o ContextGrid provê suporte para a carga dinâmica de novos componentes, visando estender o poder do sistema. Contudo, diferentemente do Solar, que tem como elemento fundamental, nesse contexto, os Operadores, o ContextGrid trabalha em uma granularidade mais elevada. Ele tem como elemento fundamental os interpretadores de contexto. Em comparação ao Solar, um interpretador conteria todo um grafo acíclico necessário para produzir um contexto. Assim como é permitido que a saída de um Operador seja consumida por outro, no ContextGrid pode-se encadear uma séries de interpretadores.

Um outro ponto importante no Solar se refere à relação entre seu componentes. Um componente gerenciador coordena as primeiras etapas da inicialização de uma aplicação. A partir desse ponto, as aplicações conversam diretamente com os elementos que lhe prevêem o contexto, sendo que estes elementos podem se comu-

nicar entre si, quase que de maneira ad-hoc. No trabalho proposto nesta dissertação, é desenvolvida uma infra-estrutura de contexto que é responsável por gerenciar todas as etapas, desde a inscrição de um componente do *middleware* até a entrega final do contexto a este. Em ambas as abordagens, a disseminação do contexto é feita através de eventos assíncronos. No entanto, são diferentes no foco final do contexto e das adaptações: enquanto no Solar o foco é a aplicação, no ContextGrid o foco é o *middleware* e a integração com dispositivos móveis.

3.4 Considerações

Assim como os trabalhos apresentados neste capítulo tiram proveito do uso de contexto e adaptação dinâmica para prover suporte a um ambiente altamente dinâmico e distribuído para coloboração ou simplesmente execução distribuída. O ContextGrid propõem um abordagem que permeia várias características apresentadas e comentadas para cada trabalho visando obter, para computação em grade, os mesmos benefícios já verificados em sistemas distribuídos de uso geral, um vez que as grades são um tipo computação distribuída.

CAPÍTULO 4

ContextGrid

As grades computacionais [29] podem ser constituídas por diversos tipos de dispositivos. Mais recentemente, dispositivos móveis têm sido integrados às grades computacionais fixas [53], representando um novo cenário. A exploração deste cenário depara-se com a natureza altamente dinâmica dos dispositivos e suas diversas restrições. Em ambientes extremamente dinâmicos e diversificados como este, uma arquitetura sensível ao contexto [22], capaz de reagir e adaptar-se, possibilita um melhor aproveitamento dos recursos envolvidos e agregação de variados tipos de dispositivos e recursos.

Este capítulo descreve a arquitetura, protocolos, modelo de contexto e implementação do ContextGrid [19], cujo objetivo é prover um infra-estrutura de suporte a contexto e adaptação ao *middleware* de grade MAG/InteGrade [34, 47].

4.1 Visão Geral

O ContextGrid [20] é uma infra-estrutura de suporte para computação sensível ao contexto e à adaptação dinâmica, desenvolvida como uma extensão do MAG/InteGrade. Ele explora as potencialidades da computação sensível ao contexto para prover adaptação dinâmica aos componentes do *middleware* de grade, visando a integração de dispositivos móveis às grades, assim como permitindo um melhor uso dos recursos disponíveis. O ContextGrid tem como objetivos: (*i*) a definição de uma arquitetura de suporte à computação sensível ao contexto, com adaptação dinâmica integrada ao *middleware* de grade, (*ii*) explorar o uso de Contexto na execução de aplicações em grades, e (*iii*) usar o Contexto para prover suporte a adaptações dinâmicas dos componentes do *middleware* de grade.

Em linhas gerais, o ContextGrid pode ser representado pela Figura 4.1 que fornece uma visão geral da arquitetura. Pode-se observar que a grade computacional é agora composta por uma Infra-estrutura de Contexto (IC), onde encontram-se os componentes responsáveis pelos serviços de contexto. A IC é alimentada por informações contextuais provindas tanto do ambiente quanto dos dispositivos

participantes da grade (2). Essas informações são processadas pela IC (4), que as interpretam e as transformam em contextos úteis aos outros componentes. Ao receber uma solicitação de execução de uma aplicação (1), o componente do *middleware* da grade responsável pelo escalonamento "A", faz uso dos serviços de contexto (3) para avaliar e interpretar o contexto da aplicação juntamente com o contexto atual da grade, de modo a executar um melhor escalonamento. Assim, como o contexto é considerado no inicio da computação, ao fim da mesma, ele é novamente avaliado (5) e, caso necessário, o resultado é adaptado pelo módulo "B" para, então, ser entregue (6). Os componentes do *middleware*, representados aqui por "C", também podem usar a infra-estrutura de contexto para se auto-adaptarem (7). Como por exemplo, migrarem de dispositivo quando este apresentar um contexto desfavorável aos seus requisitos. Outro exemplo seria a mudança da freqüência de mensagens trocadas entre os componentes que executam em dispositivos provedores de recursos e o componente de gerenciamento. Isso poderia ocorrer diante de um contexto onde os enlaces de rede apresentassem congestionamento.

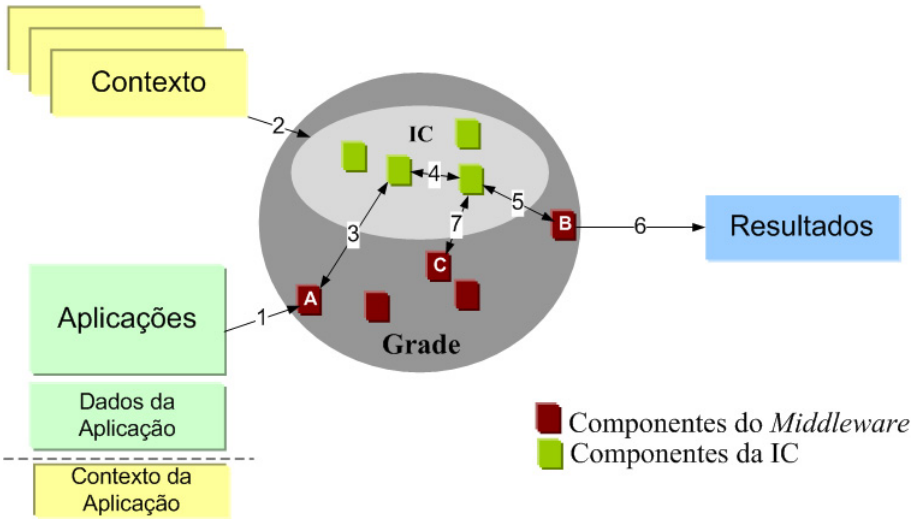

Figura 4.1: *Visão geral do ContextGrid*

4.2 Arquitetura

A arquitetura do ContextGrid foi desenvolvida tomando como base a arquitetura MAG, de forma a manter uma interação direta com o InteGrade. Muitos dos serviços básicos com os quais o ContextGrid irá atuar não estão na arquitetura MAG, como por exemplo, o escalonador de tarefas. A Figura 4.2 ilustra o modelo de camadas no qual está inserido o ContextGrid.

A camada do ContextGrid acrescenta novos mecanismos de execução de aplicações ao MAG/InteGrade, que passa a considerar o contexto dos nós envolvidos

diretamente com a submissão da tarefa, bem como o contexto geral da grade e o da aplicação. Ele introduz também novos recursos e serviços visando melhorar a integração com dispositivos móveis, o escalonamento e a interação com o usuário. Não é abordado nesse trabalho a oferta de serviços diretamente às aplicações de usuário. Além da execução de aplicações, o ContextGrid atua também na adaptação dos componentes do *middleware*.

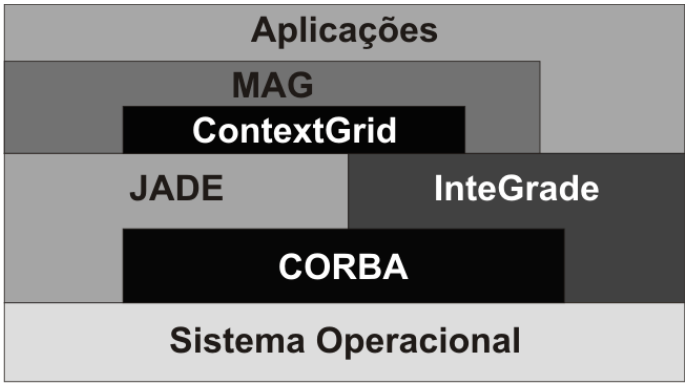

Figura 4.2: *Arquitetura em camadas do ContextGrid*

Na arquitetura proposta, a adaptação é responsabilidade do desenvolvedor de componentes para o *middleware* de grade. São providos meios através dos quais os módulos desenvolvidos podem tomar conhecimento do contexto e de suas mudanças, ficando a cargo do desenvolvedor decidir qual(ais) contexto(s) é(são) interessante(s) e qual(ais) ação(ões) deve(m) ser tomada(s) diante de um determinado contexto ou de sua mudança. Isso torna a arquitetura independente das políticas de adaptação adotadas, sendo, portanto, mais flexível no que se refere à adaptação. Estas adaptações poder ser feitas através da implementação da interface discutida na Seção 4.7.1.

4.3 Componentes

Para compor a infra-estrutura do ContextGrid são introduzidos dois novos componentes principais: o LCM (*Local Context Manager*) e o GCM (*Global Context Manager*). Outros elementos importantes na arquitetura são os *Context Users*, *Proxies* e os *Wrappers*. A Figura 4.3 mostra o conjunto de módulos do MAG/Integrade, descritos nas seções 2.2.4 e 2.2.3, respectivamente, assim como os novos módulos mencionados acima e descritos a seguir.

LCM (*Local Context Manager*) : é responsável por coletar as informações de contexto das mais diversas fontes através dos *wrappers*. Ele realiza um pré-processamento dessas informações, a fim de convertê-las em um formato

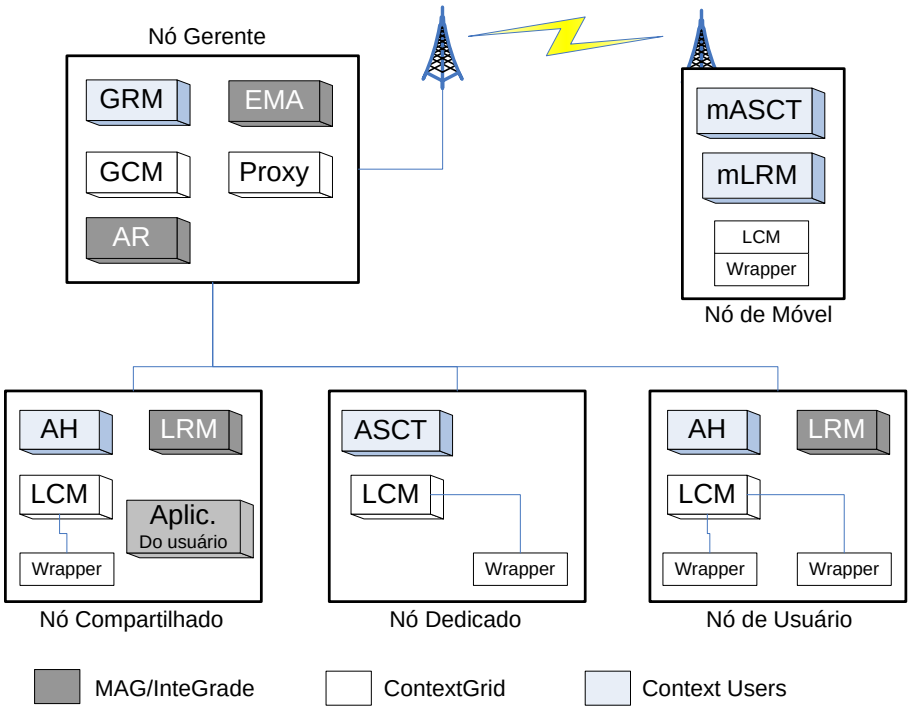

Figura 4.3: *Componentes do ContextGrid*

padronizado, em XML, para, posteriormente, enviá-las ao GCM. Isso torna a obtenção e a utilização das informações de contexto independentes das fontes ou do formato em que são fornecidas. O que permite, por exemplo, o uso dos mais variados tipos de dispositivos como fornecedores de informações, tais como sensores, PDAs e telefones celulares. Para isso, basta a criação de um novo *Wrapper* e seu registro no LCM, conforme descrito abaixo.

GCM (*Global Context Manager*) : é responsável por armazenar, processar e interpretar o contexto, transformando-o em tipos mais elaborados, seja pela junção de várias informações de contexto ou pela inferência de novas informações a partir daquelas obtidas dos LCMs. O GCM pode, por exemplo, a partir de informações geradas por um GPS (coordenadas), inferir o nome do local apontado por elas. Também é parte desse processamento a classificação dos contextos em categorias, adotando critérios como: similaridade, proximidade e/ou domínio administrativo. O GCM é responsável, ainda, por realizar as notificações de mudanças de contexto a todos os interessados. Com esta finalidade, é implementado o padrão de projeto *observer* (*publish/subscribe*) [43], que faz uso de canais de eventos por onde são feitas as notificações. Isto será abordado com maiores detalhes na Seção 4.5.

Context Users : podem ser quaisquer dos componentes do *middleware* de grade desde que tenham sido implementados ou alterados para fazer uso do contexto

e adaptarem-se ao mesmo. É dentro desses componentes que estão as políticas de adaptação e ações a serem tomadas em virtude das mudanças no contexto. Uma vez que, no escopo deste trabalho, as adaptações serão de responsabilidade dos desenvolvedores de componentes para o *middleware*,foram alterados alguns componentes do MAG e do Integrade para usarem contexto, de modo a atingir alguns dos objetivos do trabalho, como a integração de dispositivos móveis. Essas alterações também têm por objetivo, ilustrar o funcionamento da arquitetura, conforme será visto na Seção 4.7.6.

Wrapper : é responsável pela comunicação direta com as fontes de informações contextuais. Ele traduz os dados do formato das fontes para um formato que o LCM é capaz de entender. Cada tipo de sensor ou dispositivo que fornece informações contextuais possui um *wrapper* específico. Eles têm uma relação dinâmica com o LCM, de modo que novos *wrappers* podem ser adicionados para expandir as fontes de informações reconhecidas pelo LCM.

Proxy : o ContextGrid usa uma abordagem baseada em *proxy* para a integração com dispositivos móveis [53]. Este elemento é responsável por intermediar todas as atividades dos dispositivos móveis com a grade. Ele é o representante do dispositivo na rede fixa. O *Proxy*, em última instância, atua com se fosse o próprio dispositivo. Ele é responsável por aplicar adaptações, principalmente de conteúdo e comunicação, às interações entre a grade e os dispositivos móveis. Por exemplo, ele pode decidir enviar apenas o cabeçalho do resultado de uma computação ao dispositivo móvel, diante de um contexto onde há pouquíssima memória disponível para aquele dispositivo. Como representante do dispositivo na rede fixa, o *Proxy* também é responsável por assumir as funcionalidades de um dispositivo nele registrado, cujo contexto de comunicado não seja favorável ou no caso de uma desconexão total. Nesse cenário, pode haver uma avaliação do contexto do usuário que utilizava o dispositivo para decidir a melhor maneira de enviar os dados recebidos durante o período de desconexão.

4.4 Contexto no ContextGrid

No ContextGrid, a definição de contexto utilizada é aquela proposta por Dey [22], discutida no Capítulo 2.

O desenvolvimento de aplicações sensíveis ao contexto é uma tarefa complexa e o uso de ferramentas de modelagem e engenharia de *software* pode auxiliar bastante. Entretanto, por se tratar de um conceito relativamente novo, os atuais modelos para desenvolvimento de *software* não oferecem suporte para a modelagem

de tais aplicações. Novas técnicas vêm sendo apresentadas por diversos autores, como mostrado no Capítulo 2. Entretanto há uma série de características sobre o contexto e sobre o poder de expressão que estas técnicas ou modelos devem apresentar.

4.4.1 Características da Informação Contextual

A fim de que se possa definir uma técnica de modalagem para o ContextGrid, temos que delinear bem as características e nível de expressividade que a técnica deve prover. Contudo, antes que isso seja feito, tem-se que compreender algumas características da informação contextual, algumas já descritas no decorrer do Capítulo 2. Neste ponto, serão abordadas apenas aquelas que influenciam diretamente no ContextGrid:

A informação contextual possui vários níveis de temporalidade: as informações contextuais podem se apresentar em várias formas desde aquelas puramente estáticas, como por exemplo, o *login* e o nome do usuário, que descrevem as características do ambiente que são, por definição, invariáveis, até aquelas altamente dinâmicas e voláteis que, em geral, constituem a maior parte do contexto. Alguns exemplos nesta última categoria são porcentagem de uso de CPU e de memória RAM. Um aspecto importante quando se trata da temporalidade da informação de contexto é a sua persistência, ou seja, por quanto tempo aquela informação é válida e representa, realmente, o estado atual.

A informação contextual nem sempre é confiável: as informações contextuais podem ser, em um dado momento e sob certas circunstâncias,incompletas, quando alguma parte do contexto não é conhecida; inconsistentes, quando contêm informações contraditórias; incorretas, quando não refletem o real estado do ambiente. Algumas das causas para estes tipos de problemas são: o ambiente altamente dinâmico e as falhas na conectividade e nos dispositivos.

A informação contextual pode ter diferentes níveis de abstração: a mesma informação contextual pode ter diferentes níveis de abstração. Informações que são monitoradas por sensores podem não ter o nível correto para que sejam úteis às aplicações. Ao mesmo tempo, diferentes aplicações podem requerer uma mesma informação em níveis diferentes. Por exemplo, uma aplicação pode requerer as coordenadas fornecidas diretamente por um sensor GPS, enquanto que outra pode requerer o nome da sala onde o usuário está, o que nada mais é que uma elevação do nível de abstração das coordenadas GPS do usuário.

A informação de contexto é inter-relacionada: Em geral, as informações contextuais são altamente relacionais e, algumas vezes, dependentes umas das outras.

Dadas as características apresentadas acima, somadas às questões discutidas no Capítulo 2, tem-se que uma técnica para modelagem de contexto deve ser capaz de, pelo menos, expressar o seguinte:

- entidades fornecedoras de contexto;
- atributos ou informações fornecidas pelas entidades;
- inter-relacionamentos (associações) entre entidades e entre entidades e seus atributos;
- temporalidade;
- qualidade e grau de confiança; e
- alternativas para representar a mesma informação em diferentes níveis de abstração.

Considerando isto e a avaliação da expressividade dos modelos discutida no Capítulo 2, será adotada neste trabalho a técnica e a notação gráfica apresentadas por [38]. Esta escolha se deve ao fato de que uma diagramação do contexto, de forma visual, traz inúmeras vantagens para sua compreensão e modelagem, além, do que este modelo possui o poder de expressão que é desejado para o ContextGrid. Outro aspecto que determina a escolha desse modelo em detrimento de outros mais complexos foram as características limitadas dos dispositivos envolvidos. Técnicas como modelagem de contexto orientada à objetos e baseadas em ontologia, exigiriam um poder de processamento substancial para sua manipulação [62]em dispositivos portáteis.

4.4.2 Modelagem do contexto para o ContextGrid

Observando-se as características discutidas nas seções anteriores e seguindo a notação gráfica apresentada por [38], a Figura 4.4 apresenta o modelo de contexto elaborado para o ContextGrid. Este modelo e maiores detalhes da notação utilizada são comentados a seguir.

Notação

A técnica adotada é baseada no modelo proposto por [38], que é fundamentado em uma abordagem objeto-relacional, na qual o contexto é modelado em

torno de um conjunto de entidades que são elementos abstratos e representam a fonte da informação contextual. Elas foram definidas de acordo com os objetivos do trabalho. Acerca dessas entidades são definidos atributos, que correspondem às informações contextuais sobre as mesmas. Além disso, são definidos os tipos de associações entre entidades e entre entidades e seus atributos. Os atributos coletados juntamente com seus associações, caracterizam o contexto do ambiente em um dado momento. Graficamente, as entidades são representadas por retângulos com bordas duplas, os atributos por retângulos com bordas simples e as associações ou inter-relacionamentos por um segmento de reta dirigido.

A informação contextual se apresenta de diversas maneiras, conforme exposto na Seção 4.4.1. Na notação utilizada as associações caracterizam como é uma informação de contexto. Assim as associações serão classificadas da seguinte forma:

Quanto à temporalidade:

- **Estáticas:** São as associações que são fixas e imutáveis durante todo o ciclo de vida da entidade; como por exemplo, a associação entre a entidade Dispositivo e o atributo Modelo;
- **Dinâmicas:** São todas aquelas que não são estáticas podendo mudar durante o ciclo de vida da entidade como, por exemplo, Largura de Banda;

Quanto à origem dos dados:

- **Definidas:** São informações fornecidas. Em geral, são estáticas ou apresentam dinamicidade relativamente baixa, ou seja, que mudam com pouca freqüência como, por exemplo, Login;
- **Monitoradas:** São aqueles obtidos através de sensores de *hardware* ou de *software*. Em geral possuem um baixo nível de abstração e o formato varia de sensor para sensor. Em sua maioria, necessitam de um processamento para que sejam padronizadas e para que tenham seu nível de abstração elevado como, por exemplo, Recursos;
- **Derivadas:** São obtidos através de uma função de derivação, a partir de uma ou mais associações como, por exemplo, a associação entre as entidades Usuário e Dispositivo, denominada "Está próximo de". Esse tipo de associação é apresentado juntamente com a indicação das outras associações das quais ela depende.
- **Depende de:** Quando a associação é derivada, ela se associa a outra com este tipo, significando que seu valor depende do valor da outra associação.

Quanto à estrutura:

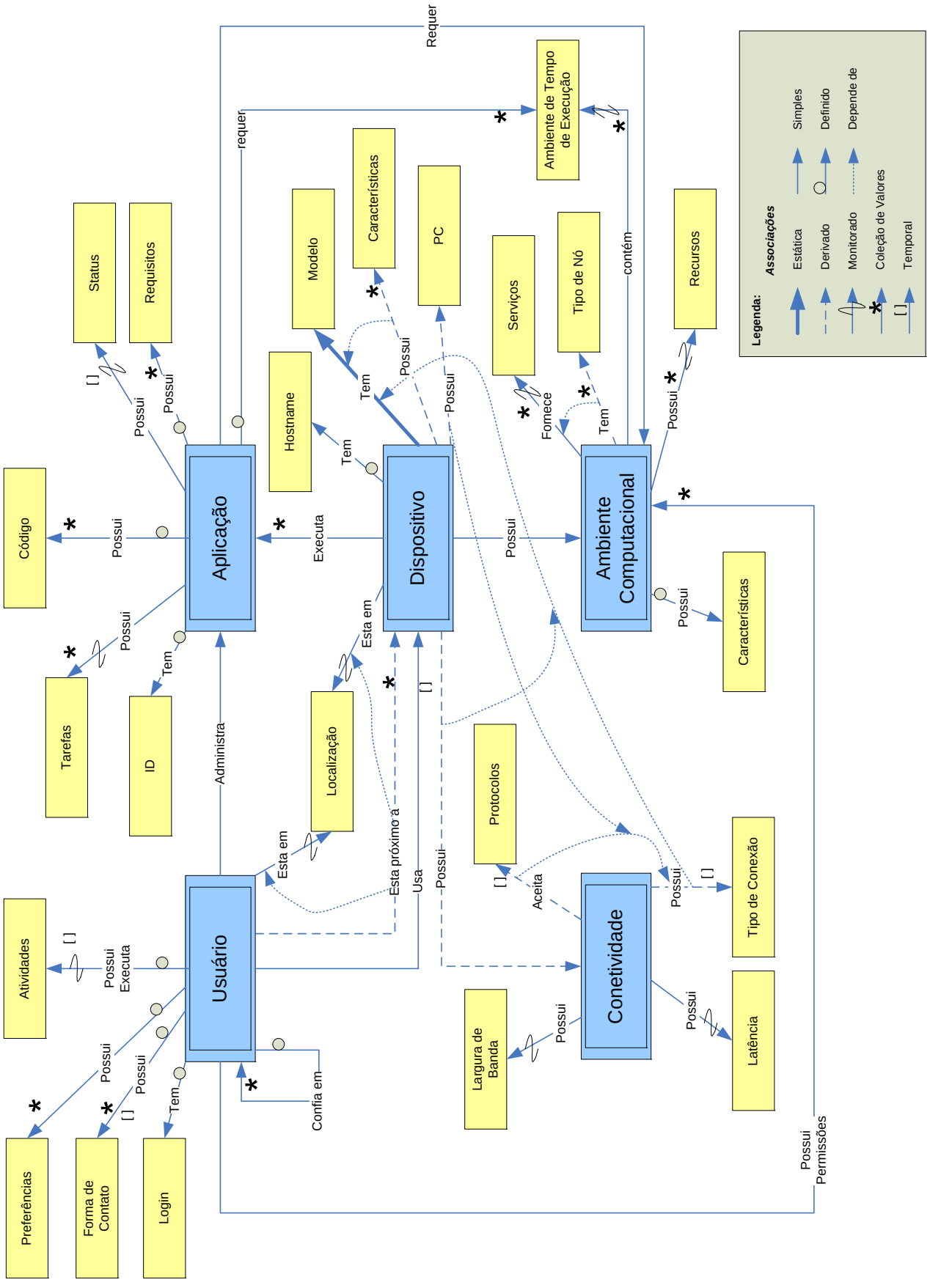

Figura 4.4: *Modelo de Contexto do ContextGrid*

- **Simples:** Possui um único valor; como por exemplo, Login;
- **Coleção de valores:** Possui vários valores ao mesmo tempo, em geral, baseados em uma estrutura chava-valor como, por exemplo, Preferências;
- **Temporal:** Possui um conjunto de valores, mas assume somente um deles por um dado período como, por exemplo, a associação entre a entidade Usuário e Dispositivo denominada "Usa".

O Modelo

Dada a notação e a semântica apresentada são discutidas a seguir as Entidades, Atributos e as Associações do modelo de contexto do ContextGrid apresentado na Figura 4.4.

Usuário: representa os usuários da grade computacional, devidamente registrados e com permissões para uso da mesma. Seus atributos e associações são:

- **Atividades:** são as atividades que estão sendo executadas pelo usuário em um dado instante ou foram agendadas por ele. Cada registro possui os seguintes campos: atividade, hora aproximada de início e hora aproximada de término. Também são aceitos os seguintes valores: atividade indefinida, início indeterminado e fim indeterminado.
  Tipo de Associação:
  Definido: o usuário pode agendar atividades.
  Temporal: em um dado instante, o usuário está executando uma única tarefa.
  Monitorado: pode-se obter as tarefas do usuário monitorando sua agenda ou as aplicações que estão sendo usadas por ele.
- **Forma de Contato:** são listadas as formas de contato com o usuário, ou seja, quais meios podem ser utilizados para fazer com que uma informação qualquer chegue a ele. Possui, para cada entrada, os seguintes dados: meio, prioridade e dados para realizar a comunicação. São aceitos os seguintes meios: E-mail, SMS e Conexão direta com o dispositivo em uso pelo usuário.
  Tipo de Associação:
  Coleção de valores.
  Definido: o usuário deve listar todas as formas de contato desejadas, bem como a prioridade de cada uma.
- **Localização:** nome do ambiente onde o usuário se encontra.

Tipo de Associação:
Monitorado: é obtida por meio de um sensor de localização. Para este trabalho, houve apenas uma simulação baseada na localização do dispositivo de onde partiu o último *login* do usuário. Caso esse dispositivo seja móvel, a localização assume o valor de indeterminada.

- **Login:** nome de *login* do usuário.
  Tipo de Associação:
  Definido: é associado no momento do cadastro do usuário na grade.
- **Preferências:** são listadas algumas preferências do usuário. Foram abordadas nesse trabalho as seguintes opções: qualidade mínima para exibição de imagens, nível de bateria abaixo do qual o dispositivo deve ser totalmente liberado, tipo preferencial de conexão, uso ou não de enfileiramento de tarefas antes de enviá-las a grade.
  Tipo de Associação:
  Coleção de valores
  Definido: o usuário escolhe para cada preferência a opção que melhor lhe atende.

Aplicação: Representa as aplicações que estão em execução na grade ou cuja execução foi solicitada. Entende-se por execução todo o ciclo de vida da aplicação, a partir do momento em que o usuário a requisita na interface do ASCT. Os atributos e associações dessa entidade são:

- **Ambiente de Tempo de Execução:** ambiente de tempo de execução exigido pela aplicação. Cada entrada possui os seguintes dados: referência para o código binário da aplicação, ambiente, versão e lista de bibliotecas exigidas. Os ambientes aceitos até o momento são: JVM, Python, Bash.
  Tipo de Associação:
  Coleção de valores
  Definido: No momento do registro da aplicação, devem ser informados todos os ambientes suportados pela aplicação.
- **Código:** para cada aplicação podem binários para diferentes arquiteturas e com diferentes versões. Cada registro contém: referência para o binário, arquitetura e versão.
  Tipo de Associação:
  Coleção de valores
  Definido: Durante o registro da aplicação são registrados, também, os binários.

- **ID:** cada aplicação registrada com um ID único, que a identifica na grade.
  Tipo de Associação:
  Definido: atribuído no momento do registro da aplicação.
- **Requisitos:** cada aplicação possui um conjunto de requisitos mínimos necessários e opcionais. Os requisitos podem ser definidos para cada binário de independente. Cada registro contém: referência para o binário, requisito, valor, tipo (necessário ou opcional).
  Tipo de Associação:
  Coleção de valores
  Definido: durante o registro da aplicação, os requisitos dever ser preenchidos. Eles podem ser alterados temporariamente no momento da solicitação de execução da aplicação na grade.
- **Status:** armazena o status global da aplicação. São aceitos os seguintes valores: executando, finalizada, suspensa e falhou.
  Tipo de Associação:
  Temporal: dentre os vários estados possíveis, só assume um a cada momento.
  Monitorado: pode-se obter o estado atual através do monitoramento de cada tarefa em execução.
- **Tarefas:** quando uma aplicação é executada na grade, cada instância que efetivamente executa é chamada, na arquitetura MAG/InteGrade, de tarefa. Este atributo com a lista de todas as tarefas da aplicação em execução na grade. Cada registro contém: referência para o binário, nó em que executa, parâmetros de execução e status.
  Tipo de Associação:
  Coleção de valores
  Monitorado: podem ser obtidos pelo monitoramento de cada solicitação de execução e do monitoramento dos nós da grade.

Dispositivo: Esta entidade reflete os dispositivos de *hardware* e seus atributos pouco variáveis que compõem a grade. São, por exemplo, os computadores pessoais de uma rede já instalada, dispositivos móveis, impressoras ou um simples sensor que forneça algum recurso à grade. Seus atributos e associações são:

- **Características:** para cada modelo de dispositivo existem inúmeras características associadas. Neste trabalho, as características foram limitadas a: tamanho da tela, arquitetura do processador, freqüência máxima do

processador, quantidade de cores suportadas e tipo de dispositivo para entrada de dados (teclado convencional, mouse ou teclado de telefone).
Tipo de Associação:
Coleção de valores
Derivado: o valor de cada característica depende do modelo do dispositivo. Os valores são carregados de um banco de dados contendo os modelos e suas características.
Depende de: Tem-Modelo

- **PC:** cada dispositivo possui um conjunto da dos que diz como se conectar a ele, ou seja, um ponto de conexão (PC). Para esse projeto os PC suportados são: IP mais porta e endereço *bluetooth* mais canal.
  Tipo de Associação:
  Derivado: os dados do PC são preenchidos de acordo com o tipo de conexão.
  Depende de: Possui-Tipo de Conexão
- **Modelo:** a cada dispositivo está associado um modelo
  Tipo de Associação:
  Estático: é uma característica constante durante toda existência do dispositivo.
- **Localização:** nome do ambiente onde esta o dispositivo.
  Tipo de Associação:
  Monitorado: é obtida por meio de um sensor de localização. Para este trabalho, houve apenas uma simulação baseada em um cadastro prévio dos dispositivos e suas localizações. Caso esse dispositivo seja móvel, a localização assume o valor de indeterminada.
- **Hostname:** nome associado ao dispositivo. Deve ser único na rede.
  Tipo de Associação:
  Definido: é associado no momento da conexão do dispositivo a rede.

Ambiente Computacional: Representa o ambiente computacional presente nos dispositivos que compõem a grade, ou seja, do nível do sistema operacional acima. Os atributos e associações de interesse para esta entidade são:

- **Ambiente de Tempo de Execução:** lista todos os ambientes de tempo de execução que o ambiente computacional possui. Cada entrada possui os seguintes dados: ambiente, versão e lista de bibliotecas. Os ambientes considerados são: JVM, Python, Bash.
  Tipo de Associação:
  Coleção de valores

Monitorado: obtido e atualizado através do monitoramento do que está instalado no dispositivo.

- **Características:** Detalha as características do ambiente computacional. Atualmente estão limitadas a: sistema operacional e versão do sistema operacional.
  Tipo de Associação:
  Definido: é definido no momento da inicialização do ambiente computacional.
- **Recursos:** o ambiente computacional exporta recursos da grade. Os tipos de recursos aceitos até o momento são: processador, memória e espaço em disco.
  Tipo de Associação:
  Coleção de valores
  Monitorado: um sensor de *software* é encarregado de monitorar constantemente esses recursos e atualizar este atributo.
- **Serviços:** lista de todos os serviços fornecidos pelo ambiente computacional de um dispositivo (servido htttp, banco de dados ou aceita tarefas para executar).
  Tipo de Associação:
  Coleção de valores
  Monitorado: são obtidos pelo monitoramento constante de tudo que está em execução em um dado momento.
- **Tipo de Nó:** na arquitetura MAG/InteGrade cada nó da grade pode ser classificado de uma forma, dentre elas: nó de usuário, nó compartilhado, nó dedicado e nó gerente. Um dispositivo pode assumir mais de um tipo de nó ao mesmo tempo.
  Tipo de Associação:
  Coleção de valores
  Derivado: com base no conjunto de serviços fornecidos pelo ambiente computacional pode-se obter o tipo de nó.
  Depende de: Fornece-Serviços

Conectividade: Esta entidade representa o tipo de conectividade que pode existir nos dispositivos que compõem a grade. Seus atributos e associações são:

- **Largura de Banda:**Largura de banda disponível em um dado momento.
  Tipo de Associação:
  Monitorado

- **Latência:** Tempo de resposta da conexão. Baseado no tempo gasto para um pacote enviado retornar ao nó de origem (*ping*).
  Tipo de Associação:
  Monitorado
- **Protocolo:** Protocolos de comunicação aceitos. Atualmente estão sendo explorados apenas TCP e UDP.
  Tipo de Associação:
  Temporal: só pode, em um dado instante, usar um dos protocolos citados.
  Derivado: depende do tipo de conexão atual.
  Depende de: Possui-Tipo de conexão.
- **Tipo de Conexão:** tipos de conexão aceitos pelo dispositivo. Neste trabalho eles estam limitados a: *Wireless* 802.11x, *Bluetooth* e *Ethernet/Fast Ethernet*.
  Tipo de Associação:
  Temporal: em um dado instante somente um dos tipos esta em uso.
  Derivado: Obtido, com base no modelo do dispositivo.
  Depende de: Tem-Modelo

Por fim, a Tabela 4.1 apresenta e detalha as associações inter-entidades.

4.4.3 Interpretação do Contexto

Um ponto que merece destaque quando se trata de aplicações sensíveis ao contexto é a interpretação do contexto. Dey [22] trata a interpretação do contexto como um processo onde se eleva o nível de abstração das informações contextuais, onde se gera informações mais elaboradas tendo como ponto de partida informações primitivas. A interpretação do contexto usa a abstração, o refinamento, a agregação, a derivação e a inferência sobre as informações contextuais, tendo como objetivo melhorar a compreensão de um contexto pelas aplicações.

No ContextGrid este aspecto é tratado no LCM e no GCM. O primeiro passo da interpretação de contexto ocorre no LCM, onde as informações de contexto são transformadas em um padrão uniforme baseado em XML e independente da fonte de onde provieram. Essa informação, agora chamada contexto, é então passada ao GCM onde, com o uso de regras de composição e transformação, o nível de abstração do contexto é elevado ainda mais. Por exemplo, as informações de coordenadas de um GPS, são transformadas no nome do local a que se referem.

Para dar suporte a essas manipulações, o ContextGrid permite a carga dinâmica dos interpretadores para cada atributo de contexto ou para um conjunto de atributos de contexto. A carga dinâmica desses interpretadores é realizada através

Associações			
Nome	**De**	**Para**	**Descrição**
Usa	Usuário	Dispositivo	Um usuário usa (esta logado) um dispositivo em um dado momento. *Temporal:* O usuário só pode usar um dispositivo por vez.
Administra	Usuário	Aplicação	O Usuário pode incluir, excluir e solicitar a execução de uma aplicação.
Está próximo de	Usuário	Dispositivo	Um usuário pode estar próximo a outros dispositivos. *Coleção de valores*; *Derivado:* depende da localização do usuário e dos dispositivos; *Depende de:* Esta em-Localização (Usuário) e Esta em-Localização (Dispositivo).
Requer	Aplicação	Amb. Comp.	Para executar uma aplicação necessita de um ambiente computacional para executar.
Executa	Dispositivo	Aplicação	Um dispositivo pode executar várias aplicações. *Coleção de valores.*
Permissões	Usuário	Amb. Comp.	O usuário possui um contjunto de permissões de acesso em um ambiente computacional. *Coleção de valores*;
Possui	Dispositivo	Amb. Comp.	O dispositivo possui um ambiente computacional instalado nele.
Possui	Dispositivo	Conectividade	Um dispositivo pode possuir *hardware* que lhe proporciona conectividade com outros. *Derivado:* Ter ou não essa conectividade depende do modelo do dispositivo; *Depende de:* Tem-Modelo.
Confia em	Usuário	Usuário	Um usuário confia em outros usuários. Em sua ausência outros podem ser contatados em seu lugar. *Coleção de valores*; *Definido:* o usuário deve informar sua rede de confiança.

Tabela 4.1: *Associações inter-entidade*

do registro dos mesmos e do(s) atributo(s) ao(s) qual(ais) está(ão) vinculado(s), assim como uma referência para sua implementação, junto ao GCM. O GCM, ao receber um contexto do LCM, verifica quais atributos esse contexto contém e se existe um interpretador em particular para esse atributo. Caso haja, o GCM faz a carga desse interpretador e repassa as informações daquele atributo a ele. Ao final do processamento, o interpretador retorna um novo contexto ao GCM no mesmo modelo passado pelo LCM. O processo é repetido. O contexto em seus diversos níveis, isto é, antes e o de depois de passar pelos interpretadores, são armazenados ao longo do processo.

4.5 Protocolos

Os componentes do ContextGrid assim como os componentes do MAG e Integrade, colaboram de maneira a atingir os objetivos propostos. Nessa seção, são detalhados os principais protocolos que ditam, de maneira consistente, como essa colaboração ocorre.

Para prover suporte a contexto e adaptação dinâmica integrado ao MAG, o ContextGrid faz uso dos componentes GCM e LCM aliados aos *Context Users*, descritos na Seção 4.3, entre os quais existem dois protocolos: um protocolo de atualização e outro de notificação de contexto.

4.5.1 Protocolo de Atualização de Contexto

O Protocolo de Atualização de Contexto permite que o GCM mantenha o contexto global do aglomerado. As informações de contexto atualizadas incluem todas aquelas apresentadas na Seção 4.4.2, que não são do tipo derivado ou estáticas.

Um aspecto importante desse protocolo é a periodicidade com que as informações de contexto são atualizadas. Caso essas atualizações sejam freqüentes demais, os dispositivos móveis teriam suas baterias drenadas e a rede poderia ser tomada por estas mensagens, comprometendo todo o desempenho do sistema. Por outro lado, se o intervalo for grande demais, o GCM não teria um contexto que refletisse o atual estado do sistema. Dessa forma, utiliza-se tempos e métricas distintas para decidir quando uma informação provinda de um sensor deve ser repassada ao GCM. Como a interface com os sensores é feita pelo LCM, é nele que, para cada sensor, são associados um intervalo, prioridades e métricas que indicarão quando a informação é passada ao GCM. Uma métrica seria, por exemplo, a percentagem de variação acima da qual o contexto deve ser atualizado. O intervalo

de atualização também pode ser regido por uma métrica como, por exemplo, largura de banda disponível ou a porcentagem de uso da CPU.

A Figura 4.5 ilustra o protocolo de atualização de contexto. Apenas para distinção, o componente denominado mLCM representa um LCM executando em um dispositivo móvel, enquanto que aquele denominado LCM executa no mesmo nó em que reside o *mobileProxyAgent*. O mLCM, ao detectar mudanças significativas nas informações de contexto de sua responsabilidade, faz uma chamada para atualizar as informações no GCM. Esta requisição é intermediada pelo *proxy* ao qual o dispositivo móvel está associado (1). O *proxy* verifica e realiza as mudanças pertinentes à mensagem (2) e a repassa para o GCM (3). Os LCM situados em nós fixos realizam esta atualização diretamente com o GCM (5). Ao receber atualizações de informações de contexto, o GCM realiza o processamento (4, 6) pertinente, atualizando o(s) contexto(s) dependente(s) da informação recebida.

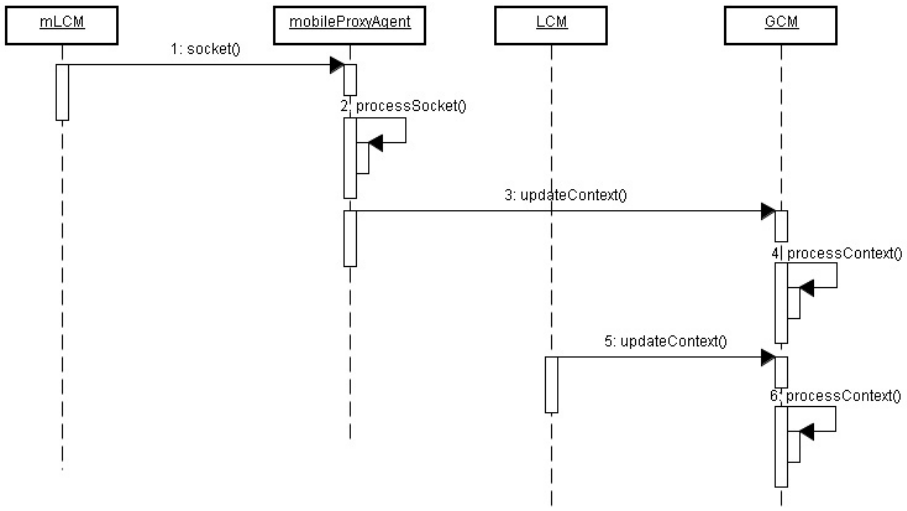

Figura 4.5: *Protocolo de Atualização de Contexto*

O processamento realizado pelo GCM se refere à interpretação do contexto discutida na Seção 4.4.3.

4.5.2 Protocolo de Notificação de Contexto

O Protocolo de Notificação de Contexto permite ao ContextGrid disseminar o contexto pela arquitetura, fazendo com que ele chegue aos *Context Users*, os quais dependem dele para executarem suas políticas de adaptação.

A notificação de contexto no ContextGrid envolve os componentes LCM, GCM e os *Context Users* interessado(s) em determinado(s) contexto(s) e suas mudanças. Esse protocolo faz uso do padrão de projeto *Observer* [43], que é baseado em canais de eventos assíncronos, nos quais os interessados devem se inscrever. Ele

é dividido em duas partes, a inscrição em um canal de eventos e a Notificação propriamente dita de mudanças de contexto.

Inscrição em um Canal de Eventos

Esta parte do protocolo trata da realização da inscrição de um *Context User* em um ou mais canais de eventos disponíveis que melhor refletem o contexto de interesse. A Figura 4.6 ilustra essa parte do protocolo.

O mASCT, componente de submissão de tarefas, quando está interessado em contexto, deve realizar um pedido ao LCM local (mLCM) (1) para a inscrição em algum canal de evento. O mLCM repassa o pedido para o *proxy* ao qual o dispositivo móvel está associado (2). O *proxy* verifica e realiza as mudanças pertinentes à mensagem (3) e a repassa para o GCM (4). Outro componente, no caso o *mobileProxyAgent*, também pode realizar um pedido ao LCM local (5) para inscrição em algum canal de evento. Ao receber um pedido de inscrição em algum canal, o LCM realiza sua inscrição junto ao GCM no canal ou canais solicitados (6). Caso o LCM já tenha se inscrito em um canal solicitado, ele apenas atualiza sua lista local de interessados.

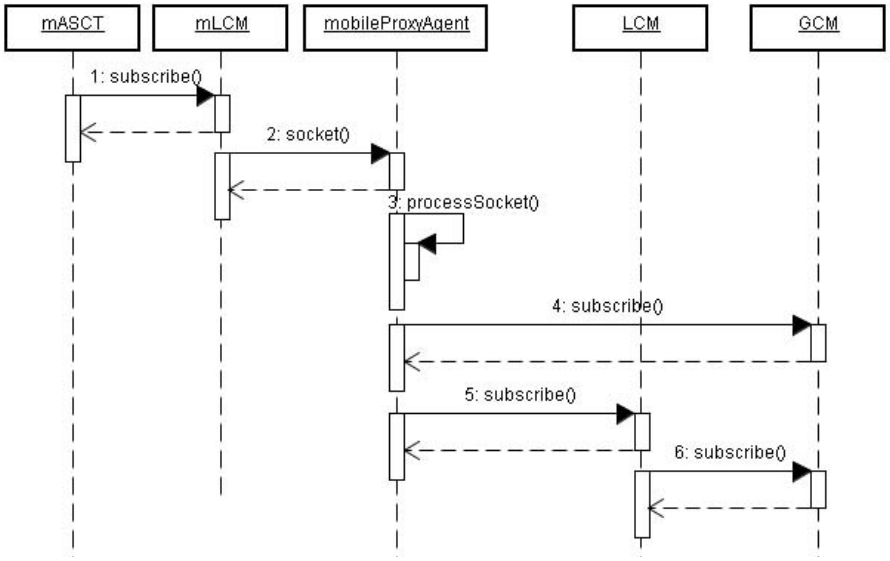

Figura 4.6: *Protocolo de Disseminação de Contexto - Subscribe*

Notificação de Mudanças no Contexto

Esta é a segunda parte do Protocolo de Notificação de Contexto, que se refere a forma como é feita a notificação dos *Context Users* quando ocorre uma mudança no contexto. Quando alguma das informações de contexto sofre alteração,

o sistema é atualizado conforme o Protocolo de Atualização de Contexto descrito na Seção 4.5.1. Com essas novas informações, o GCM faz o processamento e, após gerar um novo estado válido de contexto, notifica as mudanças ocorridas através dos canais de eventos. A escolha do canal ou canais no(s) qual(ais) as notificações serão postadas, depende de uma determinação prévia, discutida mais adiante na Seção 4.6. A Figura 4.7 ilustra esta segunda parte do Protocolo de Notificação de Contexto.

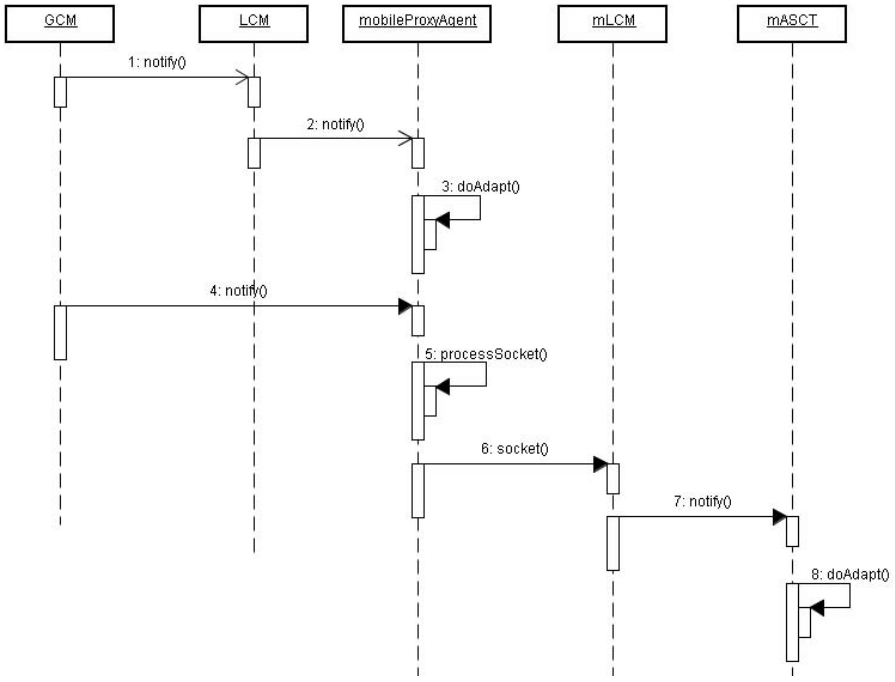

Figura 4.7: *Protocolo de Disseminação de Contexto - Notify*

Ao dar início a uma sessão de notificação, o GCM envia mensagens assíncronas em todos os canais para os quais há LCMs inscritos e houve mudança no contexto (1). O LCM, ao receber esta mensagem, repassa a notificação a todos os interessados inscritos em suas listas internas para aquele canal (2 e 7). Se esta notificação é destinada a um dispositivo móvel (4), ela é interceptada pelo *mobileProxyAgent*, devendo ser adaptada (5) e, posteriormente, repassada ao dispositivo (6). De posse do novo contexto, os componentes realizar as adaptações determinadas por suas políticas (3 e 8).

O uso do LCM como intermediário para o GCM,tem como um de seus objetivos diminuir o tráfego na rede, aumentando a escalabilidade do sistema, uma vez que ele recebe um única notificação do GCM para um dado canal, mesmo que exista inúmeros componentes inscritos no LCM para o canal em questão. Como um aglomerado pode ter centenas de máquinas e em cada máquina pode haver um grande número de componentes pertencentes ao *middleware*, o volume de mensagens disparadas pelo GCM pode inundar a rede. Outro objetivo do LCM é

permitir um menor acoplamento dos componentes em relação ao nó gerentes do aglomerado, bastando que o desenvolvedor dos componentes para o *middleware* conheça a interface do LCM.

4.6 Canais de Eventos

Para que o Protocolo de Notificação de Contexto funcione, é necessário que haja um conjunto bem definido de canais de eventos, onde cada canal deve ter definidos de forma bem clara, quais os tipos de contexto podem ser postados nele.

Neste trabalho, os canais de eventos definidos até o momento são apresentados na Tabela 4.2. Não há intenção de que esta proposta seja definitiva. É apresentado aqui um conjunto básico para uma funcionalidade mínima da arquitetura, onde tenta-se lidar com os aspectos mais utilizados. O ContextGrid, através do LCM e GCM, prevê meios pelos quais novos canais de eventos possam ser criados. Este aspecto será tratado com mais detalhes na Seção 4.7, que detalha as funcionalidades e a implementação do LCM e do GCM.

A criação destes canais tem como base o modelo de contexto definido na Seção 4.4.2. As entidades são consideradas fontes de contexto. Foi feito então, um mapeamento um-para-um entre entidade e canais de eventos. Essa abordagem permitiu uma definição clara do que pode ser postado em cada canal, ou seja, somente o contexto que envolve os atributos da entidade em questão. Outro ponto importante desse mapeamento é a equivalência com relação ao modelo abstrato apresentado (Figura 4.4), facilitando o entendimento da semântica de cada canal, pois equivale a semântica de cada entidade.

Canal	Descrição
connection	Contexto relativo à conectividade (status, largura de banda, latência)
application	Informações do contexto da aplicação (status, requisitos, tarefas que estão executando na grade)
computational Environment	Contexto do ambiente computacional dos dispositivos (versão do sistema operacional, tipo de nó para grade, recursos disponíveis, ambientes de tempo de execução disponíveis)
device	Contexto do dispositivo (tela, modelo, características de *hardware*, localização)
user	Informações de contexto do usuário (perfil, permissões, atividades, dispositivo em uso)

Tabela 4.2: *Hierarquia de canais de eventos*

4.7 Implementação

O ContextGrid foi implementado em *Python* para uso em dispositivos móveis, compreendendo o LCM e os wrappers, com versões para *Symbian OS* (Celulares) e *Windows Mobile (Pocket PC)*. Para o GCM, LCM, em nós fixos, e alterações em alguns componentes do MAG, a implementação foi feita em Java. Já para as *wrappers*, em nós fixos, e alterações em alguns componentes do InteGrade foi usado C/C++.

A escolha de *Python* para as implementações em dispositivos móveis foi feito principalmente por apresentar um melhor acesso ao *hardware* e suas informações. Esse acesso pode ser feito tanto através de recursos da própria linguagem quanto, no caso destes serem insuficientes, por extensões implementadas em C/C++. O acesso mais próximo do *hardware* foi necessário para a obtenção de informações dos dispositivos os quais compõem o modelo de contexto do ContextGrid. Um outro ponto que influenciou na escolha foi a portabilidade do interpretador Python, existindo implementações para inúmeras plataformas. Inicialmente considerou-se o uso de Java, mais especificamente J2ME. Contudo, não havendo um suporte completo para acesso às informações necessárias e, para dispositivos com MIDP versão 1.0, não houveram formas para estender ou fazer chamadas nativas do dispositivo. Mesmo que o problema fosse resolvido, com uma série de extensões, a portabilidade ficaria comprometida. Isso porque essas extensões são muito dependentes do dispositivo e do sistema operacional.

Além da implementação dos componentes do ContextGrid, alguns componentes do MAG/InteGrade foram alterados para que passassem a usar os serviços de contexto oferecidos. Sendo estes: o GRM, para suporte a escalonamento sensível ao contexto; o *AgentHandler*, principalmente para suporte a migração ciente de contexto dos *MagAgents* e conseqüentemente, das aplicações MAG; e o *MobileProxyAgent* que tem papel crucial na integração com dispositivos móveis. Ele deve ter ciência do contexto e suportar adaptações dinâmicas, não somente dele, mas também, de todos os dispositivos móveis associados a ele.

Nesta seção é descrita a implementação do LCM, GCM, Wrapper, Proxy e de alguns *Context Users*. É importante ressaltar que a implementação foi mais focada nos dois principais componentes da arquitetura e na sua integração com a arquitetura MAG/IntegGrade. Os *Context Users* implementados ou alterados para tal foram apenas aqueles que têm influência direta em alguns dos objetivos do trabalho.

4.7.1 Suporte a Contexto e Notificação

Para a implementação de eventos ou notificação utilizando o conceito de canais de eventos, foi adotado no ContextGrid o padrão de projeto *Observer* [43]. Ele define uma dependência do tipo um-para-muitos entre objetos de modo que, quando um objeto muda o estado, todos os seus dependentes são notificados e atualizados automaticamente. O padrão *Observer* é também chamado de *Publish-Subscrib* ou *Event Generator-Dependents*.

Esse padrão pode ser usado quando uma abstração tem dois aspectos, um dependente do outro. Encapsular tais aspectos em objetos separados permite que eles variem e sejam reusados separadamente. Por exemplo, quando uma mudança em um objeto requer mudanças em outros e não se sabe quantos outros objetos devem ser mudados. O padrão também é útil quando um objeto ouvinte deve ser capaz de avisar outros sem fazer suposições sobre quem são os objetos. Isto é, sem criar um acoplamento forte entre os objetos.

Para o ContextGrid foram definidas duas interfaces que, juntas, implementam este padrão: uma para o Gerador de Eventos ou *Subject* e outra para os Dependentes ou Observers . A Figura 4.8 ilustra estas interfaces, as quais são descritas a seguir.

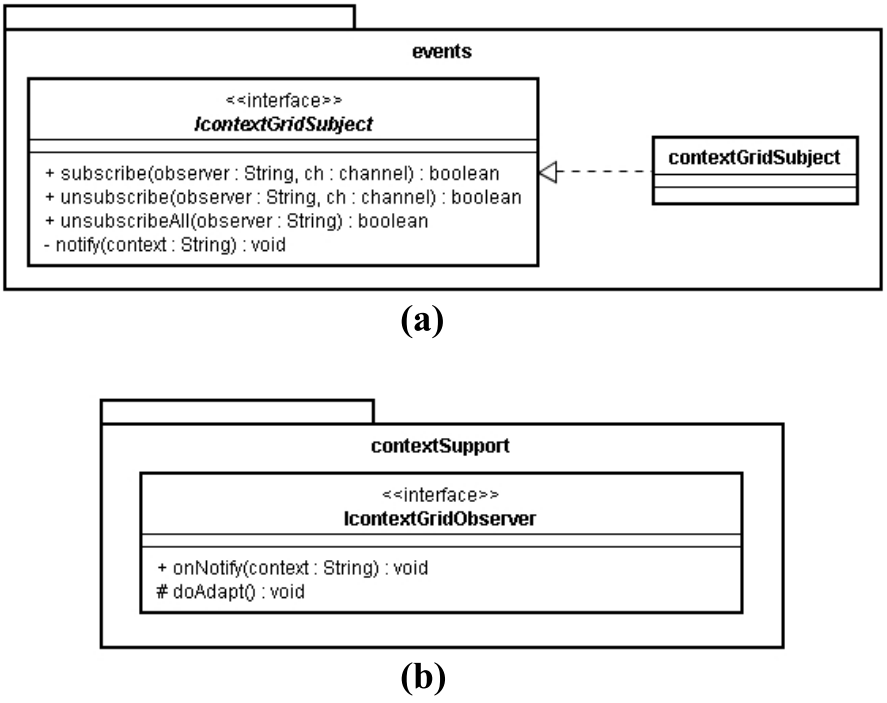

Figura 4.8: *Interfaces de Suporte ao Contexto e Notificações*

IcontextGridSubject

Esta interface define o *Subject* do padrão *Observer*. Ela deve ser implementada pelos publicadores de contexto do ContextGrid, sendo este, o GCM, que publica seus eventos aos LCMs, e estes últimos, que republicam, localmente, os eventos recebidos. Ela é ilustrada na Figura 4.8(a).

Os métodos *subscribe* e *unsubscribe* gerenciam a lista de *observers* (*Context Users*) interessados em receber os eventos de contexto de um determinado canal. O método *unsubscribeAll*, por sua vez, é uma forma rápida de cancelar a inscrição de um *Context User* de todos os canais aos quais ele está vinculado. Já o método *notify*, é o responsável por notificar a todos os *Context Users*, as mudanças no contexto. Essa notificação é realizada canal por canal.

Uma implementação concreta dessa interface é dada pela classe *contextGridSubject*, que implementa as listas de inscrições na forma de tabela *hash*, indexada pelas IORs dos *Context Users*. Seguindo o padrão de comunicação do InteGrade, notificações são feitas utilizando comunicação por meio de um ORB CORBA, mais especificamente o JacORB.

IcontextGridObserver

Para que um componente do *middleware* possa se tornar um *Context User*, ele deve implementar a interface *IcontextGridObserver*. Esta interface define um conjunto padrão de métodos que são esperados pelo ContextGrid em seus usuários de contexto. Esta interface é ilustrada na Figura 4.8(b).

O método *onNotify* é chamado por um *Subject* para enviar um evento de mudança de contexto. Este método recebe como parâmetro o contexto atual. Já o método *doAdapt*, é invocado pelos *Context Users* sempre que se desejam realizar uma adaptação. É este método que deve ser implementado as políticas de adaptaçãos adotadas diante das mudanças no contexto. Uma boa prática é que ele seja chamado de dentro do método *onNotify*. Deste modo, sempre que haver uma notificação de mudança em um contexto, pode-se decidir se há adaptações aplicáveis.

4.7.2 LCM

O LCM é o componente do ContextGrid que executa em todos os dispositivos que fazem parte da grade, seja como clientes ou provedores de recursos. As principais funções do LCM são:

Coleta e atualização de informações contextuais: o LRM é responsável por coletar as informações contextuais do dispositivo em que executa e enviá-las ao GCM, conforme descrito pelo protocolo de atualização de contexto na

Seção 4.5.1. Para tanto, ele faz uso dos *wrappers* que implementam, em última instância, a comunicação com os sensores.

Conversão das informações contextuais em um formato padrão: as informações contextuais podem ser fornecidas nos mais diversos formatos. O LCM é responsável por transformar os diversos formatos recebidos em um formato único, mais abstrato.

Gerenciar os diversos *wrappers* locais: para cada LCM pode existir um conjunto de *wrappers* associados. É função do LCM gerenciá-los de forma consistente, sendo também responsável pelo controle de registro e comunicação com os mesmos. O LCM deve permitir também a adição e remoção dinâmica de *wrappers*.

Atuar como *proxy* do GCM: o LCM é responsável por intermediar todas as solicitações de inscrições e cancelamento de inscrição em canais de eventos para todos os *Context Users* de um nó. Todos as solicitações devem ser encaminhadas ao LCM local, o qual se inscreve no GCM. Ao receber um evento de notificação, o LCM repassa as notificações aos *Context Users* locais.

Avaliar quando uma informação deve ser enviada ao GCM: ao receber uma nova informação de contexto de um *wrapper*, o LCM deve avaliar se houve um mudança significativa na mesma, de modo que justifique seu envio ao GCM. Deve ser possível configurar políticas iniciais de avaliação a partir dos próprios *wrappers*.

Adaptar-se ao contexto local: o LCM dever fazer uso do contexto para adaptar seu próprio comportamento.

Este componente é o mais crítico do ContextGrid, devendo executar em todos os nós da grade, incluindo nos dispositivos móveis. Ele executa juntamente com os componentes LRM do InteGrade e com o AgentHandler do MAG, apresentando as mesmas restrições. Como sua execução é permanente, ele deve consumir poucos recursos, de modo a não comprometer o desempenho do nó. O LCM foi desenvolvido em Java e utilizando um ORB CORBA para os dispositivos fixos. Para os dispositivos móveis, no entanto, ele foi implementado em *Python* e utilizando sockets para comunicação com um *proxy*. Por uma questão de homogeneidade e para evitar que as dependências em relação a terceiros aumentem, o ORB utilizado foi o mesmo que já vinha sendo empregado no *AgentHandler*, o JacORB. A Figura 4.9 apresenta o diagrama de classes para o LCM.

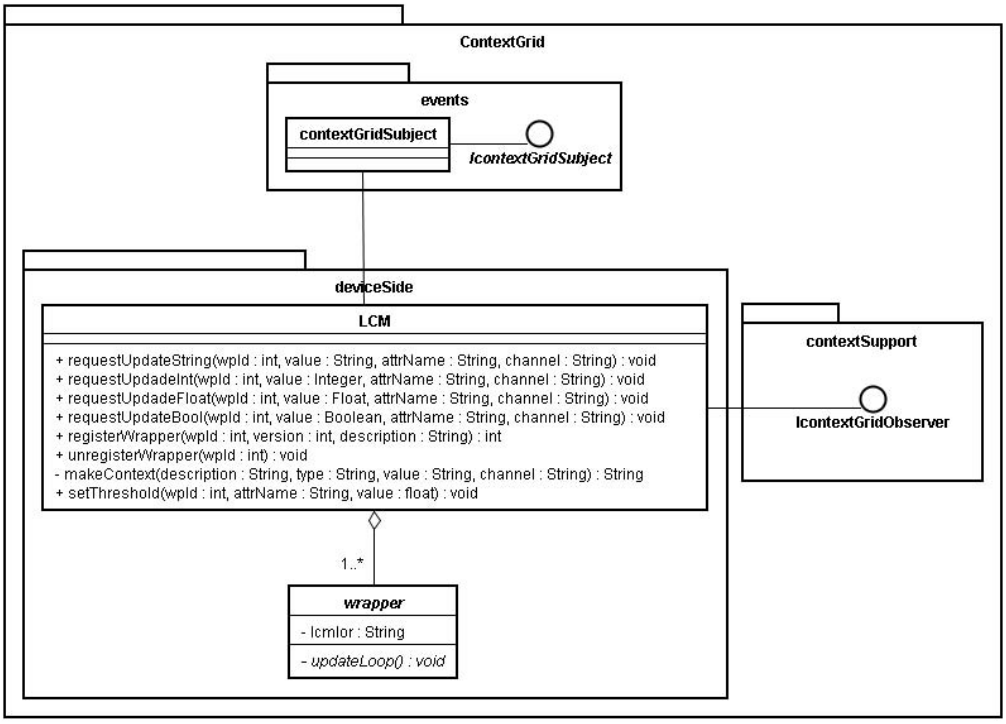

Figura 4.9: *Diagrama de classes para o LCM*

Como o pode ser notado na Figura 4.9, pela generalização da classe *contextGridSubject*, o LCM é um gerador de eventos.O LCM tem o papel de notificador local de contexto. Paralelamente, ele pode fazer uso do contexto para promover adaptações em seu comportamento. Para isso, ele deve se tornar um *Context User*, o que é concretizado através da implementação, por este componente, da interface *IcontextGridObserver*.

Outra função associada ao LCM é a gerência dos *wrappers* do nó onde ele executa. Para tanto, ele fornece em sua interface um conjunto de funções específicas para comunicação e agregação desses *wrappers*. Os métodos *registerWrapper* e *unregisterWrapper* são destinados à gerência desses componentes. Um *wrapper*, para poder fornecer informações de contexto ao ContextGrid, deve registrar-se junto ao LCM local através do método *registerWrapper*, passando como parâmetros a sua IOR e sua versão. Os *wrappers* registrados no LCM são mantidos em uma tabela *hash* indexada por sua IOR. O método *unregisterWrapper* deve ser chamando antes que um *wrapper* encerre sua execução, de modo a remover seu registro no LCM.

O método *setThreshold* configura, para cada atributo de contexto, uma métrica. Essa métrica é usada pelo LCM para decidir se houve uma variação significativa no valor de um atributo que justifique seu envio ao GCM. Estes atributos são aqueles apresentados na Figura 4.4 e descritos na Seção 4.4.2. Um exemplo de uso seria, como já discutido na Seção 4.5.1, definir para qual variação do valor de

uso da CPU será relevante atualizar o contexto.

O método *requestUpdate* usado aqui para representar suas diversas variações (*requestUpdateString*,*requestUpdateBool*,*requestUpdateInt*,*requestUpdateFloat*). É chamando pelo *wrapper* quando este deseja enviar novas informações monitoradas. Ele provê suporte para *wrappers* que produzem informações contextuais do tipo inteiro, boleano, ponto flutuante e cadeia de caracteres. Para cada um destes tipos há um processamento para sua transformação em um formato uniforme e com um nível de abstração mais elevado. É nesse método que a métrica configurada pelo método *setThershold* é avaliada.

Por fim, o método *makeContext* auxilia o método *requestUpdate* no processo de transformação do contexto. No método *requestUpdate* deve haver a separação entre a informação e seu tipo. Em seguida, ele chama o método *makeContext*, que irá converter as informações contextuais em um modelo padronizado, independente de tipo e da fonte. Esse padrão é baseado em XML e derivado do modelo de contexto (Figura 4.4), cuja estrutura pode ser vista na Figura 4.10.

```
<?xml version="1.0" encoding="UTF-8"?>
<contextBase type="localization">
    <source>
        <device>mobile1.inf.ufg.br</device> <!-- hostname -->
        <lcm>IOR:00000000000000174...3a6400020002</lcm> <!-- IOR -->
    </source>
    <channel>device</channel>
    <entity name="device">
        <attr>localization</attr>
    </entity>
    <contextValue>
        <value dataType="string">350/500</value> <!--
dataType=integer/float/boolean/string -->
    </contextValue>
</contextBase>
```

Figura 4.10: *Informações de contexto transformadas pelo LCM*

Este XML contém a informação de contexto, agora transformada em contexto, com uma abstração maior, independente da fonte. É capturado o tipo de contexto (*contextBase type=*) que na implementação atual é o nome do atributo, no exemplo acima *localization*. Ele taz informações da origem (*source*), do canal (*channel*), vinculado a entidade a qual o atributo pertence. Por fim ele traz o valor do contexto (*contextValue*) assim como o tipo do qual foi derivado.

Como o LCM utiliza CORBA para que seus métodos sejam acessíveis por outros objetos, deve ser definida uma interface em IDL. Essa interface é apresentada na Figura 4.11. Seus métodos já foram descritos acima.

```
module deviceSide
{
      interface lcm
      {

            long registerWrapper(in string name, in long version);
            void unregisterWrappers (in long wpId);
            void setThreshold(in long wpId, in string attrName, in float value);

            void requestUpdateString(in long wpId, in string value, in string entity, in string attrName, in string channel);
            void requestUpdateInt(in long wpId, in long value, in string entity, in string attrName, in string channel);
            void requestUpdateFloat(in long wpId, in float value, in string entity, in string attrName, in string channel);
            void requestUpdateBool(in long wpId, in boolean value, in string entity, in string attrName, in string channel);

            void onNotify(in string contexts);
            boolean subscribe(in string observer, in string ch);
            boolean unsubscribe(in string observer, in string ch);
            boolean unsubscribeAll(in string observer);

      };
};
```

Figura 4.11: *IDL do LCM*

4.7.3 GCM

O GCM é o coordenador dos diversos serviços do ContextGrid. É este componente que tem uma visão global do contexto da grade. Suas principais funções são:

Interpretar o contexto com base nas informações enviadas pelos LCMs: o GCM deve coletar os diversos contextos oriundos dos LCMs e interpretá-los, formando uma visão do contexto de um aglomerado em diversos níveis de abstração. Isso inclui manter tanto o contexto individual de cada entidade, quanto um contexto da grade. Ele deve permitir também a carga dinâmica de interpretadores de contexto.

Notificar os diversos LCMs sobre mudanças no contexto: o GCM é responsável por notificar as mudanças de contexto ocorridas. Ele se comunica exclusivamente com os LCMs, através dos canais de eventos, que por sua vez fazem a entrega final da notificação.

Gerenciar os canais de eventos: é de responsabilidade do GCM criar, excluir e manter canais de eventos por onde as notificações são feitas. Ele deve, ainda, controlar quais contextos podem ser postado em cada canal. A tarefa de gerência deve ser dinâmica e ter a possibilidade de ser realizada em tempo de execução.

Gerenciar inscrições em canais de eventos: o GRM é responsável por atender às solicitações de inscrição e cancelamento de inscrições em canais de eventos.

Adaptar-se ao contexto da grade e ao contexto local: o GCM deve fazer uso do contexto de modo a adaptar seu comportamento, por exemplo, mudar o interpretador de contexto para que exija menor poder de processamento diante de uma sobrecarga no nó em que executa.

O GCM foi implementado em Java utilizando como ORB o JacORB. A escolha do ORB foi orientada pelos mesmos princípios utilizados para o LCM. Ele utiliza o banco de dados XML *Xindice* [69] para armazenamento do contexto do aglomerado MAG/InteGrade. Este banco de dados já é utilizado no MAG/InteGrade para o armazenamento das informações do *ExecutionManagementAgent*. O banco de dados Xindice foi adotado por se mostrar mais apropriado ao armazenamento e manipulação de dados em XML. Esta escolha está também relacionada ao princípio de não aumentar as dependências externas adotadas até então. A estrutura em XML manipulada pelo GCM, e efetivamente armazenada no *Xindice*, pode ser observada na Figura 4.12.

```
<?xml version="1.0" encoding="UTF-8"?>
<context type="localization" level="1">
    <contextScope>
        <entity name="device">
            <attr name="hostname"><value>mobile1.inf.ufg.br</value></attr>
        </entity>
    </contextScope>

    <contextInformation>
        <contextValue>
            <item name="positionX">
                <dataType>number</dataType> <!-- number/condition/percent/string -->
                <concreteType>integer</concreteType> <!-- integer/float/boolean/string -->
                <value>350</value>
                <allowedValueRange>
                    <minimum>0</minimum>
                    <maximum>1000</maximum>
                    <step>1</step>
                </allowedValueRange>
             </item>
             <item name="positionY">
                 <dataType>number</dataType> <!-- number/condition/percent/string -->
                 <concreteType>integer</concreteType> <!-- integer/float/boolean/string -->
                 <value>500</value>
                <allowedValueRange>
                    <minimum>0</minimum>
                    <maximum>1000</maximum>
                    <step>1</step>
                </allowedValueRange>
             </item>
         </contextValue>
        <dependOf>
            <contextBase type="localization">
                <source>
                    <device>mobile1.inf.ufg.br</device> <!-- hostname -->
                    <lcm>IOR:00000000000000174...3a6400020002</lcm> <!-- IOR -->
                </source>
                <channel>device</channel>
                <entity name="device">
                    <attr>localization</attr>
                </entity>
                <contextValue>
                    <!--dataType=integer/float/boolean/string -->
                    <value dataType="string">350/500</value>
                </contextValue>
            </contextBase>
        </dependOf>
    </contextInformation>
</context>
```

Figura 4.12: *Contexto produzido pelo GCM*

O contexto definido deve ter um tipo e possuir um nível(*context type=* *level=*) que representa a atual abstração, por exemplo, se um contexto depende apena de contexto base (gerados pelo LCM) ele tem nível 1, se ele depende de um contexto de nível 1 ele é nível 2 e assim por diante. Este XML tem duas partes básicas: (1) definição do escopo onde o contexto é válido (*contextScope*), e (2) da informação do contexto (*contextInformation*). Na primeira parte é definido o escopo em termos do modelo de contexto (Figura 4.4), delimitando a(s) entidade(s) onde ele é válido. Para um escopo mais fechado, pode-se definir valores de alguns atributos para cada entidade citada. Por exemplo, o contexto apresentado na Figura 4.12 é válido para entidade *device* cujo atributo *hostname* é `mobile1.inf.ufg.br`. A segunda parte é composta do valor do contexto (*contextValue*) e das dependências desse contexto (*dependOf*), ou seja, os contextos do qual ele foi derivado. Para o *contextValue* são definidos itens e os valores desses itens, por exemplo *positonX* cujo valor é 350. É definido também o tipo de cada item. Há dois tipos, um geral, por exemplo, porcentagem, e um concreto, por exemplo, *float*. Isso quer dizer que o item tem um valor que dever ser interpretado como uma porcentagem e vem representado como um número do tipo *float*. A parte de dependências é composta pelos contextos do qual este, que esta sendo definido, depende. Essa parte pode ser composta tanto por contextos base (*contextBase*) como por contextos de qualquer nível (*context*).

A Figura 4.13 apresenta as principais classes que implementam ou que têm relação direta com o GCM. Pode-se observar, que de maneira semelhante ao LCM, o GCM é um gerador de eventos, através da generalização da classe *contextGridSubject*. Nesse caso, seu papel é notificar mudanças de contexto aos LCMs, conforme o protocolo descrito na Seção 4.5.2. De modo semelhante, ele pode fazer uso do contexto para promover adaptações em seu comportamento, tornando-se, portanto, um *Context User*. Essa característica é provida através da implementação da interface *IcontextGridObserver*.

A classe de suporte *history* auxilia o GCM na manutenção de um histórico dos últimos eventos de notificação de contexto gerados em cada canal. Esse histórico é, nessa implementação, volátil, sendo perdido quando o GCM é finalizado. Ele também é limitado a um tamanho que pode ser configurado pelo método *setHistorySize*. A pesar do contexto ser volátil, interessando apenas o último estado, o uso de um histórico pode ajudar na predição do estado futuro da grade.

A manutenção dos últimos eventos de notificação gerados é usada atualmente quando um componente do *middleware* solicita sua inscrição em um novo canal de eventos, seja porque é novo na grade ou por querer ampliar os contextos conhecidos. Nesse momento, o último evento de notificação é enviado exclusivamente para esse componente. Deste modo, ele já pode se adaptar de acordo suas políti-

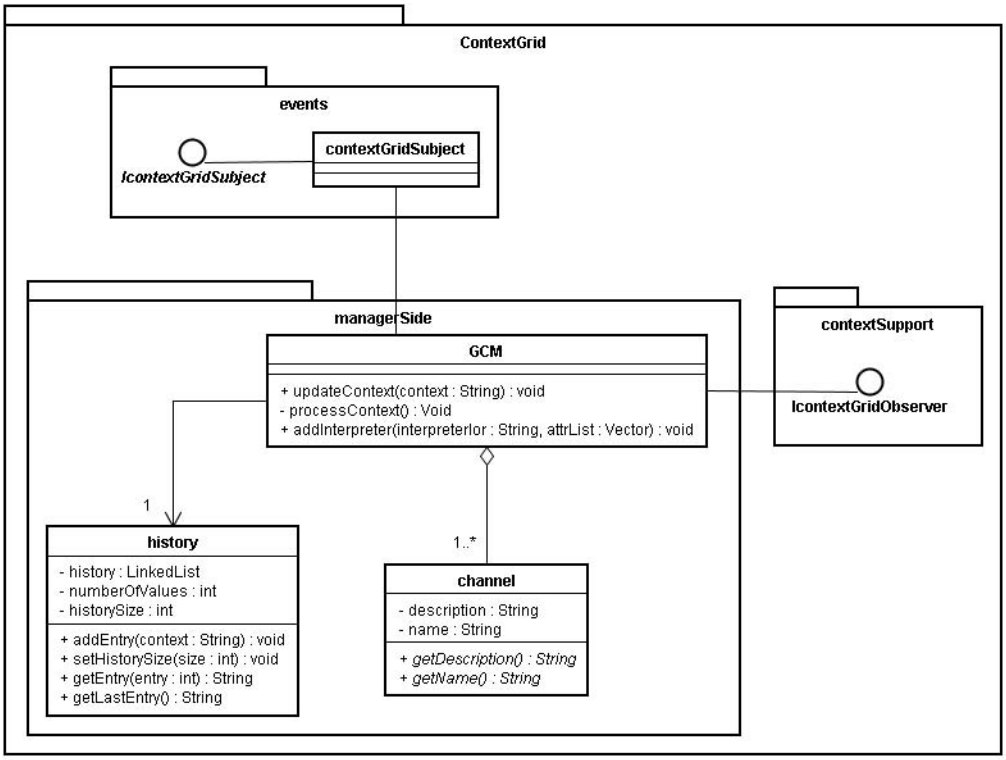

Figura 4.13: *Diagrama de classes para o GCM*

cas adaptativas. Isso evita que ele, ao entrar na grade ou ao ampliar os contextos conhecidos, opere de forma não condizente com o contexto mais atual e não tenha que esperar até que receba uma notificação. Esse mesmo processo ocorre quando um novo dispositivo, móvel ou não, conecta-se à grade.

O atributo *channelList* é um vetor que contém todos os canais atualmente reconhecidos pelo GCM. Ele pode ser gerenciado pelos métodos *addChannel* e *removeChannel*. Para cada canal, há um subconjunto dos atributos de contexto que são aceitos. Como foi usado um mapeamento um-para-um entre entidades do modelo e canais de eventos (Seção 4.6), os atributos de cada canal são os mesmos apresentados na Figura 4.4 para cada entidade. Quando um contexto depende de vários atributos ele é postado em todos os canais que aceitam pelo menos um dos atributos da sua lista. Esse mapeamento canal-atributo, é implementado por um tabela *hash* indexada pelo nome do atributo, denominada de *mapAttrChannel*. De modo semelhante, o atributo mapAttrInterpreter mapeia os atributos para os quais deve ser chamado um interpretador externo.

O método *updateContext* é usado pelos LCMs para a atualização do contexto conforme descrito pelo protocolo de atualização de contexto na Seção 4.5.1. Essa atualização ocorre após a avaliação e conversão promovida pelos LCMs. Nesse ponto, o contexto está em um formato padrão, conforme descrito na Seção 4.7.2. Após a atualização de um contexto, o GCM invoca seu método *processContext* que, por sua

vez, fará a interpretação básica do contexto, sendo também o responsável por iniciar as notificações. O processo de notificação foi descrito na Seção 4.5.2. Caso haja um interpretador que se aplique ao contexto recebido ele é instanciado pelo GCM, que passa a este interpretador uma cópia do contexto. Após finalizar a interpretação, o interpretador retorna um novo contexto ao GCM o qual, por sua vez, inicia o processo de notificação. Mesmo havendo um interpretador externo, o GCM continua com sua linha de execução, interpretando também o contexto recebido. Esse comportamento se deve principalmente ao fato de que os *Context Users* podem depender de contexto em níveis de abstração diferentes, conforme discutido na Seção 4.4.1. Assim, o GCM procede com as notificações e processamentos mais básicos, enquanto os interpretadores externos elaboram um contexto mais abstrato. O interpretador interno do GCM tem, na atual implementação, a função de transformar o contexto no modelo apresentado na Figura 4.12.

Por fim, o método *addInterpreter* é usado para o registro de um novo interpretador junto ao GCM. Esse método tem como parâmetro uma cadeia de caracteres que indica a localização da implementação do interpretador, que deve ser implementado em Java, além de um vetor contendo todos os atributos exigidos por este interpretador.

Assim como o LCM, o GCM utiliza CORBA para que seus métodos sejam acessíveis por outros objetos, sendo definida uma interface IDL para ele. Essa interface é apresentada na Figura 4.14, sendo que a implementação dos métodos já foi descrita acima.

```
module managerSide
{
      interface gcm
      {

            void updateContext(in string contexts);
            void addInterpreter(in string interpreter, in types::vectorString attrList);
            void onNotify(in string contexts);
            boolean subscribe(in string observer, in string ch);
            boolean unsubscribe(in string observer, in string ch);
            boolean unsubscribeAll(in string observer);

      };
};
```

Figura 4.14: *IDL do GCM*

4.7.4 *Wrappers*

Os *wrappers* são os elementos de *software* que fazem a ponte entre a fonte de informações contextuais e o LCM, funciona como tradutores, pegando as informações das fontes e as entregando ao LCM. Para cada fonte de informação distinta há um *wrapper* associado. Suas principais funções são:

Comunicar-se diretamente com a fonte de informação: uma das principais funções do *wrapper* é se comunicar com as fontes de informações sob sua responsabilidade e compreender os dados fornecidos pelas mesmas.

Registrar-se no LCM local: para que passe a fazer parte do ContextGrid, fornecendo-lhe informações de contexto, o *wrapper* deve se registrar junto ao LCM. Esse procedimento permite que o LCM passe a reconhecer as informações enviadas por este *wrapper*.

Converter as informações lidas das fontes de informações: antes de enviar as informações de contexto para o LCM, o *wrapper* deve converter a informação obtida para um dos formatos que são aceitos pelo LCM (método *requestUpdate*). Nesse momento, essa conversão é bastante simples, devendo seguir os tipos básicos, a saber: booleano, inteiro, ponto flutuante e cadeia de caracteres.

A implementação efetiva de um *wrapper* pode variar muito, dependendo fortemente do sensor ou do dado monitorado. Contudo, foram definidos alguns padrões para esta implementação. O *wrapper* deverá se comunicar com o LCM via CORBA. Isso confere certa interdependência de localização e permite o desenvolvimento em qualquer linguagem com suporte para CORBA, flexibilizando bastante esse componente. A Figura 4.9 apresenta o *wrapper* juntamente com o LCM.

Um *wrapper* deve, no mínimo, realizar as funções descritas acima, conhecer a IOR do seu LCM e ter um laço constante de monitoramento. No que se refere à função de converter as informações colhidas, ele deve compatibilizar os dados obtidos com uma das assinaturas do método *requestUpdate* definidas pelo LCM. No momento de sua inicialização, ele dever se registrar junto ao LCM e, quando da finalização de suas atividades, ele deve requisitar a remoção do seu registro.

Neste trabalho, foram desenvolvidos *wrappers* para monitoração de informações de dispositivos com SymbianOS, WinCE e Linux. Estas informações incluem os atributos apresentados na Figura 4.4 que são do tipo monitorado. Também foi desenvolvido um *wrapper* que monitora o *ExecutionManagementAgent* do MAG/InteGrade, utilizando as informações já armazenadas por este agente para atualizar o contexto.

4.7.5 *Proxy*

Como já mencionado na Seção 4.3, o *proxy* é o elemento do ContextGrid responsável por tratar alguns dos desafios introduzidos pela integração com dispositivos móveis, como instabilidade na conexão, diversidade de modelos e recursos restritos. É nele que o contexto começa a ser explorado para permitir uma integração

transparente ao usuário e ao mesmo tempo flexível. Ele foi baseado na implementação do **moblieProxyAgent** do MAG/InteGrade e suas principais funções são:

Gerenciar os dispositivos móveis sob seu domínio: o *proxy* pode ter vários dispositivos móveis sob sua responsabilidade. É sua função gerenciar a comunicação com esses dispositivos de maneira ordenada, além de manter meios para que a conexão com os mesmos seja possível, independentemente da sua localização. Atualmente apenas um *proxy* por aglomerado é permitido.

Representar o dispositivo junto à grade: é função do *proxy* atuar junto à grade como se fosse o próprio dispositivo podendo, caso necessário, assumir completamente as funções do dispositivo de modo transparente à grade, por exemplo, em caso de desconexão ou nível crítico de bateria. Como base no contexto e em suas políticas de adaptação, ele mesmo decide quando tomar a decisão de representar completamente o dispositivo. Ele deve esconder da grade as questões inerentes à integração com dispositivos móveis.

Monitorar o contexto de cada dispositivo sob seu domínio: o *proxy* deve monitorar o contexto de todos os dispositivos móveis associados a ele. É fundamental que ele, ao receber conexões de um dispositivo, passe a monitorar todo o contexto referente ao mesmo.

Realizar adaptações nas interações dos dispositivos móveis com a grade: o ponto fundamental de atuação do *proxy* ocorre nas interações entre os dispositivos móveis intermediados por ele e a grade. Ele deve atuar neste ponto, promovendo adaptações de acordo com as variações de contexto a que tais dispositivos estão sujeitos. Por exemplo, diante da mudança da conexão de um celular participante da grade, de *bluetooth* para GPRS, ele pode adaptar as mensagens a serem enviadas ao dispositivo, de modo a minimizar seu tamanho e/ou periodicidade. Isso porque, nesse tipo de conexão, a tarifação é realizada pelo número de *bytes* trafegados.

Adaptar-se ao contexto local: assim como o LCM e GCM, o *proxy* deve tirar proveito do contexto para promover adaptações em seu próprio comportamento, por exemplo, não aceitar mais dispositivos se o nó onde ele está executando apresenta sobrecarga.

Ao *mobileProxyAgent* do MAG/InteGrade foi adicionado a implementação da interface apresentada na Seção 4.7.1. Sendo que o método *doAdapt* foi desenvolvido de maneira que processe também as adaptações necessárias aos dispositivos

registrados no *proxy*. Quando um dispositivo móvel é registrado, o *proxy* solicita simultaneamente ao LCM um inscrição no canal de eventos *device* e passa a monitorar o contexto do dispositivo. A Figura 4.15 apresenta os principais métodos do *proxy* e são comentados a seguir.

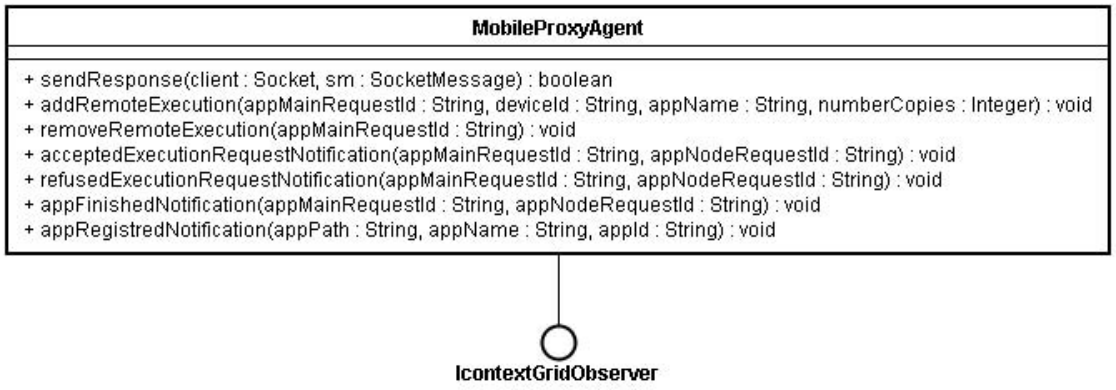

Figura 4.15: *Proxy do ContextGrid*

Os métodos *acceptedExecutionRquestNotification*, *refusedExecutionRequestNotification*, *appFinishedNotification* e *appRegistredNotification*, fazem as vezes de ASCT junto a grade. notificando o dispositivo móvel do aceite de uma aplicação para execução, da finalização dessa execução e da confirmação de registro de uma aplicação na grade (*Application Repository* do InteGrade). Os métodos *addRemoteExecution* e *removeRemoteExecution* são responsáveis pela manutenção de uma tabela contendo cada aplicação em execução e os respectivos dispositivos de onde originaram-se as solicitações. Já o método *sendResponse* é responsável por enviar, para um dispositivo móvel, o resultado da computação e repassar as notificações geradas pelo protocolo de notificação de contexto, apresentado na Seção 4.5.2.

Foi implementado apenas parte das funcionalidades do *proxy*, dentre elas: monitorar o contexto de cada dispositivo sob seu domínio, realizar adaptações nas interações dos dispositivos móveis com a grade, gerenciar os dispositivos móveis sob seu domínio e representar o dispositivo junto à grade, sendo as duas últimas já existentes na implementação do *mobileProxyAgent*. Portanto, foram apenas adaptadas. No que diz respeito a representação do dispositivo junto a grade, foi tratado apenas o aspecto de submissão de aplicação. Já para as adaptações nas interações foi definido apenas uma política simples, sendo que esta define: para uma largura de banda menor que 1Mbps, enviar apenas os 100 primeiros caracteres do resultado da computação; para um contexto de desconexão ou um nível de bateria abaixo de 15%, representar todas as funcionalidades do dispositivo junto à grade até que esse contexto mude para um mais favorável.

4.7.6 Context Users

Nessa seção são descritas as alterações realizadas em alguns dos componentes do MAG/Integrade para que se tornassem *Context Users*, fazendo uso, assim, de informações de contexto para realizar adaptações. O componentes descritos aqui, juntamente com o *proxy* discutido na Seção 4.7.5 representam apenas um pequeno uso do ContextGrid.

GRM - Escalonamento consciente de contexto

O GRM e sua função foram descritos na Capítulo 2. Ele foi alterado para tornar-se um *Context User* e assim fazer uso do contexto para se adaptar. Essa implementação tem caráter ilustrativo, portanto foi abordado o uso de contexto apenas no momento da seleção dos LRMs para o escalonamento de uma aplicação. Este refinamento no escalonamento leva em consideração, nessa versão, os contexto da aplicação e do ambiente computacional. Mais especificamente os atributos "'requisitos"' e "'ambiente de tempo de execução"'. Contudo, a idéia é que em uma próxima versão o modelo de contexto seja explorado por completo. A Figura x apresenta os principais métodos do GRM que são comentados a seguir.

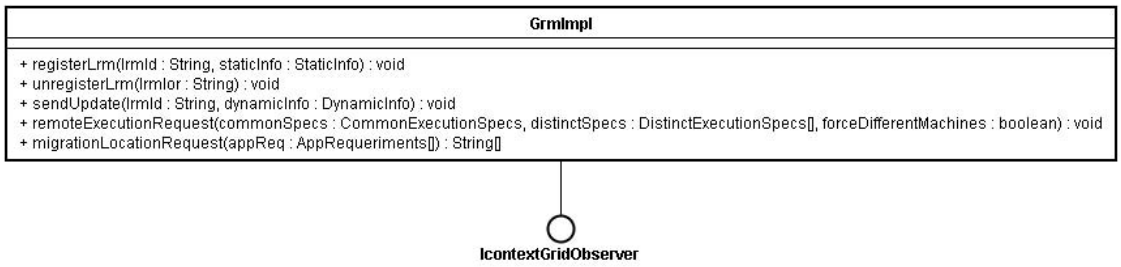

Figura 4.16: *Context User GRM*

Os métodos *risterLrm*, *unregisterLrm* e *sendUpdate*, são usados pelos LRMs para respectivamente se registrar ao GRM, cancelar seu registro, nesse caso o nó deixa de prover recursos a grade, e atualizar o GRM sobre os recursos disponíveis. O método *remoteExecutionRequest* é utilizando pelo ASCT para solicitar a execução de uma aplicação. Já o método *migrationLocationRequest*, introduzido pelo MAG, será usado na adaptação sensível ao contexto. Foi adicionado ao método *remoteExecutionRequest* código para que o GRM armazene o contexto de requisitos desejáveis da aplicação, e se inscreva no canal de eventos "'ambiente computacional"'. A cada mudança no contexto notificada nesse canal, o GRM verifica se há aplicações cujos requisitos desejáveis não estão sendo atendidos e se há algum nó nesse momento que possa atendê-los. Se houver, o GCM inicia a migração da aplicação através do

método *migrationLocationRequest*. Essa adaptação foi explorada apenas para aplicações MAG, pelo fato de estas serem encapsuladas em agentes móveis e suportarem migração forte.

AgentHandler - Migração de tarefas consciente de contexto

De modo semelhante ao GRM, o *AgentHandler*, por suportar migração, foi alterado para com base no contexto, migrar as aplicações sob sua responsabilidade. No entanto, enquanto o GRM aplica migração em todo o aglomerado, o *AgentHandler* aplica ao nó em que executa. O *AgentHandler* usa o contexto do nó em que está executando e os requisitos obrigatórios das aplicações para aplicar as adaptações necessárias, como por exemplo, ao perceber que o contexto de recursos do nó em que executa não satisfaz os requisitos básicos de algumas aplicações em execução, ele solicita a migração das mesmas para um outro nó. Isso é executado com o auxílio do GCM, mais especificamente pelo método *migrationLocationRequest*. A Figura 4.17 apresenta os principais métodos para esse agente, que são comentados a seguir.

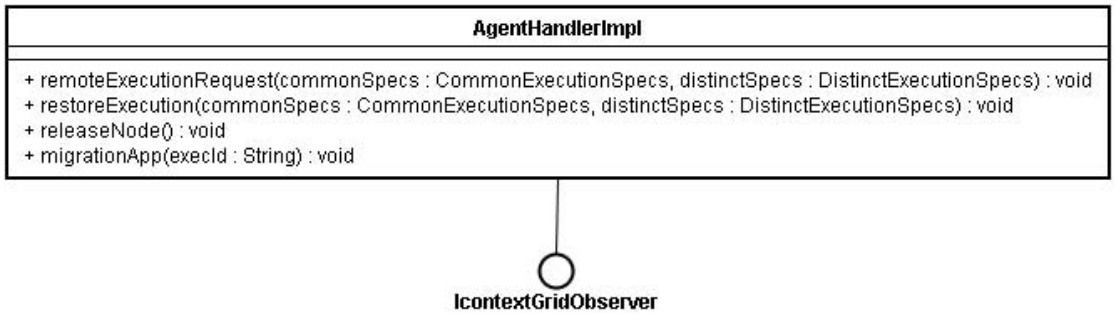

Figura 4.17: *Context User AgentHandler*

O método *remoteExecutionRequest* é usado na interação com o LCM, onde este último ao detectar que recebeu uma requisição de execução para uma aplicação MAG, a repassa ao *AgentHandler*. O método *restoreExecution* é utilizado para restaurar a computação de uma aplicação que foi migrada de outro nó do aglomerado. Além da implementação dos métodos da interface *IcontextGridObserver* foi adicionado outro, o *migrationApp*, que realiza a migração de uma aplicação especifica. Anteriormente o *AgentHandler* permitia apenas a migração de todas as aplicações através do método *releaseNode*.

4.8 Comentários

Durante o desenvolvimento do trabalho a definição do contexto para computação em grade representou um grande desafio. Optou-se por usar um gráfico visando facilitar essa definição e a compreensão das diversas entidades presentes nas grades. A definição desse modelo contribuiu significativamente para o entendimento dos aspectos inseridos na computação em grades, como por exemplo, o grau de relacionamento existente entre os componentes desse ambiente. No próximo capítulo será feita uma análise da implementação e do real uso de contexto em computação em grade.

CAPÍTULO 5

Resultados

Durante o desenvolvimento do ContextGrid foram realizados diversos testes com o objetivo de permitir uma avaliacao quantitativa do mesmo. Os resultados obtidos com estes testes são apresentados a seguir. Além disso, é discutido um cenário que ilustra a aplicação da infra-estrutura e seus benefícios.

Os testes foram realizados em um ambiente MAG/InteGrade formado por duas máquinas fixas e por dois dispositivos móveis, um celular Nokia 3650 e um *PocketPC* Dell Axim x50v. A especificação de cada dispositivo é apresentada no Apêndice A. As máquinas fixas estavam interligadas através de uma rede Fast-Ethernet, a 100Mbps. Já o *PocketPC* se conectava a seu *proxy* via rede sem fio 802.11b, com taxa de transmissão nominal de até 11Mbps. Por fim, o celular utiliza uma conexão *bluetooth*, com taxa de transmissão de até 1Mbps.

Foram realizados três experimentos quantitativos, um para cada tipo de dispositivo utilizado (fixo, *pocketPC* e celular), com o objetivo de avaliar o impacto do componente LCM. Nota-se que o componente LCM, como executa em todos os nós, inclusive nos nós móveis e por tempo indefinido, não pode consumir muitos recursos, principalmente daqueles dispositivos que já possuem recursos limitados e dependem de bateria, cujo consumo está diretamente relacionado ao uso desses recursos.

5.1 Avaliação do consumo de CPU causado pelo LCM

Nesses testes, o LCM foi avaliado quanto ao protocolo de atualização de contexto, descrito no Capítulo 4. Entende-se que, por ser uma tarefa que executa em segundo plano, sem interação direta com o usuário, e durante todo o ciclo de vida do componente, ela deve consumir o mínimo de recursos possível. O LCM não deve interferir nas atividades do usuário e nem causar um consumo excessivo de recursos e, consequentemente, de bateria, em dispositivos que dela dependem.

Os experimentos foram realizados com o LCM configurado para avaliaro as informações de contexto a cada 10 segundos e, caso fossem essas 10% diferentes das anteriores, deviam ser enviadas ao GCM. Foram realizados 3 experimentos de 5 minutos cada, para efeito do cálculo da média. Durante cada experimento foram coletadas 300 amostras.

5.1.1 Celular

A Figura 5.1 apresenta a média de consumo de CPU para o experimento realizado no celular. Pode-se notar que o consumo se mantém baixo, por volta de 3%, sendo que na maior parte do tempo está muito próximo de 0%.

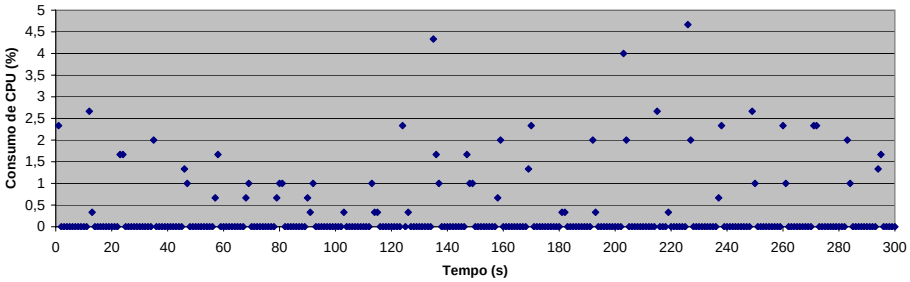

Figura 5.1: *Consumo de CPU do LCM - Celular*

Para esse tipo de dispositivo o LCM foi desenvolvido em Python, com o objetivo de facilitar a portabilidade. Observou-se que o consumo de memória total, com o LCM e com o interpretador Python foi, em média, de 1692 KB, sendo 1324 KB ocupados pelo interpretador Python e 368 KB ocupados pelo LCM. Apesar do consumo um tanto elevado, a memória restante foi suficiente para que todos os aplicativos e funções do dispositivo operassem normalmente. A implementação do LCM apresentou um tamanho elevado em memória. Isto se deve, principalmente, à carga de bibliotecas necessárias para o acesso às informações do dispositivo. Um futuro refatoramento e otimização pode reduzir significativamente este tamanho.

5.1.2 PocketPC

Para o Dell Axim x50v, alguns desafios técnicos, como falta de documentação e restrições do Windows Mobile 2003, não permitiram a coleta direta de informações sobre o uso do processador. Nesse caso, os dados coletados pelo LCM para o atributo "'recursos"' ficaram restritos à memória primária e secundária (conforme apresentado abaixo). Foi possível avaliar o impacto do LCM apenas com o uso de um *software* de código fonte fechado, denominado *RhinoStats V1.2* [1], que apresenta

[1] http://www.pocketgear.com/software_detail.asp?id=5367

os resultados em um gráfico na tela. Segundo observações dos dados apresentados por esse *software*, o LCM causou um consumo extra de CPU no dispositivo de, em média, 3% e nunca superior a 5%.

Também para este dispositivo, o LCM foi implementado em Python. O total de memória usada foi, em média, de 1960 KB, sendo 1730 KB ocupados pelo interpretador da linguagem e 230 KB ocupados pelo LCM. A memória ocupada pelo LCM nesse dispositivo foi menor que a do celular, mas por outro lado, a quantidade ocupada pelo interpretador foi maior. Isso se deve às características de cada porte do interpretador de Python. Um aspecto importante quanto ao menor tamanho do LCM nesse dispositivo foi a necessidade de importar um menor número de bibliotecas que no celular.

5.1.3 Nó fixo

Na Figura 5.2 são apresentados os resultados do experimento para o nó fixo (ver Nó Fixo 2 no Apêndice A). Em média, o LCM apresentou um consumo de CPU de 1%.

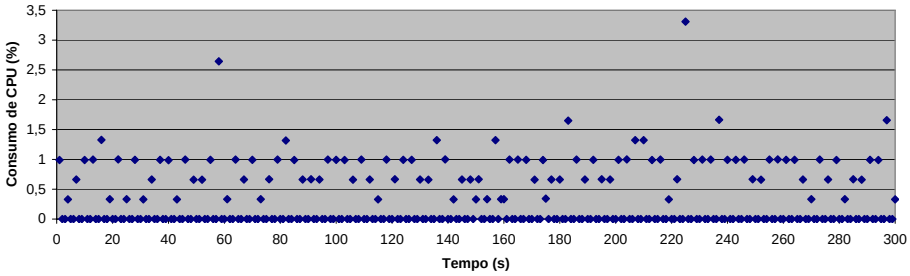

Figura 5.2: *Consumo de CPU do LCM - Nó Fixo 2*

Como pode ser visto, a grande maioria dos valores coletados do uso do processador se aproxima de 0% e nunca é superior a 3,5%, o que confirma o baixo consumo de processador por parte do componente em questão.

5.2 Cenário de Uso

O uso do ContextGrid para permitir a adaptação dos componentes do *middleware* pode ser aplicado a uma grande variedade de aspectos da grade. Como a adaptação é de responsabilidade do desenvolvedor dos componentes que vão e integrar o *middleware*, o objeto e o momento da adaptação são alguns dos aspectos mais flexíveis. Esta seção descreve um cenário de aplicação, juntamente com as adaptações que poderiam ser executadas com base na disponibilidade de informações

de contexto. Esse cenário utiliza as características presentes no *middleware* de grade MAG/InteGrade.

Cenário: Um usuário conecta-se à grade para solicitar a execução de uma aplicação com seu *PDA*. Através de um componente de submissão de tarefas (ASCT), ele detalha os parâmetros para execução e efetiva submissão. Esse dispositivo se conecta à grade por meio de rede sem fio e é associado a um *proxy*, o qual, no momento da conexão do dispositivo, passa a monitorar o contexto deste, o contexto da aplicação e o contexto do usuário. Esse monitoramento é conseguido utilizando-se o método *subscribe* do LCM local para solicitar a inscrição nos canais de eventos *device*, *user* e *application*.

O usuário passa, em seguida, a usar seu *PDA* para ler seus e-mails, enquanto o componente de submissão fica recebendo o estado da execução enviado pela grade e aguardando os resultados. Com o uso, a bateria do dispositivo passa a ter menos de 20% de carga. O *proxy*, tendo o conhecimento do contexto do dispositivo, através do monitoramento das mudanças no contexto, que são notificadas pelo GCM utilizando-se o protocolo de notificação de contexto mostrado no Capítulo 4, realiza adaptações de modo a assumir completamente o lugar do dispositivo na grade, não enviando mais qualquer informação a ele (de modo a poupá-lo). Essas adaptações devem ser implementadas no método *doAdapt* do *proxy*. O componente de submissão de tarefas no dispositivo, também deve implementar a interface *IcontextGridObserver*, apresentada no Capítulo 4, para se tornar um *Context User*. Deste modo, ele pode utilizar o método *subscribe* do LCM local para se inscrever no canal de eventos *device*, passando a tomar conhecimento do contexto do seu dispositivo. Ao perceber a baixa na bateria, o componente de submissão de tarefas entra em um estado no qual não consome recursos.

Após certo tempo, o usuário, com a bateria de seu *PDA* descarregada, recebe uma mensagem do *proxy* em seu telefone celular, informando-lhe que os resultados de sua aplicação já estão disponíveis. O envio dessa mensagem ao celular do usuário é possível através da análise do contexto do usuário, mais especificamente, do atributo *forma de contato*. Ao chegar à sua sala, o usuário executa o componente de submissão de tarefas instalado em seu computador de mesa e, nesse instante, o *proxy* é notificado de uma mudança no dispositivo em uso pelo usuário. Diante da mudança no contexto do usuário, notificada pelo GCM, o *proxy* executa o seu método *doAdapt*, que deverá conter uma adaptação para esse caso. Por exemplo, o *proxy* pode passar todo o estado por ele coletado para o componente no computador do usuário, de forma transparente, como se o usuário tivesse solicitado o processamento a partir daquela máquina.

Em um ambiente convencional, onde não existem adaptações ou ciência

do contexto, provavelmente o usuário perderia seu processamento no momento em que seu dispositivo móvel tivesse a bateria esgotada ou não estivesse mais no raio de alcance da rede sem fio. Isto se deve ao fato de que, na próxima tentativa da grade de enviar o estado do processamento ao componente solicitante (no dispositivo móvel), ocorreria um erro e não haveria mais meios para recuperar os resultados, muitas vezes dispersos por inúmeros nós. Um outro ponto se refere ao consumo de recursos no dispositivo móvel. Como o comportamento da execução do componente de submissão de tarefas não seria alterado, ele continuaria a consumir processador e, conseqüentemente, a contribuir para uma descarga mais rápida da bateria.

5.3 Comentários

Os resultados obtidos demonstram que o uso de contexto é viável, mesmo em dispositivos com poucos recursos. O LCM apresentou um consumo de CPU bastante aceitável. Uma questão não muito satisfatória foi o consumo muito grande de memória nos dispositivos móveis, principalmente para o celular, cujos recursos são extremamente restritos.

Além disso, no cenário de uso discutido, é apresentada uma idéia de onde e como o ContextGrid pode contribuir para as grades e de que maneira. Como a adaptação não é o foco do trabalho, esse cenário não foi implementado efetivamente. Contudo, em sua implementação atual, o ContextGrid comporta perfeitamente a total implementação dos aspectos tratados nesse cenário. Esta validação prática é sugerida como um trabalho futuro imediato.

CAPÍTULO 6

Conclusões e Trabalhos Futuros

Considerando os novos paradigmas de computação ubíqua e a presença cada vez maior de dispositivos móveis, hoje em dia, simplificar a integração destes dispositivos, em especial, às grades computacionais, tende a resolver muitos dos problemas relacionados à quantidade restrita de recursos presentes neste tipo de dispositivo. Esse cenário, onde não só a quantidade de recursos varia muito, mas assim como também os dispositivos presentes na grade e seu ambiente, sugere o uso de adaptação dinâmica como forma de melhorar o aproveitamento dos recursos disponíveis e a adequação dinâmica a essa variabilidade. Essas questões introduzem inúmeros desafios à concepção de mecanismos de suporte para adaptação dinâmica voltados, em especial, para middleware de grades computacionais.

Este trabalho apresentou o ContextGrid, uma infra-estrutura de suporte à computação sensível ao contexto que fornece a base para mecanismos de adaptação dinâmica dos componentes do *middleware* de grade computacional. O ContextGrid provê um conjunto de recursos para que um novo ambiente de grade seja composto, onde o *middleware* é capaz de se adapta às variações de contexto. O objetivo é promover um melhor uso dos recursos disponíveis e a integração com dispositivos móveis que, devido à extrema variedade, têm como pré-requisito a necessidade de adaptação das aplicações e do próprio *middleware*. O ContextGrid foi desenvolvido como uma extensão do *middleware* MAG/InteGrade, com o objetivo de não duplicar esforços nas implementações dos componentes básicos de um *middleware* de grade. Ele foi estruturado de forma a ser independente das fontes fornecedoras de informações contextuais, assim como do formato dos dados originários dessas fontes. Percebeu-se que, a partir do momento em que essas informações passam pelo LCM, elas adquirem um formato padronizado, semanticamente em um nível mais elevado, o que provê essa independência. Ressalte-se que, em um nível mais básico, essa padronização começa nos próprios *wrappers*. O uso desse último componente, fracamente acoplado ao LCM, mostrou-se bastante flexível, permitindo a integração de sensores dos mais variados tipos. Devido ao emprego de CORBA, obteve-se transparência de localização e uma flexibilidade na linguagem usada para implementação. Quanto aos

wrappers, esses podem ser desenvolvidos em qualquer linguagem que tenha suporte para o uso de um ORB CORBA.

A Arquitetura do ContextGrid foi estruturada de modo muito similar a sistemas de *middlewares* de computação em grades, assim como o modelo de contexto, que foi desenvolvido considerando-se esse tipo de ambiente. Sabe-se que este modelo é um ponto de partida para a construção de plataformas de *middlewares* de grade sensíveis ao contexto. Deste modo, o projeto foi concebido de forma a ser o mais flexível possível, permitindo diversas extensões de maneira transparente. A começar pela relação *Wrapper* - LCM, onde existe a possibilidade de inserção e remoção de *wrappers* e, conseqüentemente, de novas informações monitoradas, em tempo de execução. Outro ponto de flexibilidade importante está no GCM. Com a função de interpretar o contexto, o GCM deve ser capaz de possibilitar incontáveis combinações das informações, resultando em diferentes contextos. Enfim, o GCM dever ser capaz de possibilitar inúmeras interpretações dos dados. Pensando nisso, o GCM permite que sejam registrados, em tempo de execução, novos interpretadores de contexto. Estes aspectos possibilitam a expansão, de maneira relativamente simples, do contexto da grade, incluindo alterações no modelo de contexto proposto.

Quando se tem um ambiente sensível ao contexto, adaptações são intrínsecas. Contudo, especificar um padrão para se decidir o que será adaptado (e quando) pode ser uma tarefa muito complexa, além da alta probabilidade de não se conseguir atender a todas as entidades envolvidas da melhor forma. O ContextGrid provê meios através dos quais os componentes desenvolvidos para o *middleware* podem tomar conhecimento do contexto e de suas mudanças, ficando sob responsabilidade do desenvolvedor de tais componentes, decidir qual(ais) contexto(s) é(são) interessante(s) e qual(ais) ação(ões) deve(m) ser tomada(s) diante de um determinado contexto ou de sua mudança. Isso torna o ContextGrid mais flexível no que se refere à adaptação, sendo independente das políticas adotadas.

Testes de desempenho foram realizados para avaliar o impacto dos componentes do ContextGrid, LCM e GCM, no uso relativo de CPU. Estes testes tiveram como principal foco o impacto nos dispositivos móveis, que apresentam recursos limitados. Observou-se que o custo de CPU introduzido foi relativamente pequeno, podendo muitas vezes ser negligenciado para os nós fixos. Um outro teste teve como métrica o consumo de memória em nós móveis. Observou-se um consumo relativamente elevado para os padrões desses dispositivos. Percebeu-se que, esse consumo, em sua maior parte, foi causado pelo interpretador da linguagem Python e pelas diversas bibliotecas que tiveram que ser importadas para o monitoramento do dispositivo. Contudo, apesar do consumo elevado, a memória restante foi suficiente para que todos os aplicativos e funções do dispositivo operassem normalmente. Um futuro

"refatoramento"e otimização podem reduzir significativamente esse problema.

Outro aspecto discutido teve como objetivo uma avaliação qualitativa do impacto das adaptações ao contexto. No cenário descrito, observou-se uma melhora da interação com o usuário, com destaque para as adaptações na entrega dos resultados, por exemplo a notificação através do celular, e na mudança do dispositivo em uso pelo usuário. Quanto à realocação de tarefas cientes de contexto, observou-se um melhor desempenho da aplicação como um todo, mesmo diante do overhead das migrações. Contudo, esta questão deve ser avaliada mais profundamente, com o objetivo de se determinar até que ponto a migração é benéfica, ou seja, qual o valor desse overhead e quando ele pode ser efetivamente desprezado.

Entre as diversas contribuições deste trabalho, pode-se destacar o desenvolvimento de uma infra-estrutura de suporte a contexto como extensão ao *middleware* de grade MAG/InteGrade, conforme apresentado no Capítulo 4. Outra importante contribuição é a investigação do uso de adaptação sensível ao contexto em grades computacionais. Destaca-se também a exploração de dispositivos móveis nesse cenário e a avaliação do impacto causado neles, principalmente quanto ao consumo de recursos. O uso de tais dispositivos integrados à grade traz consigo a necessidade de suporte a contexto para que essa integração seja efetiva.

Observou-se que o uso de adaptação sensível ao contexto contribuiu para uma exploração mais racional dos recursos disponíveis, permitindo a adequação às peculiaridades de cada dispositivo. Isso também permitiu que essas peculiaridades fossem transparentes para o usuário, ou seja, percebeu-se que, decisões para uma melhor interação com o usuário podem ser tomadas sem o conhecimento do mesmo.

Este trabalho admite uma série de melhorias e extensões, dentre as quais podemos destacar as que se seguem.

- O uso de sensores físicos (*hardware*) como fornecedores de contexto: para uma exploração completa do contexto, é necessário o uso de sensores físicos na obtenção de um contexto mais elaborado e preciso. Neste trabalho, alguns deles foram apenas simulados, como no caso da localização. Uma extensão interessante seria explorar melhor esse tipo de sensor, com seu uso efetivo, e até mesmo a exploração de redes de sensores, ampliando o contexto monitorado pelo ContextGrid, assim como sua precisão.
- Exploração de recursos adaptáveis ao contexto: não só os componentes do *middleware* necessitam se adaptar, mas os recursos também podem se adaptar ao contexto da grade, como por exemplo, rede de sensores fornecidas como recursos cuja interação entre os sensores seja configurada e reconfigurada com base no contexto. Assim, uma possível abordagem que explore o uso

do contexto disponibilizado pelo ContextGrid, para adaptação dos recursos poderia contribuir para uma inserção cada vez mais ampla do uso de contexto em todos os níveis das grades computacionais

- Investigar o uso do contexto para adaptação das aplicações de usuário: o uso do contexto explorado nesse trabalho se limitou aos componentes do *middleware* de grade. Contudo, assim como o contexto tem contribuído para inúmeros tipos de aplicações e cenários, pode-se explorar a adaptação ciente de contexto nas aplicações que executam em grades computacionais.
- Uso de reflexão computacional ciente de contexto no *middleware* de grade: o uso de técnicas de *middleware* reflexivos em computação em grades tem merecido, cada vez mais o interesse na comunidade científica. O uso de contexto aliado a eles pode tornar a reflexão mais precisa, uma vez que a reflexão teria com insumo, também, o contexto.
- Explorar os conceitos de computação ubíqua: o contexto se apresenta como pré-requisito para computação ubíqua. Uma vez que o contexto já está sendo fornecido pelo ContextGrid, assim como meios para adaptação ciente de contexto, explorar a inserção da computação em grade em sistemas de computação ubíqua pode representar um importante passo rumo a um sistema de grade que realmente seja tão fácil de usar quanto a energia elétrica.

Referências Bibliográficas

[1] 02. **Luaforge:project info- o2 - an orb in lua**. http://luaforge.net/projects/o-two/, maio 2006.

[2] 4.0, G. T. **Globus toolkit 4.0 release manuals**. http://www-unix.globus.org/toolkit/docs/4.0/, maio 2006.

[3] BARBOSA, J. L. V. **Holoparadigma: Um Modelo Multiparadigma Orientado ao Desenvolvimento de Software Distribuído**. PhD thesis, 2002.

[4] BARBOSA, R; GOLDMAN, A. **Mobigrid: Framework for mobile agents on computer grid environments**. In: FIRST INTERNATIONAL WORKSHOP ON MOBILITY AWARE TECHNOLOGIES AND APPLICATIONS, Florianopolis, SC, Brazil, 2004.

[5] BASNEY, J; LIVNY, M. **Managing network resources in Condor**. In: PROCEEDINGS OF THE NINTH IEEE SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING (HPDC9), p. 298–299, Pittsburgh, PA, August 2000.

[6] BROWN, P. J. **The stick-e document: a framework for creating context-aware applications**. In: PROCEEDINGS OF EP'96, PALO ALTO, p. 259–272, January 1996.

[7] BRUNEO, D; SCARPA, M; ZAIA, A; PULIAFITO, A. **Communication paradigms for mobile grid users**. ccgrid, 00:669, 2003.

[8] BUYYA, R. **High Performance Cluster Computing: Architectures and Systems**. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[9] BUYYA, R; BRANSON, K; GIDDY, J; ABRAMSON, D. **The virtual laboratory: A toolset for utilising the world-wide grid to design drugs**. In: CCGRID '02: PROCEEDINGS OF THE 2ND IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, 2002.

[10] CALVI, C. Z; PESSOA, R. M; FILHO, J. G. P. **Um interpretador de contexto para plataforma de serviços context-aware**. In: XXV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAçãO, p. 1990–2004, julho 2005.

[11] CAPORUSCIO, M; INVERARDI, P. **Yet another framework for supporting mobile and collaborative work**. In: INTERNATIONAL WORKSHOP ON DISTRIBUTED AND MOBILE COLLABORATION, Linz, Austria, June 2003.

[12] CAPRA, L; EMMERICH, W; MASCOLO, C. **Carisma: Context-aware reflective middleware system for mobile applications**. IEEE Transactions on Software Engineering, 29(10):929–945, Oct. 2003.

[13] CHEN, G; KOTZ, D. **A survey of context-aware mobile computing research**. Technical report, Hanover, NH, USA, 2000.

[14] CHEN, G; KOTZ, D. **Solar: A pervasive-computing infrastructure for context-aware mobile applications**. Technical Report TR2002-421, Dartmouth College, Computer Science, Hanover, NH, February 2002.

[15] CHEN, G; KOTZ, D. **Solar: An open platform for context-aware mobile applications**. In: PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING (SHORT PAPER), p. 41–47, June 2002.

[16] CHEN, G; LI, M; KOTZ, D. **Design and implementation of a large-scale context fusion network**. mobiquitous, 00:246–255, 2004.

[17] CONDOR. **Condor project homepage**. http://www.cs.wisc.edu/condor/, maio 2006.

[18] CZAJKOWSKI, K; FERGUSON, D; FOSTER, I; FREY, J; GRAHAM, S; MAQUIRE, T; SNELLING, D; ; TUECKE, S. **From open grid services infrastructure to wsresource framework: Refactoring & evolution**, May 2004.

[19] DA SILVA JUNIOR, A. J; DE MIRANDA, M. N; COSTA, F. M. **Contextgrid – an infrastructure for context support for integration of mobile devices into the computational grid**. In: IV WORKSHOP ON GRID COMPUTING AND APPLICATIONS, Curitiba, Paraná, 2006.

[20] DA SILVA JUNIOR, A. J; DE MIRANDA, M. N; COSTA, F. M. **Contextgrid: Uma infra-estrutura de suporte a contexto para integração de dispositivos móveis a grades computacionais**. In: VII WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO (SUBMETIDO), Ouro Preto, Minas Gerais, 2006.

[21] DEY, A. K; ABOWD, G. D; BROWN, P. J; DAVIES, N; SMITH, M; STEGGLES, P. **Towards a better understanding of context and context-awareness**. In: HUC '99: PROCEEDINGS OF THE 1ST INTERNATIONAL SYMPOSIUM ON HANDHELD AND UBIQUITOUS COMPUTING, p. 304–307, London, UK, 1999. Springer-Verlag.

[22] DEY, A. K. **Providing architectural support for building context-aware applications**. PhD thesis, 2000. Director-Gregory D. Abowd.

[23] DOUGLASS, B. P. **Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[24] EXEHDA. **Exehda: Ambiente de execução direcionado à computação pervasiva**. http://www.inf.ufrgs.br/ exehda/, maio 2006.

[25] FILHO, J. V; SACRAMENTO, V; DA ROCHA, R; ENDLER, M. **Moca: Uma arquitetura para o desenvolvimento de aplicações sensíveis ao contexto para dispositivos móveis**. In: PROC. OF THE XXIV SIMPóSIO BRASILEIRO DE REDES DE COMPUTADORES (SBRC),TOOL SESSION, volume II, Curitiba, Brazil, May 2006.

[26] FORMAN, G. H; ZAHORJAN, J. **The challenges of mobile computing**. Computer, 27(4):38–47, 1994.

[27] FOSTER, I; GEISLER, J; NICKLESS, W; SMITH, W; TUECKE, S. **Software infrastructure for the i-way high-performance distributed computing experiment**. p. 562–571. IEEE Computer Society Press, 1996.

[28] FOSTER, I; KESSELMAN, C. **The globus toolkit**. p. 259–278, 1999.

[29] FOSTER, I; KESSELMAN, C; TUECKE, S. **The anatomy of the grid: Enabling scalable virtual organizations**. Int. J. High Perform. Comput. Appl., 15(3), 2001.

[30] FUNFROCKEN, S. **Transparent migration of java-based mobile agents: Capturing and reestablishing the state of java programs**. In: SECOND INTERNATIONAL WORKSHOP ON MOBILE AGENTS, p. 26–37, September 1998.

[31] GHIDINI, C; GIUNCHIGLIA, F. **Local models semantics, or contextual reasoning = locality + compatibility**. Artif. Intell., 127(2):221–259, 2001.

[32] GIRARDI, R. **An analysis of the contributions of the agent paradigm for the development of complex systems**. In: ISAS-SCI '01: PROCEEDINGS OF THE WORLD MULTICONFERENCE ON SYSTEMICS, CYBERNETICS AND INFORMATICS, p. 388–393. IIIS, 2001.

[33] GLOBUS. **The globus alliance**. http://www.globus.org/, maio 2006.

[34] GOLDCHLEGER, A. **Integrade: Um sistema de middleware para computação em grade oportunista**. Master's thesis, Instituto de Matemática e Estatística - Universidade de São Paulo, dezembro 2004.

[35] GRUBER, T. R. **Towards Principles for the Design of Ontologies Used for Knowledge Sharing**. In: Guarino, N; Poli, R, editors, Formal Ontology in Conceptual Analysis and Knowledge Representation, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[36] GSI. **Overview of the grid security infrastructure**. http://www.globus.org/security/overview.html, maio 2006.

[37] HENRICKSEN, K; INDULSKA, J. **Modelling and using imperfect context information**. In: PROCEEDINGS OF THE SECOND IEEE ANNUAL CONFERENCE ON PERVASIVE COMPUTING AND COMMUNICATIONS WORKSHOPS, p. 33–37, Orlando, FL, US, 2004.

[38] HENRICKSEN, K; INDULSKA, J; RAKOTONIRAINY, A. **Modeling context information in pervasive computing systems**. In: PERVASIVE '02: PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING, p. 167–180, London, UK, 2002. Springer-Verlag.

[39] IEEE. **Ieee 802.11 the working group setting the standards for wireless lans**. http://grouper.ieee.org/groups/802/11/, maio 2006.

[40] INTEGRADE. **Mesh networking - networking research group**. http://www.integrade.org.br, maio 2006.

[41] ISAM. **Projeto isam - infraestrutura de suporte às aplicações móveis distribuídas**. http://www.inf.ufrgs.br/ isam/, maio 2006.

[42] JADE. **Jade - java agent development framework**. http://jade.tilab.com/, maio 2006.

[43] KERIEVSKY, J. **Refactoring to Patterns (Addison-Wesley Signature Series)**. Addison-Wesley Professional, August 2004.

[44] KRAUTER, K; BUYYA, R; MAHESWARAN, M. **A taxonomy and survey of grid resource management systems for distributed computing**. Software Practice and Experience, 32(2):135–164, February 2002.

[45] LANGE, D. B; OSHIMA, M. **Seven good reasons for mobile agents**. Commun. ACM, 42(3):88–89, 1999.

[46] LITZKOW, M; LIVNY, M; MUTKA, M. **Condor - a hunter of idle workstations**. In: PROCEEDINGS OF THE 8TH INTERNATIONAL CONFERENCE OF DISTRIBUTED COMPUTING SYSTEMS, June 1988.

[47] LOPES, R. F. **MAG: uma grade computacional baseada em agentes móveis**. Master's thesis, Universidade Federal do Maranhão, Sã Luís, MA, Brasil, January 2006. In Portuguese.

[48] MCKNIGHT, L. W; HOWISON, J; BRADNER, S. **Guest editors' introduction: Wireless grids–distributed resource sharing by mobile, nomadic, and fixed devices**. IEEE Internet Computing, 8(4):24–31, 2004.

[49] MORENO, R. A; FURUIE, S. S. **Utilização de contexto para visualização de imagens médicas**. In: IX CONGRESSO BRASILEIRO DE INFORMáTICA EM SAúDE, 2004.

[50] OGSA. **Open grid system architecture**. http://www.globus.org/ogsa, maio 2006.

[51] OGSI. **Open grid system infrastructure**. http://www.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf, maio 2006.

[52] PASCOE, M. J. **Adding generic contextual capabilities to wearable computers**. In: ISWC '98: PROCEEDINGS OF THE 2ND IEEE INTERNATIONAL SYMPOSIUM ON WEARABLE COMPUTERS, p. 92, Washington, DC, USA, 1998. IEEE Computer Society.

[53] PHAN, T; HUANG, L; DULAN, C. **Challenge:: integrating mobile wireless devices into the computational grid**. In: MOBICOM '02: PROCEEDINGS OF THE 8TH ANNUAL INTERNATIONAL CONFERENCE ON MOBILE COMPUTING AND NETWORKING, 2002.

[54] RUBINSZTEJN, H; ENDLER, M; RODRIGUES, N. **A framework for building customized adaptation proxies**. In: PROC. OF THE IFIP CONFERENCE

ON INTELLIGENCE IN COMMUNICATION SYSTEMS (INTELLCOMM 2005), MONTREAL, oct 2005.

[55] SATYANARAYANAN, M. **Fundamental challenges in mobile computing**. In: SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, p. 1–7, 1996.

[56] SCHILIT, B; ADAMS, N; WANT, R. **Context-aware computing applications**. In: IEEE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS, Santa Cruz, CA, US, 1994.

[57] SCHILIT, B. N. **A Context-Aware System Architecture for Mobile Distributed Computing**. PhD thesis, 1995.

[58] SCHMIDT, A; BEIGL, M; GELLERSEN, H.-W. **There is more to context than location**. Computers and Graphics, 23(6):893–901, 1999.

[59] SERVICES, W. W. **Web services activity**. http://www.w3.org/2002/ws/, maio 2006.

[60] SGML. **Overview of sgml resources**. http://www.w3.org/MarkUp/SGML/, maio 2006.

[61] STRANG, T. **Towards autonomous context-aware services for smart mobile devices**. In: Chen, M.-S; Chrysanthis, P. K; Sloman, M; Zaslavsky, A, editors, LNCS 2574: PROCEEDINGS OF THE 4TH INTERNATIONAL CONFERENCE ON MOBILE DATA MANAGEMENT (MDM2003), Lecture Note in Computer Science (LNCS), p. 279–293, Melbourne/Australia, January 2003. Springer.

[62] STRANG, T; LINNHOFF-POPIEN, C. **A context modeling survey**. 2004.

[63] STRANG, T; MEYER, M. **Agent-environment for small mobile devices**, 2002.

[64] TRUYEN, E; ROBBEN, B; VANHAUTE, B; CONINX, T; JOOSEN, W; VERBAETEN, P. **Portable support for transparent thread migration in java**. In: ASA/MA, p. 29–43, 2000.

[65] VIANA, W; TEIXEIRA, R; CAVALCANTE, P; ANDRADE, R. **Mobile adapter: Uma abordagem para a construção de mobile application servers adaptativos utilizando as especificações cc/pp e uaprof**. In: ANAIS XXV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, p. 1914–1929. SBC, julho 2005.

[66] WANT, R; PERING, T. **System challenges for ubiquitous & pervasive computing**. In: ICSE '05: PROCEEDINGS OF THE 27TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, p. 9–14, New York, NY, USA, 2005. ACM Press.

[67] WEISER, M. **The computer for the 21st century**. p. 933–940, 1995.

[68] WSDL, W. **Web services description language (wsdl) 1.1**. http://www.w3.org/TR/wsdl, maio 2006.

[69] XINDICE. **Apache xindice**. http://xml.apache.org/xindice/, maio 2006.

[70] YAMIN, A; BARBOSA, J; AUGUSTIN, I; SILVA, L; AMD G. CAVALEIRO, C. G. **Towards merging context-aware, mobile and grid computing**. International Journal Of High Performance Applications, 17(2):191–203, 2003.

[71] YAMIN, A. C. **Arquitetura para um Ambiente de Grade Computacional Direcionado às Aplicações Móveis, Distribuídas e Conscientes do Contexto da Computação Pervasiva**. PhD thesis, 2004.

[72] ÖZTÜRK, P; AAMODT, A. **Towards a model of context for case-based diagnostic problem solving**. In: CONTEXT-97: PROCEEDINGS OF THE INTERDISCIPLINARY CONFERENCE ON MODELING AND USING CONTEXT, p. 198–208, Rio de Janeiro, RJ, Brazil, February 1997.

APÊNDICE A

Especificação dos dispositivos utilizados

Máquina	**Processador**	**Mem. RAM**	**Sistema**
Nó Fixo 1	AMD Athlon 1.25GHz	512 MB	Linux 2.6.13
Nó Fixo 2	AMD Athlon 64 2.0GHz	1 GB	Linux 2.6.13
Nó Móvel 1	Intel ARM 104MHz	3.2 MB	SymbianOS 6.1
Nó Móvel 2	Intel XScale 624MHz	64 MB	Win. Mobile 2003