Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Gustavo Bervian Brand

Ubiservices: um Modelo para Administração de Serviços em Ambientes Ubíquos

São Leopoldo, 2010

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

GUSTAVO BERVIAN BRAND

Ubiservices: um Modelo para Administração de Serviços em Ambientes Ubíquos

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

Prof. Dr. Jorge Luis Victória Barbosa
Orientador

São Leopoldo, 2010.

Agradecimentos

Agradeço à minha família, amigos e esposa pela força e empurrões constantes ao longo dos últimos dois anos em que cursei o mestrado na Unisinos. Desde o começo se enfrentam dificuldades, primeiro devido a cadeiras trabalhosas e a expectativa das notas. Depois, no segundo ano, pela auto cobrança com relação à proposta e depois com relação a dissertação propriamente dita.

Ao meu professor orientador, Barbosa, não tenho palavras para definir seu apoio constante e incentivo, tendo a visão dos caminhos possíveis para seguir em frente quando eu estava quase por desistir.

Resumo

Com a evolução acelerada dos dispositivos computacionais, diminuição de tamanho e custo, ocorre uma imersão das pessoas no mundo digital. Com o uso de mais tecnologia, abundante e barata, surgem ambientes que tiram proveito da computação ubíqua, permitindo aos seus usuários uma interatividade e acesso a informações de forma fácil, ágil e natural. No entanto, ambientes ubíquos foram definidos em função da necessidade de uma grande infra-estrutura instalada para funcionar adequadamente, algo que não é uma realidade atualmente. Este trabalho apresenta uma alternativa aos ambientes ubíquos clássicos: um modelo para administração de serviços chamado Ubiservices. O objetivo do Ubiservices é fornecer conectividade em ambientes formados através de redes *Ad Hoc* não estruturadas, com aplicações como migração, replicação, publicação e busca de informações e serviços nesta rede. A interconexão das redes *Ad Hoc* ocorre através de uma rede *overlay* de forma transparente ao usuário, utilizando tecnologias existentes que, devidamente adaptadas e aperfeiçoadas, conseguem formar uma rede que permite a diferentes redes *mesh* locais interagirem, mesmo estando em diferentes locais.

Abstract

With the fast evolution of the computing devices, its reduced size and cost, there is an immersion of people in the digital world. With the use of more technology, abundant and cheap, there are environments that can take advantage of the ubiquitous computing, allowing users to interact and have access to information easily, quickly and naturaly. However, ubiquitous environments were defined needing a large installed infrastructure to work properly, something that is not a reality nowadays. This work presents an alternative to the classic ubiquitous environments called Ubiservices. Its goal is to provide connectivity to environments made up of unstructured Ad Hoc networks, besides applications like migration, replication, publishing and search of information and services at the network. The interconnection of Ad Hoc networks occurs through an overlay network, user transparently, using existing technologies that, properly adapted and refined, can create a network that allows different mesh networks to interact, even if they are placed in different locations.

SUMÁRIO

LISTA DE ABREVIATURAS

API	Application Programming Interface
GTK	GNU Tools Kit
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
JXTA	P2P middleware (from “Juxtaposes”)
LAN	Local Area Network
MANET	Mobile Ad Hoc Network
OLSR	Optimized Link State Routing
P2P	Peer to peer
PDA	Personal Digital Assistant
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UOP	Ubiquitous Oriented Programming
WMN	Wireless Mesh Networks
WPAN	Wireless Personal Area networks
WSN	Wireless Sensor Network
XML	Extensible Markup Language

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Os dispositivos computacionais vêm diminuindo consideravelmente em tamanho e custo, sendo um exemplo disso equipamentos como celulares inteligentes, PDAs e *netbooks* (*notebooks* com pouca capacidade de processamento), cuja evolução acelerada traz para o cotidiano uma imersão cada vez maior neste mundo digital. Na medida em que a tecnologia móvel fica abundante e barata, tanto em termos de hardware disponível quanto na comunicação entre os mesmos, surgem os sistemas ubíquos [Weiser 1991][Satyanarayanan 2001].

Apesar da maior parte do hardware e da tecnologia para sistemas ubíquos estar disponível atualmente, eles ainda não se tornaram uma realidade presente no dia a dia devido a falta de padronização e hardware suficiente espalhado geograficamente para tornar a proposta de um ambiente ubíquo clássico disponível em grandes centros de forma transparente aos usuários. Para que isto aconteça ainda existem desafios a serem vencidos. Desde que Weiser concebeu sua visão de ubiqüidade [Weiser 1991], evoluções importantes no hardware puderam ser constatadas através da miniaturização de dispositivos eletrônicos, comunicação sem fio e de acesso contínuo e ininterrupto a informações.

Este cenário possibilitou o surgimento de sistemas que têm por objetivo criar o suporte para o desenvolvimento de ambientes que tiram proveito de aspectos da computação ubíqua. Como alguns exemplos disso, tem-se: Aura [Garlan 2002], Gaia [Román 2002], One-world [Grimm 2001], ISAM [Augustin 2004], EXEHDA [Yamin 2004] e PHolo [Barbosa 2007]. Além disso, existem diversos *middlewares* ubíquos que tem características interessantes para a área, como modularização, mobilidade de código e foco em ambientes *Ad Hoc*. Alguns exemplos são o Meshmdl [Hermann 2003 e 2007], Mobile Gaia [Shankar 005] e Lime [Murphy 2003] que vêm ao encontro à exploração de ambientes menos povoados de sensoriamento e estrutura [Bouillet 2008].

Para uma plena experiência dentro de um ambiente ubíquo, a interação com os demais integrantes é fundamental. Tal atividade é possível através de serviços [Edwards 2006], pois de nada adianta ter acesso à rede se através do ambiente não for possível a utilização de diferentes fontes de informação. Os serviços vêm ao encontro à esta demanda, servindo também como difusores de dados, onde o próprio usuário pode disponibilizar informações. Atualmente os dispositivos não só fazem uso de diferentes redes e de aplicativos distribuídos, como também dependem destes, ocasionando uma crescente utilização de serviços Web e dispositivos computacionais móveis [Kindberg 2001].

A administração de serviços em ambientes dinâmicos se mostra um desafio. Dentre os problemas envolvidos estão à alta dinamicidade dos nós que compreendem a rede e formas para contornar o fraco acoplamento existente entre os mesmos.

1.1 Definição do Problema e da Questão de Pesquisa

Diversos desafios da computação ubíqua foram abordados em áreas pré existentes, como o processamento paralelo/distribuído ou redes de computadores. Mas o surgimento da computação ubíqua traz novas questões, de forma que nem todas as soluções já conhecidas podem ser diretamente aplicadas e outros problemas são completamente novos, não possuindo nenhuma abordagem pronta [Weiser 1991].

Em um ambiente ubíquo as aplicações precisam ser **adaptativas** [Satyanarayanan 2001][Saha 2003], permitindo a troca ou cópia de elementos lógicos em tempo de execução, ganhando mais funcionalidade com isso. Para tal fim entra em cena a mobilidade de código [Fuggetta 1998]. A **sensibilidade ao contexto** é outra necessidade [Augustin 2004][Dey 2001], que permite as aplicações ubíquas estarem conscientes das modificações ocorridas na sua volta para adaptarem o seu comportamento. A heterogeneidade de dispositivos e redes de comunicação exige que os recursos disponíveis precisem ser utilizados da melhor forma possível. Por fim, **robustez** é outro ponto chave para suportar desconexões e falhas inerentes a esse tipo de ambiente [Satyanarayanan 2001].

Redes *Ad Hoc* possuem semelhanças com ambientes ubíquos, e podem ser a base física para o mesmo, com diversos PDAs ou telefones celulares que possam interagir sem o suporte de um ponto centralizador, se obtêm um exemplo de uma rede *Ad Hoc*. Se o *middleware* presente nos dispositivos suportar as características descritas acima, pode-se ter um ambiente ubíquo [Hadim 2006].

Nos contextos isolados de redes locais ou redes estruturadas existem diversas soluções de administração de serviços. No entanto, dentre as soluções pesquisadas e detalhadas no capítulo 3, nenhuma solução se mostra efetiva em termos de conectividade e fornecimento de serviços entre diversas redes *Ad Hoc* não estruturadas [Edwards 2006].

Neste contexto a administração de serviços tem grande importância, uma vez que serve de base para a troca de informações. O suporte à busca e distribuição de serviços deve ser ágil, exigindo um protocolo de **descoberta** e **publicação** dos serviços disponíveis. Devido às características adaptativas do ambiente, é outro requisito o suporte a **replicação** e **migração** dos serviços em execução. Com isso é possível fornecer um suporte de

administração de serviços completo, contemplando desde a descoberta até a contínua execução e entrega dos mesmos.

A mobilidade de código se mostra outro requisito importante para a replicação e migração de serviços. Este trabalho não tem por fim desenvolver tal funcionalidade, mas fazer uso de um ambiente que suporte as especificações de ubiqüidade, como mobilidade de código, e suporte a contextos.

Considerando os problemas citados, surge a questão de pesquisa:

- Como fornecer a conectividade e suporte necessários em ambientes ubíquos formados por redes *Ad Hoc* não estruturadas, possibilitando a interconexão das mesmas, troca de serviços e interação de forma transparente para o usuário?

1.2 Objetivos

O presente trabalho tem como objetivo fazer um estudo sobre ubiqüidade e sistemas ubíquos, e posteriormente especificar, implementar e validar um modelo de administração de serviços que possibilite a busca, publicação, execução, replicação e migração de serviços entre diferentes redes *Ad Hoc* de pequena escala. A interação entre estas redes ocorrerá dentro de uma rede maior presente na *Internet*, categorizada como uma rede *overlay*, a qual transcende o escopo local das diferentes redes *Ad Hoc*, permitindo a sua interconexão. Para tal fim, precisa-se de um ambiente com suporte às especificações de ubiqüidade, como mobilidade de código, e suporte a contextos.

Os objetivos específicos deste trabalho são:

- Estudar e sumarizar os sistemas ubíquos existentes na bibliografia;
- Identificar os requisitos para uma arquitetura distribuída e ubíqua;
- Criar um modelo distribuído para administração de serviços e informações;
- Especificar e implementar um protótipo para execução dentro de um ambiente controlado, onde a localização e distribuição de serviços sejam a prioridade;
- Avaliar a proposta e resultados atingidos.

1.3 Metodologia e organização da proposta

A viabilização dos objetivos propostos neste trabalho passa, em um primeiro momento, pela busca por sistemas similares e o estudo detalhado das suas capacidades de administração de serviços. A partir desde ponto pode ser projetada uma arquitetura modular e, uma vez que se tenham as interfaces funcionais, pretende-se efetuar testes de descoberta de recursos e localização no contexto e grupos associados a um ambiente que suporte as

especificações de ubiqüidade, como mobilidade de código e suporte a contextos [Satyanarayanan 2001].

No capítulo 2 são explicados conceitos e feita uma contextualização relativa aos ambientes ubíquos. Em seguida, no capítulo 3, são detalhados trabalhos relacionados. No capítulo 4 é mostrado o modelo proposto e, no capítulo 5 são discutidos aspectos de implementação e dificuldades encontradas. Após, no capítulo 6, são apresentados os cenários de avaliação para o modelo proposto no capítulo 4, finalizando no capítulo 7 com as considerações finais, contribuições e trabalhos futuros.

2 CONCEITOS BÁSICOS

Neste capítulo são apresentados conceitos sobre áreas importantes no contexto da proposta. As seções apresentam os temas de forma resumida, buscando servir como guia para o aprofundamento dos estudos. São descritos conceitos sobre serviços, redes *mesh* e computação ubíqua.

2.1 Busca e publicação de serviços

Atualmente o reaproveitamento de componentes é um fator importante em termos competitivos para o mercado de *software*. Mais do que isso, a interconexão entre diferentes sistemas é fundamental para extração e processamento de dados, pois a maior parte das empresas possui tanto sistemas legados quanto códigos recém criados para dar suporte à operacionalização de novos produtos e plataformas.

A interação entre a gama de softwares e plataformas distintas pode ocorrer tendo serviços como a base de reutilização e interoperabilidade. A Computação Orientada a Serviços [Papazoglou 2003] endereça esta tendência diretamente, pensando no agrupamento de serviços para sua reutilização e escalabilidade.

Para o usuário final, o uso de serviços deve ser algo natural. Em ambientes ubíquos, esse desafio é particularmente maior, uma vez que a permanência dos dispositivos no mesmo é dinâmica [Edwards 2006]. Para tornar possível esse cenário são necessários mecanismos de localização e descoberta de serviços. Um usuário deve poder descobrir e utilizar um serviço sem conhecimento prévio do mesmo.

Existem diversos protocolos e *middlewares* para tal fim como o mDns [Cheshire 2008a], DNS-SD [Cheshire 2008b], WS Discovery [Goland 1999], Rescue [Juszczyk 2008], em geral caracterizados pelas necessidades de seus criadores, as quais podem ir desde comércio eletrônico até gerenciamento de catástrofes. Os sistemas de localização existentes no geral deixam o usuário encontrar conteúdo de seu próprio interesse, não fornecendo explicitamente algo predefinido.

Na seção 3.1 do próximo capítulo serão detalhados protocolos de administração de serviços utilizados atualmente.

2.2 Redes mesh sem fio

Dentro deste cenário criado pela computação ubíqua, a falta de estrutura pode determinar o isolamento do usuário, que acabaria subutilizando seu PDA ou *smartphone*.

Considerando ambientes não estruturados, as redes *mesh* [Tonnesen 2009][Perkins 1998][Johnson 1998] são uma opção válida para interconexão de usuários.

Elas são uma tecnologia emergente e promissora para redes sem fio, provendo cobertura dentro de amplas áreas, pois fazem uso do sinal de todos os nós para interconectá-los e realizar o roteamento de pacotes sem a necessidade de um ponto central. Uma vez que no mínimo um nó da rede *mesh* esteja conectado a um *link* externo, todos os integrantes da rede podem conseguir acesso através dele [Akyildiz 2005].

As redes *mesh* sem fio consistem de dois tipos de nós: clientes *mesh* e roteadores *mesh*. Os nós roteadores possuem, além de funcionalidade comum de um roteador sem fio, funções extras para suportar redes *mesh*. Comparado com um roteador sem fio comum, um roteador *mesh* consegue atingir a mesma cobertura utilizando menos potência e energia para transmitir o seu sinal, uma vez que faz uso de comunicação *multi-hop*, ou seja, os pacotes podem passar por vários nós até chegar ao seu destino. As redes *mesh* podem estar inclusas dentro da categoria de MANETs (*Mobile Ad Hoc Networks*) e WMNs (*Wireless Mesh Networks*) [Akyildiz 2005].

2.3 Computação Ubíqua

A computação ubíqua consiste no acesso e uso de ambientes computacionais em qualquer lugar, a qualquer momento, através de quaisquer meios ou dispositivos, e inclui questões das áreas de computação distribuída e computação móvel [Satyanarayanan 2001].

A quantidade de desafios criados especificamente para esta área está em constante crescimento. Desde a descrição inicial de Weiser [Weiser 1991] até hoje uma grande quantidade de estudos e trabalhos científicos foram realizados na área de computação ubíqua, mas a maioria deles foi baseada em tecnologias emergentes e/ou futuristas, desprovidas de uma base de tecnologia que desse o suporte adequado para a sua avaliação e execução. No entanto, este cenário está em constante mudança na medida em que os dispositivos móveis se tornam cada vez mais comuns. Além disso, a capacidade de processamento, armazenamento e comunicação dos mesmos cada vez aumenta mais, trazendo a computação ubíqua para o mundo cotidiano.

O objetivo da computação ubíqua é resolver um conjunto de problemas, e não de criar algo completamente novo como inicialmente proposto por Weiser[Weiser 1991]. Ou seja, o ambiente computacional deve desaparecer para o usuário final, de forma que a tecnologia de última geração esteja sendo usada corriqueiramente e de forma natural [Schmidt 2006].

As inúmeras questões que precisam ser solucionadas na área de computação ubíqua não são apenas relativas a questões técnicas, mas sobre questões comportamentais e sociais também. Ou seja, a cultura para uso de equipamentos computacionais deve existir em massa e os desafios devem englobar também questões além das técnicas [Lyytinen 2002].

Nos últimos anos foram criados vários modelos para ambientes de computação ubíqua, como o Aura [Garlan 2002], Gaia [Román 2002], One-world [Grimm 2001], ISAM [Augustin 2004], EXEHDA [Yamin 2004] e PHolo [Barbosa 2007].

2.4 Considerações sobre o capítulo

Neste capítulo foram discutidos conceitos a respeito de três áreas chave para este trabalho, começando pelo que são caracterizados os serviços e sua utilização, seguindo com redes *mesh* e finalizando com idéias sobre a área de computação ubíqua em geral.

No próximo capítulo serão apresentados trabalhos relacionados às áreas de descoberta de serviços, começando por protocolos de descoberta e administração e seguindo adiante com a referência de alguns *middlewares* ubíquos recentes.

3 TRABALHOS RELACIONADOS

Atualmente o uso de protocolos de descoberta de serviços é comum na maior parte dos sistemas operacionais, seja para encontrar impressoras na rede local com o *Bonjour* da Apple [Apple 2009] ou detectar qualquer tipo de serviço com o software *Avahi* no Linux [Poettering 2009]. Ambos são referências para ambientes de rede estruturados e se baseiam em protocolos já conhecidos e publicados pela IETF [IETF 2009], como o mDns [Cheshire 2008a] e DNS-SD [Cheshire 2008b]. Eles são detalhados juntamente com outros protocolos existentes na seção 3.1, abordando tecnologias de descoberta e publicação de serviços. Em seguida, na seção 3.2, são citados *middlewares* ubíquos recentes, observando suas características relevantes para ambientes ubíquos *Ad Hoc*, além de formas de comunicação e eventual suporte a algum tipo de serviço ou protocolo para tal fim.

3.1 Tecnologias de descoberta e publicação de serviços

3.1.1 mDns

Este protocolo tem como objetivo a introdução de suporte a requisições DNS *multicast*, onde os *hosts* conectados podem cooperar entre si para a divulgação e captura de informações na rede local. Foi oficializado pelo trabalho do *Zeroconf IETF group*, um grupo de pesquisa destinado à otimização das configurações de rede [Cheshire 2008a].

Dado o domínio local (.local), uma requisição de DNS precisa ser enviada para o endereço 224.0.0.251. O protocolo especifica diferentes possibilidades de requisições: simples (*one shot)*, envio simples com múltiplas respostas acumuladas e requisições *multicast* pedindo por respostas *unicast*.

Com a sua utilização o tráfego na rede aumenta, mas diversas técnicas de supressão de requisições duplicadas existem para diminuir o *overhead* na rede.

3.1.2 DNS-SD

Juntamente com mDns, o DNS-SD traz o conceito de descoberta de serviços baseados em requisições de DNS. A idéia é utilizar a estrutura maciçamente instalada em diversas redes, como servidores DNS, para que seja agregada esse tipo de funcionalidade sem a necessidade de grandes alterações.

O DNS-SD é uma convenção de como as requisições de DNS podem ser adaptadas e utilizadas para que os nós da rede possam se adaptar e requisitar os serviços que estejam disponíveis localmente [Cheshire 2008b].

3.1.3 SSDP com UPnP

Os protocolos de UPnP focam-se muito mais na camada de aplicação, diferentemente do conjunto de mDns-SD (mDns juntamente com DNS-SD). No entanto, eles fornecem a mesma funcionalidade no que tange a descoberta de serviços, onde o SSDP e o mDns-SD têm soluções diferentes. O *draft* do SSDP foi abandonado desde 1999, pois o serviço se mostrou impraticável em redes densamente povoadas e, até mesmo com mais de dez nodos, o tráfego já aumentava significativamente [Goland 1999].

3.1.4 WS Discovery

O *WS Discovery* é um protocolo *multicast* de descoberta de recursos, onde o cliente busca pelo serviço requerido com o uso de *Web Services*. O envio é para todos, enquanto a resposta é direcionada ponto a ponto dos possíveis destinos. Ao se juntar a um grupo o nodo envia uma mensagem multicast anunciando sua presença e serviços que disponibiliza para os demais. Esse procedimento evita um *overhead* exagerado de troca de informações, uma vez que todos os nodos saberão dos novos serviços oferecidos sem precisar conferir quem dispõe do mesmo [Beatty 2005].

3.1.5 Rescue

A proposta do modelo Rescue [Juszczyk 2008] é baseada em redes *mesh* e ambientes totalmente *Ad Hoc*, sem nenhuma infra-estrutura instalada. O Rescue é voltado para uso de dispositivos móveis junto a equipes de gerenciamento de desastres naturais, como terremotos e enchentes. Nestes casos a infra-estrutura instalada pode não estar operacional ou não ser confiável devido a diversos fatores.

Nestes casos a comunicação entre os membros de uma equipe de resgate se torna um problema, que pode ser parcialmente resolvido com o *middleware* Rescue. No nível de transporte é utilizado *multicast* sobre OLSR (explicado em mais detalhes na subseção seguinte) para enviar anúncios através de pacotes UDP, caracterizando um novo tipo de suporte a serviços, possível devido ao OLSR-BMF (*Basic Multicast Forwarding Plugin for OLSRd*). Os clientes que recebem os pacotes iniciam uma comunicação *unicast* com a origem do chamado por pacotes TCP ou UDP, dependendo do tamanho das mesmas.

3.1.6 OLSR e redes *Mesh*

Redes do tipo *mesh* (em “malha”) possuem a vantagem de serem redes de baixo custo, fácil implantação e tolerantes a falhas. Para redes comuns, geralmente roteadores sem fio são

instalados no topo de edifícios e comunicam-se entre si em modo *Ad Hoc* através de múltiplos saltos para realizar o encaminhamento de mensagens. Existem projetos de *firmware* para roteadores poderem montar uma rede *mesh*, como o Freifunk [Freifunk 2009].

No que tange dispositivos móveis, existe o *daemon* OLSRd que implementa o protocolo OLSR (*Optimized Link State Routing*) [Clausen 2003] e pode ser executado em diferentes plataformas e com pouco uso de CPU (economizando bateria em dispositivos móveis).

A *middleware* Rescue faz uso de redes *mesh* e do protocolo OLSR para atingir a funcionalidade de comunicação desejada em ambientes hostis.

3.1.7 Konark

O Konark é uma das primeiras referências sobre um mecanismo de descoberta de serviços que tivesse como foco as complexidades envolvidas em ambientes *Ad Hoc* com dispositivos móveis. Ele tem como foco tanto serviços oferecidos por dispositivos físicos (como impressoras e faxes) como também serviços de software, como cenários de comércio eletrônico ou entretenimento [Helal 2003].

O seu protocolo é concebido no nível de conectividade IP, o que independe do meio de conexão a ser utilizado, como 802.11, *bluetooth* ou IrDA, por exemplo. Além disso, existe um micro servidor HTTP em cada nó da rede, e através dele é realizado o gerenciamento de serviços, incluindo a entrega e a busca dos mesmos. Com o objetivo de ser compatível com diversos dispositivos, utiliza HTTP e SOAP com mensagens em XML para efetuar os pedidos e enviar as respostas.

Dentre os pontos fracos, o Konark se baseia somente no protocolo *multicast* para conseguir enviar pedidos de publicação de serviços. Apesar de ser uma estratégia eficiente para pequenas redes, se torna inviável na Internet, pois a maior parte dos roteadores bloqueia multicast. Além disso, em redes de grande proporção o uso de XML sem um tratamento de compactação gera um *overhead* considerável [Grimm 2001].

3.1.8 GSD - *Group-based service Discovery*

O GSD [Chakraborty 2006] tem por objetivo desenvolver uma arquitetura de descoberta de serviços para ambientes ubíquos que seja eficiente, distribuída, escalável e adaptativa. Para isso, utiliza descrições semânticas explorando os recursos oferecidos por OWL para descrição dos recursos presentes em cada nó do ambiente *Ad Hoc*.

Outro ponto de destaque é o agrupamento do serviços pelo seu tipo, ou seja, a classe principal *Service* é subdividida em duas outras classes de funcionalidade: de hardware e de software. Estas, por sua vez, são novamente reclassificas pelo tipo de serviço até se chegar a funcionalidade específica que está se buscando.

3.1.9 Comparativo e observações

Dentre os protocolos de descoberta de serviços expostos na seção 3.1 pode-se perceber o uso de diferentes tecnologias utilizadas na implementação de cada um, indo desde protocolos no nível IP até *Web services*. Os protocolos mDns e DNS-SD são tidos atualmente como as alternativas mais sólidas e difundidas para a utilização de serviços dentro de redes locais. Já no caso de serviços na Internet, *Web services* são comuns.

No entanto, a busca por soluções alternativas para nichos que não se incluam nas duas abordagens acima se fazem necessárias e, como pode se observar no caso do Rescue, são de extrema valia para situações tanto genéricas como de risco, onde a tecnologia aplicada de forma eficaz pode fazer a diferença.

Outro caso que requisita soluções alternativas é a interconexão de redes locais, como é o foco desta dissertação, onde múltiplas redes *mesh* locais podem se interconectar e interagir através de serviços. Nas referências expostas acima, os protocolos ou são voltados para redes locais, ou são voltados para redes completamente *Ad Hoc*, mas a interligação entre as redes é algo incomum na prática [Edwards 2006].

Na seção a seguir são expostos *middlewares* ubíquos focados em redes *Ad Hoc* descentralizadas que, pelo menos em parte, podem fazer uso de serviços e utilizar protocolos similares aos expostos na seção 3.1.

3.2 Middlewares ubíquos

A busca por *middlewares* ubíquos atuais tem como objetivo verificar se os mesmos fazem uso de serviços e de que forma conseguem realizar a comunicação entre si, seja para o simples envio de uma mensagem ou a replicação de uma funcionalidade. A seguir são expostas algumas propostas atuais.

3.2.1 Meshmdl

O *middleware* Meshmdl foi especificado para possibilitar aplicações adaptáveis ao contexto. Segundo [Herrmann 2003], a sensibilização ao contexto e auto organização são os principais destaques deste *middleware*. O meio de comunicação principal da plataforma

formada pelo MeshMdl é o *Event Space*, baseado no conceito de *Tuple Spaces*, provendo uma comunicação assíncrona, anônima e baseada em conteúdo, desassociando os componentes da aplicação do fluxo dos dados, espaço ou tempo para sincronização. Essa característica de dissociação entre os componentes e nodos é de suma importância em um ambiente completamente distribuído [Herrmann 2007].

Os autores argumentam que *middlewares* clássicos baseados em CORBA e RMI fazem uso de comunicação síncrona e bloqueante, o que não satisfaz os requisitos apresentados em uma MANET (*Mobile Ad Hoc Network*), de mobilidade e dissociação. Outro exemplo é de mecanismos que utilizam o paradigma de comunicação *publish/subscribe*, onde existe a dissociação de espaço e fluxo contínuo, mas não ao tempo.

O Meshmdl apresenta ainda outra característica não comum a outros tipos de *middleware*: *Virtual spaces*. Isso permite ao Meshmdl ser estendido com a criação de novos módulos baseados na interface do *Virtual space.* As aplicações criadas podem interagir diretamente com as demais já existentes através do *Event Space* previamente mencionado.

Com relação à arquitetura do *middleware*, o mesmo é compatível com J2ME e com a especificação da SUN *Connected Device Configuration* [Sun 2005]. Cada dispositivo executa um *daemon* com quatro subsistemas que podem ser visualizados na Figura 1 e são detalhados abaixo.

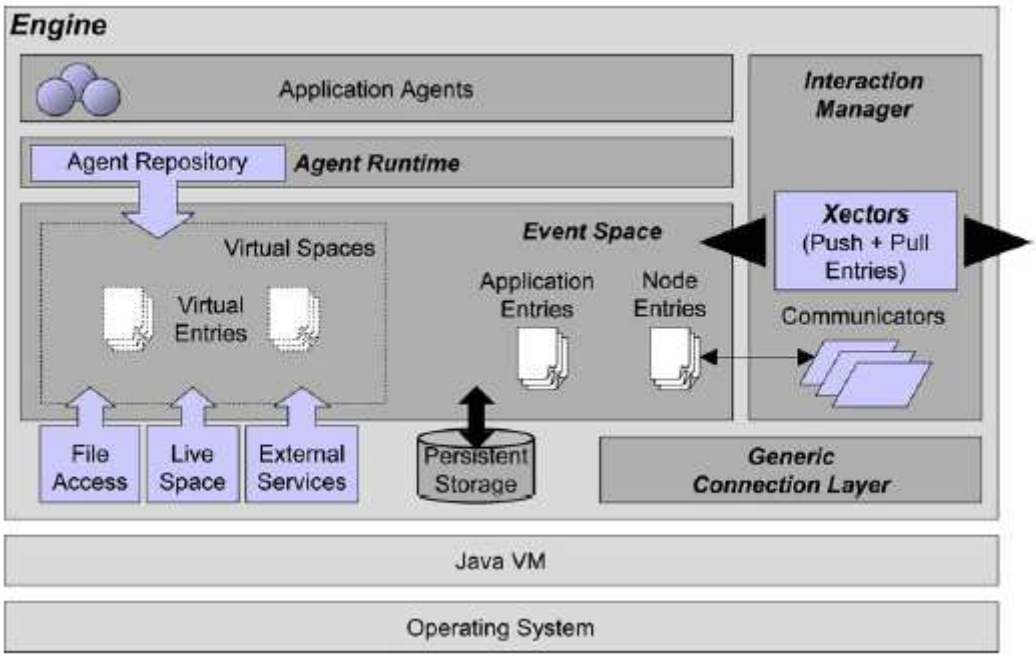

Figura 1: Arquitetura do MeshMdl [Herrmann 2003]

- O *Event Space*: é a parte principal do mecanismo e provê a funcionalidade básica do espaço de tuplas. Ele também mantém o mecanismo de *Virtual*

Spaces, que possibilita compartilhar informações de diversos subsistemas do *middleware* e de recursos externos;

- O *Agent Runtime*: é responsável pela execução dos agentes aplicativos. Ele provê uma interface para carregar os agentes e toma conta da carga das classes corretas. Ele também é composto do *Agent Repository* que provê informações sobre os pacotes de agentes e daqueles em execução. O *Agent Repository* existe através do *Virtual Space*;
- O *Interaction Manager*: é responsável por qualquer comunicação com nodos vizinhos, que estejam dentro do raio de transmissão. Para cada vizinho é mantido um objeto de comunicação que representa o canal de comunicação para o dispositivo remoto. Ainda é disponibilizado um mecanismo chamado *Xector Model* que possibilita que tuplas sejam enviadas ou recebidas dos nodos vizinhos;
- O *Generic Connection Layer (GCL)*: é uma camada de abstração da rede *Ad Hoc*. Ela disponibiliza uma API ao *Interaction Manager* para a descoberta de dispositivos e para se realizar conexões com nodos vizinhos. A GCL pode ser configurada para utilizar diversas APIs específicas para diferentes tecnologias, possibilitando a sua transição para outros tipos de rede sem afetar as camadas superiores.

O *Event Space* se mostra a parte mais importante deste *middleware*, sobre o qual podem ser implementadas funcionalidades similares a de serviços em outros ambientes. É através do *Event Space* que os aplicativos que executem o Meshmdl poderão trocar informações.

3.2.2 Expeerience e JMobiPeer

De acordo com [Bisignano 2003], o Expeerience tem como principal objetivo o compartilhamento de informações e comunicação dentro de um ambiente *Ad Hoc*. Ele utiliza JXTA como *framework* P2P, pois o mesmo provê interoperabilidade, independência de plataforma e ubiqüidade. O *middleware* possibilita o gerenciamento de conexões intermitentes, múltiplas interfaces, mecanismo eficiente de descoberta de recursos e mobilidade de código.

O principal objetivo dos autores foi o desenvolvimento de um *middleware* que fornecesse um suporte de alto nível e consistente para a programação de softwares explorando a tecnologia P2P sobre redes móveis *Ad Hoc* [Bisignano 2004]. O *middleware* desenvolvido

supre os requisitos estabelecidos pelos autores para ambientes que envolvam MANETs, ou seja:

- gerenciar um serviço de descoberta;
- atuar com múltiplas interfaces e conexões intermitentes;
- suportar mobilidade de código para aplicações que utilizem o paradigma de programação orientada a agentes móveis.

Com a evolução do JXTA surgiu o desenvolvimento do JXME (JXTA *for Micro Edition*), uma versão reduzida e otimizada do JXTA para operação em dispositivos móveis com J2ME. Algumas limitações se fizeram presentes no JXME, como a necessidade do nodo estar permanentemente conectado a um nodo na rede JXTA. Com isso uma rede isolada somente com nodos JXME não tem como funcionar corretamente, algo que não servia ao propósito dos autores. Uma das metas estabelecidas para o JMobiPeer foi estender a arquitetura do JXME, possibilitando seu uso integrado ao protocolo JXTA, sem a necessidade de se estar conectado a uma rede JXTA tradicional [Bisignano 2005].

No decorrer do desenvolvimento da plataforma JXME, inicialmente foi criada uma versão chamada de JXME *proxied*. Ela foi desenvolvida para dispositivos compatíveis com CLDC – MIDP 1.0, ou seja, dispositivos celulares com suporte a J2ME. Esta primeira versão não possui diversos recursos disponíveis na rede JXTA padrão, nem características como um XML *parser*, suporte a entrada de conexões HTTP, suporte estendido a segurança ou comunicação *multicast*. A falta destes itens força um nodo JXME a atuar somente como um *edge peer* com o ajuda de um JXTA *Relay*, ou seja, não podendo atuar de forma mais complexa dentro da rede *peer to peer*.

Após o desenvolvimento do modelo JXME *proxied*, foi iniciado o desenvolvimento do modelo *proxyless* JXME voltado para dispositivos compatíveis com CDC e CLDC – MIDP 2.0 (PDAs e celulares mais potentes). Nesta versão do JXME, ainda na sua fase inicial quando os autores iniciaram o desenvolvimento paralelo do JMobiPeer, foi fornecida uma versão mais limitada do *framework* P2P que não necessitava de JXTA *relays* para conseguir funcionar. Na Figura 2 é mostrada a arquitetura estendida proposta pelos autores, destacando as capacidades adicionais necessárias introduzidas no framework JXTA.

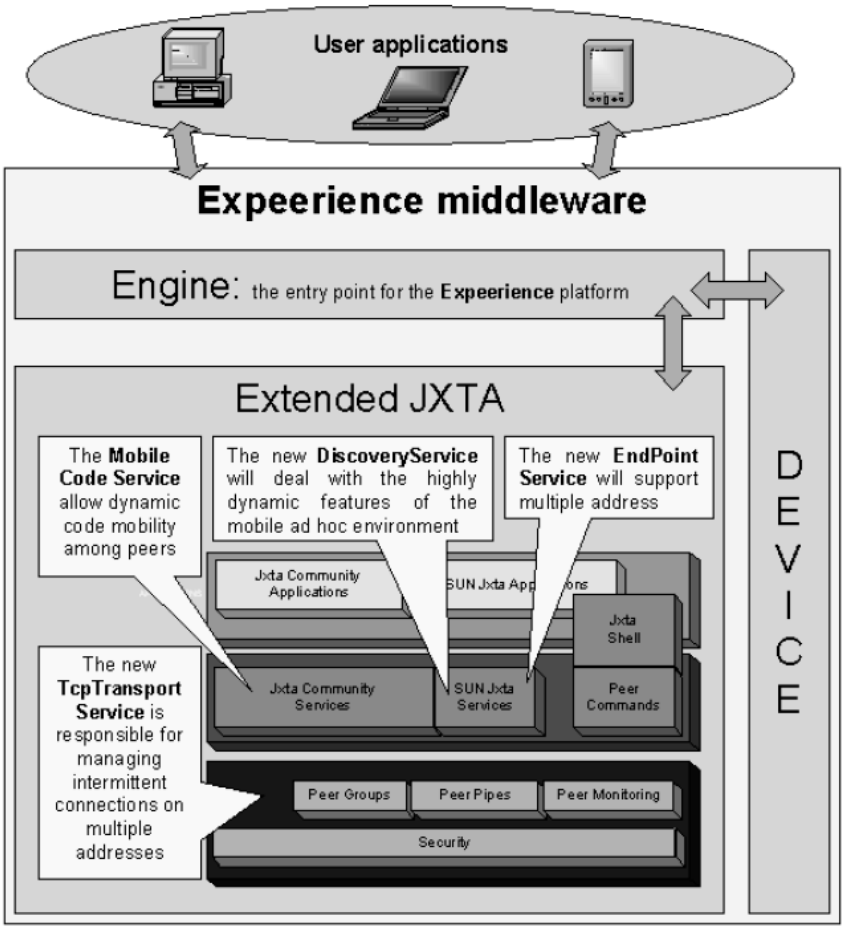

Figura 2: Arquitetura do Expeerience [Bisignano 2004]

Estas modificações e extensões do JMobiPeer vão ao encontro direto dos aperfeiçoamentos realizados no JXME *proxied* que foram necessários devido ao estado inicial em que se encontrava o desenvolvimento do último.

Aspectos como a descoberta de serviços e a mobilidade de código estão disponíveis no JMobiPeer. No entanto, cada nó precisa ter conectividade à Internet para conseguir fazer parte da comunidade JXTA e isso limita a participação de dispositivos móveis que não possuam tal recurso disponível. O JMobiPeer se baseia somente no nível de aplicação, deixando sem suporte os nós que sejam desprovidos de conectividade à rede.

3.2.3 Lime e Limone

Os *middlewares* LIME [Murphy 2001] e LIMONE [Fok 2004] focam no compartilhamento de informações e adotam uma abordagem baseada em um espaço de tuplas, estendendo o projeto Linda [Gelertner 1985]. O LIMONE é uma otimização implementada sobre o LIME, o qual estende o mesmo introduzindo gerenciamento de contexto, programação reativa, mobilidade de código e migração de agentes. No que diz respeito ao

gerenciamento de contexto, um mecanismo de descoberta eficiente é empregado, permitindo que os agentes descubram nodos vizinhos e seletivamente decidam a sua relevância dependendo dos requisitos da aplicação e da rede disponível.

A idéia principal por trás do modelo do LIME é manter o contexto transparente em se trocando de um ambiente fixo para outro móvel. O ITS (*Interface tuple space*) é o espaço de tuplas individual de cada dispositivo. Quando diversos agentes móveis podem se comunicar, seja diretamente ou não, eles formam um LIME *group*. Quando entra em um grupo, o agente pode mesclar o conteúdo de seu ITS com o de outros agentes. Quando diversos ITSs são compartilhados de forma transiente e colocados em algum *host*, esse TS (*tuple space*) é definido como um *host-level tuple space*.

Em seguida, surge o conceito do *Federated Tuple Space*, quando temos a interconexão entre os *host-level tuple spaces* e os ITSs. Quando for solicitada uma *query* no ITS de um agente, a mesma pode retornar uma tupla pertencente ao próprio ITS do agente (local), de um agente específico ou de um agente qualquer conectado através do *federated tuple space*. Esta estrutura pode ser visualizada na Figura 3, onde está representada um espaço de tuplas transiente e compartilhado sendo englobado pela mobilidade física e lógica dos agentes.

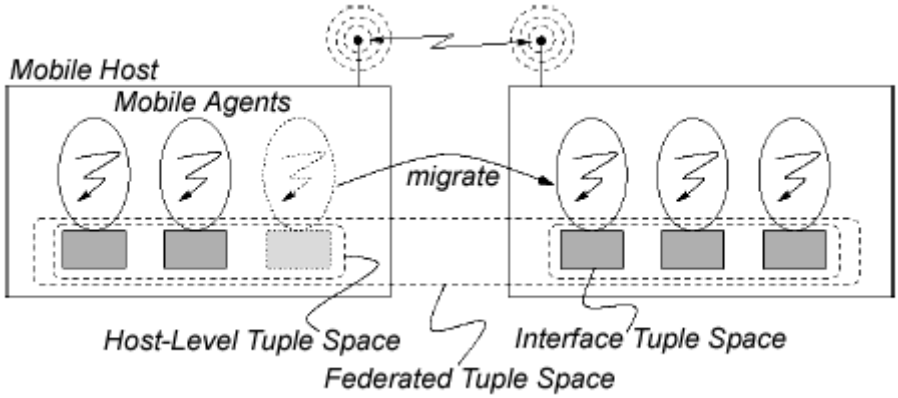

Figura 3: Espaço de tuplas transiente e compartilhado [Fok 2004]

3.2.4 Mobile Gaia

No Mobile Gaia os sistemas ubíquos *Ad Hoc* não dependem de arquiteturas pré estabelecidas, podendo agrupar diferentes dispositivos dentro de um mesmo ambiente. Os autores do *middleware* Mobile Gaia classificaram os clusters de dispositivos formados por estes sistemas em três categorias [Shankar 2005]:

- dispositivos pertencentes à mesma pessoa, servindo para gerar mais informação para o seu proprietário de forma customizada;

- dispositivos agrupados para uma determinada tarefa, como um jogo, por exemplo;
- dispositivos agrupados para compartilhamento de dados ou recursos.

O Mobile Gaia tem como objetivo o primeiro item, ou seja, tornar possível que os dispositivos do usuário formem um cluster pessoal, possibilitando o conceito de um *personal active space*.

Para tal fim, cada *active space* contém um dispositivo coordenador e diversos dispositivos clientes. O coordenador espera um *heartbeat* de cada um dos clientes, verifica o ambiente em volta na busca de novos dispositivos e processa toda a informação enviada pelos clientes, convergindo em uma central que irá disponibilizar os dados para o usuário da melhor maneira.

Um serviço de localização como o da Figura 4, por exemplo, seria formado por diversos dispositivos presentes no *cluster* pessoal, e não somente o GPS. Enquanto um aparelho de GPS e um celular forneceriam as informações necessárias para o coordenador, um PDA ou um Pager poderiam receber a informação necessária para exibi-la ao usuário do *cluster*.

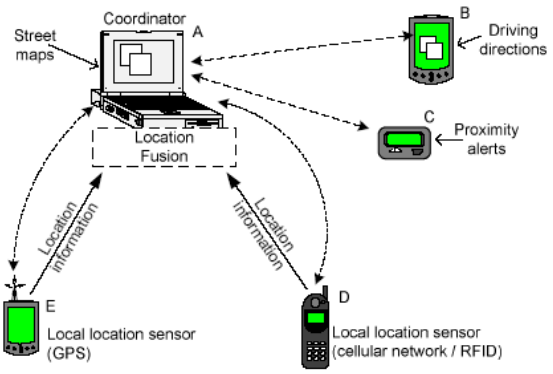

Figura 4: Serviço de localização [Shankar 2005]

No exemplo da Figura 4, o papel de coordenador ficou a cargo de um *notebook*, mas isso pode ser facilmente alterado, uma vez que a arquitetura do Mobile Gaia consiste de uma série de serviços básicos que gerenciam o cluster de dispositivos. Sua arquitetura é mostrada na Figura 5.

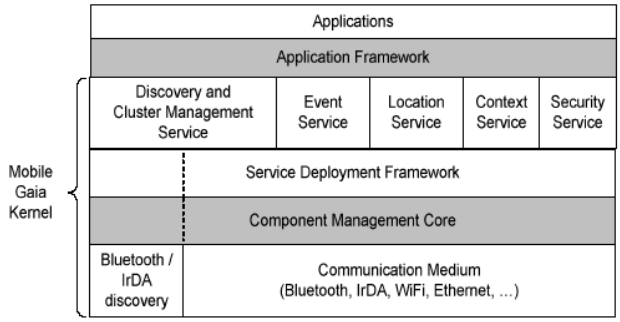

Figura 5: Arquitetura do Mobile Gaia [Shankar 2005]

Os serviços básicos que compõem o *kernel* do Mobile Gaia são:

- *Cluster management service*: descobre dispositivos que estejam nas redondezas, negocia com eles a entrada no *cluster* e a composição do próprio *cluster*, determinando quem fornece qual tipo de dado;
- *Event service*: habilita a comunicação de eventos entre os dispositivos do *cluster*;
- Serviços de localização e contexto (*location and context*): fornecem informações de contexto e localização para todos os dispositivos do cluster;
- Serviço de segurança: fornece serviços de autenticação e controle de acesso ao *cluster*;
- S*ervice deployment framework*: engloba todos os demais serviços dentro de um *framework* comum;
- *Component management core*: fornece as funções de baixo nível necessárias para criação e gerenciamento dos componentes superiores.

3.2.5 Impromptu

Outro ambiente ubíquo *Ad Hoc* encontrado foi o Impromptu [Beigl 2004]. Ambientes Impromptu são construídos para naturalmente permitir que objetos comuns sejam introduzidos de forma natural dentro do ambiente computacional ubíquo. Tais objetos podem ser dispositivos computacionais como PDAs, câmeras digitais, entre outros. O diferencial do Impromptu é a criação de ambientes onde aplicações computacionais possam se beneficiar de informações locais que sejam detectadas por alguns ou todos os dispositivos participantes e depois propagados entre os mesmos, tirando-se proveito da rede *Ad Hoc* local formada no ambiente.

Segundo os autores, realizando-se experimentos dentro de ambientes de escritório (portanto fechados), chegou-se a conclusão que os dispositivos deveriam preencher três condições para tornar um ambiente Impromptu possível:

- detectar informações do contexto interno e/ou externo;
- comunicar essa informação para todos os outros dispositivos dentro do ambiente usando a mesma linguagem de comunicação em alto nível;
- trabalhar sem configuração ou administração centralizada.

O *framework* do Impromptu é composto de:

- *Smart-its Particles*: uma tecnologia de hardware e software que visa prover poder de computação, rede e percepção de capacidades a objetos e ambientes comuns do dia a dia;
- Linguagem con-com (*context communication*): faz parte do *AwareCon communication network*. Permite a comunicação entre os dispositivos Impromptu;
- *Context perception*: uma forma padronizada para percepção do contexto, necessária para se perceber o mundo através de sensores e a comunicação de outros dispositivos Impromptu;
- Computadores *off-the-shelf*: laptops, PDAs e outros dispositivos que sejam integrados no ambiente fazendo uso de serviços adicionais.

A divulgação ampla de informações de contexto no ambiente Impromptu é uma característica interessante, pois todos os nós presentes na rede sem fio sabem exatamente quem está na sua proximidade. Isso pode, eventualmente, ser aplicado aos serviços para a informação dos demais usuários da rede.

3.2.6 Outros *middlewares Ad Hoc*

Após exemplificar *middlewares* de referência para esta pesquisa, cabe ainda citar mais três ambientes ubíquos que surgiram como destaques voltados para redes *Ad Hoc*, mas que tiveram menor prioridade:

- Steam [Méier 2002]: introduz o conceito de *middleware* baseado em serviços dentro de um ambiente móvel *Ad Hoc*. Ele suporta três tipos diferentes de filtros de eventos: assuntos, proximidade e conteúdo. O uso de filtros de conteúdo possibilita às entidades que divulgam informações a se expressarem por *queries* sofisticadas, fazendo com que a granularidade de busca dos eventos seja fina;

- XMIDDLE [Mascolo 2002]: pode ser considerado um *middleware* orientado a compartilhamento de dados, provendo mecanismos para se lidar com desconexões freqüentes e comuns em um ambiente ubíquo;
- Mate [Levis 2002]: faz parte de *middlewares* voltados para *Wireless Sensor Networks* (WSNs) que utiliza como abordagem máquinas virtuais como uma camada de abstração para implementar suas operações e para enfrentar os diferentes desafios das WSNs, as quais têm grandes similaridades com as MANETs.

3.2.7 Comparativo e observações

Analisando os *middlewares* descritos, o Expeerience tem um forte suporte aos requisitos de ubiqüidade expostos desde o capítulo 1, enquanto o Mobile Gaia apresenta uma arquitetura bem modularizada, mas é voltado para *clusters* pessoais e não para a disponibilização de serviços em grupos e diferentes contextos. Já o LIME/LIMONE é baseado na troca de informações por espaço de tuplas. Eles têm uma base sólida no projeto LINDA e continuam em pleno desenvolvimento, já tendo sofrido otimizações para executar até mesmo em clientes de redes de sensores sem fio, que geralmente tem pouca capacidade de processamento. Por fim, o MeshMdl apresentou uma estrutura dinâmica e bem construída, além de diversos itens com forte suporte à ubiqüidade.

Dentre os sistemas ubíquos expostos na seção 3.2 pode-se classificar os *middlewares* existentes no que diz respeito ao seu funcionamento e forma utilizada para construção:

- *Middleware* orientado a mensagens e baseado em eventos (MOM): são particularmente adequados para ambientes distribuídos sem uma central de controle, utilizando a troca de mensagens com mecanismos de *publish/subscribe*. A principal vantagem deste mecanismo é a facilidade para se estabelecer uma comunicação assíncrona;
- *Middleware* P2P: a principal vantagem está na fácil penetração por diferentes tipos de redes e no fato do nodo poder agir tanto como cliente ou servidor a qualquer momento, sem a necessidade de um nodo central ou agregador;
- *Middleware* baseado em componentes e com agentes móveis: a vantagem desta abordagem é a modularidade das aplicações, pois o cliente pode ter diversas funcionalidades diferentes, conforme a necessidade. Isso pode gerar, por exemplo, um menor consumo de energia;

- *Middlewares* baseados em espaços de tuplas: têm sua vantagem no fato de conseguir realizar comunicações assíncronas com sucesso, mesmo na presença de desconexões freqüentes;
- *Middlewares* baseados no compartilhamento de dados: também lida bem com desconexões freqüentes, pois tenta compartilhar os dados de maneira que sejam fornecidos mecanismos robustos que permitam maximizar a disponibilidade da informação com diversas réplicas. Isso permite que os nodos consigam se adaptar dinamicamente à desconexão de algum deles, mas é eficiente somente em redes pequenas devido ao alto *overhead* gerado;
- *Middleware* baseado em máquinas virtuais: é um sistema flexível que contém máquinas virtuais, interpretadores e agentes móveis. Permite que aplicações sejam escritas em módulos pequenos que possam ser injetados e distribuídos na rede para serem interpretados corretamente pelas máquinas virtuais. A desvantagem é o *overhead* introduzido pela necessidade de diversas instruções.

Tabela 1. Comparativo de *middlewares* ubíquos [Hadim 2006]

Legenda: F = Suporte Fraco, X = Pouco ou nenhum suporte, C = Suporte Completo		**Gerenciamento de energia**	**Abertura para aplicações**	**Escalabilidade**	**Mobilidade**	**Heterogeneidade**	**Facilidade de uso**
Nome do projeto	**Características principais**						
***Middleware* orientado a mensagens e baseado em eventos**							
STEAM	Baseado em eventos, Comunicação em grupos baseada na proximidade, Filtros, Mecanismo de publicação-subscrição.	F	X	F	F	F	C
EMMA	*Middleware* orientado a mensagens, JMS, comunicação ponto a ponto, publicação, roteamento epidêmico.	F	X	F	F	C	C
***Middleware* baseado em Peer to Peer**							
Expeerience	JXTA, *Framework Peer to Peer*, Mobilidade de código, Mecanismos de descoberta de recursos, gerenciamento de serviços.	F	C	C	C	F	C
***Middleware* baseado em componentes e agentes móveis**							
SELMA	Padrão voltado a áreas comerciais, Agentes móveis, Homzones, Descoberta de vizinhos, padrão IEEE 802.11, duplicação.	F	F	C	C	C	C
Mobile-Gaia	Componentes pequenos, *Active Spaces*, *Clusters*, Mecanismo de publicação-subscrição, "WYNIWYG", Coordenação.	F	C	C	C	C	C
***Middleware* baseado em espaço de tuplas**							
LIME	*Middleware* de compartilhamento de dados, Sistema de espaço de tuplas compartilhado, Extensão do Linda, Interface de tuplas (ITS).	F	X	F	F	F	C
MESHMdl	Espaço de tuplas orientado a objetos, Agentes móveis, J2ME, Assincronismo, Modelo Xector.	F	F	C	C	C	C
***Middleware* baseado no compartilhamento de informações**							
XMIDDLE	*Middleware* de compartilhamento de dados, Estrutura de dados robusta em forma de árvore, XML, Gerenciamento de desconexões.	C	X	F	F	C	C
***Middleware* baseado em máquinas virtuais**							
Mate	Utiliza o TinyOS, Sincronismo, Interpretador de *Byte code*, Cápsulas móveis.	C	C	C	F	F	X

Estas categorias de diferentes tipos de *middlewares* foram comparadas entre si na Tabela 1, focando em características como gerenciamento de energia, abertura dos *middlewares* para o desenvolvimento de aplicações, escalabilidade, mobilidade, heterogeneidade e facilidade de uso. Os *middlewares* Mobile Gaia e Meshmdl se destacaram de forma positiva, apresentando suporte completo na maior parte das categorias.

Com relação a essa dissertação, o estudo de *middlewares* existentes foi importante para identificar como eles fazem uso de técnicas de compartilhamento de dados e disponibilização de serviços. Através da categorização acima, percebe-se que o uso de técnicas como a modularização com agentes móveis torna possível um melhor aproveitamento dos componentes, além do seu uso sob demanda, algo importante em dispositivos com pouca memória e poder de processamento.

O uso de espaço de tuplas é outro ponto de interesse, pois causa o desacoplamento das informações do espaço e tempo, mas é algo similar aos *middlewares* baseados no compartilhamento de dados, pois para conseguir implementar um espaço de tuplas em um ambiente distribuído, uma replicação de dados intensa se faz necessária. Isso pode causar uma sobrecarga considerável na rede, algo inadequado para uma rede *Ad Hoc*.

Por fim, as técnicas empregadas em *middlewares* P2P e baseadas em eventos (*publish/subscribe*) se mostram uma alternativa adequada à proposta no que diz respeito à rede compartilhada entre todas as rede *mesh* locais, também aqui chamada de rede *overlay*, pois através da conectividade P2P é possível o acesso, no nível de aplicação, aos mais diferentes tipos de redes. Além disso, o mecanismo de *publish/subscribe* possibilita que sejam atingidos todos os nós presentes na rede *overlay*, e com o devido encaminhamento das mesmas, cheguem a todos os nós presentes em cada rede *mesh* local.

3.3 Considerações sobre o capítulo

Neste capítulo foram expostos diversos trabalhos relacionados com sistemas ubíquos *Ad Hoc* e protocolos de descoberta de serviços que fazem uso de diferentes formas de comunicação, possuem escalas distintas e focos por vezes diferentes. Ao final de cada seção foram discutidos aspectos pertinentes a essa dissertação.

Considerando-se os trabalhos pesquisados, pode-se destacar pontos importantes dos trabalhos que foram levados em conta para o modelo que será mostrado no capítulo 4:

- a proposta do *Rescue* se basear em redes *mesh* foi de fundamental importância para criar a idéia de suporte a redes ubíquas completamente descentralizadas e sem uma infra-estrutura centralizada instalada;

- o protocolo *mesh* e o *daemon* OLSRd foram a forma encontrada de viabilizar o modelo proposto, enquanto outras alternativas da seção 3.1 se mostraram limitadas a rede local, como o *mDns/DNS-SD*, ou inadequados para uso em ambientes *Ad Hoc*, como o *WS Discovery*;
- em relação as tecnologias apresentadas nos *middlewares* ubíquos da seção 3.2, nenhum apresentou a possibilidade de interatividade externa, para além da rede *Ad Hoc* local, sem ter nenhuma infra-estrutura instalada localmente, dependendo somente de seus nós vizinhos. O *Meshmdl* foi o que pareceu mais chegar próximo desta realidade, mas limitado a um escopo local. Diferentemente, o modelo do Ubiservices propõe interconexão de ponta a ponta;
- os *middlewares* fizeram uso de tecnologias como *mesh*, JXTA (via *peer to peer*), mas nenhum mostrou a integração de diferentes níveis de rede conectando dois pontos de rede local não estruturada através de uma rede *peer to peer*, por exemplo;
- aspectos como a migração e replicação de serviços foram citados, mas não ficou claro em que nível cada um dos *middlewares* suporta tais características, presentes no modelo do Ubiservices.

No próximo capítulo será descrito em detalhes o modelo proposto.

4 MODELO UBISERVICES

Neste capítulo é exposto o Ubiservices, um modelo para administração de serviços e conectividade entre redes *Ad Hoc* não estruturadas e descentralizadas. Este modelo atua como um componente modular, o qual pode ser utilizado sobre quaisquer ambientes ubíquos que suportem determinadas especificações de ubiqüidade, como mobilidade de código e suporte a contextos.

Na seção 4.1 são listados os requisitados de software e hardware que se fazem necessários no ambiente onde o Ubiservices pode atuar. Na seção 4.2 são descritas as funcionalidades a serem fornecidas pelo Ubiservices. A partir da seção seguinte é explicada a arquitetura do modelo do Ubiservices, começando com uma explanação geral na seção 4.3, seguindo com o Protocolo entre camadas na seção 4.4 e detalhando cada módulo de serviço do Ubiservices nas seções seguintes, incluindo:

- o serviço de busca e publicação (seção 4.5);
- o serviço de migração e replicação (seção 4.6);
- o serviço de armazenamento local (seção 4.7);
- o serviço de grupos (seção 4.8).

4.1 Requisitos de software e hardware do ambiente

Existem requisitos que devem ser atendidos pelo ambiente ubíquo que suportará e compartilhará funcionalidades com o Ubiservices, tanto em termos de características de hardware como de software. Para que o Ubiservices possa atuar de forma modular dentro de outro ambiente ubíquo, se fazem necessários:

- Conectividade: devido à mobilidade de dispositivos presentes em um ambiente ubíquo, os nós precisam ter disponível uma forma conectividade, seja sem fio ou cabeada para comunicação com os demais nós da rede;
- Sensibilidade ao contexto: o contexto no qual o dispositivo está envolvido pode afetar o comportamento da aplicação que será executada. Desta forma, o ambiente que suporte as especificações de ubiqüidade deve fornecer informações para cada nó/usuário nele contido;
- Conectividade externa: em cada rede *mesh* local deve existir um número mínimo de nós com conectividade externa a Internet. Por exemplo, dados 50 nós, o ideal seria que pelo menos 10% deles tivessem conectividade à Internet para possibilitar a conexão à rede *overlay*;

- Mobilidade de código: deve ser fornecida pelo ambiente que suporte as especificações de ubiqüidade, pois é um requisito que possibilita a administração da replicação e migração de serviços;
- Capacidade de memória não volátil presente em cada nó: este item é, por vezes, crítico em PDAs ou *smartphones*, que dependem de cartões externos para tal. A memória é importante para armazenamento da base de dados local, que será descrita nas seções seguintes;
- Inserção de informações por parte do usuário: itens a serem compartilhados, interesses e grupos a serem formados pelos usuários dentro de cada contexto são necessários para que o Ubiservices e o ambiente ubíquo possam fornecer informações sobre os demais usuários presentes no mesmo contexto.

4.2 Funcionalidades a serem fornecidas pelo Ubiservices

A proposta de administração de serviços em redes *Ad Hoc* pode ser vislumbrada em dois patamares diferentes. Primeiramente existe o nível de transporte, onde dispositivos próximos não tenham conexão direta a Internet ou mesmo à rede local sem fio. O segundo patamar que pode ser vislumbrado está no nível de aplicação, onde se espera conectar usuários de qualquer parte, e não somente locais. As funcionalidades que podem ser fornecidas pelo modelo do Ubiservices são:

- Formação de grupos: podem ser grupos de interesses **locais** e de interesses **distribuídos**. Grupos de interesses locais são formados por nodos fisicamente próximos, como em uma sala de aula ou qualquer ambiente que permita a detecção de outros dispositivos presentes na sua volta. Nestes casos são exemplos de serviços úteis o compartilhamento de banda, fornecimento de acesso a Internet ou o acesso a uma impressora fisicamente próxima. Já os grupos de interesses distribuídos são formados em função de um objetivo comum que pode ser tipicamente associado ao nível de aplicação. Como exemplo pode-se citar o compartilhamento de arquivos/informações, processamento distribuído, ou quaisquer serviços de escopo amplo. A volatilidade das informações de um grupo de interesse local é maior, pois a permanência de um usuário neste tipo de grupo em geral é temporária. Já as informações e interesses disponibilizados em um grupo de interesse distribuído são tratadas diferentemente, permanecendo disponíveis por tempo indeterminado e cabendo ao usuário que as disponibilizou a sua remoção, pois

independe do contexto físico ou local onde o usuário se encontra;

- Migração e replicação de serviços e informações: todos os serviços que podem ser disponibilizados precisam estar replicados em um número mínimo de nós para efeitos de manutenção no caso de queda do hospedeiro. A idéia se aplica não só a serviços, como também a informações e interesses disponibilizados pelos usuários. No caso da criação de grupos locais, não existe uma preocupação pela replicação de suas informações, mas no caso de grupos de interesse distribuído, os dados precisam ficar disponíveis aos demais usuários da rede permanentemente, o que é atingido com a replicação para outros nós, e a migração caso o nó original fique fora da rede;
- Busca e publicação de aplicações e serviços: qualquer nó pode efetuar pedidos de busca e publicação de serviços, tanto na rede local como na rede *overlay*;
- Armazenamento local de referências: cada nó precisa manter um conjunto mínimo dos serviços e informações de seu interesse para conseguir realizar uma busca ágil, localmente, antes de buscar informações na rede. O objetivo é não sobrecarregar a rede a cada pedido de busca de serviços de cada um dos nós;
- Descoberta dos grupos, informações e contextos que estejam na volta do usuário: cada usuário pode descobrir através da sua localização geográfica e dos nós que estejam mais próximos de si o contexto no qual ele está inserido (se existir), os grupos (de interesses locais ou interesses distribuídos) que estejam disponíveis e as informações/interesses de cada um destes grupos. Cada usuário pode inserir informações de contexto adicionais sobre o local onde está, disponibilizando-as para os demais usuários da rede.

4.3 Arquitetura do Ubiservices

Para contemplar as funcionalidades previstas na seção 4.2, a Figura 6 mostra a arquitetura em alto nível do Ubiservices, com os seus componentes.

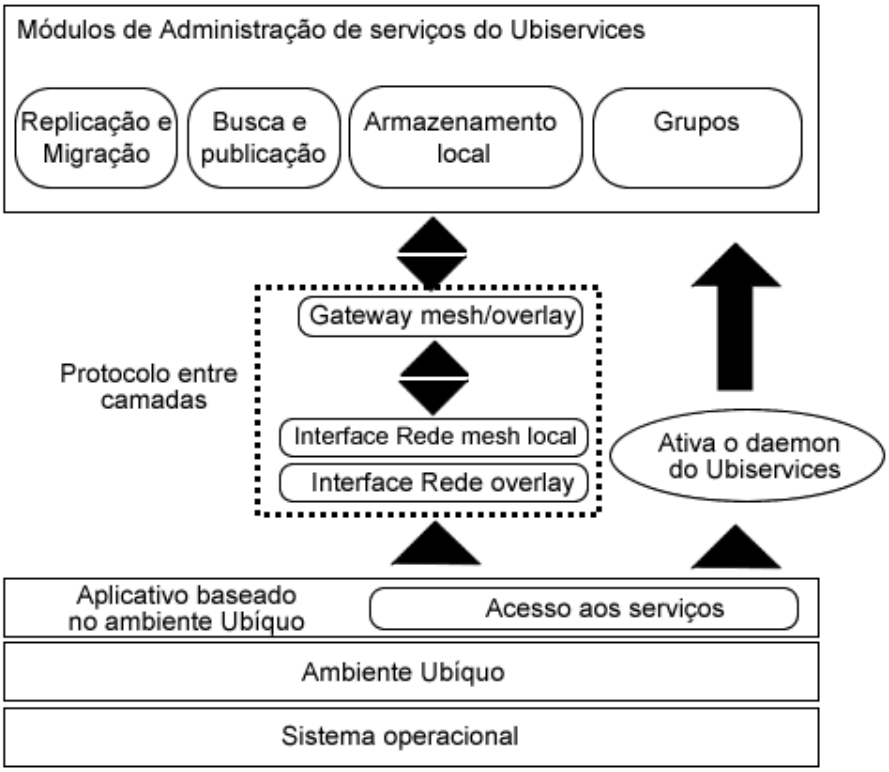

Figura 6: Modularização do Ubiservices

As principais partes da arquitetura, presentes na Figura 6, são:

- Protocolo entre camadas: permite o fornecimento de serviços e o compartilhamento de informações de uma rede *mesh* local para uma rede *overlay* no nível de aplicação, sendo de fundamental importância para todo o resto do Ubiservices;
- Serviço de busca e publicação: no Ubiservices não existe um diretório centralizado de informações, e a integração entre a rede *mesh* e a rede *overlay* transcende as camadas isoladas de uma aplicação de rede comum, fazendo com que os grupos de interesse (de interesses locais e distribuídos) tenham um papel importante de filtragem na busca e publicação de informações;
- Serviço de migração e replicação: no momento da publicação de uma informação ou serviço em um grupo de interesse distribuído, se faz necessária a replicação do mesmo em outros dois nós para efeitos de backup e tolerância a falhas do nó de origem;
- Serviço de armazenamento local: quando um nó encontra informações de seu interesse, seja nos grupos locais ou distribuídos, armazena as referências em um índice local para localização posterior.

- Serviço de grupos: é responsável por armazenar as informações de todos os grupos do usuário, criados tanto no escopo de interesses locais como de interesses distribuídos.

A Figura 6 mostra que um aplicativo no nível do usuário pode acessar o ambiente ubíquo solicitando acesso aos serviços do Ubiservices através do *gateway* formado pelo protocolo entre camadas. Ao tentar o acesso ou compartilhamento de algum serviço, um *daemon* é ativado e o nó passa a fazer parte da rede *mesh* local ou da rede *overlay*, caso disponha de conexão à Internet e não se encontre numa rede *mesh* local. A partir deste ponto as informações chegam até as pontas da rede *overlay*, onde os *gateways* tratam de replicar os pacotes para as redes *mesh* locais ou vice-versa.

O *daemon* que executa em cada equipamento possui os serviços de replicação, migração, busca, publicação, armazenamento local e grupos incorporados. Cada nó presente na rede pode fazer parte de múltiplos grupos. Os grupos de escopo local têm seus serviços divulgados somente dentro da rede *mesh*, pois o próprio serviço não tem propósito fora dela. Encaixa-se neste caso o exemplo de compartilhamento de acesso à rede para um nó que esteja isolado. Para ele entrar numa rede *mesh* local e, a partir dela, conseguir acesso à Internet, um serviço de compartilhamento de conectividade faz total sentido. No entanto, para um nó que possua uma conexão 3G no seu PDA e não esteja participando de nenhuma rede *mesh* local, utilizar um serviço de compartilhamento de banda não se aplica.

Por esse motivo, existem nós que estarão conectados via a interface de rede *mesh* da Figura 6, enquanto outros o farão diretamente pela rede *overlay*, como é o caso do PDA conectado via 3G citado previamente. Além disso, na fronteira entre cada rede *mesh* local e a rede *overlay*, os *gateways* fazem o porte dos pacotes e dos anúncios entre as camadas de rede, as quais podem ser consideradas como de transporte na rede *mesh* e de aplicação na rede *overlay*. Nas seções seguintes cada uma das partes do modelo é melhor detalhada.

4.4 Protocolo entre camadas

Para tornar possível o fornecimento de serviços tanto dentro de uma rede *mesh* local formada no nível de transporte como para uma rede *overlay* no nível de aplicação, o protocolo de roteamento é de fundamental importância. Dentre os trabalhos pesquisados, as soluções de administração de serviços são focadas ou em redes locais ou em redes de larga escala com conectividade pré existente. O mDns, por exemplo, foi arquitetado para um domínio local, assim como outros protocolos mostrados na seção 3.1.

A abordagem aqui exposta precisa de mais do que o mDns pode oferecer, requisitando

de um protocolo que seja operacional no escopo local em pequenos dispositivos, mas ao mesmo tempo que seja robusto para conseguir disponibilizar e buscar serviços dentro de uma rede ampla, como uma rede *peer to peer* na Internet, tudo isso sem centralização alguma. No caso desta proposta, o papel dos *gateways* entre a rede *mesh* e a rede do nível de aplicação são peças chave para o funcionamento de uma arquitetura similar a exposta na Figura 7.

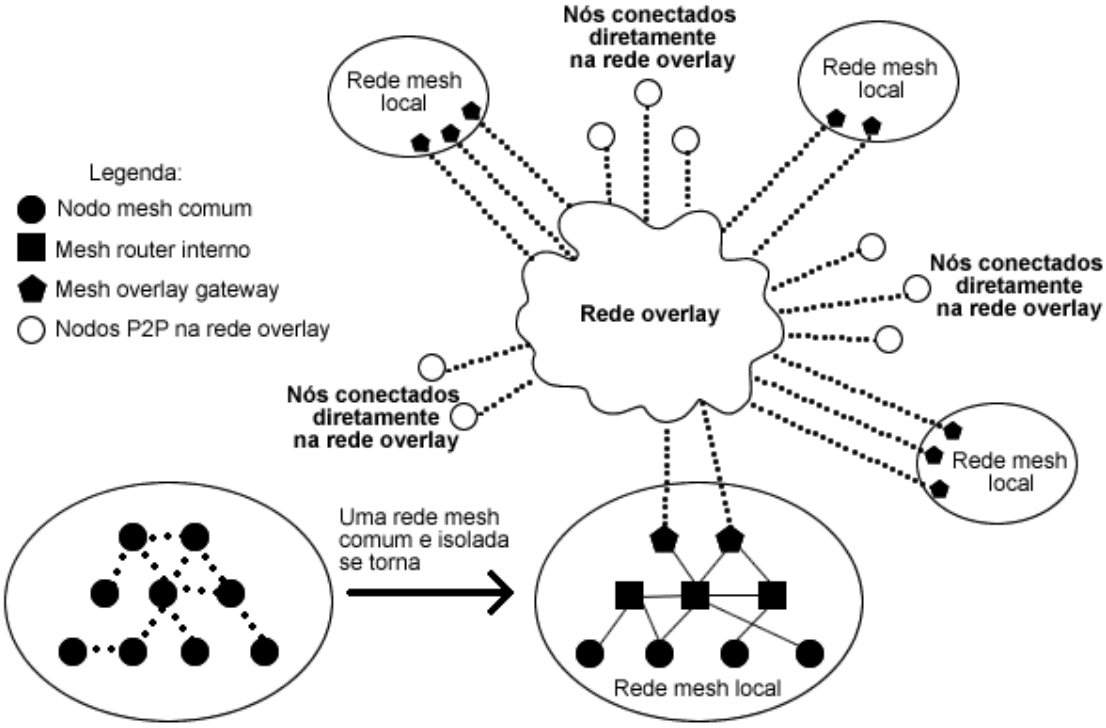

Figura 7. Estrutura de conectividade do Ubiservices

Usualmente as redes *mesh* podem dispor de poucos *gateways* para conseguir fazer o trabalho de roteamento para uma rede externa. Mas na medida em que eles diminuam em número, sobretudo em um ambiente *Ad Hoc*, a possibilidade de perda de conectividade da rede *mesh* torna-se maior. Mais importante do que o número ou percentual de nós conectados é a garantia de conectividade à rede *overlay* nos nós na rede *mesh*. Com tal disponibilidade de *gateways*, se tornará possível o adequado fornecimento de serviços.

A forma de divulgação do serviço é no estilo de *publish/request* no caso da rede *overlay*, e em forma de *multicast* na rede *mesh*. É o papel dos *gateways* tornarem um *multicast* uma publicação na rede P2P e vice-versa.

Com relação ao nível local através de uma rede *mesh* é possível conectar tais usuários e inseri-los dentro de um grupo. Já para o nível de aplicação, uma rede *overlay* é necessária (por exemplo, uma rede *peer to peer*). A junção dos dois patamares gera a possibilidade de uma rede de larga escala. Ou seja, caso nenhum usuário tenha acesso à Internet, a rede se

limita ao escopo local, já caso o acesso externo exista, pode-se conectar a qualquer usuário através da rede *overlay*.

O protocolo necessário para tal operação encapsula nos pacotes uma identificação única (UUID), com informações sobre o tipo de operação, destino, contexto e conteúdo. A Figura 8 mostra o formato do pacote.

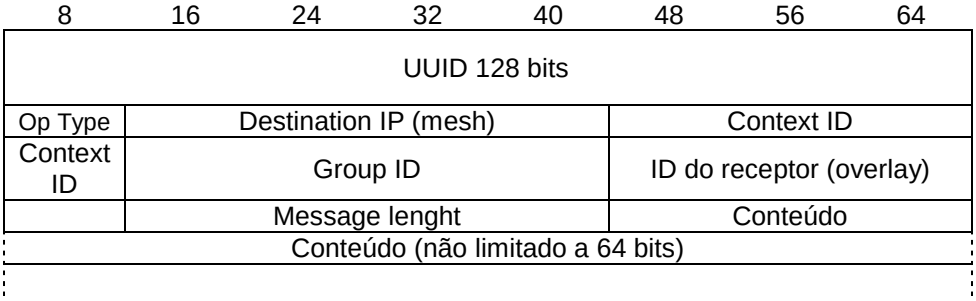

Figura 8. Pacote do protocolo de comunicação entre camadas

O campo de 128 bits do UUID presente na Figura 8 é baseado no padrão da RFC 4122 [Leach 2005] mostrado na Figura 9, o qual e tem como referência o campo *node*, em função do qual os demais campos são gerados, segundo a RFC 4122.

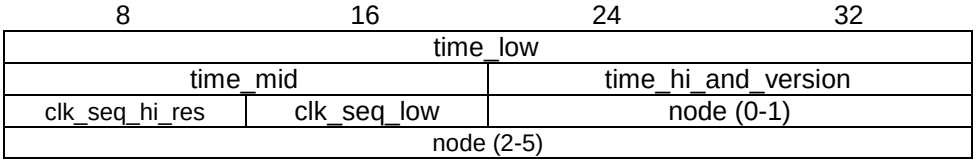

Figura 9. *Layout* do campo UUID [Leach 2005]

Interações utilizando o protocolo entre camadas são necessárias em todas as aplicações do Ubiservices, descritas nas seções seguintes, pois permitem a interação entre a rede *overlay* e a rede *mesh*. O campo *"Op Type"* de 8 bits define qual tipo de operação está sendo realizada, seja um *ack*, *nack*, *hello*, busca, publicação, migração, replicação, busca de grupo.

O campo "*destination IP*" na Figura 8 define o destino do pacote dentro da rede *mesh*, caso esteja preenchido. Caso negativo, o *ID* do receptor identifica o destino dentro da rede *overlay*. O pacote contém também informações sobre o contexto de origem, identificado numericamente, assim como o ID do grupo de origem. Em geral, o grupo de origem terá a identificação que representará o nome da pessoa, pois a maioria dos grupos será pessoal.

4.5 Serviço de busca e publicação

Em geral as abordagens de descoberta de serviços existentes podem ser classificadas de duas formas: baseadas no uso de diretórios centralizados ou não. Neste último caso o *broadcasting* ou *multicasting* entram como meio de buscar serviços [Krebs 2008]. No

Ubiservices não existe um diretório centralizado de informações, e a integração entre a rede *mesh* e a rede *overlay* transcende as camadas isoladas de uma aplicação de rede comum, sendo uma aplicação dita *cross-layer*, ou seja, que consegue interligar os nós da rede através de diferentes camadas.

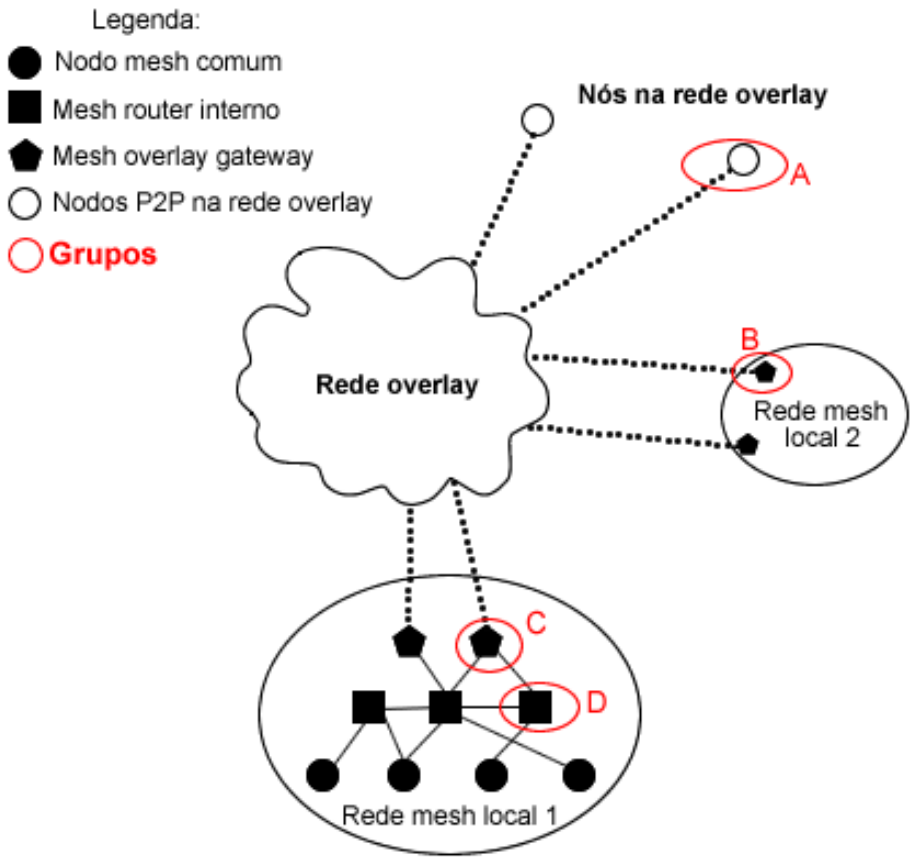

Figura 10. Nós conectados a grupos locais e distribuídos

Aplica-se neste contexto a idéia de diferentes tipos de grupos de interesses: **locais** e **distribuídos**. Grupos de interesses locais são formados por nodos fisicamente próximos, como em uma sala de aula ou qualquer ambiente que permita a detecção de dispositivos via rede sem fio de forma *Ad Hoc*. Na Figura 10 o grupo D da "Rede *mesh* local 1" poderia ser um grupo de interesse local, visto que não está diretamente conectado a rede *overlay*, tendo interesses somente locais. Nestes casos são exemplos de serviços o compartilhamento de banda, fornecimento de acesso a Internet e o acesso a uma impressora fisicamente próxima.

Já os grupos de interesse distribuído são formados por um objetivo comum que pode ser tipicamente associado ao nível de aplicação. Na Figura 10 o nó A com certeza faz parte de grupos de interesse distribuído, uma vez que está conectado somente a rede *overlay*, não tendo sentido o interesse em grupos locais. Já os nós B e C têm grande probabilidade de estarem participando de grupos tanto locais como distribuídos, uma vez que tem

conectividade própria a rede *overlay*, mas também participam da rede *mesh*. Como exemplo de interesse distribuído pode-se citar o compartilhamento de arquivos/informações, processamento distribuído, ou quaisquer serviços de escopo amplo.

Aplica-se aos grupos de interesse distribuído o conceito de que as informações locais são mais dinâmicas, ficando disponíveis enquanto o usuário estiver dentro de um contexto físico onde tem contato com os demais. Já os grupos de interesse distribuído fazem uso de replicação e migração de suas informações para outros nós, deixando-as permanentemente disponíveis aos demais usuários da rede *overlay*, e conseqüentemente de todos os usuários que tiverem interesse no seu grupo de interesse distribuído.

Cada nó pode estar associado a diversos grupos, seja para usufruir ou fornecer informações e serviços. Os nós envolvidos nesta rede *cross-layer* podem tanto ser parte de uma rede *mesh* e estarem totalmente desprovidos de uma conexão própria à rede *overlay*, como também podem estar conectados diretamente à rede *overlay*.

No caso de um nó conectado diretamente a rede *peer to peer*, pode-se tomar como exemplo o *notebook* precisando de acesso a uma impressora. Caso o mesmo não esteja ligado a rede *mesh* local, pode procurar pela mesma em grupos de interesse locais. Com a exposição destas diferentes possibilidades de conectividade entra em cena uma questão chave do Ubiservices: os *gateways* que fazem a intermediação entre a rede *overlay* e a rede *mesh*. Eles são de fundamental importância e devem existir em uma quantidade razoável na rede *mesh*, em conformidade com o número de nós presentes na mesma.

4.6 Serviço de migração e replicação

O serviço de migração e replicação atua sobre informações, interesses ou serviços publicados pelos usuários. Para isso se faz uso da mobilidade de código, necessária na migração e replicação de serviços, e os protocolos de busca, publicação e requisição de serviços. Na medida em que um nó entre com um serviço em execução ou publique uma informação em um grupo de interesse distribuído, imediatamente se faz necessária a replicação da informação, interesse ou serviço em outros nós para efeitos de *backup* e tolerância a falhas do nó de origem. Caso a publicação seja restrita a rede *mesh* local (em um grupo de interesse local), quando o nó for desligado, a informação se perde.

Dependendo do tipo de grupo ao qual o serviço é dirigido ou a informação é publicada (de interesse local ou de interesse distribuído), ela pode ficar restrita ao escopo da rede *mesh* local ou se tornar pública na rede *overlay*. As técnicas necessárias para a manutenção dos mesmos envolvem diretamente a replicação e a migração. Assim que um nó disponibiliza uma

informação ou serviço, este precisa ser imediatamente replicado em outro nodo se estiver dentro de um grupo de interesse distribuído. Entra em ação o protocolo entre camadas explicado na seção 4.4, onde a partir do nó do usuário partem mensagens do tipo “replicação”, informando o grupo de origem e o contexto(caso exista). A partir deste momento os nós que receberem a informação começam a buscar periodicamente pelo nó de origem, a fim de saber de seu status.

Com isso, caso o nó de origem da informação/serviço sofra alguma perda de sinal ou desconexão, os nós para os quais a informação foi replicada deixam de receber o *ack* do nó de origem nas suas verificações e é enviada uma mensagem de *broadcast* informando a replicação do serviço ou informação neste nó de backup. Desta forma evita-se que um serviço ou informação deixe de ser fornecida dentro do ambiente *Ad Hoc* ou da rede *overlay*.

Ainda no que tange a heurística de funcionamento da replicação, todo nó presente na rede *mesh* ao publicar uma informação ou serviço em um grupo de interesses distribuídos deve fazer a replicação destes dados para dois nós na rede *overlay*. O critério de escolha destes nós ocorre em função do tempo de resposta dos nós conhecidos armazenados pelo serviço da subseção 4.7. Os dois nós conhecidos que tiverem os menores tempos de resposta na rede *overlay* e aceitarem armazenar a replicação, são os eleitos para armazená-la. Caso o nó de origem não tenha seus pedidos de replicação aceitos, ou nem mesmo tenha conhecimento de algum nó na rede *overlay*, ele envia uma busca na rede *mesh* pedindo por endereços de nós conhecidos na rede *overlay*. Caso não receba nenhuma resposta, ou ele está fora de uma rede *mesh*, ou esta rede está isolada, não sendo possível publicar a informação desejada na rede *overlay*.

Com relação a migração, cabe destacar que ela é o objetivo principal do serviço desta subseção, uma vez que a replicação é o meio para conseguir multiplicar a informação ou serviço desejado afim de que, caso o nó de origem se desconecte, a migração efetivamente aconteça e outro nó da rede possa manter a informação ou serviço disponível. Depois que o nó de origem se desconecta da rede, é disparado um protocolo de eleição entre os dois nós de *backup* da informação, de modo que aquele que conseguir disparar por primeiro a mensagem de que está assumindo o controle do serviço ou a responsabilidade por manter a informação, é o eleito. Para tal comparação ambos recebem a mensagem do outro nó de *backup* e comparam o *timestamp* da mensagem recebida com a mensagem que foi enviada. O menor *timestamp* vence e publica um broadcast na rede informando que o serviço ou informação foi efetivamente migrado.

4.7 Serviço de armazenamento local

Na medida em que cada nó encontra serviços de seu interesse, armazena as suas referências num espaço próprio, contemplando a identificação da origem, seja o *IP* da rede *mesh* ou o *ID* da rede *overlay*. Com isso, mais tarde este nó consegue encontrar facilmente o serviço ou informação anteriormente utilizado. Isso é armazenado no formato de uma lista encadeada armazenada em memória volátil, sendo perdida quando o nó se desconecta da rede.

Nenhuma mensagem é enviada para operacionalizar este serviço, pois a informação é armazenada localmente. No entanto, quando um nó necessita buscar uma informação ou serviço, consulta primeiramente os dados presentes na lista do serviço de armazenamento local, e depois seguindo o processo normal de busca na rede.

Caso o nó de armazenamento do serviço tenha sido modificado e nenhuma das referências existentes para o mesmo serviço seja válida, se procede com a busca normal, que envolve tráfego de rede. A idéia de armazenamento local surge como uma tentativa de amenizar o *overhead* de tráfego que possa ser gerado em uma rede com um número considerável de nós. No entanto, quanto mais instáveis forem os nós que compõem a rede, mais tráfego será gerado, pois as referências estarão constantemente sendo modificadas.

4.8 Serviço de grupos

O serviço de grupos tem como objetivo disponibilizar as informações, interesses e serviços do usuário em um determinado grupo. Cada grupo pode ser de interesses somente locais ou de interesses distribuídos. O que caracteriza um grupo de interesse local é o seu escopo que gira em torno somente dos usuários presentes no mesmo local. Já um grupo de interesses distribuídos tem suas informações espalhadas na rede *overlay* e disponíveis de forma permanente. Quando o usuário publicar uma informação, caso ela não exista e ele não especifique, ele estará publicando-a dentro de um grupo de interesses locais com o seu nome.

Cada usuário pode estar conectado a vários grupos, mas estará localizado dentro de no máximo um contexto físico. Estes conceitos podem se confundir, mas a idéia é que os grupos sejam muito mais flexíveis que os contextos. Os grupos contêm informações, interesses e serviços, enquanto os contextos incluem somente informações de localização. Um grupo é formado pelos usuários, já um contexto é disponibilizado por um nó que seja permanente ao local, como uma infra-estrutura existente. Este nó que informa um contexto é mais restrito que um grupo, fornecendo somente informações de nome do local e localização geográfica.

Caso o usuário esteja dentro de uma universidade, por exemplo, *N* grupos diferentes

podem estar disponíveis simultaneamente em função de usuários espalhados na forma de redes *mesh* locais. Caso tenha interesse nos serviços oferecidos por algum destes grupos, o usuário pode localizar informações dos grupos, tendo como referência aqueles que se encontram em contextos geograficamente mais próximos de si. Desta forma, não é necessário manter uma base de dados centralizada nem permanente sobre os grupos dos nós, uma vez que esta busca pode ser feita sob demanda, conforme o interesse do usuário recém chegado.

Deixando que cada usuário insira informações pertinentes a respeito de seus próprios grupos, é possível obter diferentes perspectivas do local quando uma pesquisa for feita ao redor do usuário. Estudos a respeito deste tipo de inserção de dados de forma livre já foram efetuados no projeto *Place Lab* [Hightower 2006], onde foi verificado que ontologias e restrições para a catalogação dos nomes de locais onde os usuários se encontram não são um pré requisito para se conseguir efetuar uma boa pesquisa, com informações úteis para aqueles usuários que estejam realizando a busca por informações.

4.9 Considerações sobre o capítulo

Neste capítulo foi detalhado o Ubiservices, primeiramente de forma geral na seção 4.3, seguindo com o protocolo entre camadas na seção 4.4 e os serviços da seção 4.5 em diante. O protocolo entre camadas da subseção 4.4 permite uma interconexão entre redes *mesh* e uma rede *overlay*, algo não encontrado nos trabalhos relacionados do capítulo 2. Os serviços complementam a funcionalidade do Ubiservices, o qual, além de permitir a interconexão de diferentes redes *mesh*, proporciona uma série de funcionalidades que existem nos trabalhos relacionados, mas não para nós que originalmente podem estar isolados. O Ubiservices permite a inclusão destes nós isolados a uma rede *mesh* local (*Ad Hoc*), e através da conexão desta rede a rede *overlay*, a comunicação deles com os demais nós presentes na rede, sem a necessidade de infra-estrutura pré instalada nem centralização.

No capítulo 5 serão explicados aspectos relativos a implementação do Ubiservices.

5 ASPECTOS DE IMPLEMENTAÇÃO

Este capítulo descreve as etapas da implementação do protótipo e as tecnologias utilizadas neste processo. Tais tecnologias são detalhadas na seção 5.1, tendo na seção 5.1.1 uma descrição sobre o OLSRd, na seção 5.1.2 detalhes sobre a plataforma Gnunet e na 5.1.3 detalhes a respeito do protótipo do UOP, o ambiente ubíquo com o qual o Ubiservices está sendo integrado. O UOP (*Ubiquitous Oriented Programming*) [Garzão 2010], desenvolvido por Alex Garzão como dissertação de mestrado em 2009, surge como um modelo de programação orientado ao desenvolvimento de aplicativos ubíquos introduzindo um conjunto de diretrizes que visam facilitar o desenvolvimento de software para este fim, pois o mesmo tem uma semântica simples, a qual estimula a exploração automática da distribuição (distribuição implícita), concorrência (processamento distribuído e paralelo), adaptação ao contexto [Satyanarayanan 2001], serviços [Austin 2002], localização [Syvanen 2005] e contextos [Dey 2001].

A implementação foi feita em duas etapas: na seção 5.2 é detalhado o trabalho de implementação dividido entre a seção 5.2.1, que trata sobre a integração entre o *daemon* OLSRd/Gnunet, e a seção 5.2.2, que trata sobre a integração entre o OLSRd e o ambiente ubíquo UOP. Mais etapas teriam sido necessárias para a geração dos serviços do Ubiservices citados no modelo das seções 4.4 a 4.8, mas a implementação foi realizada parcialmente, gerando somente resultados qualitativos, ou seja, agregando conhecimento a respeito das tecnologias envolvidas e gerando uma série questões a serem avaliadas em trabalhos futuros juntamente com os cenários de avaliação que serão apresentados no capítulo 5.

Na primeira etapa da implementação descrita na seção 5.2.1 é detalhada a integração de mensagens entre uma rede *mesh* estruturada pelo OLSR *daemon* e a rede *peer to peer* do Gnunet. Dentro do Gnunet existem extensões que podem ser implementadas como *applications*. No entanto, não se chegou ao ponto de geração de uma *application* funcional. Como exemplos existem aplicações de *chat*, *advertising*, *dht*, *ping*, entre outras.

Já no OLSR *daemon*, através de *libs* pode-se adicionar conteúdo ao núcleo do *daemon*, o qual sozinho implementa somente o básico de uma rede *mesh*, em termos de funcionalidade. Como exemplos existem as *libs txtinfo*, responsável por logar informações para a saída padrão, *bmf*, responsável por enviar mensagens de forma multicast na rede em malha, entre outras.

Na segunda etapa da implementação, descrita na seção 5.2.2, é detalhada a forma como o UOP pode se integrar à *lib* do Ubiservices no OLSRd.

5.1 Tecnologias utilizadas

Na seção 5.1.1 é detalhado o *daemon* OLSRd, na seção 5.1.2 a plataforma Gnunet é explorada e na 5.1.3 detalhes a respeito do protótipo do UOP são colocados.

5.1.1 OLSRd

O *daemon* OLSRd foi escolhido frente a outras implementações como o Inria [Clausen 2003] ou NRL [Adamson 2008], pois está em atualização constante e consegue lidar com um número superior a 500 nós, se mostrando robusto para qualquer cenário. Detalhando a estrutura do OLSR que foi utilizada junto ao protótipo, a *lib* do Ubiservices concentrou em três serviços do *daemon* a sua maior interação: o *socket parser*, o *packet parser* e o *event scheduler*.

Dentro do OLSRd o *socket parser* é a entidade responsável por escutar por dados de determinados *sockets*, os quais juntamente com as suas correspondentes funções de *parsing* são registrados em tempo de execução. A Figura 11 mostra que o *socket parser* é dinâmico para possibilitar receber quaisquer pacotes e encaminhá-los para as respectivas funções no *packet parser*.

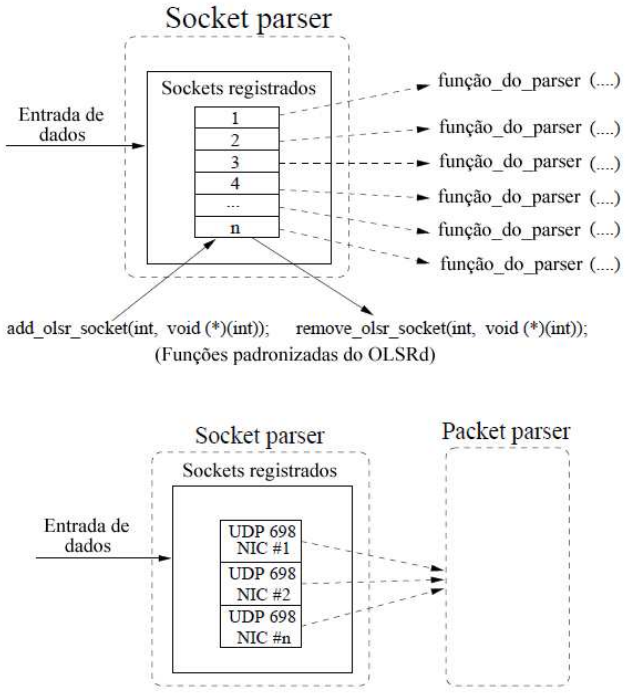

Figura 11. O *socket parser* encaminhando o tráfego ao packet parser

O *packet parser*, por sua vez, é executado assim que o *socket parser* recebe uma pacote destinado a ser tratado por determinada função. Seu comportamento é mostrado na

Figura 12, onde ele verifica se o tamanho reportado no cabeçalho da mensagem OLSR confere com os dados reais presentes no pacote, analisando então o pacote e passando a mensagem para a função de *parse* previamente registrada em tempo de execução, ao carregar-se a *lib*. Caso o pacote recebido não tenha o mesmo tamanho do anunciado no *header*, ele é silenciosamente descartado.

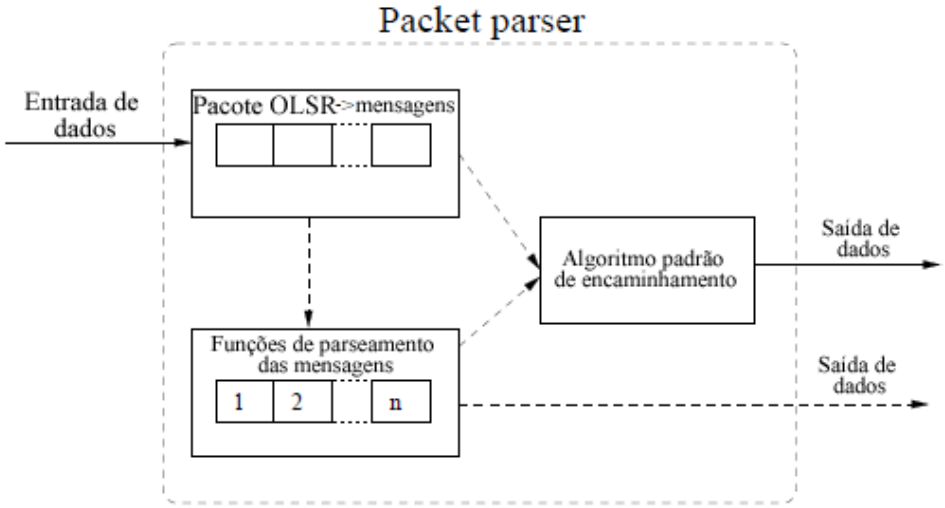

Figura 12. O *packet parser*

As informações necessárias para armazenamento de conteúdo em tempo de execução são mantidas de forma dinâmica, podendo ser alteradas, por exemplo, em função de uma nova topologia da rede ou de algum *timeout*. Toda informação compartilhada é mantida em tabelas dinâmicas, as quais em geral são verificadas e modificadas com freqüência. O formato destas tabelas é na forma de listas duplamente encadeadas e árvores normais ou binárias. Existem ainda funções de *hash* disponíveis na própria API do OLSRd para localização ágil dos elementos dentro das listas encadeadas.

Outro mecanismo importante presente no OLSR é o *scheduler* mostrado na Figura 13, através do qual é possível se detectar mudanças e onde se podem registrar tarefas a serem executadas regularmente. Por padrão a cada 0,1 segundos cada tarefa poderia ser executada, mas isso é completamente configurável. Para evitar condições de corrida com dados compartilhados em outros processos, pode-se fazer uso de regiões críticas com o uso de um *mutex* para cada caso.

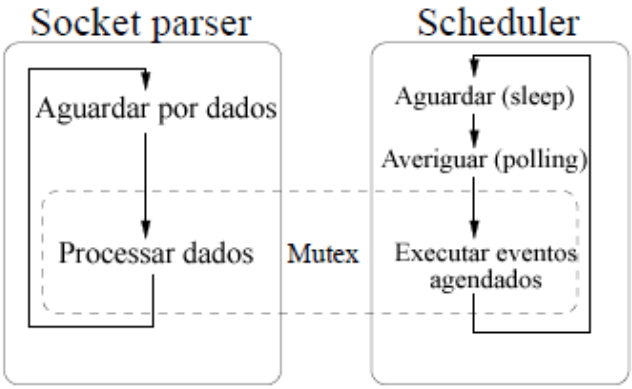

Figura 13. Código comum protegido por região crítica

Com relação à forma de funcionamento interno do *core* do OLSR, quatro tipos básicos de mensagens existem, sendo as duas principais as mensagens HELLO e TC (*topology control*). Com as mensagens HELLO é possível a cada nó se anunciar na rede e descobrir quais vizinhos existem e a qualidade de seus links para com eles. Através das TC *messages* é possível anunciar isso ao demais nós da rede, compartilhando esse conhecimento.

O esquema de detecção de *links* implementado no OLSR verifica que, ao perder o retorno de três mensagens de HELLO seguidas, o *link* seria considerado inválido. No entanto, este processo é mais refinado multiplicando-se um intervalo de mensagens pelo quantificador de qualidade do *link*, o qual varia entre 0 e 1. Desta forma um *link* não é sumariamente removido da base de dados de cada nó no caso de uma qualidade ruim, mas se estima que uma resposta originada no mesmo demore mais para retornar, caso não exista uma rota alternativa de melhor qualidade.

5.1.2 Gnunet

Outra tecnologia utilizada foi o Gnunet, o qual é definido como um *framework* para redes *peer to peer* descentralizadas, o qual oferece serviços de criptografia, descoberta e alocação de recursos.

Há dois fluxos de dados típico no Gnunet, em um deles um cliente como o *gnunet-search* ou *gnunet-inserv* se conecta ao *tcpserver* (através de uma conexão TCP segura, tipicamente de *loopback*). A operação de conexão é iniciada através de bibliotecas auxiliares escritas para a aplicação específica, neste caso são chamadas a *libgnunetfsui* e *libgnunetecrs*, que por sua vez chama *libgnunetfs*, enviando o pedido via TCP usando *libgnunetutil*. Um pedido comum é entregue pelo *tcpserver* ao core do Gnunet para o serviço de FS (*File Sharing*), por exemplo, que passa então a consulta para o serviço interno *GAP* para o

roteamento. Este, por sua vez, produz uma consulta que é enfileirada na conexão para o núcleo de transmissão para os outros. Depois de algum tempo, o buffer inteiro é liberado para ser enviado ao(s) destino(s) com a ajuda dos módulos de transporte, quando a mensagem *peer to peer* é transmitida através de um serviço de transporte como o UDP.

O segundo caminho ocorre após o *gnunetd* receber uma mensagem de um mecanismo de transporte. A mensagem é decodificada de acordo com seu tipo; se é uma consulta do *GAP* onde módulos do FS realizaram uma pesquisa e, eventualmente, enviaram uma resposta, a mensagem retorna pelo mesmo caminho, chegando ao ponto onde é transmitida para a interface do usuário. Os principais arquivos e bibliotecas disponíveis são o *gnunet_protocols.h*, o *gnunet_transport.h*, o *gnunet_core.h* e o *gnunet_util.h*.

O servidor *gnunetd* é responsável pelo carregamento dos módulos, demultiplexação das mensagens originadas de clientes e outros nós da rede, além de mensagens encriptadas vindas de outros nós também.

Além disso, aplicações (como a de compartilhamento de arquivos) são construídas em cima do *core* do Gnunet, o núcleo em si não tem noção de compartilhamento de arquivos de aplicativos, pois múltiplos serviços podem utilizar o *CoreAPIForApplication* para acessar o *core* do Gnunet. Se necessário, as aplicações são responsáveis pela inserção de confiabilidade (através da retransmissão) na camada de rede, comumente usando o serviço de *RPC*.

Enquanto o núcleo irá impor limitações de largura de banda definida pelo usuário, os serviços deverão implementar melhores estratégias de aplicações, comunicando estas estratégias para o núcleo e dando a sua mensagem uma prioridade e um tempo desejado para envio. O núcleo, por sua vez, considera a largura de banda disponível, prazos e prioridades.

Uma aplicação para o Gnunet se divide em duas partes principais: a primeira parte reside no espaço de processos do *gnunetd*, o qual é implementado como um *plugin* de aplicação. O núcleo notifica a camada mais baixa do serviço de mensagens quando chegarem pacotes direcionados para a aplicação. Essa camada da aplicação é responsável por todas as interações entre os nós e, normalmente, ela é constituída por vários módulos de serviço. Por exemplo, o compartilhamento de arquivos (aplicação FS) usa o serviço de roteamento e BPA para o serviço e *sqstore* para o banco de dados.

A segunda parte de cada serviço é uma interface que é invocada pelo usuário, pois a idéia é que o usuário inicie um programa em separado (como *gnunet-search*, o *gnunet-gtk* ou *gnunet-download*) que fornece uma interface de usuário para o serviço. Normalmente, a interface de usuário é dividida em uma biblioteca compartilhada que fornece a implementação real e um cliente concreto, fazendo com que a biblioteca compartilhada foque sobre a

interação com *gnunetd* através da conexão TCP local (*loopback*) para conversar com o processo do serviço, enquanto o cliente concreto trata da interação com o usuário real (como, por exemplo, uma aplicação GTK).

O core do Gnunet fornece uma maneira de registrar determinados tipos de mensagens originados das aplicações. A implementação do *tcpio* fornece funções para enviar e receber mensagens via TCP, fazendo com que a ligação TCP entre a aplicação e o *gnunetd* seja totalmente segura, via *loopback*, podendo-se especificar a lista de endereços IP confiáveis (ou seja, a LAN) que têm permissão para conectarem-se como clientes ao *gnunetd*.

5.1.3 Protótipo UOP

Um fator chave para a validação é fazer uso de um ambiente que suporte as especificações e requisitos de ubiqüidade detalhadas na seção 4.1, como mobilidade de código e suporte a contextos [Satyanarayanan 2001]. Isso se faz necessário pois os paradigmas existentes (imperativo, funcional, lógico e orientado a objetos) não foram planejados tendo-se em vista os cenários de ubiqüidade. Alguns novos paradigmas como *Context-Oriented Programming* [Gassanenko 1998], *Subject-Oriented Programming* [Harrison 1993] e o Holoparadigma [Barbosa 2002] exploraram esta área na tentativa de suportar contextos ubíquos, mas com limitações.

Optou-se pelo UOP (*Ubiquitous Oriented Programming*) [Garzão 2010], que surge como um modelo de programação orientado ao desenvolvimento de aplicativos ubíquos introduzindo um conjunto de diretrizes que visam facilitar o desenvolvimento de software para este fim, pois o mesmo tem uma semântica simples, a qual estimula a exploração automática da distribuição (distribuição implícita), concorrência (processamento distribuído e paralelo), adaptação ao contexto [Satyanarayanan 2001], serviços [Austin 2002], localização [Syvanen 2005] e contextos [Dey 2001].

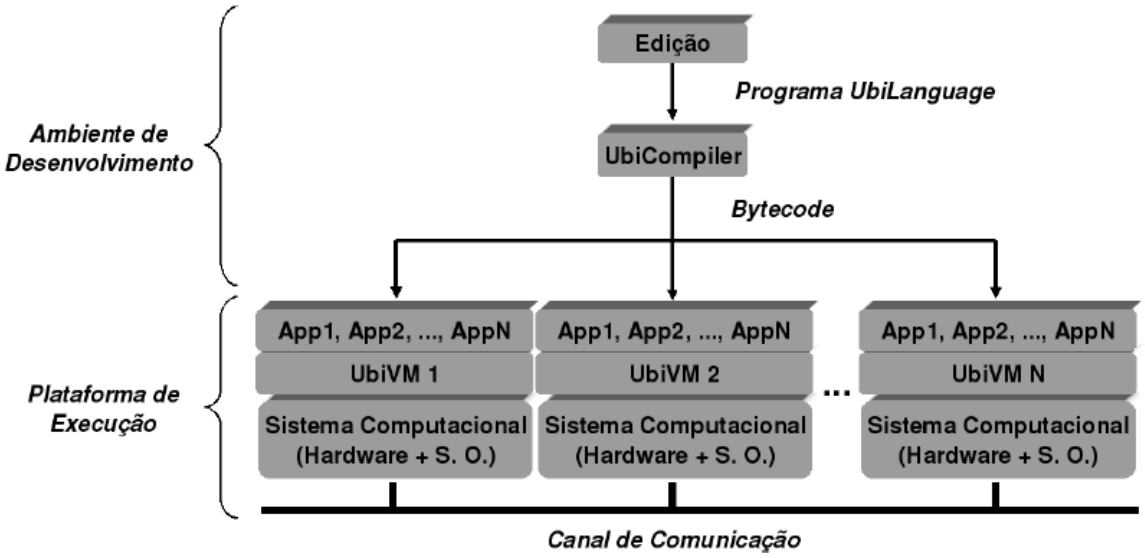

Figura 14. Ambiente de execução de aplicativos UOP [Garzão 2010]

O ambiente para desenvolvimento e execução de aplicativos proposto para o UOP pode ser visto na Figura 14. O Ambiente de Desenvolvimento é composto pela *UbiLanguage* (abreviadamente UbiL) e pelo *UbiCompiler* (abreviadamente UbiC). UbiL é a linguagem de programação que concretiza os conceitos propostos no UOP enquanto que o UbiC é o compilador que traduz código escrito em UbiL para *bytecode*. A Plataforma de Execução é composta pelo Sistema Computacional (Hardware + Sistema Operacional) e pela *Ubiquitous Virtual Machine* (abreviadamente UbiVM). A UbiVM é a máquina virtual ubíqua responsável por executar o *bytecode* gerado pelo UbiC. As UbiVMs interagem entre si para gerenciar a distribuição dos conteúdos e serviços existentes nos contextos, os quais são criados e gerenciados dinamicamente pelos aplicativos em execução.

Segundo [Garzão 2010], no UOP os contextos são criados dinamicamente agrupando dados e serviços. A localização indica a região física onde o dispositivo se encontra e os serviços são tarefas disponibilizadas para um contexto. O UOP utiliza conceitos existentes na programação orientada a objetos (como propriedades, métodos, mensagens, sobrecarga de métodos, herança e polimorfismo) e em *Web Services* (como Provedores de Serviços e Consumidores), complementando as mesmas com a inserção de suporte as abstrações da computação ubíqua, como contextos, adaptação, sensibilidade ao contexto, concorrência, mobilidade e distribuição.

Dentro da contextualização sobre UOP, as entidades são fundamentais, pois são os objetos relevantes para a interação entre um usuário e uma aplicação [Dey 2001]. No UOP uma entidade estática representa um conjunto de objetos com características próprias, como as

classes do paradigma orientado a objetos, porém estende as mesmas com os conceitos de serviços e adaptação ao contexto. Em suma, uma entidade estática pode definir:

- Estados (propriedades);
- Comportamento dos objetos (métodos);
- Serviços que disponibiliza em cada contexto.

Assim como toda classe é instanciada através de objetos no paradigma orientado a objeto, uma entidade estática é instanciada através de entidades dinâmicas, as quais armazenam os estados através de seus atributos e são capazes de tomar uma ação baseada nas mensagens recebidas, se relacionando com outras entidades dinâmicas através da troca de dados e serviços. Os serviços, por sua vez, são disponibilizados nos contextos para utilização por outras entidades dinâmicas, podendo estas disponibilizarem e/ou consumirem *N* serviços dentro dos contextos em que estão envolvidas. Como uma opção simplificada de implementação de serviços, pode ser utilizado um processo similar ao utilizado em *Web Services* com um seqüência de *Find/Bind/Execute* em um registro central de contexto. No entanto, em se tratando de um ambiente ubíquo *Ad Hoc*, o modelo proposto estende esta implementação para uma administração completamente distribuída de serviços, capaz de transcender diferentes níveis de rede.

O contexto de uma entidade tem relação direta com a sua adaptação ao ambiente, pois cada vez que um nó entra em um novo contexto pode publicar seus serviços e usufruir de novos serviços fornecidos pelas demais entidades, fazendo parte de vários contextos e grupos simultaneamente. A característica de adaptação ao contexto é a reação às alterações no ambiente, administrando os recursos disponíveis e as necessidades das aplicações. Qualquer alteração em uma entidade dinâmica ou no ambiente pode forçar uma adaptação, e dependendo das alterações sofridas no contexto, pode ser necessária outra característica importante do UOP: a mobilidade forte de código [Fuggetta 1998][Naseen 2004], colocada em prática com a migração e a replicação de serviços proposta no modelo.

Baseado no modelo do UOP foi criado um protótipo funcional [Garzão 2010]. A LibUVM foi desenvolvida para auxiliar na prototipagem do UbiC e da UbiVM. A LibUVM é uma biblioteca de código que auxilia a leitura e a geração do arquivo UVM. Esta biblioteca é utilizada tanto pelo UbiC (para gerar o arquivo UVM) como pela UbiVM (para carregar o arquivo UVM).

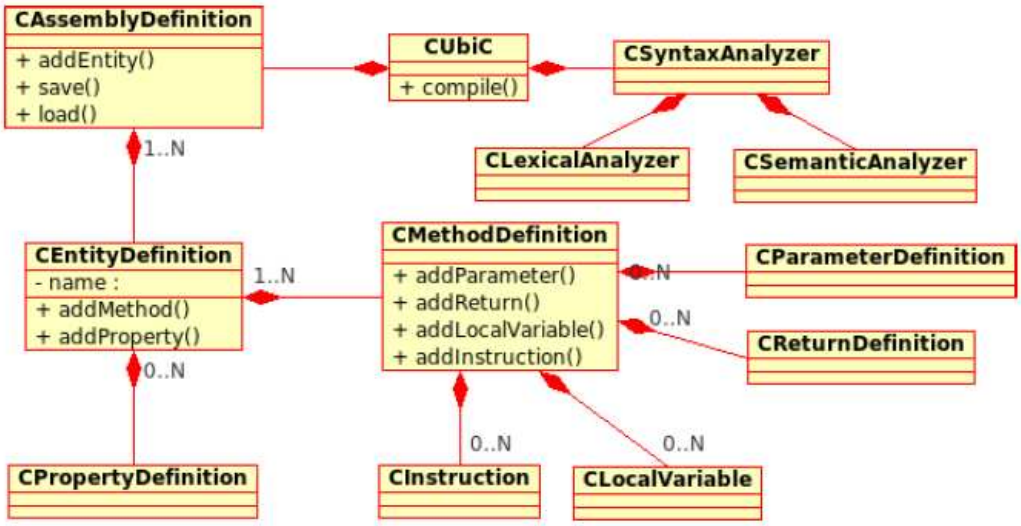

Figura 15. Diagrama de classes da LibUVM [Garzão 2010]

O diagrama de classes da LibUVM na Figura 15 contém as principais classes existentes nesta biblioteca. *CAssemblyDefinition* é a classe principal; a partir dela é possível definir várias entidades (classe *CEntityDefinition*) e obter o *bytecode* propriamente dito. *CEntityDefinition* contém a definição de uma entidade e possibilita a definição das propriedades, métodos, serviços e eventos de uma entidade. *CMethodDefinition* contém a definição de um método e possibilita a identificação dos parâmetros, valores de retorno, variáveis locais e instruções de um método.

5.2 Protótipo do Ubiservices

Esta seção detalha as duas etapas de implementação que ocorreram, abordando na seção 5.2.1 a integração entre as tecnologias OLSRd/Gnunet para o desenvolvimento do protocolo entre camadas. Na seção 5.2.2 é detalhado o esforço a respeito da integração do protótipo do UOP para funcionar com a *lib* do Ubiservices no OLSRd.

5.2.1 Integração OLSRd-GNUNET

O protocolo entre camadas utiliza o protocolo OLSR [Tonnesen 2009] para prover *multicast* sobre redes *mesh*. Na rede *overlay*, a LIB *gnunetlib* é utilizada como uma extensão do OLSR no nível de aplicação, logando na rede Gnunet [Bennett 2009] para integrar a rede *mesh* na rede P2P.

A interligação entre os níveis de rede ocorre através dos *gateways* representados na Figura 16 pelos *peer to peer GnutNet node*, fundamentais para o funcionamento do Ubiservices.

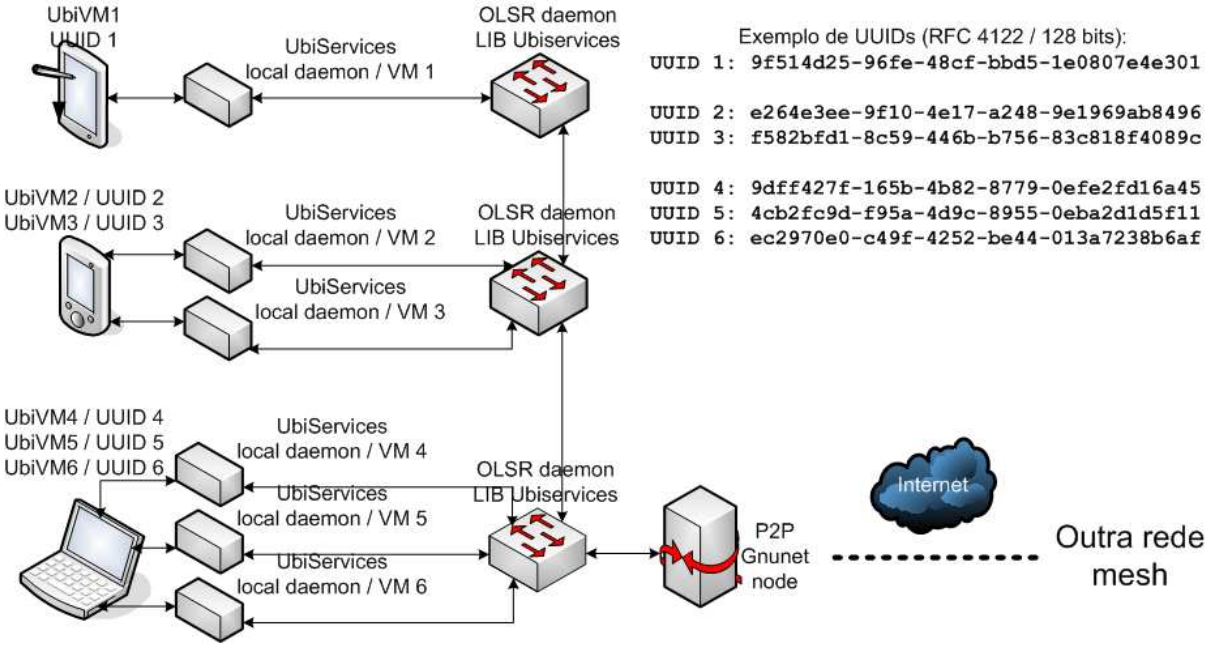

Figura 16. Interligação da arquitetura do Ubiservices

A Figura 16 extrapola a integração do OLSRd com o Gnunet, representando também parte da UbiVM do UOP, cuja interligação com o OLSR é detalhada na subseção 5.2.2. A *lib* Ubiservices, o local *daemon* e o próprio nó *peer to peer* do Gnunet são implementados em C++, possibilitando a sua integração com aplicações implementadas para o ambiente ubíquo. Conforme a Figura 16, múltiplas UbiVMs podem executar na máquina ou PDA, mas em geral somente uma fará sentido pois só um usuário utilizaria um PDA, por exemplo. O Ubiservices *local daemon* é parte da UbiVM, implementado através de uma biblioteca representada pelo *Communication Provider* da UbiVM, o qual é detalhado na subseção 5.2.2 na integração OLSR-UOP. Em cada dispositivo computacional entra em execução a UbiVM, uma instância do OLSR *daemon* e, caso tenha-se acesso direto a Internet, a interface de comunicação com o Gnunet dentro do OLSRd é ativada, fazendo que ao inicializar o OLSRd instancie-se como um nó na rede *peer to peer* do Gnunet.

Em termos de organização de código, a Figura 17 mostra como estão estruturados os módulos do Ubiservices internamente.

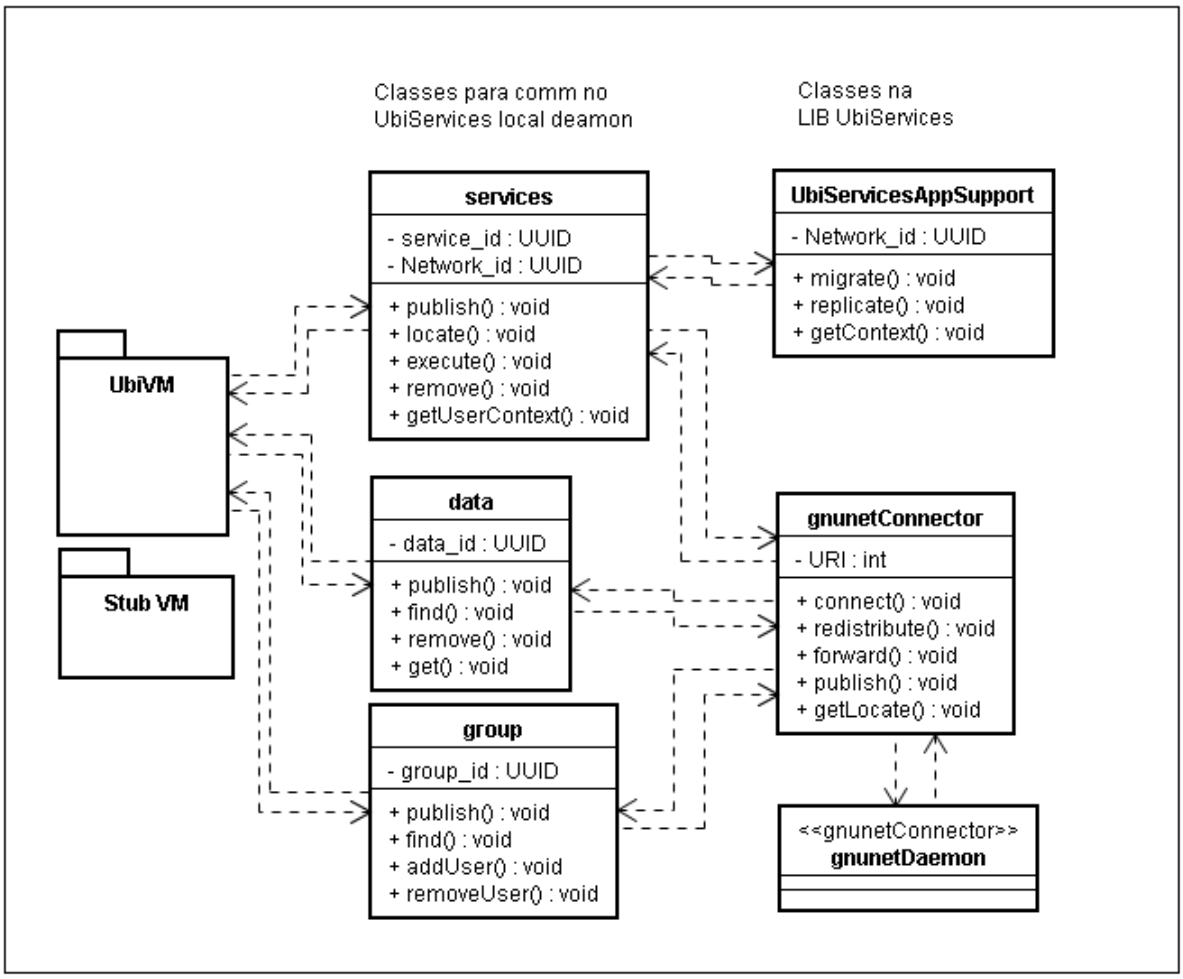

Figura 17. Diagrama dos módulos da implementação

Na Figura 17 é mostrado um diagrama que representa as classes implementadas no Ubiservices como módulos, os quais podem ter mais subclasses que não chegaram a se concretizar na implementação. Na Figura 17 é representada a interligação não só do OLSRd com o Gnunet, como também do OLSRd com a UbiVM. Para a UbiVM são disponibilizados métodos como para manipulação de serviços, como *services.publish*, *services.locate*, *services.execute*, *services.remove* e *services.getUserContext*. Assim como existe uma classe para representação dos serviços, foi feita outra para representação de informações (*data*) com seus métodos de publicação(*data.publish*), busca (*data.find*), obtenção(*data.get*) depois de uma busca feita com sucesso, e remoção (*data.remove*).

Além disso, a classe *group* identifica os grupos que o usuário pode criar, sejam de interesses locais ou distribuídos, onde um usuário pode ser adicionado(*group.AddUser*) ou removido (*group.removeUser*), assim como uma informação pode ser publicada através do grupo (*group.publish*).

Ainda na Figura 17, os métodos das classes *services*, *data* e *group* são disparados a partir do OLSRd, e podem interagir com outra classe, a *UbiservicesAppSuport*, para operacionalizarem os serviços de migração e replicação explicados na seção 4.5. Por fim, a classe *gnunetConnector* faz a interface direta com um módulo presente no *daemon* do Gnunet (*gnunetDaemon*), o qual não chegou a ser implementado de forma funcional. A classe *gnunetConnector* trata de fazer as devidas conversões das informações da rede *mesh* para a rede *peer to peer* do Gnunet, utilizando-se dos métodos mostrado na Figura 17.

Voltando ao OLSR e ao principal foco desta subseção, foi criada a *lib* Ubiservices. Dentre os arquivos que a compõem, existem:

- *olsrd_plugin.c*: que contém interfaces comuns aos *plugins* de inicialização do OLSRd, como o *olsrd_plugin_init()*, o qual chama a função *ubiOlsrInit()*. Esta, por sua vez, entra em funções relacionadas ao arquivo *ubiservice.c*;
- *ubiservice.c*: é onde a função *gnunetInit()* trata de iniciar a *thread* para se registrar no *socket parser* do OLSRd e passa a receber os pacotes que lhe cabem;
- *ubiApps.c*: é onde se encontram os métodos responsáveis pelas aplicações e comportamentos de alto nível, como *ubiMigrateInstance()* e *ubiReplicateInstance()*, além do *ubiGetUserContext()* e *ubiSetUserContext()*, os quais foram parcialmente implementados.

Ainda na inicialização no *olsrd_plugin.c*, como é mostrada na saída padrão mostrada na Figura 18, três parâmetros são passados no arquivo de configuração *olsrd.conf* com relação ao *plugin* do Ubiservices: *name* na linha 22, *context* na linha 21 e *shared info* na linha 20. Cada um deles fica armazenado de forma global em memória durante a execução do *plugin*, de forma que estas informações podem ser alteradas a qualquer tempo, principalmente com relação ao *context*, o qual representa o contexto no qual o nó está inserido. Este campo é de livre entrada, textual, podendo ser estabelecido ou modificado pelo usuário a qualquer tempo por interfaces disponíveis através de um *local daemon* que é agendado para ser executado pelo *scheduler* do OLSRd e permitir alterar informações locais.

```
1.  Checking eth0:
2.     Not a wireless interface
3.     Metric: 0
4.     MTU - IPhdr: 1472
5.     Index 2
6.     Address:192.168.1.102
7.     Netmask:255.255.255.0
8.     Broadcast address:255.255.255.255
9.  New main address: 192.168.1.102
10. Using 'etx_ff' algorithm for lq calculation.
11. TC: add entry 192.168.1.102
12. RIB: add prefix 192.168.1.102/32 from 192.168.1.102
13. ----- LOADING LIBRARY olsrd_ubiservice.so.0.1 ---------
14. *** constructor
15. OLSRD Ubiservice plugin 0.0.1
16. Checking plugin interface version:  5 - OK
17. Trying to fetch plugin init function: OK
18. Trying to fetch parameter table and it's size...
19. Sending parameters...
20. "sharedinfo"/"info geral... "... sharedinfo: OK
21. "context"/"PIPCA"... context: OK
22. "name"/"testStation0"... name: OK
23. Running plugin_init function...
24. ----- LIBRARY olsrd_ubiservice.so.0.1 LOADED ----------

25. ALL PLUGINS LOADED! --

26. Main address: 192.168.1.102

27. Scheduler started - polling every 0.05 seconds
28. *** olsr.org - 0.5.6-r8
29. (2010-04-07 00:12:59 on gbrand_slack_vm) ***
```

Figura 18. Inicialização do OLSR *daemon*

No que tange a parte do Gnunet do Ubiservices, a mesma se divide em duas partes. A primeira é uma biblioteca dinâmica que se conecta ao núcleo do *gnunetd*. Esta biblioteca fornece uma única função que é invocada pelo núcleo quando começa o *gnunetd*, registrando alguns *callbacks* junto ao *core* do Gnunet para lidar com determinadas mensagens *peer to peer*. Cada mensagem é limitada no tamanho (cerca de 60*Kbytes*) e deve ser ligada ao módulo de aplicação adequada usando um identificador único de mensagem definido em *src/include/gnunet_protocols.h*. A função registrada pela aplicação é chamada sempre que uma mensagem *peer to peer* com um identificador de mensagem própria dela for registrada. O módulo de aplicação pode enviar mensagens para outros nós usando o *CoreAPIForApplication*, e geralmente o módulo também faz uso do *gnunet tcpserver* para se comunicar com uma interface de usuário.

A segunda parte da aplicação do Ubiservices no Gnunet é a interface com o usuário. Neste caso, ela precisa se comunicar com o módulo de serviço que foi registrado junto ao *core* do Gnunet, fazendo uso do módulo *tcpio* para uma conexão via o *local tcpserver* (*loopback*). No caso do Ubiservices, a interface com o usuário não passou de um *output* da saída padrão,

para visualização e *debugging*. Qualquer tratamento dos pacotes vindos do OLSRd foi feito na *application* anexada diretamente ao *core* do Gnunet, sem uma interface em separado.

Deve-se salientar, no entanto, que a implementação da solução proposta foi somente parcialmente implementada. A integração entre os dois, considerada a primeira etapa da implementação, se mostrou complexa, exigindo um bom nível de conhecimento de ambas as implementações, e não resultou em uma plataforma funcional para integração na segunda etapa da implementação. A próxima subseção descreve o esforço para integração do OLSR com o UOP.

5.2.2 Integração OLSR-UOP

A integração do OLSR com o UOP foi iniciada pela modificação do módulo *Communication provider* da UbiVM [Garzão 2010], cujo *header* é mostrado na Figura 19. Merecem destaque alterações nos processos *sendBroadCastPacket* na linha 14, *sendUnicastReply* na linha 15, além de *processReceivedPacket* na linha 19 e *processRequestOperation* na linha 20 precisaram ser alterados para enviar pacotes na mesma porta do *socket* que está configurado para receber pacotes no OLSRd. O cabeçalho abaixo mostra parte dos métodos utilizados.

```
1.  class CCommunicationProvider {

2.  public:
3.  static CCommunicationProvider* getInstance()
4.  {
5.  static CCommunicationProvider *instance = NULL;
6.  return instance ? instance : (instance = new CCommunicationProvider());
7.  }
8.  void setConfig(std::map<std::string, CContext*>* contextList, uint bindPort, uint sendPort);
9.  void run();
10. uint getNextPacketNumber()
11. {
12. return _packetNumber++;
13. }
14. void sendBroadcastPacket(const char* packet, size_t length);
15. void sendUnicastReply(const char* packet, size_t length, udp::endpoint& endpoint);

16. private:
17. CCommunicationProvider() { };
18. void threadedCode();
19. void processReceivedPacket(const char* char_packet, size_t lenght, udp::socket& sock, udp::endpoint&
    sender_endpoint);
20. void processRequestOperation(CBinString& requestPacket, udp::socket& sock, udp::endpoint& sender_endpoint,
    SPacketHeader& requestHeader);
```

```
21. void processReplyOperation(CBinString& requestPacket, udp::socket& sock, udp::endpoint& sender_endpoint,
    SPacketHeader& requestHeader);
22. std::string dumpPacket(const char* packet, uint length);

23. boost::thread*  _thread;
24. uint          _packetNumber;
25. uint _bindPort;
26. uint _sendPort;
27. boost::asio::io_service _io_service;
};
```

Figura 19. Funções principais do *Communication Provider*.

Juntamente com a alteração dos módulos atingiu a inserção de métodos no OLSR *scheduler* e no OLSR *packet parser*. Foi possível atingir uma versão funcional do UOP funcionando dentro de uma rede mesh com o *daemon* OLSR. Na Figura 1 é mostrada a aplicação “Oportunidades pedagógicas” sendo executada em um nó da de uma rede *mesh*. Neste exemplo, as informações iniciais são preenchidas da linha 3 a 9, onde o interesse geral e específico são importantes para o compartilhamento dos objetos de aprendizagem que são cadastrados logo a seguir.

```
1. sh-3.1# ../../ubivm/ubivm oportunidades_pedagogicas
2. UbiVM - Release 0.1.0 (development release)

3. OPORTUNIDADES PEDAGOGICAS
4. NOME.................:
5. gustavo
6. INTERESSE GERAL.......:
7. redes
8. INTERESSE ESPECIFICO..:
9. p2p

10. OBJETOS DE APRENDIZAGEM
11. OBJETO
12. aaaaaaaaaaaa
13. TIPO
14. livro
15. QUEM_PUBLICOU
16. bbbbbbbbbb
17. ANO
18. 2000

19. OBJETOS DE APRENDIZAGEM
20. OBJETO
21. fim.
```

Figura 20. Execução de “Oportunidades pedagógicas” na rede *mesh*

Uma vez que na Figura 20 um usuário cadastrou seu interesse geral em redes na linha 7 e um objeto de aprendizagem nas linhas 10 a 18, a Figura 21 mostra um segundo usuário se cadastrando no mesmo interesse geral na linha 7 e recebendo, após cadastrar um objeto de aprendizagem que possui da linha 10 a 18, os objetos de aprendizagem dos usuários que estão

na rede *mesh* com o mesmo interesse que o seu. Neste caso, o objeto do aprendiz Gustavo, nas linhas 22 a 31.

```
1. sh-3.1# ../../ubivm/ubivm oportunidades_pedagogicas
2. UbiVM - Release 0.1.0 (development release)
3. OPORTUNIDADES PEDAGOGICAS
4. NOME.................:
5. jorge
6. INTERESSE GERAL.......:
7. redes
8. INTERESSE ESPECIFICO..:
9. ubiquas

10. OBJETOS DE APRENDIZAGEM
11. OBJETO
12. ccccccccccc
13. TIPO
14. livro
15. QUEM_PUBLICOU
16. sssssssssss
17. ANO
18. 2001

19. OBJETOS DE APRENDIZAGEM
20. OBJETO
21. fim

22. APRENDIZ
23. gustavo
24. OBJETO
25. aaaaaaaaaaaa
26. TIPO
27. livro
28. QUEM_PUBLICOU
29. bbbbbbbbbb
30. ANO
31. 2000
```

Figura 21. Recebimento de informações de mesmo interesse

Nesta implementação funcional se procurou seguir o padrão do formato dos pacotes enviados pelo OLSRd, que é mostrado na Figura 22 para comunicação entre o UOP e o OLSR *daemon* executando a *lib* do Ubiservices.

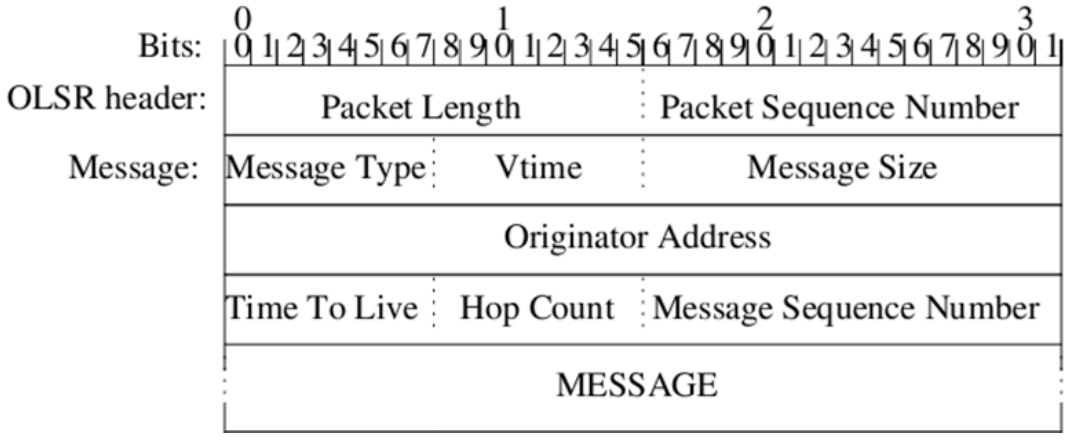

Figura 22. Pacote OLSR genérico [Tonnesen 2009]

Detalhando os campos da Figura 22, são necessárias as seguintes informações para se conseguir uma comunicação com o OLSR:

- *Packet Length*: comprimento em *bytes* do pacote completo, incluindo o cabeçalho;
- *Packet Sequence Number*: número seqüencial incrementado cada vez que uma nova mensagem OLSR é transmitida pelo nó do usuário. Um número separado de seqüência é mantido para cada interface, permitindo aos pacotes transmitidos através de uma mesma interface permanecerem seqüenciais;
- *Message Type*: um número inteiro que identifica o tipo da mensagem, sendo o intervalo de 0-127 reservado ao OLSR, enquanto o espaço 128-255 é considerado “privado”, podendo ser utilizado para extensões personalizadas do protocolo;
- *Vtime*: indica por quanto tempo após a recepção da mensagem um nó irá considerar as informações contidas na mensagem como válidas;
- *Message size*: o tamanho da mensagem, incluindo o cabeçalho da mensagem, em bytes;
- *Originator address*: endereço principal do nó origem mensagem;
- *Time To Live* (TTL): número máximo de saltos que esta mensagem percorre ao ser enviada. Usando este campo pode-se controlar o raio do *flooding*, algo importante para evitar *overhead* desnecessário na rede *mesh*;
- *Hop Count*: número de vezes que a mensagem tenha sido encaminhada;
- *Message Sequence Number*: número seqüencial incrementado a cada vez que um novo pacote OLSR é transmitido pelo mesmo *host* de origem.

Foi utilizada por padrão a porta UDP 698 para transmissão em broadcast dos pacotes.

6 ASPECTOS DE AVALIAÇÃO

A validação do Ubiservices havia sido proposta de forma totalmente funcional, ou seja, validando a funcionalidade do protótipo gerado a partir do modelo. A princípio seria criado um ambiente real e estruturado com um número mínimo de nós para validar a proposta. No entanto, mesmo não tendo sido isso possível, a fim de validar o modelo três cenários foram propostos: educação ubíqua, comércio ubíquo e redes sociais.

Para efeitos de comparação, os cenários expostos abaixo são semelhantes aos propostos por [Garzão 2010], mas foram expandidos para tirar vantagem do modelo oferecido pelo Ubiservices. Na seção 6.1 é abordado o cenário de Educação Ubíqua, com detalhes do cenário na subseção 6.1.1 e uma discussão a respeito das vantagens do Ubiservices neste cenário na subseção 6.1.2. Na seção 6.2 é colocado o cenário de Comércio Ubíquo, seguindo a mesma estrutura de subseções, com a 6.2.1 detalhando o cenário através de uma tabela de interações e a subseção 6.2.2 discutindo as particularidades apresentadas na subseção anterior. Por fim, na seção 6.3 é colocado o cenário de Redes Sociais, com foco diferente dos anteriores, segue na subseção 6.3.1 com a apresentação e detalhamento do cenário, ocorrendo uma discussão a seu respeito na subseção 6.3.2.

6.1 Educação Ubíqua

A disseminação da computação ubíqua ocasionará um impacto significativo na área da Educação [Yau 2003][Rogers 2005][Barbosa 2006]. No cenário da educação apoiada pela computação ubíqua, novos pressupostos educacionais devem ser pensados, uma vez que os recursos pedagógicos podem ser acessados a qualquer momento e em qualquer lugar. O suporte ubíquo permite a construção de programas de aprendizagem relacionados com questões dinâmicas do contexto do aprendiz. O ambiente controla as aplicações orientadas à educação, possibilitando que o contexto seja vinculado com os objetivos pedagógicos. A educação neste cenário é dinâmica e os recursos educacionais estão distribuídos em contextos. Baseado nos objetivos do aprendiz, o sistema pode gerar intervenções do tipo: um material/pessoa/dispositivo que se relaciona com seu objetivo está disponível para você agora.

O cenário proposto é o Relacionamento por interesses similares entre aprendizes, e foi baseado no cenário proposto em [Barbosa 2008][Barbosa 2006]. Cada aprendiz informa, para a aplicação, seus interesses e os objetos de aprendizagem que gostaria de compartilhar. A partir deste momento, conforme o aprendiz move-se de um ambiente para outro, as novas oportunidades de aprendizado são apresentadas. Novas oportunidades são identificadas

quando aprendizes com interesses similares estão na mesma localização simbólica, e ao se movimentarem dentro de áreas com diferentes contextos, mudanças de interesse podem ocorrer em função dos alunos que se apresentam, identificando novas oportunidades de aprendizado e conhecimento.

6.1.1 O cenário

Um cenário de educação Ubíqua que pode ser pensado para o contexto explicado na subseção 6.1.1 é mostrado na Tabela 2, baseado no cenário criado por [Garzão 2010] para contextos delimitados em uma mesma área, e que presuma conectividade de todos os usuários da rede. A cada tempo é feita uma descrição do personagem atuante naquele instante e das ações geradas por ele. Com isso, consegue-se montar relações de efeito e suas conseqüências para os demais integrantes do grupo.

Em um ambiente real em que nem todos os PDAs e *smartphones* tenham conectividade em tempo real, o modelo do Ubiservices consegue contornar isso e permitir aos dispositivos isolados a interação com os demais a sua volta. Além disso, dentro da idéia de grupos locais e distribuídos, uma vez que o usuário cadastre seus interesses, não só os usuários locais podem ver isso, como também usuários que fizerem parte do grupo de interesse distribuído conectados através da rede *peer to peer*. A Tabela 2 mostra o detalhamento desta possibilidade de integração entre o UOP e um ambiente distribuído utilizando o Ubiservices.

Neste cenário, Jorge está localizado no PIPCA (Centro de pós graduação da computação na Unisinos) com seu PDA e tem interesse somente em grupos de interesses locais. Então ele cadastra seus interesses e disponibiliza seus objetos de aprendizagem. Logo em seguida, Gabriel publica seus interesses tanto de forma local como distribuída através da rede *peer to peer*, entrando também no contexto do PIPCA.

Já Graciele está na sua casa conectada via a rede *peer to peer* somente, fora de qualquer contexto, mas participando de pelos um grupo de interesses distribuídos. Alex, por outro lado, se dirige primeiramente ao laboratório geral localizado dentro do PIPCA, e mais tarde segue para o para o Mobilab (Laboratório de pesquisa sobre computação ubíqua) que fica no mesmo prédio do PIPCA, porém em andares e, conseqüentemente, contextos diferentes. Alex, junto ao seu PDA¸ mostra interesse somente em informações distribuídas na rede *peer to peer*, sem importar o que esteja sendo compartilhado localmente (apesar de poder informar aos demais usuários locais o que possui).

Tabela 2. Cenário de Educação Ubíqua

Tempo	Personagem	Ações
1	Jorge	Executa a aplicação "Oportunidades pedagógicas" no seu PDA. Cadastra o interesse geral "Compiladores", interesse específico "Análise semântica", e seus dois objetos de aprendizagem, configurados **como de interesse local**: - Artigo "Advanced compiler optimizations for supercomputers", ACM, 1986. - Artigo "Dependence graphs and compiler optimizations", ACM, 1981.
1	PDA do Jorge	UbiVM utiliza o Ubiservices para publicar os interesses de Jorge, sua localização e as informações sobre os seus objetos no contexto do Laboratório Geral. **O Ubiservices cria um grupo local de interesse, neste caso pessoal chamado Jorge (a não ser que ele especifique um nome).** **Associado a ele, armazena no seu nó da rede mesh esta informação e não a envia para um nó pra rede P2P.**
2	Gabriel	Executa a aplicação "Oportunidades pedagógicas" no seu PDA. Cadastra o interesse geral "Compiladores", interesse específico "Geração de código", e seu objeto de aprendizagem, configurados como de **interesse local e distribuído:** - Artigo "Code generation in the Columbia Esterel Compiler", EURASIP, 2007.
2	PDA do Gabriel	UbiVM utiliza o Ubiservices para publicar os interesses de Gabriel, sua localização e as informações sobre o seu objeto no contexto do Mobilab. **O Ubiservices cria um grupo local de interesse, neste caso pessoal chamado Gabriel (a não ser que ele especifique um nome) e outro de interesse na rede P2P.** **Associado a ele, armazena no seu nó da rede mesh esta informação e envia a informação para um nó pra rede P2P**.
3	Graciele	Executa a aplicação "Oportunidades pedagógicas" no seu computador fora do ambiente comum aos demais usuários, de casa, configurados como de **interesse distribuído somente**: Cadastra o interesse geral "Sistemas operacionais" e interesse específico "Gerência de memória virtual".
3	PDA da Graciele	**O daemon do GnutNet utiliza a UbiVM para publicar os interesses de Graciele e sua localização em um grupo de interesse, distribuído.**
4	Alex	Executa a aplicação "Oportunidades pedagógicas" no seu PDA. Cadastra o interesse geral "Compiladores", interesse específico "Otimização de código", e seu objeto de aprendizagem, configurados **como de interesse distribuído somente**: - Livro "Compilers: Principles, Techniques, and Tools", Addison Wesley, 1988.
4	PDA do Alex	UbiVM **utiliza o Ubiservices** para publicar os interesses de Alex, sua localização e as informações sobre o seu objeto no contexto PIPCA. **O Ubiservices cria um grupo de interesse distribuído na rede P2P, neste caso pessoal chamado Alex (a não ser que ele especifique um nome).**
5	Alex	Dirige-se ao Laboratório Geral.
5	PDA do Alex	UbiVM identifica a localização simbólica "Laboratório geral",

		publica sua nova localização e envia solicitação para listar quem está no contexto.
6	PDA do Jorge	UbiVM recebe solicitação e responde com os interesses de Jorge (somente locais).
7	PDA do Alex	UbiVM filtra por interesses similares, mesma localização, e não identifica oportunidades pedagógicas.
8	Alex	Dirige-se ao Mobilab.
8	PDA do Alex	UbiVM identifica a localização simbólica "Mobilab", publica sua nova localização e envia solicitação para listar quem está no contexto.
9	PDA do Gabriel	UbiVM recebe solicitação e responde com os interesses de Alex.
10	Gabriel	Está no Mobilab, e **busca por interesses similares no local e em grupos de interesse distribuídos**.
10	PDA do Gabriel	UbiVM filtra por interesses similares, na mesma localização, e identifica que Jorge também tem interesse no mesmo assunto. **Também recebe como resposta o interesse de Graciele, que não está fisicamente no mesmo local, mas compartilha informações de interesse comum.** Solicita os objetos de aprendizados publicados por eles nos contextos através do Ubiservices;
11	PDA do Jorge	UbiVM recebe solicitação vinda do PDA do Gabriel e responde com seus objetos de aprendizagem.
12	Comp. da Graciele	UbiVM recebe solicitação vinda do PDA do Gabriel e responde com seus objetos de aprendizagem.
13	PDA do Gabriel	UbiVM recebe e lista dos objetos de aprendizagem de Jorge e Gabriel.
13	Gabriel	Visualiza a mensagem sobre os interesses de Jorge e Graciele, com a lista dos seus objetos.
14	Gabriel	Finaliza a execução do seu aplicativo e sai do PIPCA.

Ao cenário expandido de Educação ubíqua ainda pode-se adicionar situações de erro e falha, como a desconexão de Graciele (em casa conectada via *peer to peer*) e a manutenção de suas informações por outros nós da rede. Ela estaria desconectada, e sua presença apareceria como *offline* para os demais, mas as suas informações de interesse e compartilhamentos continuariam disponíveis para os demais que tiverem interesse no conteúdo disponibilizado através de seu grupo de interesses distribuídos (não local).

6.1.2 Discussão

Dentro do mesmo local pode-se notar diferentes interesses, alguns gostariam de compartilhar somente informações locais, outros somente distribuídas.

Jorge, por exemplo, no tempo 1 está munido de seu PDA e apresenta interesse somente nos grupos locais, uma vez que publicou seus objetos de aprendizagem e o interesse em compiladores somente como de interesse local. Isso significa que esta informação ficará restrita ao escopo da rede *mesh* local e aos usuários a sua volta, uma vez que esta rede não tem ponto centralizador e é formada pelos usuários dentro do mesmo ambiente.

Já Gabriel tem interesses tanto locais como distribuídos no tempo 2, ou seja, ele deseja que seu objeto de aprendizagem possa ser utilizado pelos demais colegas presentes a sua volta, mas também quer que seus interesses possam ser visualizados por qualquer usuário na rede *peer to peer*, pois esta informação ficará permanentemente disponível, replicada em outros nós da rede *peer to peer*.

Graciele, no tempo 3, está completamente fora do ambiente comum a Gabriel e Jorge, não apresentando, portanto, nenhum interesse local, mas somente em pesquisar e oferecer algo aos demais usuários da rede *peer to peer*. Gabriel poderia pesquisar as informações de Graciele, mas Jorge não, pois este teve como opção seu escopo restrito ao grupo local.

Por fim, Alex no tempo 4, mesmo tendo a mobilidade de um PDA e estando presente no mesmo local de Jorge e Gabriel, não quis publicar suas informações localmente, ficando restrito aos grupos de interesse distribuído, podendo, neste caso, obter informações e serviços de Gabriel e Graciele, mas não de Jorge.

O que se pode notar com o uso do Ubiservices é a dinamicidade que se proporciona a cada membro da rede, sem a necessidade de que todos estejam conectados a mesma rede física ou a Internet, pois a rede *mesh* é formada entre cada dispositivo presente. Certamente pelo menos uma das pessoas que está no mesmo local dos demais precisa dispor de um serviço de acesso a Internet para que todos possam utilizar rede *peer to peer*. Caso isso não aconteça, a rede *mesh* continua podendo ser formada, mas de forma isolada dos grupos de interesses distribuídos na rede *peer to peer*.

6.2 Comércio Ubíquo

Com relação ao cenário de comércio ubíquo, a computação ubíqua vem gerando novas oportunidades para esta área [GalanxhiJanaqi 2004][Gershman 2002]. A área de pesquisa dedicada à exploração de negócios em ambientes ubíquos é denominada Comércio Ubíquo.

No cenário do comércio apoiado pela computação ubíqua, novos pressupostos comerciais devem ser pensados, uma vez que produtos e serviços podem ser ofertados em qualquer lugar e a qualquer momento. O comércio nesse cenário é dinâmico e as oportunidades estão distribuídas em grupos. Por exemplo, baseado nos desejos de consumo do cliente, um sistema pode gerar intervenções do tipo: "foi encontrado um lojista nesse local, que possui uma oferta para o seu desejo de consumo".

O cenário proposto é o "Relacionamento entre Cliente e Lojista no Shopping", e foi baseado no cenário proposto em [Franco 2009]. Os lojistas publicam suas ofertas na praça de alimentação enquanto que o cliente cadastra o desejo por um determinado tipo de almoço.

Após visualizar as ofertas que atendem 100% do seu desejo, o cliente seleciona uma e realiza o pedido no próprio PDA, utilizando o serviço disponibilizado pelo lojista.

O cenário comércio ubíquo tem um foco voltado para um ambiente de venda dinâmico, auto informativo, onde os serviços e produtos que um vendedor esteja oferecendo possam ser oferecido diretamente ao cliente baseado nos seus interesses. Ambos os lados devem ter registrados seus interesses e produtos para que a venda direcionada aos interesses do cliente seja efetiva e funcional.

6.2.1 O cenário

Graciele está a caminho do Shopping Center para almoçar com sua família. Ela dispõe de R$ 15,00 e gostaria de comer um prato a base de massa. As ofertas do Restaurante da Vovó Olga e da Lancheria do Vovô Victório já foram publicadas. Quando Graciele entra na praça de alimentação do shopping, a UbiVM em execução no seu PDA verifica o "Repositório de Ofertas" de todas as lojas (Restaurantes e Lancherias), disponível na Praça de Alimentação, tentando identificar pratos a base de massa que estejam dentro do valor máximo pretendido. Neste instante, a UbiVM em execução no PDA da Graciele emite um alerta, permitindo que ela visualize as opções. Após verificar as ofertas, Graciele decide por um prato de "Massa a Carbonara" que custa R$ 13,00. Através do serviço de "Registro de pedido", disponibilizado pelo restaurante do prato escolhido, Graciele recebe o número do seu pedido, e se encaminha ao caixa para realizar o pagamento. Enquanto isso o seu pedido já está sendo preparado e em minutos será disponibilizado.

A Tabela 3 apresenta os detalhes desta execução, com uma evolução temporal das ações de cada participante. Neste cenário, o contexto "/shopping/praca_alimentacao" compartilha as informações sobre as ofertas dos lojistas. Como a localização do cliente ou lojista é irrelevante, este cenário não utiliza sensores e não define localização simbólica ou física.

Enquanto isso, Jorge está no trabalho e vê que não terá tempo suficiente para sair para almoçar, sendo necessário realizar um pedido de tele entrega. Ao invés de começar a acessar diversos sites de restaurantes ou ter que ligar diretamente para eles, ele verifica no seu PDA os serviços de restaurante disponível que atendam ao seu endereço. Estes serviços são disponibilizados por cada restaurante em grupos de interesse distribuído, podendo um restaurante fazer parte de diversos grupos. Tomando como exemplo uma lancheria, ele poderia participar de grupos como:

- “Bairro da Liberdade”: a respeito da cidade, shopping ou bairro onde esteja;

- “Baurus”: sobre a especialidade ou destaque servido;
- “Restaurantes de custo médio baixo”: a respeito do custo cobrado no estabelecimento;

Desta forma, Jorge pode pesquisar quais restaurantes estejam dentro destes grupos para conseguir escolher sem dificuldades de onde pedir. Jorge pode marcar, então, o restaurante escolhido como um interesse seu, relativo à área alimentícia. Se esse registro for feito como um interesse distribuído, outros usuários podem ter acesso ao mesmo.

Digamos que no dia seguinte Alex, um colega de Jorge, passa pela mesma situação no tempo 11 e, pesquisando não só os grupos de interesse de restaurantes, como também os grupos de interesse dos colegas da empresa mais próximos, descobre o mesmo restaurante de forma ágil, sem perda de tempo.

Tabela 3. Cenário de Comércio Ubíquo

Tempo	Personagem	Ações
1	Restaurante	Executa a aplicação "Ofertas Restaurante". **Utiliza o Ubiservices para cadastrar a oferta "Massa a Carbonara" com suas características**: - Lojista: Restaurante da Vovó Olga - Categoria: refeição - Componente base: massa - Ingredientes: Ovos, creme de leite, bacon, queijo parmesão, macarrão - Valor: R$ 13,00 - Serviço para realizar o pedido: restaurante_checkout - **Grupos locais cadastrados: Restaurante Vovó Olga** - **Grupos distribuídos cadastrados: massas, bairro da liberdade, restaurantes de custo médio**,
1	PDA do Restaurante	UbiVM, **através do Ubiservices**, publica a oferta no contexto "/shopping/praca_alimentacao". Também publica o serviço restaurante_checkout, utilizado pelos clientes para realizarem seus pedidos.
2	Lancheria	Executa a aplicação "Ofertas Lancheria". **Através do Ubiservices cadastra a oferta "Pizza presunto" com suas características:** - Lojista: Lancheria do vovô Victório - Categoria: lanche - Componente base: massa - Ingredientes: Presunto, queijo, orégano - Valor: R$ 11,00 - Serviço para realizar o pedido: lancheria_checkout **- Grupos locais cadastrados: Lanheria do vovo Victorio** **- Grupos distribuídos cadastrados: lancheria, pizzas, bairro da liberdade, restaurante de custo médio baixo.**
2	PDA da Lancheria	UbiVM, através do Ubiservices, publica a oferta no contexto "/shopping/praca_alimentacao". Também publica o serviço lancheria_checkout, utilizado pelos clientes para realizarem seus pedidos e os **grupos para pesquisa de informaçõe**s.
3	Graciele	Executa a aplicação "Busca ofertas". Cadastra o desejo "Prato de massa" com as características: - Categoria: refeição

		- Componente base: massa - Valor: R$ 15,00
3	PDA da Graciele	UbiVM, **através do Ubiservices,** verifica as ofertas de todas as lojas no contexto "/shopping/praca_alimentacao". Ao identificar que uma das ofertas atende 100% das características do desejo de Graciele, gera um aviso para Graciele com a oportunidade identificada, disponibilizando o serviço restaurante_checkout.
4	Graciele	Visualiza a mensagem com a oferta do prato de "Massa a Carbonara" e solicita a realização do pedido.
4	PDA da Graciele	UbiVM, **através do Ubiservices,** utiliza o serviço do restaurante para realizar o pedido, e fica aguardando o recebimento do número do pedido.
5	PDA do Restaurante	UbiVM, **através do Ubiservices**, executa a solicitação de serviço, vindo do PDA da Graciele. Este serviço executa uma ordem para a fila de preparo e de faturamento do pedido. Após isso, envia o número do pedido para o PDA da Graciele.
6	PDA da Graciele	UbiVM, **através do Ubiservices**, recebe o número do pedido e exibe para Graciele. Este aviso contém o número do pedido a ser informado no caixa, tanto para pagamento como para a retirada do prato.
6	Graciele	Verifica o número do pedido e dirige-se ao caixa. Seu prato já está na fila de preparo.
7	Jorge	**Executa a aplicação "Busca ofertas distribuídas".** Cadastra o desejo "Lanche" com as características: - Categoria: refeição - Componente base: bauru - Valor: R$ 10,00 **Dentro dos grupos: restaurantes de custo médio baixo, bairro da liberdade, baurus**
7	PDA do Jorge	**A UbiVM, através do Ubiservices, verifica as ofertas de todos os restaurantes que se incluem nos grupos de interesse distribuído mencionados, encontrando uma lista, da qual Jorge escolhe um, alertando Jorge que tem disponibilizado o serviço "tele_entrega".**
8	Jorge	Escolhe o alimento que deseja e efetua o seu pedido.
8	PDA do Jorge	**A UbiVM, através do Ubiservices, envia o pedido para o serviço do restaurante através da rede P2P disponível devido ao grupo de interesse distribuído.**
9	Serviço do Restaurante	**Recebe os dados do pedido, processa e envia de volta o número do pedido, tempo de entrega e valor a ser pago no ato da entrega.**
10	PDA do Jorge	**A UbiVM, através do Ubiservices, recebe a confirmação do pedido.**
10	Jorge	**Recebe a mensagem do pedido e aguarda a chegada.**
11	Alex	**No dia seguinte, executa a aplicação "Busca ofertas distribuídas".** Cadastra o desejo "Lanche" com as características: - Categoria: refeição - Componente base: bauru - Valor: R$ 10,00 **Dentro dos grupos: "restaurantes de custo médio baixo", "bairro da liberdade", "baurus" e de alguns de seus colegas de trabalho, como "Jorge" (que publica as informações que deseja no seu grupo pessoal)**
11	PDA do Alex	**A UbiVM, através do Ubiservices, verifica as ofertas de todos os restaurantes que estejam incluídos nos grupos de interesse distribuído mencionados, encontrando uma lista, da qual Jorge escolhe um, alertando Jorge que tem disponibilizado o serviço "tele_entrega".**

12		E seguem-se os mesmos passos do pedido de Jorge, mencionado anteriormente, do tempo 8 a 10.

6.2.2 Discussão

Novamente, assim como no cenário da seção 6.2.1, existem diferentes perspectivas de utilização pra o cenário de Comércio ubíquo que podem ser identificadas nos tempos 3, 7 e 11. Enquanto Graciele no tempo 3 utiliza-o na forma de grupos de interesse puramente local, pois está presente no local onde pode usufruir dos serviços oferecidos, Alex no tempo 11 e Jorge no tempo 7 utilizam as informações presentes nos grupos de interesse distribuído.

Graciele, ao chegar ao local no tempo 3, verifica localmente os restaurantes que podem lhe atender, enquanto Jorge faz uso das informações publicadas pelo restaurante em diferentes grupos distribuídos na rede peer to peer nos tempos 7, 8 e 10. Esta informação fica disponível de forma permanente, replicada em outros nós da rede se o nó do restaurante não puder estar presente. Utilizando a UbiVM em conjunto com o Ubiservices, Alex não precisou pesquisar exaustivamente pelo que queria, ou mesmo ligar para obter essas informações. Tudo pôde ser feito através de seu computador ou PDA do tempo 11 em diante.

Alex, além de utilizar a mesma estrutura que Jorge utilizou, ainda pode aproveitar-se das informações compartilhadas por seus colegas, como Jorge, em grupos de interesse distribuído, economizando tempo e encontrando de forma mais otimizada o que buscava.

6.3 Redes Sociais

O cenário de redes sociais é voltado para interesses gerais, indo desde opiniões e gostos pessoais até discussões, fotos, vídeos e outras formas de interação possíveis. A idéia é que pessoas que estejam em um mesmo contexto ou ambiente físico possam compartilhar interesses sociais e estabelecer uma comunicação direta de forma fácil, após registrarem suas informações junto à aplicação ubíqua.

Para tirar um maior proveito dos LBS (*Location Based Services*) [Vaughan-Nichols 2009], além de utilizar uma conexão a uma rede tecnológica, pode-se conectar as pessoas fisicamente presentes em um local em uma rede social [Costa 2009]. Fazer parte de uma rede social já é processo usual em muitos serviços que usamos diariamente, tais como *Orkut*, *Facebook*, *LinkedIn*, *Twitter*, *Wikipedia*, *digg*, *YouTube* e *flickr*.

Além da construção de espaços virtuais e do estímulo a auto-expressão, as redes sociais tem alterado nosso estilo de comunicação, apesar de manter os princípios da interação social humana [Kleinberg 2008]. Por causa disso, nada mais natural, e, portanto mais próximo

do conceito de computação ubíqua, do que aliar a interação ubíqua e a oferta de serviços baseados em localização com a formação de uma rede social espontânea, ou seja, uma rede social formada por pessoas presentes em locais específicos utilizando dispositivos móveis [Mani 2009]. As redes sociais são estruturas formadas por indivíduos, pessoas ou organizações, que são conectadas por algum tipo de interdependências, tais como interesses comuns, relações de amizade, etc. [Watts 2003].

6.3.1 O cenário

O cenário proposto por [Garzao 2010] é "Relacionamento entre pessoas e serviços na visita ao parque". O parque disponibiliza serviços e conteúdos, além de proporcionar a interação entre as pessoas fisicamente presentes no local. O parque disponibiliza conteúdos como roteiros de caminhadas e localização dos seus serviços. Como serviços, o parque oferece um sistema para reserva de vagas (estacionamento, time de basquete e time de futebol) e um sistema de reserva de canchas e quadras. Um biólogo disponibiliza o serviço de visita assistida ao parque. A medida que a pessoa percorre o parque, através da informação de localização, este serviço disponibiliza informações descrevendo a fauna e a flora daquele local.

Além disso, pessoas que estejam fora do parque podem se conectar e diferentes pontos de observação através de câmeras disponibilizadas na Internet. Com isso, a interatividade não só entre aqueles presentes no local pode existir, como destes para quem está conectado via grupos de interesse distribuído relacionados ao parque.

A possibilidade de realizar a reserva de “itens” no parque, como uma vaga e estacionamento, ou um lugar no time de basquete, torna-se possível de forma natural através do uso da UbiVM com o Ubiservices. Em um cenário remoto, a reserva de uma vaga de estacionamento pode ser feita antes mesmo antes de se chegar ao local, ou ainda antes de sair de casa através dos grupos de interesse distribuídos fornecidos pelo parque.

Por outro lado, já dentro do parque, a visibilidade de que uma partida vai acontecer em determinada hora e o fato de poder reservar um lugar no time de futebol ou basquete, por exemplo, se torna fácil com o uso dos grupos de interesses locais fornecidos pela UbiVM com o Ubiservices.

Tabela 4. Cenário de Redes Sociais

Tempo	Personagem	Ações
1	Parque	Executa a aplicação "Gerência do Parque".
1	PDA do Parque	UbiVM**, através do Ubiservices**, disponibiliza no contexto parque os conteúdos "Roteiros de caminhada" e "Localização

		de serviços", e os seguintes serviços, nos grupos abaixo: - Reserva de vagas (estacionamento, time de basquete e time de futebol) - Reserva de quadras e canchas **Nos grupos de interesse local: "parque A", "reserva de vagas parque A"** **E nos grupos de interesse distribuído: "reserva de vagas a distância parque A"**
2	Thiago	Executa a aplicação "Visita Assistida no Parque".
2	PDA do Thiago	UbiVM, **através do Ubiservices**, disponibiliza o serviço "Visita assistida" e informações sobre ele no contexto parque.
3	Alex	Antes de sair de sua casa, acessa via seu PDA **os serviços de grupo de interesse distribuídos** e reserva uma vaga no estacionamento.
3	PDA do Alex	UbiVM**, através do Ubiservices,** utiliza o serviço "Reserva de vagas no estacionamento", existente no **grupo de interesse distribuído do** parque.
4	PDA do Parque	UbiVM, **através do Ubiservices**, recebe a solicitação do serviço, vinda do PDA do Alex. Envia o número da vaga disponível.
5	PDA do Alex	UbiVM, **através do Ubiservices**, recebe o número da vaga, exibindo um aviso para Alex.
5	Alex	Visualiza, **se dirige para o parque e estaciona o carro na vaga informada.** Dirige-se para a pista de caminhada.
6	Gabriel	Executa a aplicação que reserva uma vaga no time de basquete.
6	PDA do Gabriel	UbiVM, **através do Ubiservices**, utiliza o serviço "Reserva de vagas no time de basquete", existente no contexto parque.
7	PDA do Parque	UbiVM, **através do Ubiservices**, recebe a solicitação do serviço, vindo do PDA do Gabriel. Como existem vagas em um time, faz uma reserva e envia a informação com o horário do jogo.
8	PDA do Gabriel	UbiVM, **através do Ubiservices**, recebe as informações sobre a reserva, e gera um aviso para Gabriel.
8	Gabriel	Visualiza as informações sobre o horário, dirigindo-se para a quadra.
9	Graciele	Executa a aplicação que exibe informações sobre o serviço "Visita assistida", ativando-o.
9	PDA da Graciele	UbiVM, através do Ubiservices, solicita informações sobre o serviço "Visita assistida", existente no contexto parque.
10	PDA do Thiago	UbiVM, através do Ubiservices, recebe a solicitação vinda do PDA da Graciele, e envia as informações sobre o serviço.
11	PDA da Graciele	UbiVM, através do Ubiservices, recebe as informações sobre o serviço, e gera um aviso para Graciele.
11	Graciele	Visualiza as informações e ativa o serviço. Graciele move-se pelo parque em busca dos pontos interessantes para serem visitados.
12	PDA da Graciele	**Sempre que ocorre uma mudança na localização física, a UbiVM, através do Ubiservices, envia a nova localização para o serviço "Visita assistida".**
12	Graciele	**Jorge, namorado de Graciele, não pôde ir ao parque junto com ela por estar machucado, mas ficou em casa e quer acompanhar o passeio. Isto se torna possível pois, a cada local que Graciele chega, ela compartilha a informação da sua localização no seu grupo pessoal de interesse distribuído, permitindo que Jorge acesse as câmeras disponíveis no parque e acompanhe o passeio de forma parcial com Graciele.**
12	PDA da Graciele	Publica a informação de sua localização no seu **grupo**

		pessoal distribuído, de nome “Graciele”
13	Dispositivo do Jorge	Buscando as informações contidas no **grupo de interesse distribuído de Graciele**, lista as suas atualizações.
13	Jorge	Recebe a informação de onde Graciele se encontra e acessa a câmera do parque correspondente. Opcionalmente, conversa com Graciele ao telefone (de forma completamente independente da UbiVM/Ubiservices).
14	PDA do Thiago	UbiVM, **através do Ubiservices, recebe a localização e verifica que não existem informações específicas nesta localização**. Informa isso ao PDA da Graciele.
15	PDA da Graciele	UbiVM, através do Ubiservices, recebe a resposta de que não existem informações específicas nesta localização.
15	Graciele	Move-se novamente tentando encontrar os pontos interessantes.
16	PDA da Graciele	UbiVM, através do Ubiservices, envia a nova localização para o serviço "Visita assistida".
17	PDA do Thiago	UbiVM, através do Ubiservices, recebe a localização e verifica que existem informações específicas nesta localização. Informa isso ao PDA da Graciele.
18	PDA da Graciele	UbiVM, através do Ubiservices, recebe as informações específicas desta localização, e gera aviso para Graciele.
18	Graciele	Visualiza as informações.
19	Alex, Graciele, Gabriel e Jorge	Fim do passeio.

6.3.2 Discussão

Diferentes interesses são apresentados no cenário de Redes sociais. A maior vantagem de um ambiente ubíquo dentro de uma rede social é a publicação e captação instantânea das informações, tornando possível a uma pessoa compartilhar conteúdos e interesses com pessoas com quem nunca conversou antes, e de forma direcionada encontrá-la pessoalmente depois de uma pesquisa de alguns segundos.

O cenário da seção 6.3.1 mostra a facilidade possível para se conseguir uma vaga no estacionamento, antes mesmo de sair de casa, nos tempos 3, 4 e 5 com Alex, ou a reserva de uma vaga no time de basquete nos tempos 6, 7 e 8. Alex fez uso do grupo de interesse distribuído disponibilizado pelo parque para conseguir acessar a informação de quais vagas estavam disponíveis e, através do serviço fornecido para o mesmo grupo, realizar a reserva de uma vaga.

Estas tarefas, atualmente, ainda precisam ser feitas via telefone ou, no melhor dos casos, através do *web site* da empresa, de forma não interativa. Na realidade atual, para realizar a operação através de um PDA ou *smartphone*, depende-se de uma conexão a Internet de uma operadora de telefonia. Se for um parque afastado da cidade, a probabilidade de conseguir fazer as reservas via um *smartphone* é baixa.

Além disso, o cenário da seção 6.3.1 mostra como Graciele pôde compartilhar seus momentos no parque com Jorge nos tempos 12 e 13, algo que sem o uso da UbiVM com o

Ubiservices e uma infra-estrutura ubíqua presente no parque seria difícil de acontecer, pelo menos não de uma forma natural como a mostrada na Tabela 4.

7 CONSIDERAÇÕES FINAIS

Este capítulo traz uma reflexão sobre as principais contribuições do Ubiservices para a área de Administração de Serviços em ambientes ubíquos com redes *Ad Hoc*, além de mostrar as próximas etapas.

7.1 Conclusões

O trabalho de pesquisa sobre tecnologias existentes se mostrou detalhado, citando referências a diversos *middlewares* e tecnologias atuais, gerando as idéias para o modelo. Este, por sua vez, para ser implementado precisou de ambientes existentes como o OLSRd e o Gnunet. Apesar de ambos serem softwares bem estruturados e reconhecidos nos seus respectivos nichos, a integração das tecnologias foi complexa. Fica a certeza de que o modelo é interessante e válido, mas com a necessidade de um estudo mais profundo para uma adequada implementação.

Os testes funcionais com UOP acabaram sendo prejudicados pela falta de um protótipo totalmente funcional, mas as idéias dos conceitos propostos pelo Ubiservices mostraram-se válidas, uma vez que a implementação do UOP [Garzão 2010] superou as expectativas, gerando resultados no seu escopo, e deixando o a base operacional pronta para uma adaptação do seu módulo de comunicação para fazer uso de outro modelo de rede, como o Ubiservices.

7.2 Contribuições

Atualmente existem soluções adequadas para a administração de serviços no escopo de redes locais ou na Internet. No entanto, quando se trata de um ambiente ubíquo *Ad Hoc*, na maioria das vezes pouca estrutura está disponível para os usuários. Conseqüentemente, os recursos ficam subutilizados devido ao isolamento a que um usuário é submetido por não possuir, por exemplo, um link permanente com a Internet ou o fornecimento de serviços que satisfaçam as suas necessidades computacionais locais.

O Ubiservices trata deste tipo de questão com o fornecimento compartilhado de serviços entre redes locais e a interconexão das mesmas através de uma rede *overlay*, permitindo não só que usuários que estariam isolados se comuniquem efetivamente, mas também que usuários que já possuam um *link* estável para a Internet possam entrar na mesma rede, usufruir e compartilhar serviços.

Dentre a bibliografia pesquisada, esta característica de interconexão massiva foi pouco explorada. E justamente por isso, esta proposta traz a integração de usuários e serviços com a sua maior contribuição. Esteja o usuário onde estiver, poderá de alguma forma se integrar na rede para compartilhamento de serviços.

Com a migração e replicação de serviços e informações, a continuidade no fornecimento dos serviços oferecidos por um nó é garantida dentro de ambientes instáveis como redes *Ad Hoc*. Além disso, a validação do modelo fazendo uso de UOP garante que o ambiente suporte adequadamente as especificações de ubiqüidade, como mobilidade de código e suporte a contextos [Satyanarayanan 2001], e não seja apenas uma adaptação parcial de um modelo existente para funcionar em um ambiente ubíquo. Por sua vez, os grupos locais e de interesse fornecem um escopo definido para cada rede *mesh* local e, dentro da rede *overlay*, interesses específicos também podem ser identificados através dos grupos.

Tecnologias complexas foram exploradas e estudadas, como o OLSRd e o Gnunet. O OLSR permite formar redes *mesh* com relativa facilidade, e tem um modularização versátil, que está pronta para receber qualquer tipo de aplicação através das suas *libs*, mas exige estudo aprofundado. O Gnunet se enquadra em outra categoria de rede, em um nível de aplicação, mas também com código aberto, como o OLSR, permite a adição de qualquer módulo através das *applications*.

Por fim, uma das contribuições deste trabalho foram os cenários detalhados no capítulo 6. Tendo como base os cenários criados por [Garzão 2010], se procurou expandi-los para além de cenários com escopos estritamente locais, fazendo uso dos grupos de interesses locais em conjunto com os grupos de interesses distribuídos. Foram criadas situações em que tanto usuários remotos puderam acessar informações e serviços de um local definido (o laboratório, restaurante ou o parque), como também interagir com os usuários que estavam presentes nestes locais.

Foram levantados pontos importantes que se na integração de interesses locais e distribuídos:

- Acesso a informações de qualquer lugar;
- Interação total entre usuários locais e remotos;
- Compartilhamento de informações e imagens (com alguma infra-estrutura como no cenário 3 do capítulo 6);
- Acesso ágil as informações já conhecidas de usuários fisicamente próximos.

7.3 Trabalhos futuros

O Ubiservices modelado no capítulo 4 é um trabalho ainda incompleto e que pode gerar bons resultados e comparativos quando devidamente alinhado com a implementação do UOP realizada por Alex Garzão [Garzão 2010]. Pode-se também citar como trabalhos futuros identificados:

- A utilização do OLSRd e do Gnunet podem ser questionadas e comparadas com outras plataformas de rede *mesh* e *peer to peer* existentes, tanto em termos de aplicabilidade ao modelo como em termos de métricas quantitativas para medição de testes funcionais;
- Uma gama de serviços adicionais podem ser criados sobre uma implementação totalmente funcional do modelo, indo além dos serviços apresentados de replicação, migração, busca, publicação, armazenamento local e grupos;
- A criação de simulações tanto para a rede *mesh* como o Gnunet, criando representações dos cenários descritos no capítulo 6, e expandindo-os para dezenas ou centenas de nós de forma sistemática;
- Questões de segurança podem ser incluídas na especificação do modelo e, conseqüentemente, nos aspectos de implementação;
- O conceito dos grupos de interesses locais e distribuídos pode ser expandido abrangendo permissões de entrada e saída do grupo dos demais usuários, permissões de execução e controle sobre o compartilhamento de informações;
- Cenários que demandem mais interação tanto localmente como remotamente (através dos grupos de interesses distribuídos) podem ser criados e estudos específicos de cada cenário podem levar a necessidades de serviços diferenciados por parte do Ubiservices;

Referências Bibliográficas

Adamson, B.; Bormann, C.; Handley, M.; Macker, J. **Multicast Negative-Acknowledgement (NACK) Building Blocks**. Internet Engineering Task Force (IETF) RFC 5401, November 2008.

Akyildiz, I. F.; Wang, X.; Wang, W. **Wireless mesh networks: a survey, Computer Networks**, vol. 47, no. 4, pp. 445–487, March 2005.

Apple Inc. **Bounjour Tecnology**. Disponível em http://www.apple.com/macosx/technology /bonjour.html. Ultimo acesso em junho de 2009.

Augustin, I.; et al. ISAM, **Joing Context-awareness and Mobility to Building Pervasive Applications**. Mobile Computing Handbook. New York, CRC Press, páginas. 73-94, 2004.

Austin, Daniel; et al. **Web services architecture requirements**. W3C working draft, 2002.

Barbosa, J.; et. al. **Holoparadigm: a multiparadigm model oriented to development of distributed systems**. International Conference on Parallel and Distributed Systems, IEEE Press, 2002.

Barbosa, J.; et al. **Local: Um modelo para suporte à aprendizagem consciente de contexto**. Simpósio Brasileiro de Informática na Educação (SBIE), 2006.

Barbosa, Jorge L. V. ; Hahn, Rodrigo ; Bonnato, Daniel ; Cecin, Fábio R.; Geyer, Cláudio F R. **Evaluation of a Large Scale Ubiquitous System Model through Peer-to-Peer Protocol Simulation**. In: 11th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT´07), Chania. Proceedings of 11th IEEE DS-RT. Los Alamitos : IEEE Computer Society. p. 175-181, 2007.

Barbosa, J.; et al. **Local: a model geared towards ubiquitous learning**. SIGCSE Bull., ACM, New York, NY, USA, v. 40, n. 1, p. 432-436, 2008.

Beatty, John; Kakivaya, Gopal ; Kemp, Devon; et al. **Web Services Dynamic Discovery specification (WS-Discovery)**. Microsoft Corp, April 2005.

Beigl, M.; Zimmer, T.; Krohn, A.; Decker, C.; Robinson, P. **Creating Ad-Hoc Pervasive Computing Environments**, presented at Second International Conference on Pervasive Computing, Linz/Vienna, Austria, 2004.

Bennett, Krista; et al. **Gnet (Gnunet first complete whitepaper)**. Disponível em http://gnunet.org. Ultimo acesso em maio de 2009.

Bisignano, M.; Calvagna, A.; Modica, G. Di; Tomarchio, O. **Design and development of a Jxta middleware for mobile ad-hoc networks**. Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks, 2004.

Bisignano, M.; Calvagna, A.; Modica, G. Di; Tomarchio, O. **Expeerience: a Jxta middleware for mobile ad hoc networks**, in proc. third international conference on P2P computing, 2003.

Bisignano, M.; Modica, G. Di; Tomarchio, O. **JMobiPeer: A Middleware for Mobile Peer-to-Peer Computing in MANETs**. Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops, p. 785 – 791, 2005.

Bouillet, Eric; Feblowitz, Mark; Liu, Zhen; Ranganathan, Anand; Riabov, Anton. **Semantic Models for Ad Hoc Interactions in Mobile**, Ubiquitous Environments. ICSC 2008.

Chakraborty, D.; et al. **Toward Distributed Service Discovery in Pervasive Computing Environments**. IEEE Transactions on Mobile Computing, Vol. 5, No. 2, February 2006.

Cheshire, Stuart; Krochmal, Marc. **DNS-Based Service Discovery**. IETF Internet Draft, September 2008 (b).

Cheshire, Stuart; Krochmal, Marc. **Multicast DNS**. IETF Internet Draft, September 2008 (a).

Clausen, T.; Jacquet, P. **Optimized Link State Routing Protocol**. Project Hipercom, INRIA. IETF RFC 3626, October 2003.

Costa, C. A. da. **Mingle: Gerenciamento de interação ubíqua em uma rede social espontânea**. Projeto de pesquisa em andamento na Universidade do Vale do Rio dos Sinos (UNISINOS), 2009.

Dey, A. K. **Understanding and using context**. Personal Ubiquitous Computing, 2001.

Edwards, W.K. **Discovery systems in ubiquitous computing**, Pervasive Computing, IEEE , vol.5, no.2, pp. 70-77, April-June 2006.

Fok, C. L.; Roman, G. C.; Hackmann, G. **A Lightweight Coordination Middleware for Mobile Computing**, in proc. 6th Intern'l Conf. on Coordination Models and Language, De Nicola, R., Ferrari, G., and Meredith, (editors), Lecture Notes in Computer Science 2949, Springer-Verlag, Pisa, Italy, pp. 135-151, Feb. 2004.

Franco, L. K. et al. **Um modelo para exploração de oportunidades no comércio ubíquo**. XXXV Conferência Latino Americana de Informática (CLEI), p. 1-10, 2009.

Freifunk. **Freifunk site**. Disponível em http://start.freifunk.net. Ultimo acesso em maio de 2009.

Fuggetta, A., Picco, G. P.; Vigna, G. **Understanding code mobility**. Software Engineering, IEEE Transactions on, 24(5):342–361, 1998.

Galanxhi-Janaqi, H.; Nah, F. F.-H. **U-commerce: emerging trends and research issues**. Industrial Management and Data Systems, v. 104, n. 9, p. 744-755, 2004.

Garlan, David; Siewiorek, Daniel P.; Steenkiste, Peter. **Project Aura: Toward Distraction-Free Pervasive Computing**. IEEE Pervasive Computing, 2002.

Garzão, Alex S. **UbiModel: Um Modelo de Programação orientado ao Desenvolvimento de Sistemas Ubíquos**. Dissertação de mestrado. Unisinos. Fevereiro, 2010.

Gassanenko, M. L. **Context-oriented programming**. 1998. Disponível em: http://www.complang.tuwien.ac.at/anton/euroforth/ef98/gassanenko98a.pdf. Ultimo acesso em junho de 2009.

Gelertner, D. **Generative Communication in Linda**, ACM Computing Surveys, Vol 7, No 1, pp.80-112, Jan. 1985.

Gershman, A. **Ubiquitous commerce - always on, always aware, always pro-active**. In: SAINT. [S.l.: s.n.], p. 37-38, 2002.

Goland, Yaron Y.; Cai, Ting; Leach, Paul; Gu, Ye. **Simple Service Discovery Protocol**. IETF Internet Draft, October 1999.

Grimm, Robert; Davis, Janet; Hendrickson, Ben; et al. **Programming for pervasive computing environments**. In Proceedings of the 18th ACM Symposium on Operating Systems Principle, Chateau Lake Louise, Banff, Canada., October 2001.

Hadim, S.; Al-Jaroodi, J.; Mohamed, N. **Trends in Middleware for Mobile Ad Hoc Networks**, invited paper in the Journal of Communications, 2006.

Harrison, W.; Ossher, H. **Subject-oriented programming: a critique of pure objects**. In: OOPSLA '93: Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications. ACM Press, v. 28, n. 10, 1993.

Helal, S.; Desai, N.; Verma, V.; Konark, C. Lee. **A Service Discovery and Delivery Protocol for Ad-hoc Networks**. In Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC), New Orleans, USA, March 2003.

Herrmann, Klaus. **MESHMdl - A Middleware for Self- Organization in Ad hoc Networks**. in Proceedings of the 1st International Workshop on Mobile Distributed Computing (MDC'03), May 19 2003.

Herrmann, Klaus; Muhl, Gero; Jaeger, Michael A. **MESHMdl event spaces - A coordination middleware for self-organizing applications in ad hoc networks**, Pervasive and Mobile Computing. Volume 3, Issue 4, Pages 467-487, Middleware for Pervasive Computing, August 2007.

Hightower, J.; LaMarca, A.; Smith, **I.E. Practical Lessons from Place Lab**, Pervasive Computing, IEEE , vol.5, no.3, pp.32-39, July-Sept. 2006.

IETF. **IETF: The Internet Engineering Task Force**. Disponível em http://www.ietf.org. Ultimo acesso em junho de 2009.

Johnson, D.; Maltz, D.A.; Broch, J. **The dynamic source routing protocol for mobile ad hoc networks (Internet-draft)**, in: Mobile Adhoc Network (MANET) Working Group, IETF, 1998.

Juszczyk, L.; Dustdar, S. **A middleware for service-oriented communication in mobile disaster response environments**. In Proceedings of the 6th international Workshop on Middleware For Pervasive and Ad-Hoc Computing. ACM, New York, NY, 2008.

Kindberg, T.; Barton, J. **A Web-based nomadic computing system**. Computer Networks (Amsterdam, Netherlands: 1999), v. 3, n. 4, p. 443–456, 2001.

Kleinberg, J. **The convergence of social and technological networks**. Commun. ACM, ACM, New York, NY, USA, v. 51, n. 11, p. 66-72. ISSN 0001-0782, 2008.

Krebs, Martin; Krempels, Karl-Heinz; Kucay, Markus. **Service Discovery in Wireless Mesh Networks**. IEEE WCNC proceedings, 2008.

Leach, P.; Mealling, M.; Salz, R. **A Universally Unique IDentifier (UUID) URN Namespace**. IETF RFC 4122, July 2005.

Levis, P.; Culler, D. **Mate: A Tiny Virtual Machine for Sensor Networks**, in proc. Inter. Conf. on Architectural Support for Programming Languages and Operating Systems, Oct. 2002.

Lyytinen, K.; Yoo, Y. **Issues and challenges in ubiquitous computing**. New York, NY, USA: ACM, December, 2002.

Mani, M.; Ngyuen, A. M.; Crespi, N. **What's up: P2P spontaneous social networking**. Pervasive Computing and Communications, IEEE International Conference on, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 1-2, 2009.

Mascolo, L.; Capra, S.; Zachariadis, W. Emmerich. **XMIDDLE: A Data-Sharing Middleware for Mobile Computing**, in Wireless Personal Com., Kluwer. 21. pp. 77-103. 2002.

Meier, R.; Cahill, V. **STEAM: Event-based middleware for wireless ad hoc networks**, in the 22nd Intern. Conf. On Distributed Computing Systems Workshops (ICDCSW '02), Austria, July 2002.

Murphy, A. L.; Picco, G. P.; Romjan, G. C. **Lime: A coordination middleware supporting mobility of hosts and agents**. Technical Report WUCSE-03-21, Washington University, Department of Computer Science, St. Louis, MO (USA), 2003.

Murphy, L.; Picco, G. P.; Roman, G. C. **Lime: A Middleware for Physical and Logical Mobility**, in proc. of the 21st Intern. Conf. On Distributed Computing Systems (ICDCS-21), May 2001.

Naseen, M. K.; et al. **Implementing strong code mobility**. Information Technology Journal, pages 188–191, 2004.

Papazoglou, M.; Georgakopoulos, D. **Introduction: Service-oriented Computing**. Communications of the ACM, n.46, v.10, p.24-28, October 2003.

Perkins, C.E.; Royer, E.M. **Ad hoc on demand distance vector (AODV) routing (Internet-draft)**, in: Mobile Ad-hoc Network (MANET) Working Group, IETF, 1998.

Poettering, Lennart; Lloyd, Trent. **Avahi Zeroconf implementation**. Disponível em http://avahi.org. Ultimo Acesso em junho de 2009.

Rogers, Y; et al. **Ubi-learning integrates indoor and outdoor experiences**. Communications of the ACM, ACM Press, v. 48, n. 1, p. 55-59, January 2005. Disponível em http://eprints.ecs.soton.ac.uk/11790.

Román, M. et al. **Gaia: A Middleware Infrastructure to Enable Active Spaces**. IEEE Pervasive Computing, Oct-Dec 2002.

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

Satyanarayanan, M. **Pervasive computing: vision and challenges**. Personal Communications, IEEE, Volume 8, Issue 4, Pages:10 - 17, Aug. 2001.

Schmidt, A.; et al. **Real-world challenges of pervasive computing**. IEEE Pervasive Computing, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 5, n. 3, p. 91–93, c3, 2006.

Shankar, Chetan; Al-Muhtadi, Jalal; Campbell, Roy; Mickunas, M. Dennis. **Mobile Gaia: A Middleware for Adhoc Pervasive Computing**, IEEE Consumer Communications & Networking Conference (CCNC 2005), Las Vegas, Jan. 2005.

Sun Microsystems, **CDC: An Application Framework for Personal Mobile Devices**. http://java.sun.com/products/cdc/wp/cdc-whitepaper.pdf, May 8, 2006; June 2005.

Syvanen, A.; Beale, R.; Sharples, M.; Ahonen, M.; Lonsdale, P. **Supporting pervasive learning environments: adaptability and context awareness in mobile learning**. Wireless and Mobile Technologies in Education, 2005.

Tonnesen, Andreas; et al. **UniK OLSR Daemon (OLSRD)**. Institute for informatics at the University of Oslo (Norway). Disponível em http://www.olsr.org. Ultimo acesso em junho de 2009.

Vaughan-Nichols S. J. Will Mobile. **Computing's Future Be Location, Location, and Location?** Computer, 42:14-17, 2009.

Watts, D. J. **Six Degrees: The Science of a Connected Age**. [S.l.]: W. W. Norton & Company. Paperback. ISBN 0393325423, 2003.

Web Services Dynamic Discovery (**WS-Discovery**) April 2005

Weiser, M. **The Computer for the 21st Century**. Sci. Amer., Sept. 1991.

Yamin, A. C. **Arquitetura para um Ambiente de Grade Computacional Direcionado às Aplicações Distribuídas, Móveis e Conscientes do Contexto da Computação Pervasiva**. Tese apresentada como requisito parcial para obtenção do grau de Doutor em Ciência da Computação, 195 pág., UFRGS, Porto Alegre, 2004.

Yau, S. S.; et al. **Smart classroom: Enhancing collaborative learning using pervasive computing technology**. In: In ASEE 2003 Annual Conference and Exposition. [S.l.: s.n.], p. 13633-13642, 2003.