UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Omnipresent - Um Sistema Ciente de Contexto baseado em Arquitetura Orientada a Serviço

Damião Ribeiro de Almeida

(Mestrando)

Cláudio de Souza Baptista, PhD

(Orientador)

Campina Grande – PB
Abril de 2006

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Omnipresent - Um Sistema Ciente de Contexto baseado em Arquitetura Orientada a Serviço

Damião Ribeiro de Almeida

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal de Campina Grande, como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Sistemas de Informação e Banco de Dados

Orientador: Cláudio de Souza Baptista, PhD

Campina Grande – PB
Abril de 2006

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

A447o 2006

Almeida, Damião Ribeiro de
Omnipresent- Um sistema ciente de contexto baseado em arquitetura orientada/ Damião Ribeiro de Almeida. ─ Campina Grande, 2006.
93f.: il.

Referências.
Dissertação (Mestrado em Informática) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Orientador: Cláudio de Souza Baptista.

1─ Aplicação de Redes Gerais 2─ Geoprocessamento 3─ Banco de Dados Móveis 4─ Aplicação Ciente do Contexto I─ Título

CDU 004.77

"OMNIPRESENT – UM SISTEMA CIENTE DE CONTEXTO BASEADO EM ARQUITETURA ORIENTADA A SERVIÇO"

DAMIÃO RIBEIRO DE ALMEIDA

DISSERTAÇÃO APROVADA EM 16.05.2006

PROF. CLÁUDIO DE SOUZA BAPTISTA, Ph.D
Orientador

PROF. ULRICH SCHIEL, Dr.
Orientador

PROFª VALÉRIA CESÁRIO TIMES, Ph.D
Examinadora

CAMPINA GRANDE – PB

“Penso noventa e nove vezes e nada descubro; deixo de pensar, mergulho em profundo silêncio – e eis que a verdade me é revelada.”

(Albert Einstein)

Sumário

Abreviações

ADT	Abstract Data Type
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASR	Area Service Register
BNF	Backus-Naur Form
CHTML	Compact HTML
CRS	Coordinate Reference System
DAML+OIL	DARPA Agent Markup Language + Ontology Integration Language
ECG	Eletrocardiograma
GMLC	Gateway Mobile Location Center
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HP	Hewlett-Packard
HTTP	HyperText Transfer Protocol
JAX-RPC	Java API for XML-based Remote Procedure Call
JPEG	Joint Photographic Experts Group
KB	Knowledge Base
LBS	Location Based Service
LDT	Location-Determining Technologies
MIDP	Information Device Profile
MPC	Mobile Positioning Center
MVC	Model-View-Controller
OGC	Open Geospatial Consortium
OpenLS	OpenGIS Location Services
OWL	Web Ontology Language
PC	Personal Computer
PDA	Personal Digital Assistants
PHP	Hypertext Preprocessor
PHS	Personal Handphone System
PoI	Point of Interest
RDF	Resource Description Framework

RDFS	RDF Schema
RGB	Red, Green and Blue
RMI	Remote Method Invocation
SIG	Sistema de Informação Geográfica
SRS	Sistema de Referência Espacial
SOAP	Simple Object Access Protocol
SVG	Scalable Vectorial Graphics
SVGT	Scalable Vectorial Graphics Tiny
UDDI	Universal Description, Discovery and Integration
URI	Universal Resource Identifier
URL	Universal Resource Locator
XHTML	eXtensible Hypertext Markup Language
XML	eXtensible Markup Language
WFS	Web Feature Service
WMS	Web Map Service
WSDL	Web Services Description Language
X11	X Window System

Lista de Figuras

Lista de Tabelas

Resumo

O aumento no uso de sensores e dispositivos móveis, tem proporcionado o crescimento de aplicações capazes de perceberem o ambiente em que se inserem e oferecerem serviços mais personalizados aos usuários. Essas aplicações, chamadas de aplicações cientes de contexto, proporcionam maior comodidade, tornando as atividades diárias mais simples e melhorando a qualidade de vida dos usuários.

Construir aplicações capazes de perceber o ambiente em que o usuário está envolvido traz uma série de desafios. É preciso fazer uso de sensores para perceber o ambiente e transmitir os dados para um sistema remoto, o qual deve analisar e executar alguma ação de interesse do usuário.

O usuário está sempre se movendo, alterando suas condições físicas e fisiológicas. Devido a essa complexidade do contexto, torna-se importante construir um sistema capaz de inferir sobre os dados obtidos do ambiente e que possa apresentar alguma informação com base nesse novo conhecimento.

Este trabalho propõe uma arquitetura distribuída, baseada em dispositivos móveis e orientada à serviços, capaz de monitorar o contexto do usuário e ativar mensagens com base nas mudanças do contexto. Um usuário poderá monitorar o seu próprio contexto, ou de seus parentes e amigos, de forma que ele possa ser alertado quando determinadas situações ocorrerem. Diferente de outras aplicações, essas situações são descritas através de regras que podem ser inseridas ou removidas a qualquer momento pelo usuário. Além disso, novas categorias de contexto podem ser adicionadas ao sistema em tempo de execução, devido ao uso de ontologia para descrever o contexto. A ontologia permite deduzir novas informações que não estão explícitas, mas que podem ser obtidas através de regras de inferência.

Abstract

The rise of sensors and mobile devices has demanded for applications which enable environmental monitoring and may offer personalized services to users. These applications are known as context aware ones. They provide more comfort, by simplifying daily activities and improving the quality of life.

The development of such context aware applications brings new challenges. It is necessary to use sensors to perceive the environment and transmit the data to a remote system, which should analyze and execute some action concerning user's interest.

The user is always moving, changing both physical and physiologic conditions. Due to the complexity of context, it seems to be mandatory the provision of a system with capabilities of inference on surrounding data. Also, the system should be able to present information based on this new knowledge.

This dissertation proposes a distributed architecture, based on mobile devices and service-oriented architecture, which enables the monitoring of user's context and may fire messages based on context changes. A user may monitor his own context, or relatives and friends ones. He may be alerted when certain conditions happen. Different from other applications, these conditions are described through rules that can be either inserted or removed by the user at any time. Furthermore, new context categories may be added to the system at runtime, because the context is described through an ontology. The ontology permits to deduce new information that is not explicit, but may be obtained through inference rules.

Capítulo 1 . Introdução

O crescimento do uso de dispositivos eletrônicos cada vez menores e com sua capacidade de processamento e armazenamento aumentando cada vez mais, permite que as pessoas possam carregar esses aparelhos a todo tempo, proporcionando o desenvolvimento de aplicações cada vez mais sofisticadas e que possam fornecer informações mais personalizadas.

Essa visão do futuro, na qual o poder computacional estará presente em qualquer lugar, a qualquer hora, executando tarefas de interesse dos usuários, é chamada de Computação Ubíqua ou Computação Pervasiva (*Pervasive Computing*) [Aug04] [GPZ04] [AYS04]. Segundo Weiser [Wei06], no futuro, os computadores estarão espalhados pelo ambiente físico, mas invisíveis ao usuário, deixando-o concentrar-se mais nas suas tarefas do que nas ferramentas. Nesta nova geração da computação, cada pessoa estará continuamente interagindo com centenas de computadores próximos, interconectados por uma rede sem fio. Para chegar a esse ponto, são necessários novos tipos de computadores de todos os tamanhos e formas para estarem disponíveis para cada pessoa. Hoje em dia, já existem vários dispositivos portáteis que permitem as pessoas se moverem enquanto utilizam algum recurso computacional e de rede. O principal objetivo da computação ubíqua é fornecer ao usuário um acesso imediato à informação e, transparentemente, permitir a execução de suas tarefas [Wei06]. Nesse novo futuro de uma sociedade informatizada, as pessoas estarão rodeadas por um ambiente eletrônico sensível às suas necessidades, personalizados aos seus requisitos e que respondem prontamente aos seus comportamentos, enfatizando o uso amigável, tornando a vida diária das pessoas mais confortável e melhorando a qualidade de vida [BB02]. O desenvolvimento da computação ubíqua nos levará a um mundo em que funcionalidades computacionais estarão em todos os tipos de objetos, que serão capazes de reconhecer e responder às necessidades individuais de cada um, por exemplo, um quarto de hotel que adapta a temperatura e a música ambiente de acordo com as preferências do cliente. Uma importante característica das aplicações desse tipo é a capacidade de reconhecerem o contexto do usuário de forma mais transparente possível, tornando os dispositivos eletrônicos do ambiente menos perceptíveis para o usuário.

Dessa forma, as aplicações obterão vantagens das características dinâmicas do ambiente para serem mais efetivas e adaptativas às necessidades de informação do usuário sem consumir muito de sua atenção [CK00].

A computação ubíqua provavelmente será a próxima geração de ambiente computacional, onde a tecnologia da informação e comunicação estará presente em toda parte e a toda hora. A tecnologia estará integrada à vida das pessoas e a vários produtos, como: carros, relógios de pulso, espalhados pela casa e pelas ruas. A primeira onda de tecnologia de informação foram os mainframes e os PCs, a segunda foi a Internet e a comunicação sem fio. A computação ubíqua tende a ser a terceira onda da tecnologia de informação [FAJ04]. Uma característica da computação ubíqua é a idéia de um único usuário usar muitos computadores. Hoje em dia, é possível notar o surgimento dessa tendência a partir do aumento no uso de celulares, PDA, notebooks, entre outros dispositivos que proporcionam acesso computacional em qualquer lugar.

Na era dos mainframes, havia muitas pessoas para um computador e na época dos computadores pessoais, a idéia era uma pessoa para um computador. A tendência atualmente é a existência de vários computadores para um único usuário. A Figura 1.1 apresenta um gráfico que mostra o crescimento dessa nova tendência.

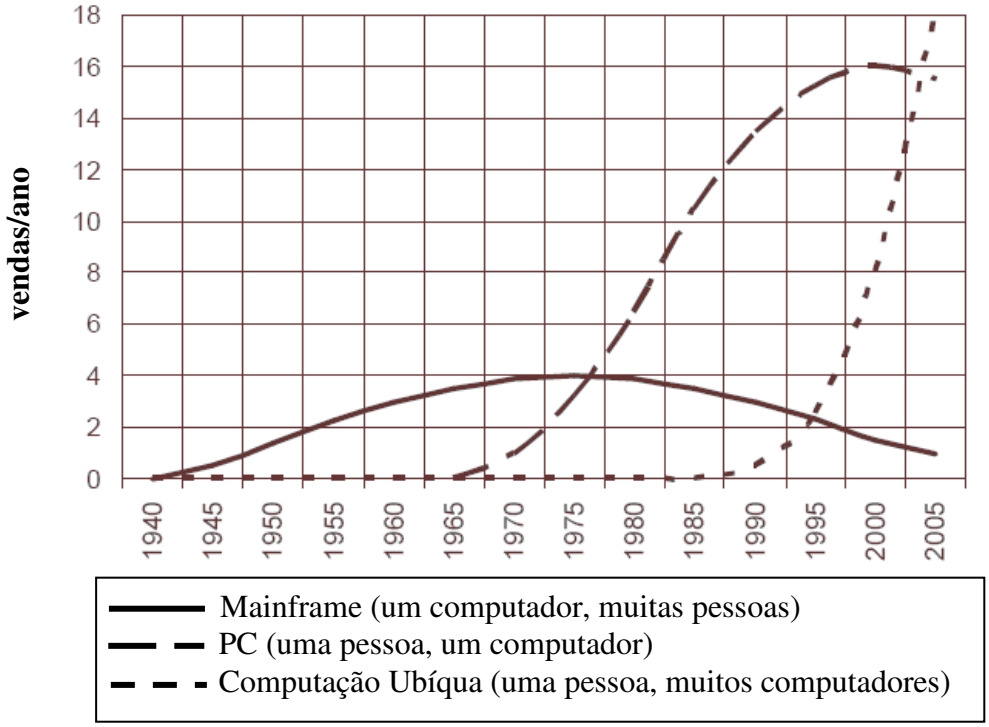

Figura 1.1 A nova tendência em computação (extraído de [ABR04])

Esse novo mundo com computadores espalhados em toda parte tem proporcionado o surgimento de um novo tipo de aplicação, que são as aplicações cientes de contexto. Contexto é o conjunto de estados do ambiente que determina o comportamento de uma aplicação [CK00]. Se uma informação pode ser usada para caracterizar a situação de um participante em uma interação com o serviço, então essa informação é contexto.

As informações de contexto podem ser dividas em:

- contexto do usuário – é toda informação relacionada ao usuário, como: emoção, estado fisiológico, agenda, perfil, etc;
- contexto do ambiente – são informações relativas ao ambiente, como: localização, temperatura, pessoas e objetos ao redor, dispositivos computacionais próximos, entre outros.

A mobilidade do usuário cria situações nas quais o contexto é mais dinâmico, fazer com que o usuário passe todas as informações do contexto para o computador a cada situação é difícil e cansativo. Dessa forma, o computador deve coletar informações do contexto automaticamente e com mínima intervenção do usuário. Algumas informações do contexto podem ser obtidas de forma transparente através do uso de sensores e outras podem ser deduzidas a partir de outros meios, como uma agenda eletrônica que informa a atividade do usuário.

1.1 Objetivos

O principal objetivo desse trabalho é desenvolver um sistema que seja capaz de reconhecer o contexto do usuário e do ambiente ao seu redor para oferecer serviços personalizados de forma transparente ao usuário.

O sistema deverá apresentar as seguintes características:

- analisar o contexto do usuário para fornecer serviços personalizados, auxiliar a monitorar seu contexto e o de outras pessoas, como por exemplo, monitorar os batimentos cardíacos de seus parentes, lembrar de ligar para um amigo quando ele sair do trabalho, avisar quando seu filho deixa a escola, entre outros. Esse sistema de monitoração funciona como um alarme que é ativado de acordo com as mudanças do contexto do usuário e do ambiente;
- analisar o contexto do ambiente. Esse tipo de contexto apresenta informações que auxiliarão a ferramenta a descobrir o tipo de ambiente em que o usuário está inserido e personalizar o serviço de acordo com as condições desse ambiente. Por

exemplo, apresentar mapas no seu dispositivo que sejam de acordo com sua localização, mostrar os seus amigos próximos de sua posição geográfica e localizar produtos ou serviços de interesse que foram especificados previamente.

Uma característica importante desse sistema é que ele é baseado em Web services. Dessa forma, ele será composto por um conjunto de serviços heterogêneos que poderão executar tarefas em paralelo de forma padronizada e transparente ao usuário. Por exemplo, um servidor de roteamento poderá consultar um serviço de condições do tráfego para encontrar a melhor rota entre o usuário e um ponto de interesse (PoI).

1.2 Motivação

Sistemas de análise de contexto tornam a vida do usuário mais confortável, melhorando a sua qualidade de vida, podendo fazer com que suas necessidades, antes esquecidas, venham à tona quando uma oportunidade aparece. O sistema ciente de contexto descrito nesta dissertação é chamado de Omnipresent. Este sistema proporciona ao usuário, os seguintes benefícios:

- economia de tempo e dinheiro. O usuário poderá pesquisar por produtos ou serviços de acordo com o seu orçamento e com as suas preferências, por exemplo, ele poderá registrar no sistema o desejo de comprar um carro da marca BMW, cor prata e preço abaixo de R$ 120.000. Quando este usuário estiver próximo a alguém que esteja oferecendo um produto semelhante, ele receberá no dispositivo as informações necessárias para entrar em contato com o vendedor, como e-mail, telefone e a localização. A monitoração do contexto poderá também auxiliar o usuário a lembrar de compromissos que antes foram esquecidos, por exemplo, se o usuário vai a um supermercado comprar um produto qualquer, e o sistema lembra que ele também está precisando de um outro item importante, então neste caso, o usuário supre duas necessidades ao buscar a execução de uma simples tarefa;
- o sistema permitirá monitorar o contexto dos usuários (emoção, status, localização e estado fisiológico) proporcionando uma maior tranqüilidade para quem monitora e segurança para quem é monitorado. O usuário também poderá descrever lembranças baseadas no contexto do ambiente ao invés do tempo, como é comum nas agendas e celulares;
- toda análise do contexto deverá ser o mais transparente possível para o usuário, ou seja, o sistema irá tentar descobrir o contexto sem intervenção humana, com

exceção das informações que devem ser passadas durante o cadastro, como as preferências do usuário e descrição das tarefas como numa agenda.

Algumas ferramentas cientes de contexto apenas analisam parte do contexto do usuário para prover serviços personalizados [DHN+04] [NTT+04] [FM04] [LML+04]. Este trabalho não apenas tem o propósito de ampliar a análise do contexto do usuário, mas também, considerar o contexto do ambiente para saber em que situação o cliente se encontra. Outras ferramentas, também não levam em consideração a consulta às fontes de dados externas para a busca de serviços que não são de sua responsabilidade, como por exemplo, consultar um serviço de previsão do tempo para apresentar no mapa digital as regiões em que irá chover durante a tarde. Algumas arquiteturas cientes de contexto podem ser expandidas, desde que, as expansões sejam da mesma natureza, ou seja, mesma linguagem de programação. Outras ferramentas tentam implementar todas as funcionalidades de um serviço ciente de contexto em um único servidor. Serviços distribuídos permitem que a carga de processamento seja divida entre os serviços, além de possibilitar que os usuários acessem apenas os serviços que lhe serão necessários, por exemplo, se um usuário deseja apenas visualizar mapas no seu dispositivo, não faz sentido ele pagar por um servidor roteamento.

1.3 Estrutura da Dissertação

A dissertação está organizada da seguinte forma:

No Capítulo 2, é realizada uma revisão bibliográfica acerca dos estudos realizados sobre computação ubíqua, computação ciente de contexto, dispositivos móveis, sensores e uma subdivisão da computação ciente de contexto que são os sistemas baseados em localização (LBS).

No Capítulo 3, são apresentados alguns trabalhos já desenvolvidos sobre computação ciente de contexto, mostrando as suas principais vantagens e desvantagens. No final, um comparativo é realizado entre eles.

No Capítulo 4, é descrita a modelagem do contexto realizada, usando ontologias para que se tenha uma base de conhecimento sobre contextos e para que novas informações possam ser deduzidas a partir de dados lidos dos sensores.

O Capítulo 5 apresenta a arquitetura desenvolvida, o Omnipresent, que é composto por alguns Web services, a ontologia que representa o contexto e os tipos de clientes que acessam os serviços (browser e dispositivo móvel). Cada interface de

comunicação dos Web services é detalhada, juntamente com o diagrama de classe dos parâmetros necessários para comunicação com esses serviços.

No Capítulo 6, é apresentado um cenário de testes para avaliar o sistema Omnipresent. Foram definidos os requisitos funcionais e não funcionais que o sistema deve atender e em seguida são apresentados os resultados dos testes.

No Capítulo 7, são apresentadas as conclusões acerca do trabalho e uma discussão sobre trabalhos futuros.

Capítulo 2 . Visão Geral de Sistemas Cientes de Contexto

2.1 Introdução

O avanço da computação móvel permite que o usuário se desloque enquanto continua executando as suas tarefas, ou seja, através de dispositivos móveis, o usuário consulta informações a qualquer hora em qualquer lugar. Por se tratar de um dispositivo móvel, o usuário pode desejar acessar informação geográfica, como sua posição corrente ou de lugares próximos. Porém, além da localização outras informações podem ser percebidas e processadas por um serviço remoto graças ao avanço na tecnologia de sensores. Com o avanço dessas tecnologias, estão surgindo cada vez mais aplicações capazes de produzir informação personalizada de forma transparente ao usuário. Além da localização, é possível monitorar outras informações do usuário, como batimento cardíaco e temperatura, ou informações do ambiente como tráfego e condição do tempo. Nas seções a seguir, são apresentadas as características desse novo paradigma da computação.

2.2 Contexto

Numa conversação entre humanos, estes são capazes de usar, implicitamente, informações da situação, ou contexto, para aumentar a compreensão da conversa, mas o mesmo não ocorre na conversação homem-máquina ou máquina-máquina. Se aumentarmos o acesso do computador ao contexto, nós aumentaríamos a riqueza de comunicação entre homem e máquina.

A mobilidade do usuário cria situações nas quais o contexto é mais dinâmico, ou seja, as informações do contexto mudam constantemente, tornando complicado para que o usuário passe todas as informações do contexto para o computador a cada situação. Dessa forma, o computador deve coletar informações do contexto automaticamente de forma transparente para o usuário [DA00a].

Entender o que é contexto e como ele pode ser usado permitirá a construção de aplicações mais eficientemente e a melhor escolha do contexto a ser usado no

desenvolvimento de aplicações [ABR04]. Várias publicações apresentam diferentes definições para contexto [SAW94] [BBC97] [ST94]. Porém, algumas dessas definições apenas apresentam exemplos de contexto ou são incompletas, ficando difícil de serem aplicadas. Por exemplo, Schmidt [SAT+99] define contexto como "o conhecimento sobre o estado do usuário e do dispositivo, inclusive ambiente, situação e localização". Dey [DA99] define contexto como "qualquer informação que pode ser usada para caracterizar a situação de uma entidade". Uma entidade é uma pessoa, lugar ou objeto que é considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação. Porém, Chen [CK00] apresenta uma definição que engloba todas as outras definições:

"Contexto é o conjunto de estados do ambiente e configurações que determinam um comportamento da aplicação, incluindo o próprio evento da aplicação que ocorre e é de interesse do usuário"

Se uma informação pode ser usada para caracterizar a situação de um participante em uma interação, então essa informação é contexto. Para que o computador consiga informações do contexto, ele pode usufruir da leitura de sensores estáticos e sensores de baixo custo presentes em dispositivos móveis [KMK+03]. Por exemplo, sensores que detectam condições de tráfego e informam ao usuário, através de alguns dispositivos conectados à Internet, quais estradas estão congestionadas; ou aparelhos de GPS que são colocados em carros e que ajudam a localizá-los quando são roubados. Existem vários outros tipos de sensores que ajudam a capturar informações de contexto, por exemplo, sensores de temperatura, microfone para detectar o nível de barulho do ambiente, sensores de presença, câmeras de vídeo e biosensores que medem o batimento cardíaco, pressão sanguínea, temperatura corporal, entre outros.

Sendo o sensoriamento de localização uma importante informação de contexto e bastante crítico para várias aplicações cientes de contexto [CK00], essas aplicações foram divididas em *indoors* e *outdoors*. Aplicações *indoors* são aquelas destinadas à localização dentro de ambientes fechados, como, *shopping centers*, edifícios e lojas. Como o sinal GPS não funciona (ou é muito fraco) dentro de ambientes fechados [CK00], essas aplicações geralmente usam *BlueTooth* ou sinais ultra-sonoros para obterem uma representação simbólica da localização [HHS+99] [ACK03], ou seja, representam a localização através símbolos abstratos como, por exemplo, o número do

andar de um edifício, a distância ou o sentido em relação a um outro objeto (a direita de, a esquerda de) [Sch95]. As aplicações *outdoors* são destinadas à localização em ambientes abertos, como, por exemplo, encontrar o menor caminho até uma loja usando um mapa digital de uma cidade. A tecnologia de localização *outdoor* mais precisa é a do GPS [ACK03]. Essas aplicações também podem usar o modelo simbólico, mas geralmente usam o modelo geométrico que representa a localização como um sistema de coordenadas geográficas [CK00], como, por exemplo, um sistema que mostra, num mapa, um posto de gasolina mais próximo do carro do usuário, juntamente com os passos necessários para se chegar ao destino, por exemplo: siga em frente, vire a direita e vire a esquerda.

2.2.1 Categorias de Contexto

A categorização do contexto pode ajudar os desenvolvedores a descobrir que tipos de informações de contexto serão úteis para sua aplicação. Segundo Feng, Apers e Jonker [FAJ04], o contexto é dividido em duas categorias:

1) contexto do usuário: é toda informação relacionada ao usuário. Esse contexto é ainda dividido em:
   a) *Perfil*: representa o perfil do usuário, tais como, músicas preferidas, comida predileta, programa de TV preferido e outras informações subjetivas do usuário;
   b) *Comportamento dinâmico*: representa as tarefas a serem cumpridas durante o dia, semelhante a uma agenda;
   c) *Estado Fisiológico*: estas informações são relevantes para o monitoramento da saúde e são obtidos através de sensores que são colocados no corpo do usuário. Por exemplo, temperatura corporal, batimento cardíaco, nível de glicerina, entre outros;
   d) *Estado Emocional*: esse contexto pode ser capturado através de câmeras (análise visual), interpretação de sinais acústicos, batimento cardíaco ou fornecido manualmente pelo usuário.
2) contexto do ambiente: são informações relativas ao ambiente. Esse contexto é dividido em:
   a) *Ambiente Físico*: representa características do ambiente físico, tais como, localização, temperatura, tempo, nível de luminosidade e nível de ruído;

b) *Ambiente Social*: representa o ambiente social, tais como, congestionamento, a localização de pessoas ao redor, informações de descontos de lojas, entre outros;
c) *Ambiente Computacional*: representa o ambiente computacional, tais como, largura de banda da rede, capacidade da rede, dispositivos ao redor, como impressoras e computadores.

Segundo Dey [DA00a], a classificação acima envolve importantes aspectos do contexto, tais como, "onde o usuário está", "com quem está" e "quais recursos estão próximos". Porém, nem todos os aspectos desta classificação são importantes para uma dada aplicação, isto varia de situação para situação. Por exemplo, em alguns casos, informações do ambiente físico são importantes e em outros casos não. Dessa forma, Dey apresenta um outro tipo de categorização. Uma categorização inicial seria: contexto primário e secundário. Localização, identidade, tempo e atividade são tipos de contexto primário que caracterizam a situação de uma entidade. Esses tipos de contexto são mais importantes que outros. Eles procuram responder as perguntas: "quem é?", "onde está?", "quando?" e "o que o usuário está fazendo?", para assim, determinar o porquê da situação estar ocorrendo. Além disso, contexto primário serve como índice para outras fontes de informação de contexto [BB02]. Por exemplo, dada a identidade de uma pessoa, pode-se adquirir muitas outras informações relacionadas, tais como, número de telefone, endereço, e-mail e lista de amigos. Com a localização de uma entidade, pode-se determinar que outros objetos ou pessoas estão próximas à entidade. As informações de contexto que são encontradas através do contexto primário são chamadas de contexto secundário.

2.2.2 Computação Ciente de Contexto

Um sistema ciente do contexto, do inglês *context-aware*, é aquele que usa o contexto para prover informações relevantes ou serviços ao usuário [DA00a]. São consideradas aplicações cientes do contexto aquelas que têm seu comportamento modificado de acordo com as informações do contexto ou aplicações que simplesmente mostram ao usuário, informações do contexto.

As aplicações cientes de contexto podem ser divididas em três categorias [DA00a]:

1) sensoriamento do contexto (*contextual sensing*): é a habilidade de detectar informações sobre o contexto e apresentá-las ao usuário. Nesta categoria,

estão as aplicações que buscam informações para o usuário sobre o contexto, por exemplo, o sistema que apresenta informações sobre a impressora disponível mais próxima;

2) adaptação ao contexto (*contextual adaptation*): é a habilidade de executar ou modificar um serviço automaticamente, baseando-se no contexto atual. São baseados numa simples regra de *if-then-else*. Um comando é executado quando existe uma certa combinação de contexto. Por exemplo, em uma casa equipada com sensores, o sistema pode acender as luzes quando alguém entra na sala;
3) aumento do contexto (*contextual augmentation*): é a habilidade de associar informações ao contexto para recuperar posteriormente. São aplicações que permitem associar dados digitais ao contexto do usuário. Por exemplo, um usuário pode associar uma nota virtual a uma televisão quebrada e quando outro usuário estiver próximo ou tentar usar a televisão, ele verá a nota virtual.

Além da classificação acima, Schilit [SAW94] apresenta uma outra classificação um pouco semelhante:

- ciente de contexto ativo: uma aplicação adapta-se automaticamente ao contexto descoberto, mudando o comportamento da aplicação. Por exemplo, considere um usuário que esteja lendo os e-mails no seu PDA e quando começa a dirigir o carro, o PDA muda automaticamente para o modo de voz para que o usuário apenas escute as mensagens e não desvie a atenção do volante do carro;
- ciente de contexto passivo: uma aplicação apenas apresenta informações do contexto ao usuário ou as armazena para que o usuário possa consultar posteriormente. Por exemplo, um PDA que exibe na tela a localização do usuário em um mapa digital.

2.3 Sensoriamento

Um dos principais meios de obter informações sobre o contexto é através do uso de sensores. O sensoriamento do contexto é dividido em sensoriamento de baixo nível e sensoriamento de alto nível [CK00]. Contexto de baixo nível é a informação bruta medida do ambiente, como por exemplo, temperatura, localização geográfica, tempo, luminosidade, umidade, entre outros. Contexto de alto nível são informações que tentam responder a perguntas, tais como *o que o usuário está fazendo agora?*. Obter o contexto

de alto nível é um grande desafio no sensoriamento. A maioria das aplicações geralmente usam três formas possíveis de tentar obter o contexto de alto nível:

- processamento de imagens de câmeras;
- consultar uma agenda do usuário para supor o que ele está fazendo agora. Porém, às vezes, essa informação torna-se imprecisa, pois o usuário pode não seguir o que foi programado e pode nem sempre programar todas as suas atividades;
- combinar vários sinais de sensores de baixo nível para reconhecer um contexto complexo, como por exemplo, se um usuário está no banheiro e o chuveiro estiver ligado e a porta fechada, o sistema pode concluir que o usuário está tomando banho.

2.3.1 Sensores de Posicionamento

Um dos mais importantes sensores nessa área é o de posicionamento. A posição do usuário é fundamental para entrega de serviços personalizados e é obtida através de dispositivos móveis com LDT (*Location-Determining Technologies*) que o usuário carrega consigo. A posição do usuário é transferida via rede do dispositivo móvel para o provedor de serviço e o usuário recebe a informação do provedor no seu dispositivo móvel.

As tecnologias de LDT mais comuns são GSM e GPS. GSM é uma tecnologia da segunda geração da telecomunicação móvel e que proporciona uma taxa de transmissão de dados de 9,6 Kbps. GSM usa a tecnologia de posicionamento baseado em célula, onde a posição do celular é encontrada usando a localização conhecida da estação base a qual o celular está conectado [ACK03]. Portando, essa tecnologia é bastante imprecisa, pois determina a localização do aparelho celular dentro de centenas de metros. Uma tecnologia LDT mais precisa é o *Global Positioning System* (GPS), pelo qual através de 26 satélites espalhados pela órbita da Terra, é possível determinar a latitude, longitude e altura em qualquer ponto da Terra com uma precisão de 5 a 10 metros [GPS04]. Os satélites enviam mensagens específicas que são interpretadas por um receptor GPS. Alguns receptores de GPS podem variar entre precisão e funcionalidade [CCH+96]. Por exemplo, alguns incluem programas que fazem transformações entre sistemas de coordenadas ou possuem dados de saída compatíveis com os sistemas de informações geográficas (SIG) disponíveis no mercado. Inicialmente, a tecnologia de GPS foi criada pelo Departamento de Defesa dos Estados Unidos para uso militar, mas em 1980, o governo decidiu tornar o sistema de posicionamento disponível para as indústrias no mundo. Desde então, muitas indústrias

têm aproveitado a oportunidade para acessar dados de posicionamento através do GPS e usá-los para enriquecer seus produtos e serviços [Sch04].

Uma grande variedade de dispositivos móveis com tecnologia de comunicação e LDT estão disponíveis no mercado. Os dispositivos podem ser agrupados em três categorias: celulares, PDA e *smartphones* [ACK03]. Já existem no mercado vários celulares que suportam receptores GPS *Bluetooth*, porém, os celulares têm um limitado poder computacional e interface multimídia limitada para mostrar imagens de baixa resolução. PDA têm um maior poder computacional que os celulares podendo apresentar imagens digitais, animação e videoclipes. Contudo, muitos PDA não têm tecnologia de comunicação embutida, precisando às vezes, usar cartões de extensão para prover esta funcionalidade. O s*martphone* é uma mistura de celular e PDA. Um *smartphone* tem muitas facilidades multimídia do PDA e tecnologia de comunicação embutida. Um *smartphone* tem mais poder computacional do que um celular e similar ao de um PDA.

2.4 LBS

Avanços na comunicação sem fio, dispositivos móveis e LDT nos últimos anos, têm proporcionado o desenvolvimento de LBS (*Location Based Services*). LBS é um tipo de aplicação dentro da computação ubíqua que oferece serviços personalizados e que utilizam tecnologias de determinação de localização, como o GPS, para obter a posição do usuário [ACK03]. A posição do usuário é fundamental para entrega de serviços altamente personalizados e é obtida através de dispositivos móveis com LDT que o usuário carrega consigo. A posição do usuário é transferida via rede do dispositivo móvel para o provedor de serviço e o usuário recebe a informação do provedor no seu dispositivo móvel.

LIF (*Location Interoperability Forum*) [ACK03] tem dividido LBS em três categorias:

- *Basic Service level*: implica no uso de tecnologia baseada em células. A acurácia é pobre, mas existem vários terminais disponíveis que suportam esse nível de serviço. As aplicações que podem usufruir desse tipo de serviço são aquelas nas quais a precisão da localização não é tão importante, como por exemplo, localização de pontos turísticos e entretenimento;

- *Enhanced Service level*: são aplicações que requerem maior acurácia (cerca de dez metros), como por exemplo, um serviço que rastreia a localização de uma pessoa, anúncios de produtos, entre outros;
- *Extended Service level*: são aplicações que precisam de alta acurácia (cerca de poucos metros), por exemplo, aplicações de navegação passo-a-passo, nas quais um usuário pode chegar ao ponto destino através das instruções que são mostradas no seu dispositivo.

O projeto de aplicações LBS pode ser classificado em dois tipos de serviços [Sch04]:

1) serviço *push* – implica que o usuário recebe informação sem explicitamente ter requisitado. A informação pode ser enviada ao usuário com ou sem o seu consentimento. Em outras palavras, em uma aplicação *push*, a informação é entregue ao consumidor sem que este tenha controle de quando isto ocorre;
2) serviço *pull* – o usuário busca informação sempre que é preciso. Esta informação pode envolver localização, como por exemplo, consultar pelo cinema mais próximo.

Numa aplicação LBS baseada em *push*, o usuário se inscreve em um certo serviço e este serviço pode lhe fornecer informação se um determinado critério for satisfeito. Quando o critério é satisfeito, a informação é enviada ao usuário em um tempo não controlado por ele. Apesar de possuir elementos *pull* (a inscrição do usuário), esse tipo de aplicação é chamada de *push* por possuir o serviço de entrega de informação como atividade principal. Um caso simples de aplicação *push* é aquela na qual o usuário informa em sua agenda eletrônica que precisa comprar um novo produto. Quando o usuário casualmente se aproxima de uma loja, ele recebe, automaticamente, em sua agenda uma propaganda da loja informando sobre o produto de seu interesse e possíveis promoções e descontos, além da localização da loja.

A transparência e comodidade oferecidas pelos serviços *push* tem um alto preço. Aplicações *push* são muito caras comparadas aos serviços *pull*. Por exemplo, eles requerem maior quantidade de recurso de rede porque a localização do usuário precisa ser atualizada constantemente. Outra desvantagem dos serviços *push* é a questão da privacidade. A própria noção de atualização constantemente da posição dá a idéia de rastreamento em tempo real [Sch04].

2.5 OpenLS

O *Open Geospatial Consortium* (OGC) é um consórcio internacional para o desenvolvimento de especificações na área de geoprocessamento. A interoperabilidade é um dos resultados obtidos pelas indústrias que adotam essas especificações. Dentre as especificações propostas, existe o *Open Location Services* (OpenLS) com o objetivo de facilitar o uso de localização e outros tipos de informação espacial em redes sem fio. OpenLS é uma especificação que define um conjunto de interfaces para acessar uma plataforma LBS [Sch04]. Esta especificação é composta pelos seguintes serviços [OGC06]:

1) *Location Utilities Service*: é composto por outros dois serviços *Geocode* e *Reverse Geocode*. O *Geocode* determina a posição geográfica passando o nome do local, endereço ou CEP. O *Reverse Geocode* determina o nome do local, endereço e CEP a partir da posição geográfica;
2) *Directory Service*: este serviço permite acesso a um diretório *on-line* para encontrar um lugar específico, produto ou serviço mais próximo. A pesquisa pode ser restrita a um lugar específico ou a uma área (*bouding box*). O resultado da busca são as posições e uma completa descrição dos locais, produtos ou serviços ordenados segundo um critério de busca, como nome ou distância. Como em sistemas de páginas amarelas, o usuário também pode especificar o tipo de diretório que deseja pesquisar, como por exemplo: restaurantes, hotéis, lojas, entre outros;
3) *Presentation Service*: este serviço trata da visualização de informação espacial como apresentação de mapas, rotas, pontos de interesse e informação textual (por exemplo, descrição da rota) em um terminal móvel. A forma de consulta a este serviço é dividida na seguinte forma:
    a) *Output Parameters*: define o tamanho e formato do mapa consultado. É composto pelos atributos:
        i) *Width*: a largura do mapa em pixels;
        ii) *Height*: a altura do mapa em pixels;
        iii) *Format*: o formato do mapa (JPEG, SVG, etc);
        iv) *Transparency*: define a transparência do fundo do mapa;
        v) *Background Color*: define a cor de fundo do mapa;

vi) *Content*: define como o resultado deve ser retornado, se através de uma URL ou codificado em base64, que é um método de codificação binária que pode ser representado usando apenas caracteres do código ASCII.

b) *Context*: define qual área do mundo está sendo consultada. Essa consulta pode ser definida de duas formas:
   i) *Bounding Box*: dois pontos em um específico SRS (Sistema de Referência Espacial) que define a extensão do mapa;
   ii) *Center Point and Scale*: define o centro do mapa em uma escala e projeção particulares.

c) *Overlays*: define a lista de informações a serem postadas no mapa, como PoI, rotas e posição dos usuários;

d) *BaseMap*: define a lista de camadas que devem aparecer no mapa e o estilo de cada uma.

4) *Gateway Service*: esta é a interface entre este serviço e o serviço de localização que reside em um GMLC (*Gateway Mobile Location Center*) ou MPC (*Mobile Positioning Center*), através do qual é possível obter a posição do terminal móvel;

5) *Route Determination Service*: encontra a rota entre dois pontos. A rota pode ser a mais rápida, a menor ou a que tem menos tráfego. O cliente também pode especificar os pontos intermediários os quais a rota deverá incluir.

A especificação também fornece um conjunto de XMLSchemas que descrevem como se deve proceder a troca de mensagens entre o cliente e serviço.

2.6 Considerações Finais

Neste capítulo, foi abordada a fundamentação teórica relacionada ao tema desta dissertação. Primeiramente, foi apresentada uma definição mais geral do que é contexto e as formas de categorizá-la segundo Feng, Apers, Jonker [FAJ04] e Dey [DA00a]. As informações do contexto são obtidas principalmente através de sensores, destacando o uso de sensores de localização, essencial em aplicações LBS.

No próximo capítulo serão abordadas aplicações que utilizam algumas informações do contexto para produzir uma resposta de acordo com a situação atual do usuário.

Capítulo 3 . Trabalhos Relacionados

3.1 Introdução

Este capítulo descreve alguns trabalhos desenvolvidos na área de computação ciente de contexto, apresentado as suas vantagens e desvantagens. A Seção 3.2 descreve sobre a arquitetura SOCAM. A Seção 3.3 apresenta um sistema de lembranças, o CybreMinder. A Seção 3.4 apresenta a arquitetura AROUND, que possui a característica de oferecer serviços baseados em escopo. A Seção 3.5 apresenta o protótipo de um guia turístico para a cidade de Aalborg. A Seção 3.6 descreve sobre o Flame2008, um sistema para integração de Web services com o intuito de obter ofertas significantes baseadas na situação e no perfil do usuário. A Seção 3.7 apresenta a plataforma Nexus, que procura integrar vários modelos de contexto com o objetivo de construir um modelo global. A Seção 3.8 apresenta um sistema de redirecionamento de mensagens telefônicas, o ICAMS. A Seção 3.9 apresenta o FieldMap, uma ferramenta que permite adicionar comentários em mapas apresentados em dispositivos móveis. A Seção 3.10 apresenta um sistema de monitoração de arritmia. Por fim, a Seção 3.11 apresenta uma comparação entre as ferramentas.

3.2 SOCAM

SOCAM [GPZ04] (*Service-Oriented Context-Aware Middleware*) é uma arquitetura para a construção de um serviço móvel ciente de contexto. O modelo do contexto é baseado em ontologia que provê um vocabulário para representar e compartilhar conhecimento de contexto em um domínio da computação onipresente. O projeto da ontologia do contexto é composto de dois níveis hierárquicos. Um nível contém ontologias individuais sobre vários subdomínios, por exemplo, o domínio de uma casa, um escritório, um veículo, etc. Um nível mais alto contém conceitos gerais sobre as outras ontologias, esse nível é chamado de ontologia generalizada ou ontologia de alto nível.

O domínio específico de uma ontologia pode ser dinamicamente re-alocado. Por exemplo, quando um usuário deixa sua casa para dirigir um carro, a ontologia do

domínio da casa será trocada automaticamente pela ontologia do veículo. A ontologia é escrita em OWL [W3C04].

A arquitetura SOCAM é composta pelos seguintes elementos:

- Provedores de contexto: provêem uma abstração do sensoriamento de baixo nível. Cada provedor de contexto precisa ser registrado em um serviço de registro, o SLS (*Service-locating service*) para que outros usuários possam descobrir esses provedores. Os provedores de contexto podem ser externos ou internos. Os provedores de contexto externo obtêm contexto de fontes externas, por exemplo, um serviço de informação do tempo ou um serviço de localização *outdoor*. Os provedores de contexto interno obtêm contexto diretamente de sensores localizados em um subdomínio, como uma casa, por exemplo;

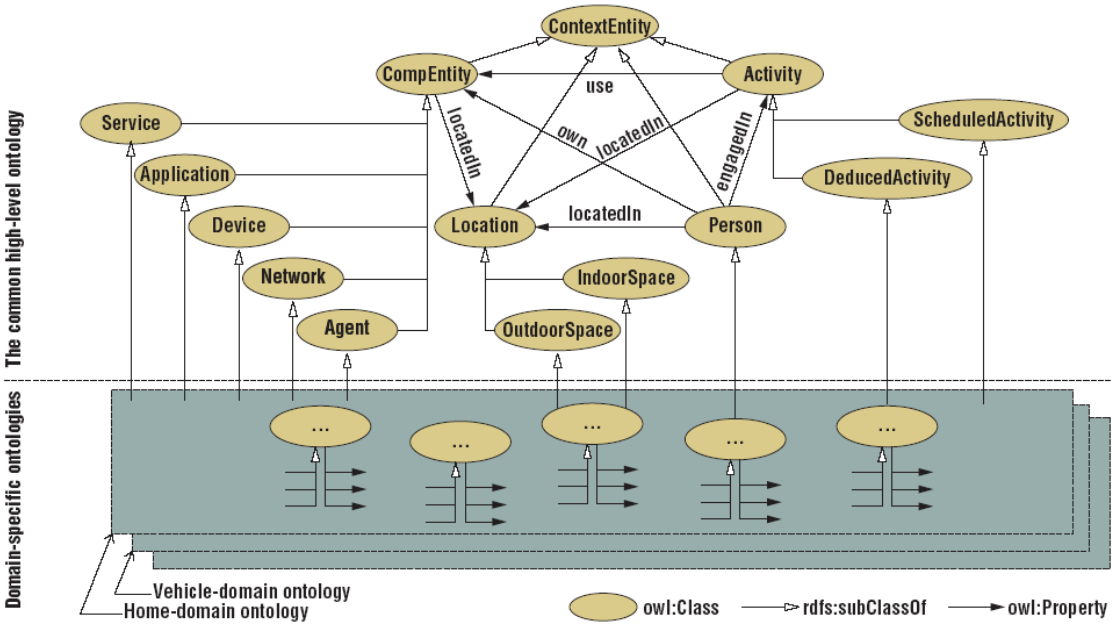

Figura 3.1 Diagrama hierárquico de uma ontologia de contexto (extraído de [GPZ04])

- Serviço de Localização de Serviço (SLS): permite usuários, agentes e aplicações descobrirem e localizarem diferentes provedores de contexto;
- Interpretador de contexto: provê contexto de alto nível através da interpretação de contexto de baixo nível. Dessa forma, o interpretador também é tratado como um provedor de contexto e pode ser registrado no SLS. O interpretador de contexto é dividido em *reasoner* e *knowledge base* (KB). O *reasoner* tem a função de prover contexto de alto nível baseado no contexto de baixo nível e detectar inconsistência e conflitos na base de conhecimento. KB provê um conjunto de API para que outros componentes possam consultar, adicionar, deletar ou modificar conhecimento de

contexto, além de conter a ontologia de um contexto de um subdomínio e suas instâncias. Essas instâncias podem ser especificadas pelos usuários, no caso de contexto definido, ou adquirido de vários provedores de contexto;

- Serviços cientes de contexto: são aplicações que fazem uso dos diferentes níveis da arquitetura SOCAM. Os desenvolvedores dessas aplicações podem predefinir regras e especificar que métodos devem ser invocados quando uma condição for verdadeira. Todas as regras são salvas em um arquivo e pré-carregadas no *reasoner*.

SOCAM foi projetado como um serviço de componentes independentes que podem ser distribuídos numa rede heterogênea e podem se comunicar entre si. Todos os componentes são implementados em Java. Para a comunicação entre os componentes distribuídos é usado Java RMI, que permite objetos distribuídos invocarem métodos de outros objetos remotos. A interação entre os componentes do SOCAM ocorre, resumidamente, da seguinte forma: um provedor de contexto pode adquirir dados sobre o contexto de vários sensores heterogêneos. Diferentes provedores de contexto registram seus serviços no SLS. As aplicações móveis ciente de contexto são capazes de localizar um provedor e obter dados sobre um determinado contexto. O interpretador de contexto e o serviço móvel ciente de contexto também podem ser registrados no SLS.

A arquitetura SOCAM segue uma arquitetura semelhante ao padrão Web Service, na qual os serviços são registrados em um diretório público e podem ser encontrados e utilizados por outros serviços. Porém, a arquitetura não é independente de linguagem de programação, pois os componentes trocam mensagens usando Java RMI, tornando difícil a integração entre servidores heterogêneos. Além disso, as regras de contexto devem ser carregadas previamente no sistema para que passem a funcionar.

3.3 CybreMinder

CybreMinder [DA00b] é uma ferramenta ciente de contexto que ajuda a criar e gerenciar lembranças. Uma lembrança é um tipo especial de mensagem que enviamos para nós mesmos ou para outros, para informar sobre atividades futuras que devemos tratar. Por exemplo, um colega de trabalho nos envia uma mensagem (ou seja, uma lembrança) pedindo para levarmos uma cópia do trabalho para a próxima reunião. O CybreMinder usa informações de contexto para informar o usuário sobre uma determinada lembrança. Uma importante informação de contexto usado pela ferramenta é a localização. A localização pode ser em relação a algum lugar ou em relação a uma

outra pessoa, por exemplo: “lembrar de comprar um presente ao chegar a um determinado lugar”; “quando estiver próximo de meu colega de trabalho, me avise de comentar sobre a reunião de trabalho”.

O sistema foi implementado em Java e é dividido em duas partes principais:

- criar lembranças: o CybreMinder apresenta uma interface semelhante ao de um e-mail, com assunto, corpo da mensagem, uma lista de outras pessoas a quem a lembrança interessa, nível de prioridade, data e hora de expiração e a situação (contexto) que a mensagem deve ser entregue, por exemplo, o lugar e a hora;
- entregar lembranças: a lembrança é entregue quando a situação (contexto) associada for satisfeita ou porque o tempo expirou. O CybreMinder decide qual a melhor forma de entregar a mensagem ao usuário. A forma de entrega *default* é mostrar a lembrança num visor disponível junto com um sinal de áudio, mas isso pode ser configurado pelo usuário para cada lembrança. A lembrança pode ser entregue via e-mail, celular, *handheld*, entre outros.

CybreMinder também permite fazer lembranças complexas, como por exemplo, Joana precisa fazer uma chamada telefônica para Pedro quando ela chegar ao seu escritório, quando ela tiver tempo livre na sua agenda e seu amigo não estiver ocupado. Para criar esta situação, o usuário precisa criar três sub-situações: Joana está em seu escritório, o status de atividade de Pedro está baixo e Joana tem pelo menos uma hora livre antes do próximo compromisso.

Os tipos de contexto percebidos pela ferramenta são: agenda do usuário, ambiente físico e social. Além disso, a determinação de situação é complexa, ou seja, a forma que o CybreMinder utiliza para programar uma situação de contexto em que a lembrança deve ser entregue, não é fácil de ser usada por um usuário comum.

Para perceber o contexto associado com a lembrança, o CybreMinder usa o Context Toolkit. O Context Toolkit [SDA99] é um software que ajuda a construir aplicações cientes de contexto. Ele promove três principais conceitos para construir aplicações cientes de contexto: separar a aquisição de contexto do uso de contexto; agregação de contexto e interpretação de contexto. Agregação e interpretação facilitam a aplicação a obter o contexto requerido.

O Context Toolkit consiste de três blocos básicos:

- widgets: agregação e interpretador de contexto (ver Figura 3.2). Widgets são responsáveis principalmente por coletar e encapsular informação sobre um dado contexto, como localização. Os widgets também dão suporte a serviços que permitem

afetar o ambiente, como por exemplo, controlar a intensidade de luz de um local de acordo com a luminosidade detectada pelos sensores;

- agregação: responsável por unir toda informação de contexto de uma entidade particular (pessoa, lugar ou objeto). O interpretador de contexto é usado para abstrair ou interpretar o contexto. Por exemplo, um *widget* pode fornecer um contexto de localização na forma de latitude e longitude, mas uma aplicação pode requerer a localização como o nome da rua. Na arquitetura, isso é realizado pelos interpretadores de contexto. O mecanismo de comunicação entre os componentes é XML sobre HTTP;
- aplicação: são as aplicações que usufruem dos componentes do *Context Toolkit*.

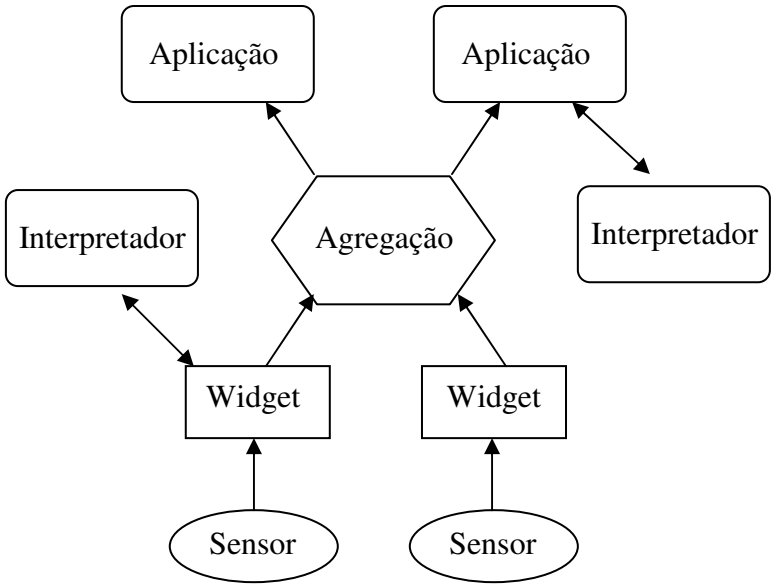

Figura 3.2 Componentes do Context Toolkit (extraído de [SDA99])

3.4 AROUND

A arquitetura AROUND [JMRD03] provê uma infra-estrutura de localização de serviços que permite as aplicações selecionarem serviços que são associados a uma determinada área espacial. A principal diferença entre a arquitetura AROUND e as outras aplicações LBS é que ela utiliza dois modelos distintos de seleção de serviços:

- modelo baseado em distância. Neste modelo, o cliente seleciona os serviços localizados dentro de sua região de proximidade, ou seja, um raio é criado ao redor da posição atual do cliente e ele seleciona os serviços de seu interesse que estiverem

dentro desse raio. A desvantagem desse modelo é que quanto maior o raio, mais coisas que não são de interesse do usuário estarão dentro de sua área de proximidade;

- modelo baseado em escopo. Neste modelo, cada serviço tem seu escopo associado a um espaço físico. O cliente seleciona aqueles serviços que o escopo inclui em sua própria localização, isto é, o cliente é capaz de descobrir um serviço se o mesmo estiver localizado dentro do escopo desse serviço. Por exemplo, um servidor de mapas de municípios de um estado que é oferecido apenas naquela região coberta pelos mapas. Na Figura 3.4, um cliente representado pelo circulo 'C' está dentro dos escopos 1, 2, 3, 4 e 6, portanto, ele pode selecionar os serviços que são oferecidos nesses escopos.

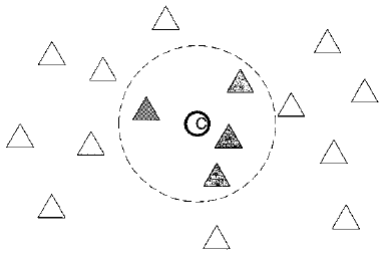

Figura 3.3 Seleção baseada em distância

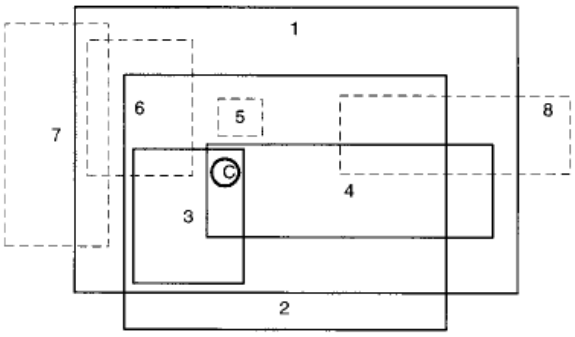

Figura 3.4 Seleção baseada em escopo

No modelo baseado em distância, o foco está na localização do provedor de serviço, enquanto no modelo baseado em escopo, o foco está na área geográfica definida pelo uso do serviço. Estas diferenças fazem com que cada modelo seja o melhor para um determinado tipo de serviço. O modelo baseado em distância é o mais adequado para serviços que tenham uma forte associação com um específico ponto no espaço, como por exemplo, um restaurante. Por outro lado, o modelo baseado em escopo é o mais adequado para serviços que não têm uma ligação com um ponto específico no espaço, tais como, serviços de mapas e previsão do tempo.

O modelo baseado em escopo é o mecanismo utilizado pela arquitetura AROUND para associar serviços com localização. O escopo de um serviço é registrado em um conjunto de contexto de localização. Neste caso, contexto de localização é uma representação simbólica de uma área num espaço físico, como por exemplo, um departamento de um campus universitário, um laboratório dentro de um departamento, uma praça pública, um bairro, etc. Na arquitetura AROUND, os contextos de localização podem ser ligados por relacionamentos unidirecionados, formando um grafo, onde o objetivo é aumentar o processo de descoberta de serviços. Relacionamentos entre contextos estabelecem um mecanismo para a propagação de consultas de um contexto fonte para um contexto destino.

Na arquitetura, existem dois tipos de relacionamentos:

- contido: se refere a inclusão espacial de áreas dentro da área de um outro contexto. A Figura 3.5 apresenta um exemplo em que um serviço "A" registrado no contexto "Campus" está disponível em todo lugar porque todos os outros contextos estão contidos no contexto "Campus". Por outro lado, o serviço "C" está registrado no contexto "Lab. 1", portanto, esse serviço está somente disponível neste contexto;
- adjacente: expressa a proximidade espacial entre dois contextos de localização, por exemplo, os quartos de um edifício, onde cada quarto tem um contexto de localização. Isso permite que usuários consultem serviços próximos mesmo estando fora do escopo.

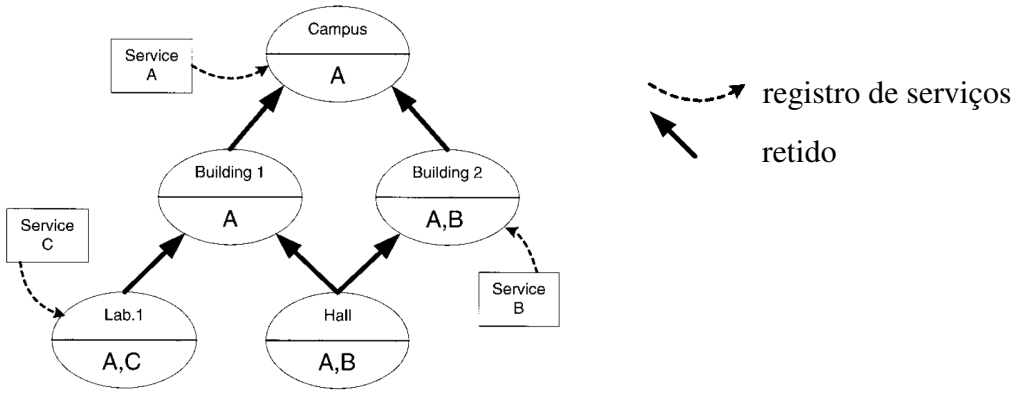

Figura 3.5 Localização de serviços através de relacionamentos entre os contextos de localização (extraído de [JMRD03])

Quando o usuário, carregando um dispositivo com um cliente AROUND, se move para uma nova área, os serviços que estão registrados nessa área são descobertos e os ícones são mostrados na tela do dispositivo do usuário. Este pode ativar um serviço clicando no ícone. Por exemplo, um serviço de informação de ônibus que informa os

ônibus que passam na rua onde o usuário está localizado e os pontos de ônibus mais próximos.

A principal desvantagem da arquitetura AROUND é que ela não leva o contexto do usuário em consideração ao apresentar os serviços. Apenas a localização e o deslocamento do usuário são utilizados pela ferramenta para descobrir em que área de serviço ele está. A comunicação entre os componentes é baseada em CORBA [OMG06].

3.5 Online Aalborg Guide

Online Aalborg Guide [ACK03] é um protótipo construído usando o *framework* baseado em LBS desenvolvido no departamento de ciências da computação da Universidade de Aalborg. O *framework* é usado para implementar um guia *on-line* para turistas que visitam a cidade de Aalborg. As características básicas da ferramenta são:

1. Visualizar a localização dos PoI (Point of Interest) mais próximos;
2. Permite ao usuário salvar os PoI para uso posterior, como o *bookmark* de um *Web browser*;
3. Permite ao usuário adicionar novos PoI, submetendo o nome e a descrição do mesmo;
4. O usuário pode adicionar novos comentários e fotos a um PoI já existente;
5. Os provedores podem enviar anúncios de acordo com as preferências e localização do usuário;
6. Permite ao usuário encontrar o menor caminho para um PoI;
7. Obter informações sobre outros usuários;
8. O usuário pode editar o seu perfil antes e durante uma viagem;
9. Serviço de mapas. Um mapa do ambiente é exibido a toda hora com uma indicação da posição do usuário e os PoI mais próximos.

O Online Aalborg Guide usa uma mistura de tecnologia *push* e *pull*, como se pode perceber pelas características anteriores. Por exemplo, os PoI mais próximos são continuamente atualizados e mostrados na tela do celular. Dessa forma, o usuário não precisa interagir com o dispositivo para ver que PoI estão próximos. Porém, se o usuário deseja obter mais informações sobre um PoI, ele deve fazer uma requisição ao servidor e a informação é apresentada na tela do dispositivo.

O protótipo desenvolvido utiliza o celular Nokia 7650 conectado via Bluetooth a um Emtac GPS. O programa que é instalado no celular é chamado de GPSOne. Como

se pode perceber na Figura 3.6, o Emtac GPS constantemente provê coordenadas ao celular. Quando o celular recebe as coordenadas, o GPSOne começa a prover serviços LBS para o usuário. O celular se conecta ao servidor através do protocolo HTTP. Se o GPSOne precisa fazer o *download* de algum mapa *raster*, uma conexão ao servidor de mapas é estabelecida e o mapa é retornado no formato JPEG. O protótipo implementado contém apenas as características 1, 2, 6, 9 e 10 das que foram listadas anteriormente.

Mapas rasters são os dados mais estáticos na aplicação LBS. Porém, esses mapas demandam uma maior capacidade de armazenamento. Por causa do pequeno poder de armazenamento do terminal do cliente, não é possível carregar todos os mapas. No protótipo, uma *cache* de mapas tem sido implementada para carregar unicamente os mapas necessários dependendo da localização do usuário. Somente os mapas ao redor do usuário são baixados do servidor. Depois de um certo tempo, a memória do dispositivo do usuário estará cheia devido à quantidade de mapas que foram baixados. Uma estratégia adotada para resolver essa situação é apagar os mapas menos recentemente usados.

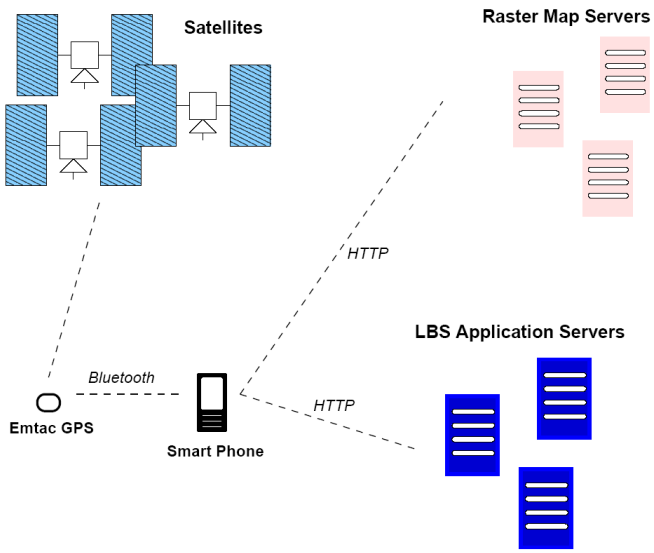

Figura 3.6 Arquitetura do Online Aalborg Guide (extraído de [ACK03])

3.6 Flame2008

Flame2008 [WVG04] é um protótipo de uma plataforma de integração para Web services inteligentes personalizados para as olimpíadas de 2008 em Beijing. A principal idéia desse projeto é, através de um dispositivo móvel, realizar '*push*' de ofertas significantes baseadas na situação atual e no perfil do usuário.

Todas as dimensões do contexto (localização, calendário, perfil, etc) são representadas através de ontologias. Com agregação dessas dimensões, o Flame2008

define situações que são registradas no sistema. Por exemplo, através de uma notação em F-Logic é possível registrar a seguinte situação:

```
WatchingCompetition : BeingAtEvent [
        position ->> loc#Stadium;
        localAction ->> act#AnyAction;
        userState ->> cal#Leisure].
```

Este exemplo descreve a situação "assistindo a uma competição". O usuário se encontra nessa situação se estiver localizado no estádio executando qualquer ação e na sua agenda aquele horário está marcado como hora de lazer. O Flame2008 pode ser composto de várias ontologias, como mostrado no exemplo anterior, pois, 'loc', 'act' e 'cal' são *namespaces* para diferentes ontologias.

Definida a situação em que o usuário se encontra e o seu perfil, o sistema se encarrega de buscar serviços que se encaixam nesses parâmetros. O perfil é composto por interesses, preferências e dados pessoais do usuário. Interesses são informações estáticas que não se alteram com a mudança de contexto, por exemplo, se o usuário possui interesse por música clássica e obras de arte. As preferências podem depender da situação, por exemplo, um usuário em sua cidade pode preferir comida italiana quando vai a um restaurante, mas quando ele está viajando pode preferir experimentar a comida local.

Cada usuário pode manter seu perfil através de um conjunto de propriedades que o caracterizam. Além disso, há sensores, que obtêm informações do ambiente atual do usuário (localização e tempo). O resultado são instâncias de uma ontologia de alto nível que são usadas para implicitamente construir um "perfil de situação" que é semanticamente comparada com os perfis de todas as situações conhecidas pelo sistema, e implicitamente comparada com todos os perfis de serviços registrados.

A desvantagem é que ele não trata o relacionamento com outros usuários, ou seja, o sistema não monitora o contexto de contatos do cliente. Além disso, a busca por produtos e serviços é baseada apenas na categoria do produto e no perfil do usuário, não se importando com outras características do produto ou propriedades do serviço.

3.7 Nexus

Nexus [DHN+04] é uma plataforma com o propósito de dar suporte a todos os tipos de aplicações cientes de contexto através de um modelo de contexto global. Servidores de contexto podem ser integrados à plataforma para compartilhar e usufruir das

informações providas pelos outros servidores de contexto. Esses servidores de contexto são chamados de modelos locais (ou modelos de contexto).

Os modelos locais contém diferentes tipos de informações do contexto. Por exemplo, um modelo que representa as estradas, um modelo que representa as casas em uma cidade, um modelo que representa as pessoas, entre outros. No caso do modelo de pessoas, são usados sensores para manter os dados atualizados no modelo.

A plataforma Nexus é organizada como mostra a Figura 3.7 dada a seguir.

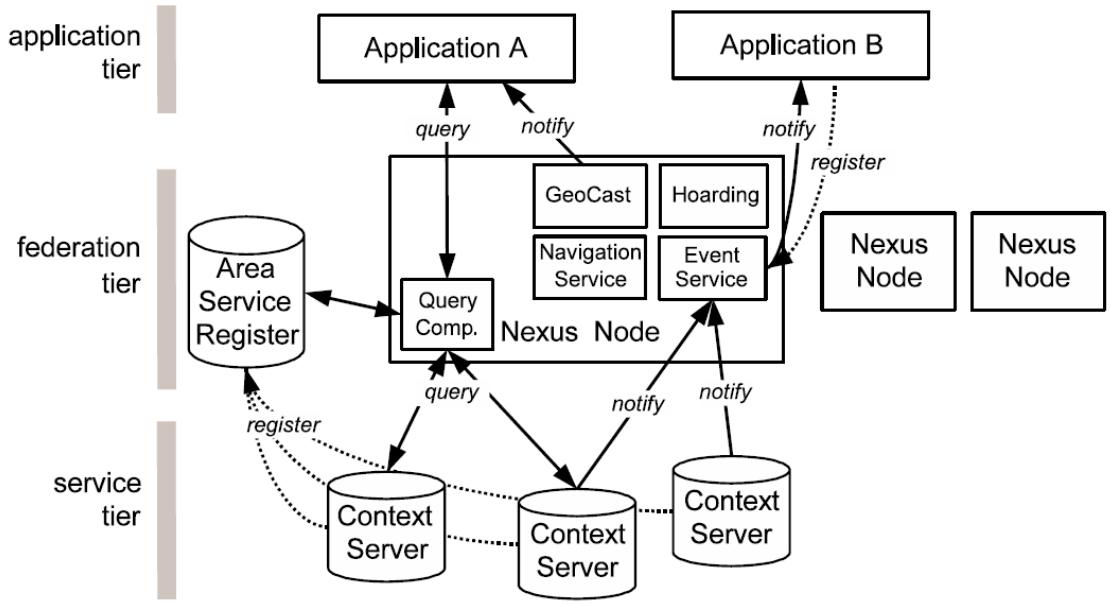

Figura 3.7 Plataforma Nexus (extraído de [DHN+04])

Os servidores de contexto (*context servers*) presentes na camada de serviço (*service tier*) armazenam os modelos locais. Para ser integrado a plataforma, os serviços devem seguir uma certa interface descrita em XML e registrados em um serviço chamado *Area Service Register* (ASR) para que possam ser descobertos dinamicamente.

A camada de federação (*federation tier*) funciona como um mediador entre as aplicações e os servidores de contexto. Ele possui a mesma interface dos servidores de contexto, mas não armazenam modelos. Ele analisa a consulta da aplicação, determina os servidores de contexto que podem responder a consulta e distribui a consulta para esses serviços. O nodo Nexus também possui a capacidade de adicionar serviços que possuem suas próprias interfaces e que usam os modelos de contexto para processar informação e fornecê-la para as aplicações. A Figura 3.7 mostra quatro tipos de serviços da plataforma Nexus:

- o serviço de evento (*event service*) monitora eventos espaciais e permite o processamento de predicados espaciais, tais como: “monitorar quando eu estiver próximo à dois amigos”;
- o serviço de navegação (*navigation service*) calcula a rota usando os dados disponíveis nos modelos locais;
- o *geocast* é responsável por enviar mensagens para todos os usuários em uma determinada área geográfica;
- o serviço *Hoarding* é responsável pelo processo de desconexão da aplicação de uma área de comunicação, ou seja, se uma aplicação está prestes a sair de uma área de comunicação de rede sem fio, o serviço *Hoarding* transfere as informações do contexto necessárias para essas aplicações com antecedência.

No topo da arquitetura estão as aplicações cientes de contexto que podem usar a plataforma de três formas diferentes:

1) as aplicações podem enviar consultas e obter informações sobre o ambiente ao redor, como por exemplo, PoI, amigos próximos, etc;
2) as aplicações podem registrar um evento para receber uma notificação quando um certo estado do mundo ocorrer;
3) as aplicações podem usar os serviços do nodo Nexus, como os serviços de navegação, para enriquecer as funcionalidades da aplicação.

A plataforma Nexus possui uma arquitetura semelhante ao padrão Web service [ACKM04]. Os servidores de contexto funcionariam como provedores de serviços que são registrados em um serviço de diretório; o ASR funcionaria como um serviço de diretórios semelhante ao UDDI no padrão Web service. Porém, na plataforma Nexus, a descoberta de serviços é realizada pelo nodo Nexus na camada de federação e não pelas aplicações como no padrão Web service.

A plataforma Nexus monitora o espaço físico, ou seja, o contexto do ambiente. O Nexus é capaz de transmitir anúncios para vários usuários em uma determinada área, mas não faz busca automática de produtos ou serviços de interesse do usuário. Além disso, a plataforma não é capaz de deduzir informação a partir dos dados do contexto.

3.8 ICAMS

ICAMS [NTT+04] é um sistema cliente-servidor que usa celulares para tornar a comunicação mais eficiente através de informações de localização e agenda dos usuários. O sistema usa o serviço de localização PHS (*Personal Handphone System*)

oferecida pela companhia de telecomunicações japonesa NTT DoCoMo. O celular atualiza a sua localização a cada 15 minutos e tem uma precisão de 100 metros. Basicamente, o ICAMS auxilia no redirecionamento de mensagens telefônicas ou e-mail.

Os usuários do sistema ICAMS são amigos que desejam compartilhar localização e outras informações. Quando um usuário acessa o servidor, um script PHP (*Hypertext Preprocessor*) consulta o banco de dados e retorna um arquivo CHTML chamado *TopPage*. Este arquivo apresenta os outros usuários do sistema (os amigos) em ordem de proximidade. Ícones e setas são usados para indicar se um amigo está se movendo e em que direção em relação ao usuário. Quando o usuário clica no nome de um amigo no *TopPage*, um segundo PHP consulta o banco de dados e retorna um CHTML chamado *MemberPage*. Essa página contém mais detalhes sobre aquele amigo: nome, o último local em que esteve, se o amigo está parado ou se movendo, a distância em relação ao usuário e as opções de contato (telefones, e-mails e outros). O usuário pode clicar no número de telefone ou no e-mail para estabelecer uma comunicação. Se o amigo, por exemplo, estiver em sua casa, o telefone residencial é listado primeiro no *MemberPage*. Mas se o amigo estiver numa reunião no trabalho, o seu e-mail de escritório será o primeiro na lista.

Através do Web browser do celular, o usuário pode entrar com sua programação (agenda) selecionando o dia, hora e conteúdo da programação. Em seguida, o usuário pode ordenar os contatos para a programação criada em ordem de preferência. Dessa forma, seus amigos saberão qual a melhor forma de entrar em contato para cada situação do dia.

Esse sistema possui algumas desvantagens:

- baixa precisão em relação à localização;
- as únicas informações do contexto que são analisadas pelo sistema são a localização do usuário e sua agenda;
- o redirecionamento das mensagens não é automático.

3.9 FieldMap

FieldMap [FM04] é uma ferramenta escrita em Java para mostrar mapas e permitir anexar comentários. Ela foi construída para ser usada em PDAs, desktops e laptops. O usuário pode adicionar novos pontos de interesse, desenhar no mapa, macar com pontos

e polígonos e adicionar comentários escritos ou por voz para ajudar a recordar sobre informações do ambiente.

Pode-se perceber que esta ferramenta não faz qualquer análise do contexto do usuário. Apenas a localização e o caminho percorrido pelo usuário são mostrados no mapa para que ele possa se orientar. Essa ferramenta é mais utilizada na área de pesquisa na qual o ambiente está sendo estudado, como, por exemplo, na área de arqueologia.

3.10 AMS

O AMS (*Arrhythmia Monitoring System*) [LML+04] é um trabalho na área de telemedicina com o objetivo de prever ataques cardíacos conhecidos como arritmia. O AMS coleta sinas de um ECG (eletrocardiograma) combinado com os dados de um GPS e os transmitem a uma estação remota para visualização e monitoração. A arquitetura do sistema é composta dos seguintes componentes:

- *wearable sever*: é um pequeno coletor de dados que o paciente fica conectado. Ele é composto por um ECG (eletrocardiograma) que coleta as atividades elétricas do músculo cardíaco através de biosensores;
- *central server*: é um pequeno servidor localizado próximo ao cliente. Ele executa várias funções: compressão, tratar sinais do GPS e detectar sinais de arritmia rudimentares. O *central server* é geralmente um dispositivo móvel como um Palm, que conectado ao *wearable server* e a um GPS, transmite esses sinais para o *call center*;
- *call center*: é através do qual um profissional médico monitora o sinal ECG e transmite um alerta caso detecte alguma situação de risco.

Os dados dos biosensores são armazenados em um buffer e a cada 20 segundos, esse buffer é transmitido para o *call center*. O sistema também possui um botão de "pânico" pelo qual o cliente pode enviar um alerta para o sistema central que se encarrega de chamar um serviço de emergência (190, por exemplo) passando a localização do usuário.

3.11 Comparação entre as Ferramentas

As ferramentas estudadas foram comparadas conforme as características funcionais de um sistema pervasivo capaz de analisar o contexto, tanto do usuário quanto do ambiente. As principais características usadas para comparar as ferramentas foram:

1) Monitoração em tempo real: para perceber alterações no contexto, é indispensável que as informações obtidas dos sensores sejam as mais atuais possíveis;
2) Monitoração de vários tipos de contexto: as ferramentas foram comparadas conforme a classificação do contexto descrita na Seção 2.2.1;
3) Capaz de monitorar o contexto de terceiros: além de monitorar o próprio contexto, um sistema ciente de contexto deve ser capaz de perceber o contexto de terceiros (parentes e amigos);
4) Sistema SIG onipresente: essa característica permite que o usuário visualize a sua localização, a localização de seus contatos e os PoI através de um mapa digital disponível em seu dispositivo móvel;
5) Análise do perfil do usuário para o fornecimento de mapas personalizados: através da análise do perfil do usuário, informações sobre suas preferências podem ser extraídas para construir um mapa com os PoI que estejam de acordo com esse perfil;
6) Anúncio de produtos ou serviços: esta característica faz parte do contexto do ambiente e permite com que o usuário receba anúncios de produtos ou serviços que estejam próximos de sua localização.
7) Orientado a serviço: no âmbito desse trabalho, essa característica não é tão importante quanto as anteriores, porém ela permite que novos serviços heterogêneos sejam adicionados a arquitetura de forma padronizada, acrescentando mais funcionalidades ao sistema.

A Tabela 3.1 mostra quais ferramentas possuem as características citadas anteriormente. Os campos marcados com ‘X’ indicam que a ferramenta possui a determinada funcionalidade. Os campos sem marcação indicam que a ferramenta não possui a funcionalidade especificada. Quando alguma ferramenta possui alguma funcionalidade, mas com restrições, essas restrições são descritas no campo correspondente na tabela.

Tabela 3.1: Comparativo entre os sistemas analisados.

Características / Sistemas	1	2	3	4	5	6	7
SOCAM	X	X	X				
Cybre Minder	X	X	X				
AROUND	X			X			
Online Aalborg Guide	X	Apenas localização e perfil		X	X		
Flame 2008	X	Apenas localização e perfil		X	X	Apenas por tipo do produto	X
Nexus	X	Apenas o contexto do ambiente	X	X			
ICAMS		Apenas localização e agenda		X			
FieldMap		Apenas a localização		X			
AMS	X	Apenas a localização e o batimento cardíaco					

3.12 Considerações Finais

Neste capítulo, foram descritos alguns sistemas cientes de contexto, como funcionam, suas principais vantagens e desvantagens. Em seguida, uma tabela comparativa mostra quais os sistemas apresentados possuem as funcionalidades desejáveis para um sistema ciente de contexto capaz de monitorar o contexto dos usuários e realizar pesquisas por produtos dinamicamente.

No Capítulo 4 será apresentado um breve estudo sobre as principais características das ontologias e no Capítulo 5 será mostrado como elas foram usadas para modelar os vários tipos de contexto.

Capítulo 4 . Representando Conhecimento Usando Ontologias

4.1 Introdução

Este capítulo descreve sobre ontologias, as linguagens usadas para expressar os termos da ontologia e seus relacionamentos. A seguir, é apresentado o funcionamento do *framework* Jena. Esta ferramenta ajuda a manipular os termos de uma ontologia usando a linguagem de programação Java.

4.2 Ontologias

Ontologia é um termo vindo da filosofia que se refere à ciência de descrever as entidades no mundo e como elas estão relacionadas. Em Ciência da Computação, as Ontologias foram desenvolvidas, primeiramente, em Inteligência Artificial para facilitar o compartilhamento e o reuso do conhecimento. Ontologias têm sido estudadas por vários ramos da Inteligência Artificial, como: engenharia do conhecimento, processamento de linguagem natural e representação do conhecimento. A principal razão das ontologias estarem se tornando popular é devido ao que elas prometem: um entendimento compartilhado e comum de um domínio que pode ser comunicado entre pessoas e sistemas de aplicações [DFH03].

Segundo Berners-Lee [BHL02], as ontologias fornecem um entendimento comum e compartilhado de um domínio, que pode ser comunicado através de pessoas e sistemas de aplicação, tornando-se um fator chave para o desenvolvimento da Web Semântica. A ontologia envolve uma taxonomia de conceitos (classes, subclasses e relações entre elas) e um conjunto de regras de inferência que permitem extrair conclusões do domínio de conhecimento, além disso, elas podem resolver problemas inerentes ao vocabulário da linguagem natural tais como homonímia, sinônima e meronímia.

Dentre as principais linguagens de definição e instanciação de ontologias estão o RDFS (RDF Schema) [W3C06a] e o OWL (Web Ontology Language) [W3C04]. RDFS é uma extensão semântica do RDF (Resource Description Framework) que é um padrão

para descrever recursos na Web. A estrutura de qualquer expressão RDF é uma coleção de triplas, cada uma formada por um sujeito, um predicado (também chamado de propriedade) e um objeto. O conjunto de triplas é chamado de grafo RDF.

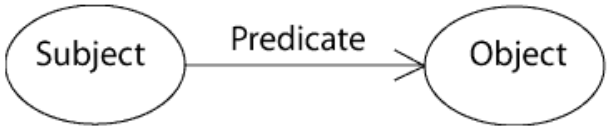

Figura 4.1: Formação de um tripla no RDF

As propriedades do RDF podem ser pensadas como atributos de recursos no sentido tradicional *atributo-valor*. Propriedades em RDF também podem representar relações entre recursos. Porém, o RDF não provê mecanismos para descrever essas propriedades e nem qualquer mecanismo para descrever a relação entre propriedades e recursos. RDFS (RDF Schema) define classes e propriedades que podem ser usadas para descrever e dá significado aos recursos e propriedades do RDF [W3C06b]. No RDFS, recursos são separados em grupos chamados de classes e os membros de cada classe são chamados de instâncias.

Duas importantes propriedades do RDFS são os *rdfs:domain* e o *rdfs:range*. O *rdfs:domain* é usado para informar que qualquer recurso que tem uma dada propriedade é instância de uma ou mais classe especificada pelo *rdfs:domain*. Por exemplo, supondo que exista uma classe *Estado* e uma classe *Governador*, o fragmento de código a seguir descreve que a propriedade *governado_por* tem como domínio as instâncias da classe *Estado* e os objetos dessa propriedade são instâncias da classe *Governador*:

```
<rdf:Property rdf:about="#governado_por">
  <rdfs:domain rdf:resource="#Estado"/>
  <rdfs:range rdf:resource="#Governador"/>
</rdf:Property>
```

Além dessas propriedades, o RDFS possui outras que aumentam a capacidade semântica do RDF, como a *rdfs:subClassOf* e *rdfs:subPropertyOf*, que permite que classes e propriedades possam ter subclasses e subpropriedades.

O DAML+OIL [ACKM04] é uma linguagem que estende RDF e RDFS. Ela possui um rico conjunto de termos usados para descrever e representar um conhecimento do domínio. Ela permite dizer tudo o que o RDFS permite e algo mais. Porém, logo mais tarde, como resultado de uma revisão do DAML+OIL, no qual

algumas construções foram removidas e outras acrescentadas, surgiu a linguagem OWL [ACKM04].

Devido ao seu poder de expressão, a linguagem OWL foi usada na arquitetura Omnipresent para descrever as entidades do contexto, seus relacionamentos e deduzir conhecimento através de regras de inferência. A principal vantagem da OWL é que ela permite descrever características das classes e propriedades que antes não eram possíveis com as outras linguagens [HP06]. Por exemplo, OWL permite dizer se uma propriedade é transitiva, simétrica, funcional ou inversa.

Uma propriedade P é dita transitiva se P(x, y) e P(y, z) implica que P(x, z), nas quais x, y e z são instâncias de uma classe da ontologia. Por exemplo, em OWL, é possível descrever uma propriedade transitiva chamada *localizado_em* e informar que a instância *Paraíba* é um Estado localizado na região Nordeste e que a cidade Campina Grande está localizada no estado da Paraíba. Como a propriedade é transitiva implica dizer então que a cidade Campina Grande também está localizada na região Nordeste. O código OWL a seguir mostra como esse exemplo pode ser descrito.

```
<owl:ObjectProperty rdf:ID="localizado_em">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdfs:domain rdf:resource="&owl;Thing" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>

<Region rdf:ID="Paraiba">
  <locatedIn rdf:resource="#Nordeste " />
</Region>

<Region rdf:ID="Campina_Grande ">
  <locatedIn rdf:resource="#Paraiba " />
</Region>
```

Informando que a propriedade é transitiva

Uma propriedade P é dita simétrica se P(x, y) implica em P(y, x). Por exemplo, se uma cidade A é vizinha de uma cidade B então a cidade B também é vizinha da cidade A.

Se uma propriedade P é funcional, então P(x, y) e P(x, z) implica dizer que y é igual a z. Essa propriedade é usada para informar que uma propriedade de uma instância só pode assumir um único valor.

Se duas propriedades P1 e P2 são inversas, então P1(x, y) implica em P2(y, x), por exemplo, as propriedades *governar* e *governada_por* são declaradas inversas no

código a seguir e quando é declarado que o Estado de São Paulo é governado por Geraldo Alckmin, implica dizer também que Geraldo Alckmin governa o Estado de São Paulo.

```
<owl:ObjectProperty  rdf:ID=”governa”>
    <owl:inverseOf    rdf:resource”#governadaPor”>
</owl:ObjectProperty>

<Region rdf:ID="Sao_Paulo">
 <governado_por rdf:resource="#Geraldo_Alckmin " />
</Region>
```

Pode ser inferido que Geraldo_Alckmin governa o Estado de São Paulo

Além dessas características sobre as propriedades, OWL apresenta várias restrições sobre as classes e propriedades que enriquece as descrições na ontologia. Por exemplo, classes ou propriedades podem ser declaradas como equivalentes a outras classes ou propriedades, como por exemplo, uma classe *Carro* pode ser declarada como equivalente a classe *Automóvel*. As classes também podem ser declaradas como disjunta, complemento, união ou intersecção de outras classes, algo que não era possível nas outras linguagens.

A linguagem OWL é dividida em três tipos:

- OWL Lite: tipo mais simples de OWL na qual todos os relacionamentos tem cardinalidade 0 ou 1;
- OWL DL: permite o máximo de expressividade sem perder a decidabilidade, ou seja, tudo que é representado pode ser entendido e decidido por uma máquina. Ele inclui todas as restrições e cardinalidade da linguagem OWL;
- OWL Full: é o tipo de OWL que tem o maior poder de representação, embora não se tenha garantia de que o que ela seja representa computacionalmente. Por exemplo, em OWL Full, uma classe pode assumir o papel de instância em alguns momentos e vice-versa.

Para auxiliar na construção e manipulação de ontologias descritas em RDFS, DAML+OIL e OWL, a *Hewlett-Packard Development Company* (HP) desenvolveu um *framework* para ajudar os desenvolvedores a inserir, alterar ou remover recursos da ontologia. Essa ferramenta é descrita na seção seguinte.

4.3 Framework Jena

Jena é um *framework* para Web semântica desenvolvido em Java pela HP [HP06]. Ele provê um conjunto de APIs para lidar com documentos RDF, RDFS, DAML+OIL e OWL, permitindo trabalhar com os arquivos em memória ou persisti-los em um banco de dados.

Cada arco no grafo que representa a ontologia é representado como um *statement*. O *statement* representa um fato na ontologia e contém as partes: sujeito, predicado e objeto. Os *statements* podem ser usados para fazer consultas na ontologia, como por exemplo, listar todas as subregiões localizadas na região Nordeste:

listStatements (null, localizado_em, Nordeste)

Neste exemplo, quando o Jena listar os resultados, ele irá levar em consideração as características das propriedades (simétrica, funcional, etc), dessa forma, as subregiões também serão listadas no resultado da função.

O Jena possui um importante subsistema de inferência que permite extrair novos conhecimentos da ontologia. Ele permite que novos fatos sejam inferidos a partir das instâncias e descrições das classes. No Jena, o código que executa a tarefa de extrair informação adicional da ontologia é chamado de *reasoner*. Novos *reasoners* podem ser adicionados a arquitetura do Jena, porém ele já inclui os seguintes *reasoners*:

1. *transitive reasoner*: trata apenas das propriedades simétricas e transitivas. Por ser mais leve, ele apresenta uma maior performance e uso mais eficiente de memória;
2. *RDFS reasoner*: suporta quase toda a especificação RDFS;
3. *DAML micro reasoner*: um incompleto *reasoner* sobre o DAML que provavelmente não será aperfeiçoado com as próximas versões do Jena;
4. *OWL reasoner*: um dos mais importantes *reasoners* do Jena. Devido a complexidade do OWL, este *reasoner* é dividido em três tipos: *micro*, *mini* e *full*. Dependendo da performance desejada para a aplicação, o desenvolvedor pode optar por um dos três tipos do *OWL reasoner*, como por exemplo, o *OWLMicro* suporta o RDFS mais as características de intersecção e união, porém, ele não trabalha com as restrições de cardinalidade. Apesar disso, o *OWLMicro*, por ser mais leve, é o que possui maior performance comparado com os outros tipos de *OWL reasoner*. O *OWL mini* suporta um conjunto maior de construções OWL, porém em algumas delas, ele possui algumas

restrições, como por exemplo, ele suporta restrições de cardinalidade apenas com valores 0 ou 1. Por último, existe o *OWL full* que suporta a maioria das construções OWL;

5. *generic rule reasoner*: *reasoner* baseado em regras. Pode ser usado para implementar *RDFS* ou *OWL reasoner*, mas também para uso geral. Este é o *reasoner* usado pelo Omnipresent para adicionar novas regras de inferência e novas primitivas.

Neste *reasoner* baseado em regras, estas últimas são definidas a partir de uma seqüência de termos que forma o corpo da regra e outra seqüência que compõe a conclusão. As regras podem ser adicionadas ao *framework* através da classe *Rule* ou editadas num arquivo para que o *reasoner* possa extraí-las.

De acordo com o BNF da Figura 4.2, uma regra pode ser descrita apenas por uma seqüência de termos (*bare-rule*), em que a conclusão é apontada por uma seta (representada pelos caracteres '->' e '<-').

```
Rule       :=   bare-rule .
           or   [ bare-rule ]
                or   [ ruleName : bare-rule ]

bare-rule  :=   term, ... term -> hterm, ... hterm
           or   term, ... term <- term, ... term

hterm      :=   term
           or   [ bare-rule ]

term       :=   (node, node, node)
           or   (node, node, functor)
           or   builtin(node, ... node)

functor    :=   functorName(node, ... node)

node       :=   uri-ref
           or   prefix:localname
           or   ?varname
           or   'a literal'
           or   'lex'^^typeURI
           or   number
```

Figura 4.2 Uma descrição informal da construção de regras (extraído de [HP06])

Uma regra pode ser descrita entre colchetes e também pode conter um nome para que possa ser identificada. Isso é útil quando várias regras são adicionadas à ontologia e futuramente, seja preciso consultá-las ou removê-las. A conclusão de uma regra (*hterm*) pode ser composta por outras regras de inferências. Os termos da regra (*term*) podem

ser formados por um *statement* ou uma chamada a uma função. Os parâmetros da função e os componentes do *statement* (sujeito, relacionamento e objeto) são formados por nodos. Um nodo pode ser uma URI, uma variável indeterminada (representada por '*?varname*'), um literal, um número ou um valor da variável seguido pelo tipo, por exemplo:

"valor"^^http://www.w3.org/2001/XMLSchema#string.

Uma função usada na regra é chamada de primitiva. O Jena possui uma coleção de primitivas que podem ser úteis na realização de algumas comparações necessárias na regra, como por exemplo: *lessThan()*, *greaterThan() e equal()*.

As regras podem ter três tipos de encadeamento:

- *forward-chaining*: a execução de regras desse tipo é baseada no padrão RETE [For82], o qual executa todas as regras de uma só vez, acumulando-se as conclusões geradas, de tal maneira que futuras consultas a uma regra obterão respostas imediatas. Exemplo: a regra a seguir diz que se uma instância *?f* é pai da instância *?a* e uma instancia *?u* é irmão de *?f*, então significa que *?u* é tio de *?a*;

[rule1: (?f ont:pai ?a) (?u ont:irmao ?f) -> (?u ont:tio ?a)]

- *backward-chaining*: usa um mecanismo de lógica de programação semelhante ao Prolog. Exemplo: a regra a seguir descreve o relacionamento entre subclasses.

[r1: (?A rdfs:subClassOf ?C) <- (?A rdfs:subClassOf ?B) (?B rdfs:subClassOf ?C)]

Segundo a documentação do Jena não existe diferença entre o *backward-chaining* e o *forward-chaining* (além do algoritmo de inferência), a conclusão se encontra sempre no final da flecha;

- execução híbrida: é a combinação do *forward-chaining* e o *backward-chaining*.

Neste projeto, foi necessário acrescentar algumas primitivas para auxiliar em algumas regras. Por exemplo, a regra a seguir deduz que um usuário está próximo a outro usando uma primitiva chamada *near* que calcula a distância euclidiana entre os dois usuários.

[near: (?u1 ont#near ?u2) <- (?u1 ont#located_at ?l1) (?u2 ont#located_at ?l2) near(?l1, ?l2)]

As primitivas são implementadas em Java e registradas no *framewok* para que ele possa reconhecer e permitir que sejam usadas nas regras. A primitiva deve retornar um valor booleano, pois a conclusão da regra é inferida se todas as premissas forem verdadeiras.

4.4 Considerações Finais

Este capítulo discorre sobre ontologias e como elas podem ser usadas para representar e extrair conhecimento de um domínio. A arquitetura Omnipresent usa a linguagem OWL para descrever a ontologia e o *reasoner* de propósitos gerais para processar as regras de inferência usadas para inferir novos conhecimentos.

No próximo capítulo, será abordado como as ontologias foram usadas para representar o contexto na arquitetura Omnipresent, além de mostrar como a arquitetura foi implementar para analisar o contexto e oferecer funcionalidades SIG, como mapas e roteamento.

Capítulo 5 . Omnipresent – Um Sistema para Aplicações Cientes de Contexto

5.1 Introdução

Este capítulo discorre sobre a modelagem de contexto usando ontologias e sobre a arquitetura do sistema Omnipresent, contendo discussões a respeito dos principais elementos da arquitetura, compostas pelos Web services, as aplicações Web e os tipos de clientes que podem acessar e usufruir das funcionalidades que o sistema oferece. Primeiramente, serão abordados os principais requisitos do sistema.

5.2 Requisitos do Sistema

5.2.1 Requisitos Funcionais

Monitoração em tempo real

Como se trata de um sistema de análise de contexto, é fundamental que os dados analisados sejam os mais atuais possíveis. Porém, atualização constante de vários usuários simultaneamente pode sobrecarregar o sistema. Uma das formas de se evitar essa sobrecarga é atualizando os dados apenas quando eles sofrem uma grande alteração. Como por exemplo, quando o dispositivo obtém uma nova coordenada do GPS e esses dados não apresentam uma diferença significativa em relação às coordenadas anteriormente obtidas (por exemplo, 10 metros), elas não precisam ser atualizadas no *LBS Service*.

Permitir registrar e remover lembranças que são ativadas por contexto

O cliente descreve o contexto em que a lembrança deve ser entregue através de um documento XML e a qualquer momento a regra pode ser removida pelo usuário. Uma outra opção é deixar a regra como inativa, ou seja, a regra existe na ontologia, mas o usuário pode optar por não querer receber mensagens quando o contexto for satisfeito, mas futuramente ele pode mudar de idéia e ativar novamente a regra. Por exemplo, um

usuário deseja monitorar o seu filho para se certificar que ele esteja mesmo na escola, mas durante as férias escolares, não faz sentido monitorar constantemente a localização da criança. Portanto, a regra pode ser marcada como inativa durante as férias e ativada novamente no início das aulas.

Monitorar contexto do usuário e do ambiente

O *LBS Service* deve ser capaz de monitorar o contexto do usuário e do ambiente ao seu redor. Atualmente, as informações de contexto do usuário que são monitorados são: emoção, estado fisiológico (batimento cardíaco e temperatura), estado atual (trabalhando, livre, dormindo, etc) e localização. O usuário é capaz de monitorar o próprio contexto ou o contexto de seus contatos. Através do contexto de localização, o sistema deverá ser capaz de monitorar quando dois usuários estiverem próximos ou quando o usuário estiver próximo a algum PoI.

Mapa personalizado

Além do mapa possuir os pontos de interesse personalizados de acordo com as preferências do usuário, ele apresenta os contatos próximos e informações de produtos e serviços desejados. A carga do mapa deve acompanhar o movimento do usuário, ou seja, quando ele estiver próximo do limite do mapa que está sendo apresentado no dispositivo, um novo mapa deve ser consultado do *Presentation Service*.

5.2.2 Requisitos Não Funcionais

Arquitetura Orientada a Serviço

Web Services [ACKM04] fornecem uma maneira estruturada de formatar dados, de tratar transações e um padrão para descrever o que o serviço faz e como torná-lo acessível a outros serviços. Um Web Service é uma interface que descreve uma coleção de operações acessíveis via rede através de troca de mensagens padronizadas em XML, que é a linguagem padrão de comunicação na Web. A interface esconde a implementação do serviço, permitindo que seja usado independentemente de plataforma, favorecendo a comunicação entre sistemas heterogêneos. Web services também auxiliam no processo de desenvolvimento de software, pois permite que funcionalidades distintas sejam desenvolvidas por equipes diferentes, agilizando o

desenvolvimento do sistema e permitindo que os serviços sejam reaproveitados em várias aplicações.

Resposta em tempo razoável

O tempo de resposta rápido é fundamental principalmente em sistemas na área de telemedicina na qual cada segundo é importante para a vida do paciente. Na arquitetura Omnipresent, o processo de apresentação do mapa, construção dos pontos de interesse e contatos do usuário no mapa ainda é um pouco demorado devido ao complexo processo de construção do mapa no dispositivo. Porém, as informações referentes ao estado fisiológico, devido a sua importância, devem possuir, no máximo, um atraso de 20 segundos.

Com bases nesses requisitos, as seções seguintes descrevem como a arquitetura foi implementada para que eles fossem atendidos.

5.3 Modelando o Contexto na Arquitetura Omnipresent

O modelo do contexto é baseado em ontologias usando OWL. Seguindo a categorização de contexto apresentado na Seção 2.2.1, a ontologia da arquitetura Omnipresent (apresentada na Figura 5.1) é organizada da seguinte forma:

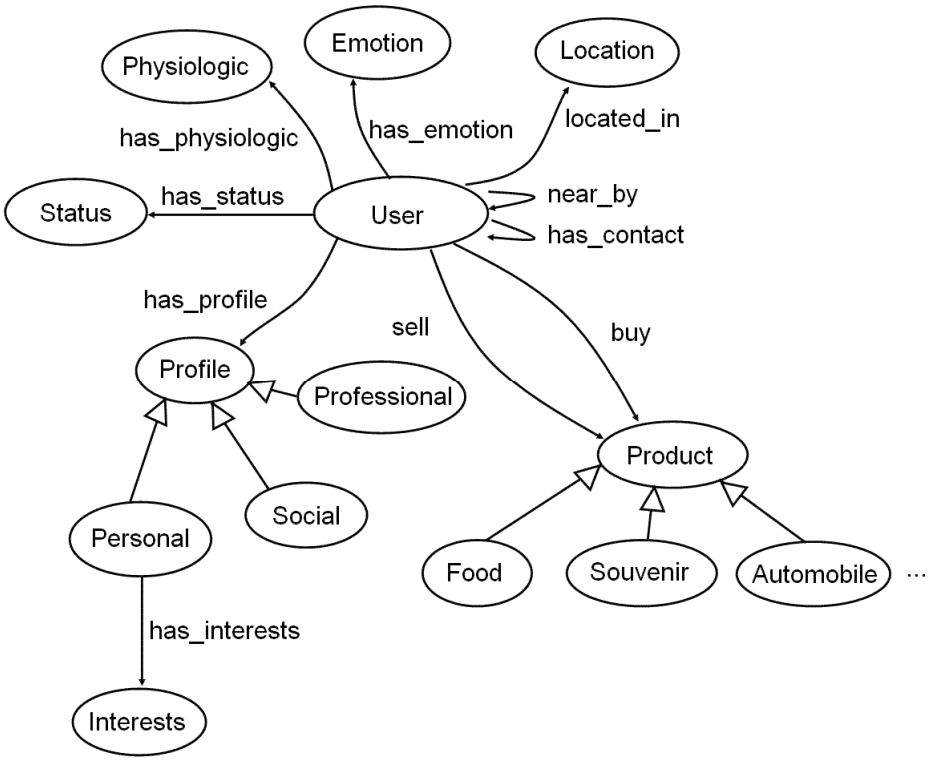

Figura 5.1 Modelagem do contexto usando ontologia

- *Perfil*: é representado pela classe *Profile*, que contém o perfil do usuário. Este perfil é classificado em três categorias: pessoal, profissional e social. O perfil pessoal contém informações sobre o usuário em si, como por exemplo, data de nascimento, estado civil, os tipos de PoI que ele deseja visualizar no mapa entre outras informações subjetivas. O perfil profissional contém informações acerca das atividades de trabalho, tais como nome da empresa, área e e-mail de trabalho. O perfil social contém informações sobre o comportamento social do usuário, tais como, religião e freqüência com que costuma beber ou fumar. A classe *Interests* representa os interesses do usuário. Esses interesses são relacionados entre si de forma a permitir que o sistema execute busca semântica e aproximada de produtos ou PoI. Por exemplo, considere a ontologia da Figura 5.2, se o usuário desejar localizar os restaurantes japoneses mais próximos, mas estes não são encontrados, o sistema pode fazer a busca pelo elemento mais próximo, neste caso por restaurantes chineses;

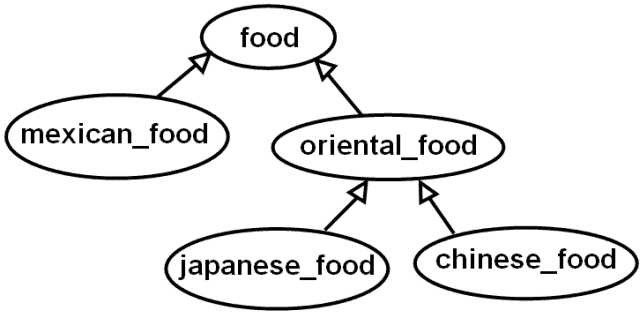

Figura 5.2 Exemplo de uma arquitetura de interesses do usuário

- *Comportamento Dinâmico*: é representado pela classe *Status* e pelas propriedades *sell* e *buy*. A classe *Status* indica a situação do usuário no momento atual, como por exemplo, trabalhando, dormindo, livre, almoçando, entre outros. As propriedades *sell* e *buy* indicam o interesse do usuário em vender ou comprar algum produto. Neste caso, a classe *Product* se refere tanto a um produto quanto a um serviço que é oferecido. O interesse do usuário em algum produto pode ser deduzido a partir de uma regra de inferência;
- *Estado Fisiológico*: é representado pela classe *Physiologic*. Essa classe contém informações sobre o estado fisiológico tais como batimento cardíaco, temperatura e pressão sanguínea. Limites inferiores e superiores podem ser definidos para cada tipo de estado fisiológico, dessa forma, se o valor ultrapassar esses limites, um alerta ou e-mail pode ser enviado para um serviço de emergência ou parente próximo;

- *Estado Emocional*: é representado pela classe *Emotion* que indica o estado emocional do usuário: triste, feliz, nervoso, mal-humorado, entre outros;
- *Ambiente Físico*: representado pela classe *Location*. Essa classe contem informações acerca da localização do usuário. Através dela é possível deduzir quais contatos e PoIs estão próximos de um usuário;
- *Ambiente Social*: informações sobre o ambiente social são representadas pelos relacionamentos *near_by*, *has_contact* e pelas regras que ativam os anúncios de produtos ou serviços quando o contexto é satisfeito. A propriedade *has_contatc* é dividida em subpropriedades que definem o tipo de relacionamento entre os usuários, como por exemplo: amigos, pai, mãe, filho, primo.

Várias regras de inferência sobre a ontologia podem extrair novas informações de contexto, por exemplo, uma regra de inferência pode deduzir que um usuário está doente baseado no seu estado fisiológico. Porém, essas regras devem ser inseridas no Jena no momento em que o *reasoner* é instanciado, para que este possa criar um grafo de inferências. Essa restrição dificulta a inserção e remoção das regras enquanto o sistema está em funcionamento. Portanto, para extrair informação de contexto de mais alto nível, as regras são adicionadas no sistema antes de iniciá-lo.

5.4 A arquitetura

O foco da arquitetura Omnipresent é oferecer serviços conforme o contexto do usuário e do ambiente no qual está inserido. Dados sobre o contexto são transmitidos para o sistema que se encarrega de analisá-los e apresentar informações de relevância para o usuário. Devido à grande variedade de serviços que devem estar disponíveis, foram desenvolvidos vários Web services para atender às requisições do usuário. Dessa forma, diferentes requisições podem ser processadas em paralelo, melhorando a eficiência do sistema.

Conforme pode ser visto na Figura 5.3, a arquitetura Omnipresent é dividida em três níveis distintos: clientes, aplicação Web (*Web application*) e Web services. Esses níveis são detalhados nas seções a seguintes.

5.4.1 Clientes

Os serviços apresentados na camada Web services podem ser acessados por um browser (através de uma aplicação Web) ou por um dispositivo móvel contendo um pacote J2ME Web services. O dispositivo móvel acessa diretamente os serviços como um

cliente Web service. Para criar aplicações para os dispositivos móveis foi utilizado o *J2ME Wireless Toolkit*. A versão 2.2 dessa ferramenta é baseada no *Information Device Profile* (MIDP) 2.0 que possui suporte a Web services. O *J2ME Wireless Toolkit* possui uma ferramenta JAX-RPC chamada *Sub Generator*. Essa ferramenta é usada para produzir *stubs* (ou proxy) que podem ser usados por uma aplicação no dispositivo móvel para fazer chamadas remotas a um Web service. O *Stub Generator* tem como entrada um arquivo WSDL e o nome do pacote no qual as classes Java são geradas. Essas classes são necessárias para que o desenvolvedor trabalhe apenas com código Java, evitando que ele tenha que descer de nível e tratar de conexões de rede e mensagens SOAP. A Figura 5.4 mostra um exemplo de como gerar um *stub*. Neste caso, um *stub* do *Presentation Service* está sendo gerado para que o dispositivo possa consultar mapas a esse serviço. Temos como parâmetro de entrada o arquivo WSDL do *Presentation Service* localizado na URL *http://buchada.dsc.ufcg.edu.br:9191/pService/presentationService*. As classes Java serão salvas no pacote *pService* no diretório *C:\WTK22\apps \omnipresent\src*.

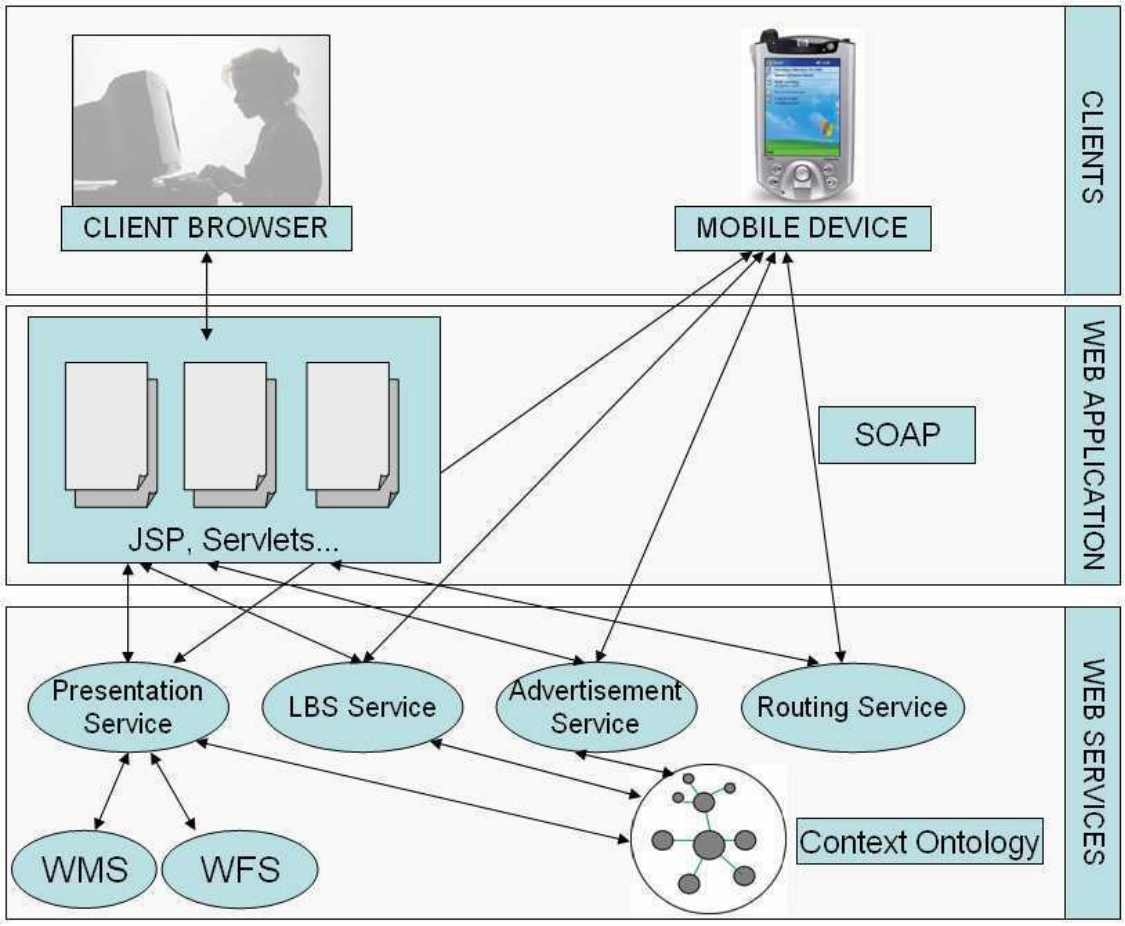

Figura 5.3 A arquitetura do Omnipresent

Com o *stub*, a aplicação Java acessa o Web service semelhante a uma chamada RMI local. Todas as chamadas remotas são mapeadas em chamadas locais para objetos *stubs*. Para acessar todos os serviços, o dispositivo móvel deve possuir o *stub* de cada um deles.

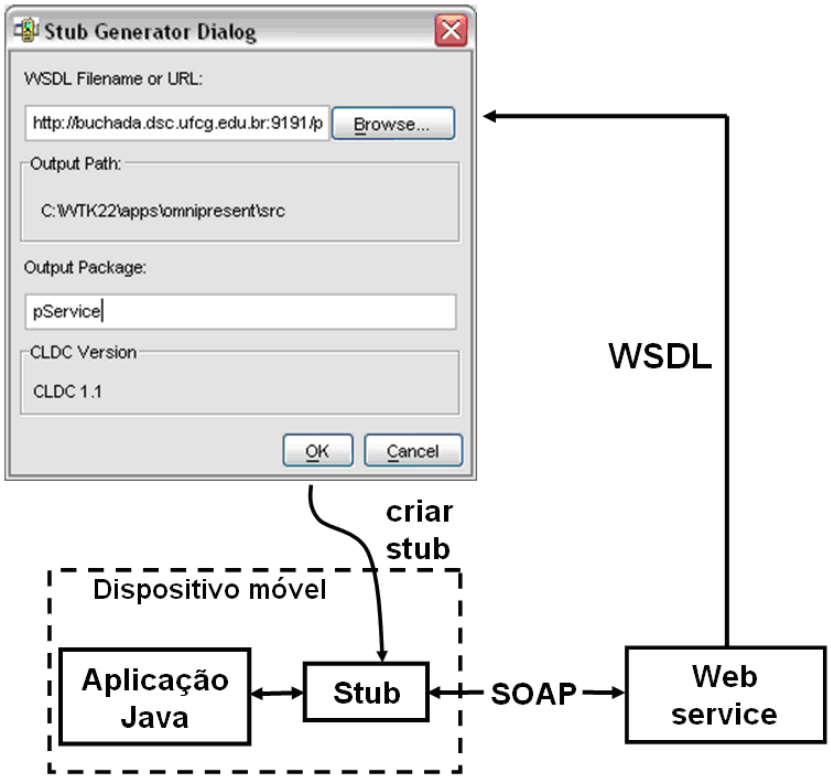

Figura 5.4: Gerando um *stub* com o *Stub Generator*

Conforme pode ser visto na Figura 5.5, a aplicação Java no dispositivo móvel é composta por dois simuladores: GPS e *Physiologic*.

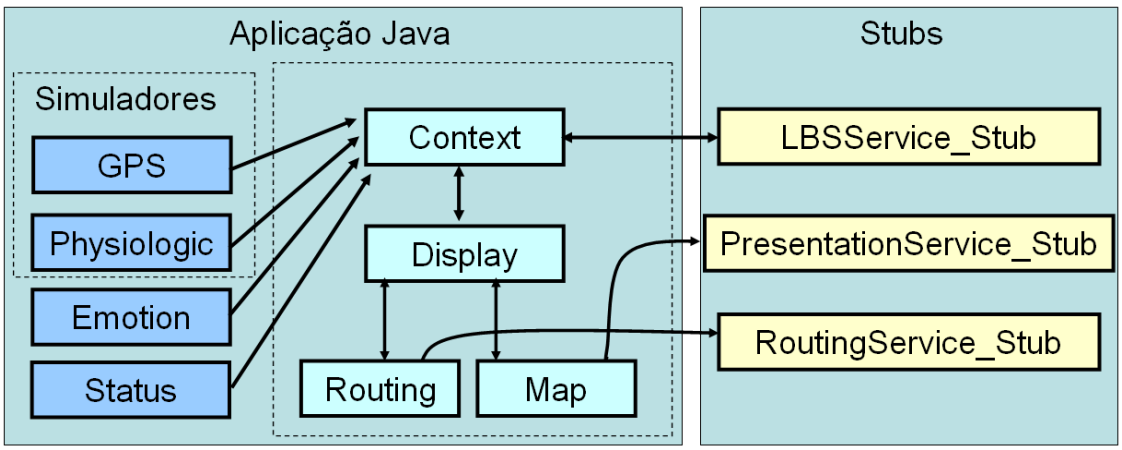

Figura 5.5 Arquitetura da aplicação Java no dispositivo móvel

O simulador de GPS irá fornecer alguns pontos representando o movimento do usuário do dispositivo. *Physiologic* é o simulador do estado fisiológico. Ele fornece

alguns dados representando o usuário em certas situações, como, febre, parada cardíaca ou pressão alta, para testar os casos em que o cliente corre perigo de vida. Além dos simuladores, existem também os componentes *Emotion* e *Status* que representam a emoção e a atividade do usuário, respectivamente. O componente *Emotion* apresenta na tela do dispositivo um conjunto de emoções no qual o usuário pode escolher aquela que define seu humor atual. Da mesma forma, o componente *Status* apresenta ao usuário, um conjunto de possíveis situações que ele esteja executando, como por exemplo: trabalhando, livre, almoçando, dormindo, etc. A Figura 5.6 mostra como o usuário define a emoção e o status através do dispositivo móvel. Para obter a lista de possíveis emoções e status, é necessário que o dispositivo se conecte ao *LBS Service* para carregar esses valores. Isso permite que novas emoções e status sejam adicionados no Web service sem que seja necessário fazer alguma modificação no dispositivo. O objetivo desses componentes é simular o comportamento de sensores na sua tarefa de coletar dados do contexto.

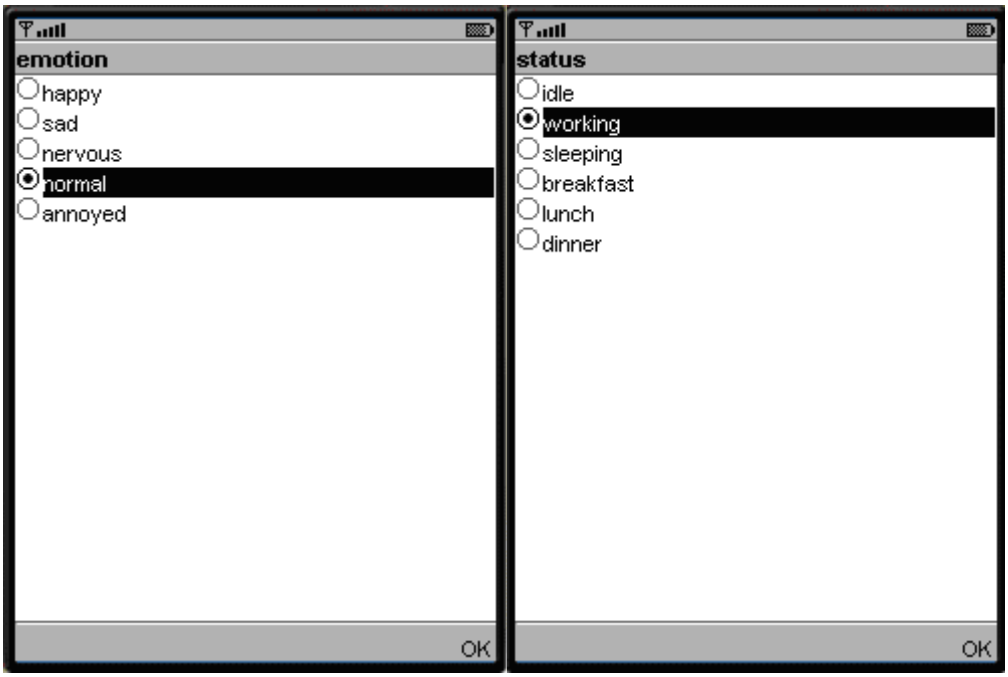

Figura 5.6 Usuário do dispositivo móvel selecionando a emoção e o status

As threads *Context*, *Routing* e *Map* são responsáveis por consultar os Web services e atualizar os dados na tela do dispositivo. A thread *Context* se encarrega de obter os dados dos sensores (simuladores), atualizar o mapa com a posição atual do usuário e transmitir os dados dos sensores para o *LBS Service*. Esses dados são também atualizados no mapa pela thread *Context*. A thread *Routing* é responsável por consultar o *Routing Service* para calcular a rota até um PoI ou um amigo próximo. Essa thread

também é responsável por desenhar a rota no mapa. Por fim, a thread *Map* tem a função de calcular o tamanho do mapa a ser exibido, consultar o *Presentation Service* e enviar o mapa recebido para o componente *Display* para que ele se encarregue de desenhar o mapa na tela do dispositivo. Atualmente, apenas mapas no formato *Scalable Vectorial Graphics Tiny* (SVGT) são carregáveis no dispositivo móvel. SVGT [W3C05] é uma linguagem XML para descrever, de forma vetorial, desenhos e gráficos bidimensionais da mesma forma que a linguagem SVG, porém, com menos recursos para que possa ser suportada por dispositivos com poucos recursos computacionais.

Quando o dispositivo transmite os dados dos sensores para serem atualizados no *LBS Service*, este responde com um objeto *Response* contendo os alertas, contatos, áreas de monitoramento e lugares próximos. A Figura 5.7 apresenta o diagrama de classe da resposta do *LBS Service* para o dispositivo móvel. A classe *Alert* contém um alerta que deve ser mostrado no dispositivo. Ele é composto pelo identificador, título e pela mensagem do alerta. O relacionamento *nearUsers* são os contatos do usuário do dispositivo e *nearPlaces* são os pontos de interesses próximos. Ambas as informações são compostas pela classe *GeoObject* que contém a coordenada e o nome do objeto espacial. A classe *Area* contém a área espacial a ser monitorada pelo dispositivo. Essa classe encapsula a área (ponto central e raio) que representa um determinado local que é monitorado por um usuário. Por exemplo, um usuário deseja monitorar quando seu filho sai da escola. A área dessa escola é representada por um objeto da classe *Area*, o qual será transmitido para o dispositivo da criança. O próprio dispositivo irá se encarregar de verificar quando a sua posição geográfica estiver fora da área especificada.

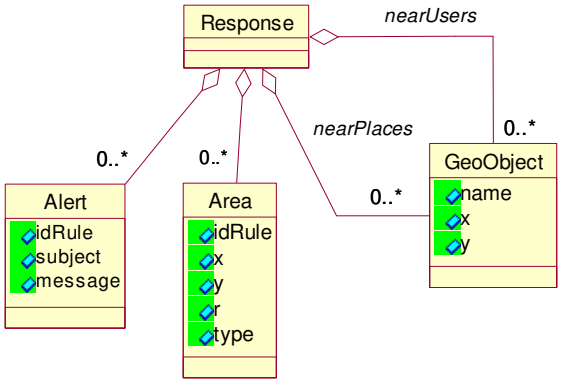

Figura 5.7 Diagrama de classe da resposta enviada para o dispositivo móvel

5.4.2 Aplicação Web (*Web Application*)

Este nível contém as aplicações Web que acessam os serviços e processam as informações para que os usuários possam usufruir dos Web services através de um *browser*. O principal objetivo dessa camada é permitir que os usuários possam acessar os serviços através de um PC e não apenas de um dispositivo móvel. Além disso, usando um *browser*, o usuário pode executar tarefas que seriam inadequadas para um dispositivo móvel, como por exemplo, o registro de lembranças ou o cadastro de produtos para a venda. Devido aos limites desses aparelhos, o fornecimento de dados como nome, e-mail, descrições, entre outras, seriam bastante demorados e incômodos para o usuário.

Da mesma forma que a aplicação Java no dispositivo móvel, a aplicação Web deve conter todos os *stubs* para trocar informações com os serviços. A aplicação Web desenvolvida é baseada no Padrão de Projeto *Model-View-Controller* (MVC). O propósito desse padrão é separar dados e interface para produzir um sistema mais desacoplado, permitindo múltiplas visões da mesma informação. Esse projeto divide a aplicação em três partes:

- *data model*: contém a parte computacional do programa. O modelo de dados (*data model*) é responsável pelo acesso direto aos dados, neste caso, acesso aos Web services através dos *stubs*;
- *view*: trata da apresentação dos dados ao usuário. Aqui estão as páginas JSP que montam a interface gráfica pelo qual o usuário pode interagir com os Web services;
- *controller*: são *servlets* que recebem o *request* do cliente e decidem que ação de negócio (*data model*) será executada e qual a próxima interface será apresentada ao usuário através do *view*.

Através da aplicação Web, o usuário pode descrever seu perfil e lembranças, encontrar contatos, receber alertas, visualizar mapas e fazer uma busca de produtos ou serviços. Essas funcionalidades são oferecidas pelos Web services. A aplicação Web apenas apresenta uma interface agradável para que o usuário possa preencher esses dados.

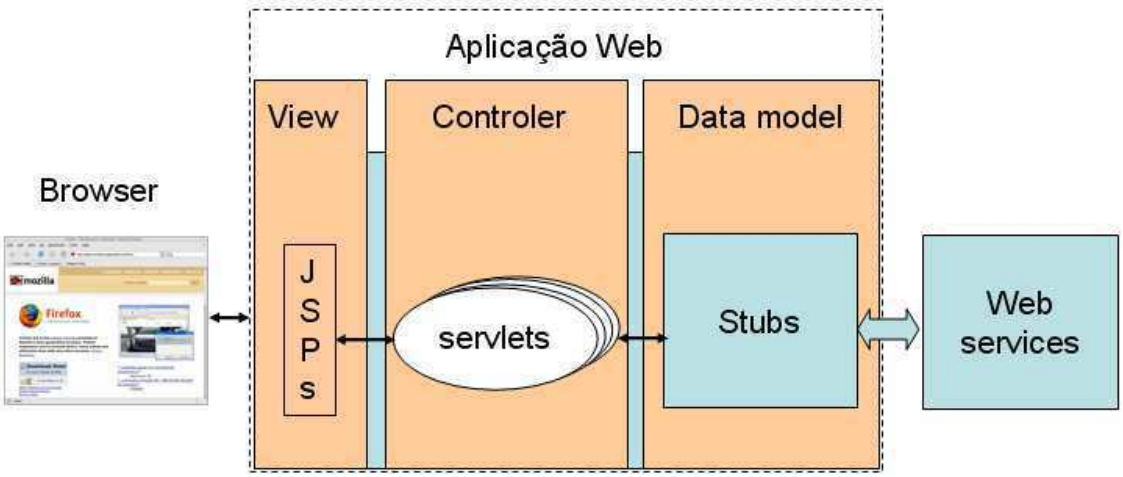

Figura 5.8 Aplicação Web

5.4.3 Web Services

Este nível agrupa os serviços utilizados pela arquitetura, bem como a modelagem do contexto representada pela ontologia. O sistema possui atualmente quatro Web services principais: *Presentation Service*, *LBS Service*, *Advertisement Service* e *Routing Service*. Todos os Web services foram implementados usando o *Java Application Server* da Sun [Sun05]. Esta plataforma possibilita o rápido desenvolvimento de aplicações J2EE, além de possuir um servidor para rodar essas aplicações. Uma grande vantagem do *Application Server* é que ele possibilita o desenvolvimento de Web services sem que o desenvolvedor precise conhecer detalhes da implementação da interface WSDL ou da transferência de dados através de mensagens SOAP. A ferramenta *wscompile*, disponibilizada pelo *Application Server* consegue transformar uma interface Java em um arquivo WSDL e vice-versa. A troca de mensagem SOAP entre o Web service e o cliente é feita de forma transparente pelo servidor. Portanto, para o *Application Server*, o desenvolvedor necessita apenas programar em Java para desenvolver Web services e os clientes destes Web services. Mais detalhes sobre os Web services da arquitetura são descritos a seguir:

Serviço de Apresentação (*Presentation Service*)

O *Presentation Service* [OGC06] é um serviço descrito pela especificação OpenLS da OpenGeospatial. Este serviço é responsável por apresentar mapas em um dispositivo móvel. Porém, o *Presentation Service* implementado para o Omnipresent, além de disponibilizar mapas tanto para dispositivos móveis quanto para uma aplicação Web

rodando em um PC, também possui a capacidade de fornecer mapas de acordo com o perfil do usuário. Por exemplo, se um usuário está visitando uma cidade e ele gosta de ver pinturas de arte, o sistema pode localizar museus e exposições e enviar para o usuário um mapa da região contendo tais PoIs.

A interface WSDL desse Web service possui duas funções por onde o cliente pode consultar mapas:

- WMSMapRequest(): essa função retorna um mapa no formato SVG ou um documento JPEG. O objetivo dessa função é para que outras aplicações, além dos dispositivos móveis, possam usufruir do *Presentation Service*. Dessa forma, qualquer aplicação Web pode acessar o serviço permitindo que o cliente consulte mapas através de um browser, por exemplo;
- openLSMapRequest(): essa função é usada pelos dispositivos móveis para obter um mapa de acordo com a localização do usuário. Essa função retorna uma URL indicando em que local o mapa foi gerado ou o próprio conteúdo do mapa no formato SVG Tiny. O objetivo da função *openLSMapRequest* é oferecer mapas mais leves que possam ser usados por dispositivos com pouca capacidade de processamento e armazenamento. O *Presentation Service* possui uma classe chamada SVG2SVGT. Esta classe se encarrega de receber um mapa no formato SVG do WMS e transformar em SVGT. Devido à simplicidade do mapa fornecido pelo WMS, poucas mudanças são necessárias para converter SVG para SVGT neste projeto. Além do cabeçalho do documento SVG, as outras mudanças principais são referentes às cores. O SVGT dá suporte a apenas 16 palavras chaves do código de cores do XHTML e não dá suporte ao nome dos códigos de cores X11, ou seja, textos que são mapeados para os valores das cores em RGB, como por exemplo: bege, azul escuro, azul violeta, coral, etc. Utilizando *xpath* é possível selecionar todos os elementos do código SVG que contenham cores e atualizá-las para os tipos dados suporte pelo SVGT. Por exemplo, utilizando a expressão xpath "//*[@fill] | //*[@stroke]", significa que estamos consultando todos os elementos do arquivo XML que contenham algum atributo "*fill*" ou algum atributo "*stroke*", pois esses atributos indicam as cores de preenchimento e de borda de alguma geometria no SVG. Uma vez de posse dos elementos, é possível alterar os seus valores para os tipos disponibilizados pelo SVGT.

Para ambas as funções descritas anteriormente, os parâmetros da requisição são especificados através da classe *MapRequest*, cujo diagrama de classe é apresentado na Figura 5.9. O projeto dessa classe foi desenvolvido baseado na forma de requisição do

mapa descrito na especificação OpenLS. Segundo esta especificação, a requisição do mapa é dividida em três níveis:

- *Output*: contém informações sobre o tamanho do mapa (*width* e *height*), formato (*format*), transparência (*transparent*), e o tipo de conteúdo a ser retornado (*content*), que pode ser uma URL que aponta para onde o mapa foi gerado ou o próprio mapa através de uma String. O *Output* também contém uma classe chamada *BBoxContext* que possui dois pontos, o sistema de referência espacial e a dimensão que especifica o *bounding box* da área de interesse. Somente a classe *BBoxContext* e os atributos *width*, *height* e *format* são obrigatórios na requisição;
- *Basemap*: especifica as camadas que serão usadas no mapa. Cada camada é composta por um nome e um estilo. De acordo com a especificação OpenLS, o BaseMap não é obrigatório. Porém, neste projeto, foi decidido tornar o seu uso obrigatório para que houvesse compatibilidade com a especificação Web Map Service (WMS), pois nesta especificação, deve-se fornecer pelo menos uma camada e um estilo na requisição do mapa;
- *Overlays*: especifica uma lista de tipos abstratos de dados (ADT) para serem apresentadas no mapa. Como os PoI são obtidos automaticamente no *LBS Service* através do perfil do usuário e as rotas, através do *Routing Service*, foi decidido não colocar *Overlays* na requisição do mapa neste projeto.

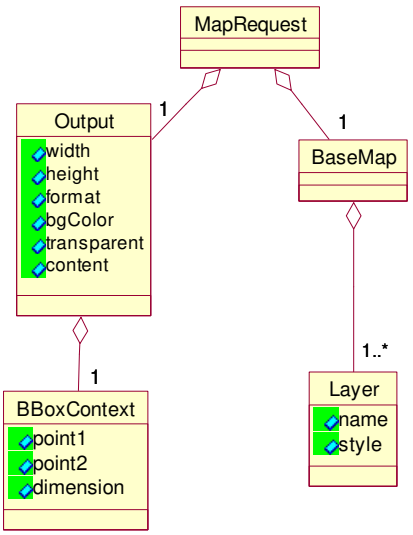

Figura 5.9 Diagrama de classe do MapRequest

WMS e WFS

O *Presentation Service* funciona com a ajuda de dois outros serviços implementados pelo iGIS [ABS+05]: o *Web Map Service* (WMS) e o *Web Feature Service* (WFS). Estes serviços são baseados nas especificações *Web Map Service* e *Web Feature Service* da OpenGeoSpatial. Ao invés de realizar uma consulta através da passagem de vários parâmetros na URL, como é descrito no padrão, interfaces Web services foram implementadas para esses serviços. Dessa forma, além de tornar a consulta a tais serviços mais fácil, também é possível publicá-los em um serviço de diretórios, como UDDI, para prover invocação e descoberta automática.

A interface WSDL do WMS possui as seguintes funções:

- wmsGetCapabilities(): retorna um metadado do serviço que descreve informações relevantes incluindo número máximo de camadas que pode existir no mapa, as camadas e os estilos disponíveis, sistema de referência de coordenadas (CRS), bounding box e escala. Todas essas informações são retornadas em um arquivo XML que está de acordo com o XMLSchema da especificação WMS;
- wmsGetMap(): retorna um mapa de acordo com os parâmetros recebidos. Os parâmetros de requisição do mapa são encapsulados em um JavaBean chamado *GetMapRequest*. Essa classe possui os mesmos atributos de requisição do mapa detalhados na especificação WMS. Na Figura 5.10 é apresentado um trecho de código que mostra como os parâmetros de requisição são passados para o JavaBean *GetMapRequest* que é usado como parâmetro na função *wmsGetMap()* na hora de consultar um mapa no WMS. Neste exemplo, o cliente do Web service está requisitando um mapa com tamanho de 800x600 pixels, contendo os limites do estado da Paraíba dentro do bouding box "-38.8, -8.4, -34.7, -6.0" e no formato SVG;
- wmsGetFeatureInfo(): retorna informações sobre uma determinada feição, por exemplo, um nome de um hotel, endereço entre outras descrições.

A especificação WFS possibilita ao usuário, consultar e atualizar dados geoespaciais independente de plataforma. Este serviço contém as seguintes operações:

- wfsGetCapabilities(): responsável por descrever as feições disponíveis e as operações disponíveis para cada feição;
- wfsDescribeFeatureType(): obtém a estrutura de um dado tipo de feição, que é descrito em um XMLSchema;

- wfsGetFeature(): possibilita ao usuário, especificar que propriedades deseja consultar de uma determinada feição, como por exemplo, obter o nome e a geometria de um teatro.

```
GetMapRequest req = new GetMapRequest();
req.setService("WMS");
req.setRequest("GetMap");
req.setVersion("1.3.0");
req.setBBOX("-38.8,-8.4,-34.7,-6.0");
req.setCRS("CRS:84");
req.setFormat("image/svg+xml");
req.setHeigth("600");
req.setLayers("ParaibaLimites");
req.setStyles("layoutFronteira");
req.setWidth("800");
```

Figura 5.10 Consultando um mapa no WMS

A Figura 5.11 apresenta a arquitetura do serviço de mapas. Existe uma única interface Web service, que encapsula os serviços WMS e WFS. Usando o *stub*, o *Presentation Service* acessa esses serviços remotamente. O WMS e WFS foram implementados como uma extensão do *framework* IGIS. Essa ferramenta permite que os desenvolvedores construam aplicações SIG na Web que acessam algum Banco de Dados Geográfico.

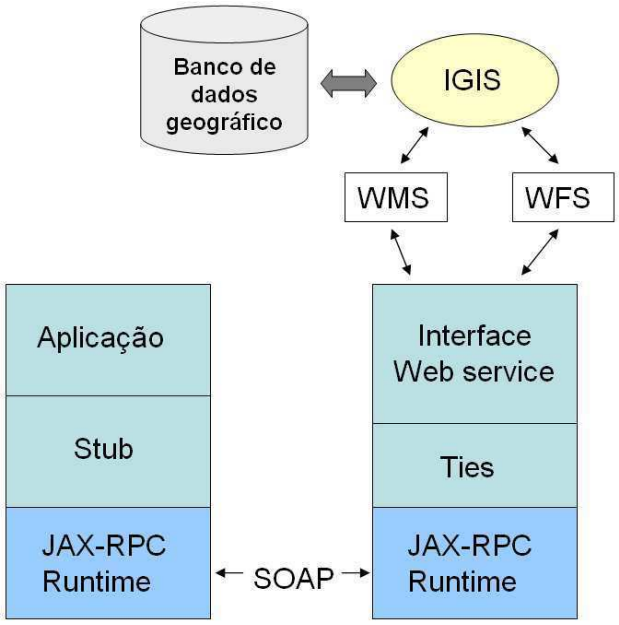

Figura 5.11 Arquitetura do serviço de mapas

Serviço de LBS (*LBS Service*)

O LBS Web service é o principal serviço da arquitetura. Ele é responsável por receber e gerenciar informações de contexto. Através desse serviço, o cliente registra o seu perfil, adiciona regras que são ativadas de acordo com o contexto e atualiza sua posição através de um dispositivo móvel equipado com GPS. Os dados capturados pelos sensores como os sensores de batimento cardíaco, por exemplo, são transmitidos para este serviço, que por sua vez, se encarrega de atualizar a ontologia e gerenciar as ações que devem ser disparadas.

As principais operações deste Web service são:

- registerUser(): registrar um usuário no Web service juntamente com o seu perfil;
- updatePosition(): atualiza a localização geográfica no sistema, e analisa o contexto procurando por ações que são relevantes para o contexto do usuário. Cada vez que a posição é atualizada o contexto é analisado. Em seguida, um objeto *Response* é enviado para o dispositivo contendo todas as informações de contatos, PoIs próximos, áreas de monitoramento e possíveis alertas;
- updateEmotion(): atualiza o estado emocional do usuário;
- updateStatus(): atualiza o status do usuário, podendo também ser deduzido a partir de regras de inferência;
- updatePhysiologic(): atualiza o estado fisiológico. Esta função recebe como parâmetro o tipo de estado fisiológico e o seu novo valor. Por exemplo: updatePhysiologic("Maria", 35, "heart_beat");
- addRule(): adiciona novas regras de monitoramento do contexto. As regras são descritas na forma de um documento XML. O *LBS Service* apresenta um XMLSchema que descreve como o cliente do serviço deve transmitir as regras de contexto. O XMLSchema pode ser obtido pela aplicação cliente através da função *ruleSchema()* do *LBS Service*. A estrutura do XMLSchema pode ser vista na Figura 5.12. O elemento *near* é a parte da regra que descreve o desejo do cliente em ser avisado quando alguns usuários (contatos) especificados estiverem próximos. O elemento *user* contém outros elementos que descrevem o contexto do usuário, como: *status*, *physiologic* e *emotion*. Através do elemento *nearPoint*, o cliente especifica qual a distância mínima de um determinado ponto no mapa, no qual, algum contato deve estar para ativar a regra. Por

exemplo, um usuário pode querer receber um aviso no seu dispositivo se o seu filho sair de casa durante o dia. Com o *nearPlace*, o cliente especifica qual a distância mínima de um determinado lugar (por exemplo, banco, loja, hospital, etc) que ele ou um contato deve estar para ativar o alerta. A diferença entre *nearPoint* e *nearPlace* é que no primeiro, o usuário especifica um ponto qualquer no mapa e no segundo, o usuário informa o tipo de lugar, ou seja, ele escolhe uma das camadas do mapa que estão disponíveis pelo WMS.

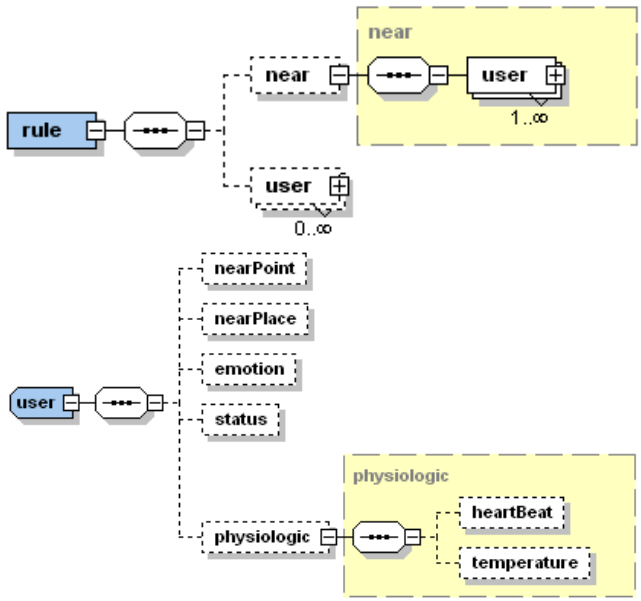

Figura 5.12 Arquitetura do XMLSchema que descreve as regras

Como pode ser visto na Figura 5.13, ambas as interfaces Web services utilizam o *LBSManager* para atualizar dados na ontologia, receber dados do *Presentation Service* e *Advertisement Service*, através dos *stubs*, para produzir a resposta personalizada para o usuário. A ontologia é armazenada em um Banco de Dados através do sistema de persistência do *framework* Jena. Os dados na ontologia são gerenciados pela classe *OntologyManager* que se encarrega de inserir, atualizar e remover dados na ontologia. *RuleManager* gerencia as regras de contexto.

Para diminuir o tamanho do arquivo *stub* usado para acessar o *LBS service* a partir de um dispositivo móvel com J2ME Web service, duas interfaces WSDL foram desenvolvidas para o mesmo serviço, na qual uma teria apenas as funções acessadas pelo dispositivo, ou seja, as funções: *updatePosition()*, *updateStatus()*, *updateEmotion()* e *updatePhysiologic()*. Essa interface com menos recursos é chamada de

LBSServiceThin. A outra interface que contém mais recursos é acessada por outras aplicações, nas quais o tamanho do arquivo *Stub* não irá influenciar no seu desempenho.

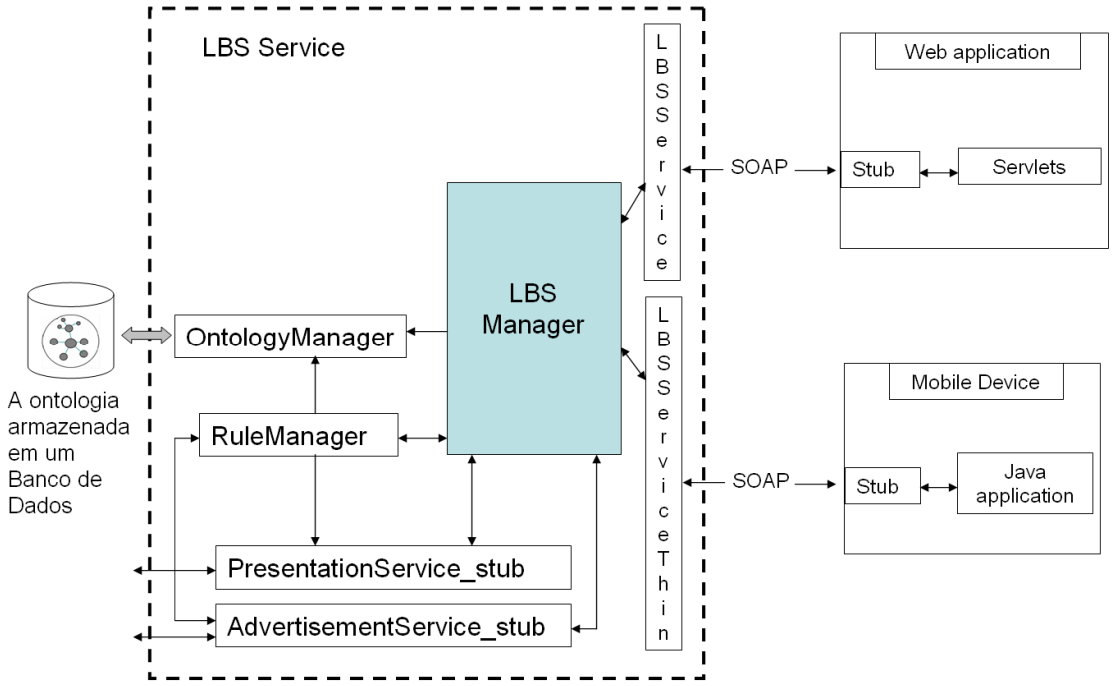

Figura 5.13 Arquitetura do *LBS Service*

A principal funcionalidade do *LBS Service* é o sistema de regras ativado por contexto. Como o sistema de regras do Jena só permite a carga de regras no momento em que o *Reasoner* é instanciado, impossibilitando a remoção e adição de novas regras enquanto o sistema está em funcionamento, um artifício baseado em padrões de projeto *Observer* foi empregado. Esse artifício é demonstrado na Figura 5.14. Na ontologia, tudo que herda da classe *Context* pode ser monitorado. A classe *Context* representa toda informação relacionada ao contexto do usuário, como por exemplo: emoção, estado, fisiologia e localização. Toda informação de contexto é monitorada pela classe *Context_listener*. Para indicar que esta classe está interessada na mudança de estado de algum contexto, existe uma propriedade chamada *listener*, que tem como sujeito a classe *Context_listener* e objeto a classe *Context* e suas subclasses. Para indicar, por exemplo, que um *Context_listener* está interessado nas mudanças de localização de um usuário, basta adicionar o *statement* (my_context_listener, listen, location) na ontologia, ou seja, criar um relacionamento do tipo *listen* entre a instância da classe *Context_listener* e a instância da classe *Location*. Quando a localização é atualizada todos os *context_listeners* que estejam interessados neste contexto são avisados da

mudança. Pra obter todos os *Context_listener* basta listar os *statements* que possuem o predicado *listen* e o objeto como instância da classe *Location*.

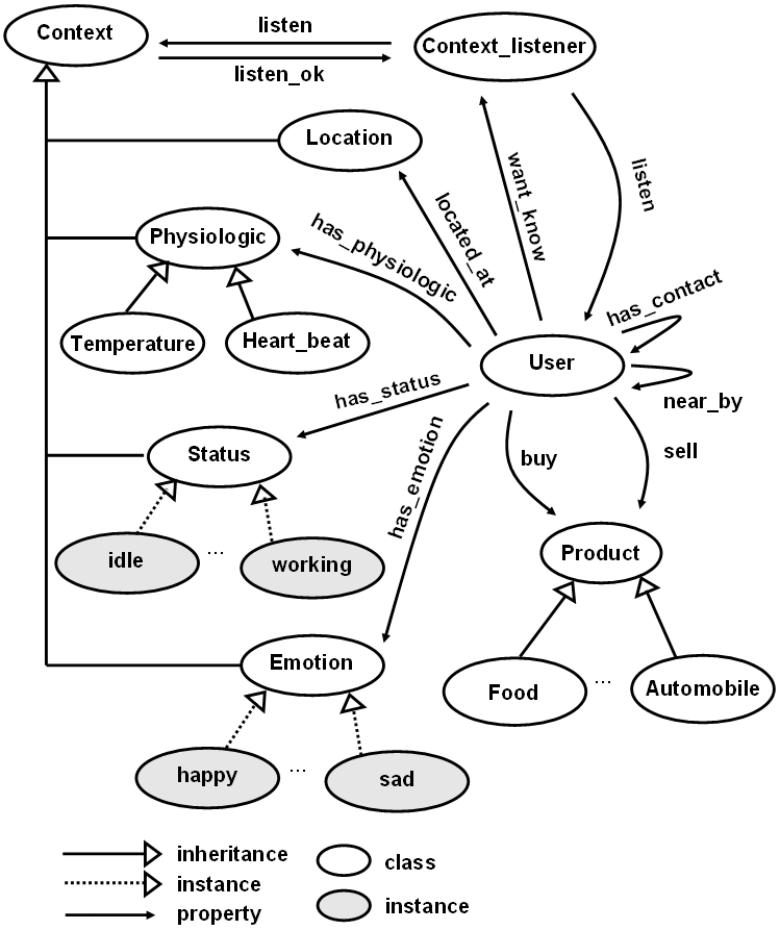

Figura 5.14 Ontologia com o *Context_listener*

O algoritmo a seguir mostra o processo que é executado quando alguma informação de contexto é atualizada.

```
01  atualizar_contexto(Contexto c) {
02  |  obter todos os Context_listen através do statement (null, listen, c)
03  |  para cada Context_listen 'cl' obtido {
04  |  |  comparar o contexto 'c' com o que era esperado pelo 'cl'
05  |  |  se o valor do contexto 'c' era o esperado pelo 'cl' {
06  |  |  |  criar o statement (c, listen_ok, cl)
07  |  |  |  se o contexto cl é satisfeito
08  |  |  |    enviar alerta
09  |  |  }
10  |  }
11  }
```

Quando uma informação de contexto é alterada na ontologia, é feita uma busca por todas as instâncias de *Context_listener* que estejam interessadas no contexto que foi atualizado. Quando a mudança no contexto é considerada satisfeita, é criado um relacionamento chamado *listen_ok*, que é inverso à propriedade *listen*, entre o contexto e a instância da classe *Context_listener*. Dessa forma, torna-se mais simples e rápido descobrir se um *Context_listener* é satisfeito ou não, bastando verificar se para cada relacionamento *listen* existe um relacionamento contrário *listen_ok*. Um *Context_listener* é dito satisfeito se todas as informações de contexto possuem o valor esperado pela regra descrita pelo usuário. O objetivo do *Context_listener* é monitorar o contexto de acordo com as regras descritas pelo usuário, para que este receba um aviso quando o contexto se encontrar no estado desejado. Por exemplo, Carla deseja ligar para André e Patrícia quando eles estiverem em casa e desocupados. A Figura 5.15 mostra como isso é representado na ontologia. A instância 'Carla' possui um relaciona *want_know* que aponta para uma instância de *Context_listener*. Essa instância escuta (*listen*) quatro instâncias de *Context*: a localização e o status de José e os de Patrícia. Quando essas informações são alteradas, o sistema de regras do *LBS Service* verifica se os novos valores são aqueles esperados pelo *Context_listener*, ou seja, André e Patrícia estão em casa e o status deles é "*idle*" (desocupado). Caso algum desses valores seja o esperado, o sistema cria um relacionamento *listen_ok* da localização e do status para a instância de *Context_listener*. Esse relacionamento acelera o processo de descobrir se um *Context_listener* é satisfeito. Por exemplo, Patrícia está em casa e seu status é *idle*; José está em casa e seu status é *sleeping* (dormindo). Quando o status de José é atualizado para *idle*, o sistema irá executar o algoritmo de atualização de contexto descrito anteriormente. No momento de checar se o contexto é satisfeito, ele não irá precisar verificar novamente a localização e o status de Patrícia para saber se estão de acordo com o que era esperado pelo *Context_listener* de Carla. O sistema apenas irá verificar se o número de *listen* é igual ao de *listen_ok*. Caso isso ocorra, significa que a regra de monitoração do contexto foi satisfeita e o alerta será enviado para os seus destinatários.

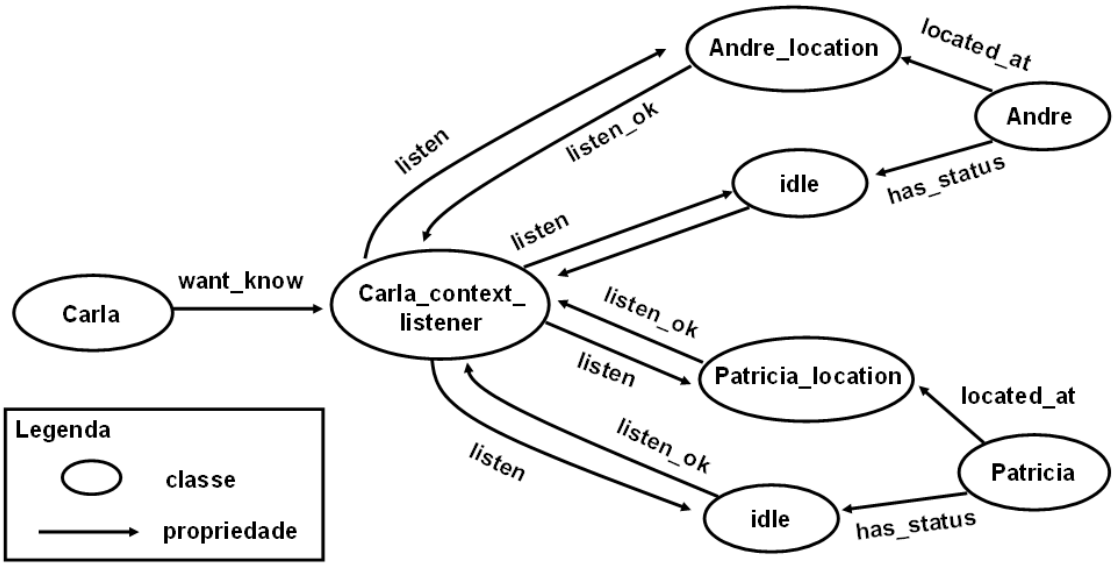

Figura 5.15 Exemplo de um *Context_listener* na ontologia

Para evitar que os alertas sejam constantemente enviados para o usuário enquanto o contexto se encontra no estado satisfeito, o *LBS Service* envia os alertas somente quando o *Context_listener* passa do estado "*não satisfeito*" para o estado "*satisfeito"*. Em qualquer outra mudança de estado o serviço não envia o alerta, pois ou ele já foi enviado ou o contexto não satisfaz as regras descritas na lembrança.

Com o objetivo de diminuir a carga no servidor, a parte das regras relativa a detectar a presença do usuário em alguma parte do mapa é tratada de forma diferente em relação às outras regras de contexto. Depois que o usuário descreve a regra na aplicação Web informando que deseja ser avisado quando um determinado contato estiver dentro ou fora de uma determinada área, as informações dessa área (x, y, raio e tipo) são salvas no banco de dados. Quando o dispositivo móvel estabelece alguma comunicação com o serviço, seja para atualizar a posição, emoção ou estado fisiológico, o *LBS Service* retorna um objeto *Response* que contém as informações das atualizações necessárias no dispositivo, como, novos PoI encontrados, contatos próximos, alertas que foram disparados por alguma regra e as informações das novas áreas que devem ser monitoradas pelo dispositivo. Dessa forma, o dispositivo móvel será encarregado de monitorar constantemente a sua localização utilizando o cálculo da distância euclidiana. Quando a distância entre a localização e o ponto (x, y) for maior ou menor do que o raio (dependendo do tipo de monitoração: "*inside*" ou "*outside*"), o dispositivo transmite a informação para o *LBS Service*, o qual se encarrega de atualizar a ontologia criando o relacionamento *listen_ok* para informar que a regra foi satisfeita.

Nesta seção, serão descritos alguns exemplos de como o sistema de monitoração do contexto funciona.

Para descrever as regras de monitoração do contexto, o usuário apresenta as regras de monitoração na forma de um documento XML e escreve a mensagem que deve ser entregue semelhante a um e-mail, com título, contatos para os quais a mensagem se destina, a forma de entrega (e-mail ou alerta no dispositivo) e o conteúdo da mensagem.

Quando o usuário submete uma regra para o *LBS Service*, esse serviço se encarrega de monitorar as informações descritas na mesma. O diagrama de seqüência da Figura 5.16 descreve o processo de cadastro de uma nova regra.

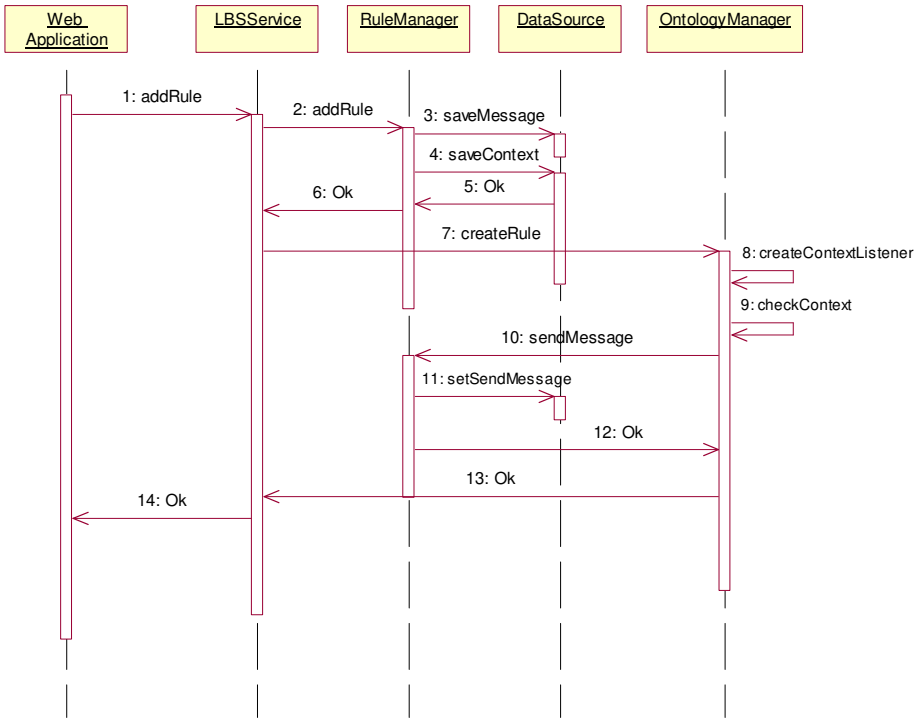

Figura 5.16 Diagrama de seqüência referente ao registro de regras

Quando a interface *LBSService* recebe a regra da aplicação Web através da função *addRule()*, os valores a serem comparados com o contexto (por exemplo, temperatura maior que 38 ºC, estado trabalhando, entre outros) e os dados da mensagem como: tipo (alerta ou e-mail), os contatos para quem a mensagem deve ser entregue e a própria mensagem são salvas em um banco de dados através da classe *RuleManager*. Em seguida, a classe *OntologyManager* cria o *Context_listener* referente a regra na ontologia e verifica o contexto, ou seja, compara os valores do contexto com aqueles

que estão descritos na regra. Se os valores do contexto na ontologia estão de acordo com os especificados na regra, então um alerta deve ser enviado para o dispositivo. A classe *RuleManager* marca no banco de dados que a mensagem deve ser enviada para todos os contatos a quem ela se destina, ou seja, coloca o atributo *send* da tabela *message_to* para "YES". Quando algum destinatário da mensagem estabelecer conexão com o *LBS Service* e verificar se existe algum alerta a ser exibido, ele irá carregar no dispositivo a mensagem marcada pelo *OntologyManager*.

A Figura 5.17 descreve o processo de atualização do contexto e como o alerta é transmitido para o dispositivo. No *LBS Service*, a classe que recebe os dados do dispositivo é o *LBSServiceThin*. Quando qualquer informação do contexto é transmitida do dispositivo móvel para o Web service através de uma função *updateXXX()* (onde XXX pode ser *Position, Phisiologic, Emotion* ou *Status*), se no banco de dados houver algum *send* igual a "YES" na tabela *message_to* para este dispositivo, as mensagens serão enviadas para ele. Este processo ocorre da seguinte forma: a classe *OntologyManager* atualiza a informação na ontologia e consulta por todos os *Context_listeners* que estão interessados nesse contexto que está sendo atualizado. Para cada *Context_listener*, o *RuleManager* compara o contexto recebido do dispositivo com aquele que era esperado pelos *Context_listeners* e retorna aqueles que tiveram o valor esperado. Por exemplo, supondo que o contexto sendo atualizado é a temperatura e o valor é de 38 ºC e que uma instância da classe *Context_listener,* chamada "*cl_1*", esteja esperando um valor maior que 37 ºC, então "*cl_1*" é um *Context_listener* retornado pelo *RuleManager* que teve essa parte da regra satisfeita. Para cada *Context_listener* retornado pelo *RuleManager* é verificado se ele é satisfeito, caso positivo, as mensagens são marcadas no banco de dados semelhante ao processo de cadastro de regras. No final, o *LBSServiceThin* consulta no banco de dados por todos os alertas destinados ao usuário que invocou a função *updateXXX()*.

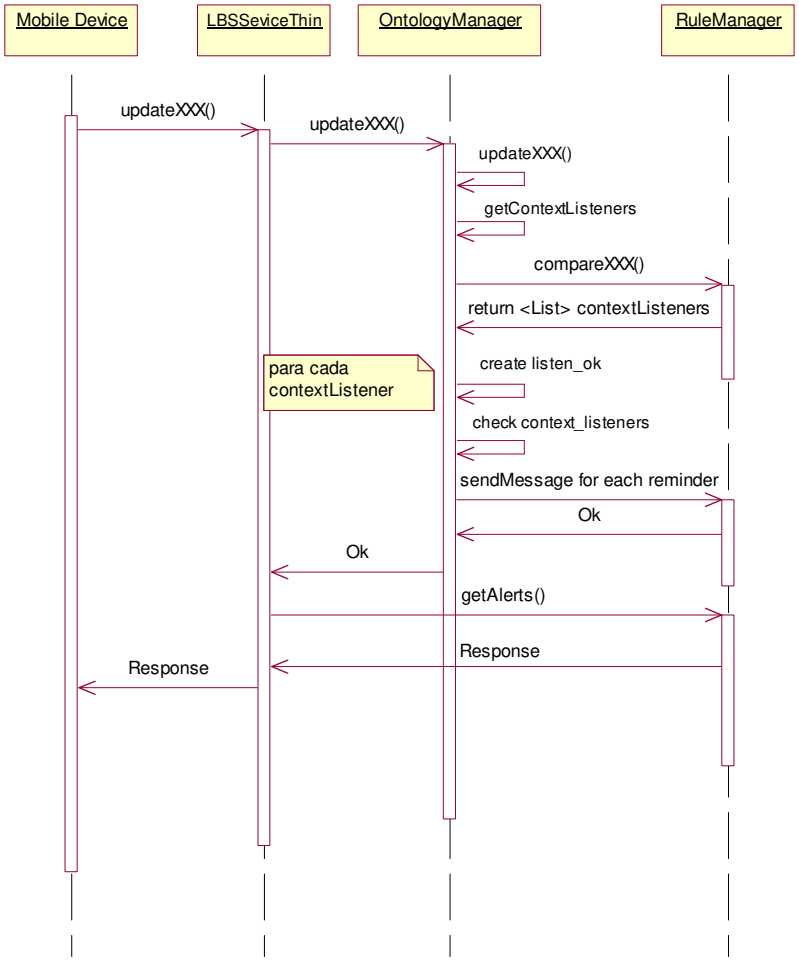

Figura 5.17 Diagrama de seqüência referente a atualização do contexto

Serviço de Anúncio (Advertisement Service)

Este Web service gerencia o anúncio e a busca de produtos e serviços oferecidos pelos usuários. Produtos são organizados de acordo com sua categoria na ontologia. Por exemplo, na Figura 5.18, as categorias carro, moto e caminhão são subcategoria de uma superclasse veículo. Novas categorias podem ser adicionadas no serviço, neste caso, uma nova classe é adicionada na ontologia. Para criar uma nova categoria, vender ou comprar novos produtos, o cliente do serviço descreve o produto através da classe *Produto* (Figura 5.19). Essa classe contém informações sobre o dono do produto (como nome, e-mail, telefone, etc), para que o cliente possa entrar em contato, além das coordenadas geográficas para que ele possa visualizar no mapa o caminho para encontrá-lo.

A classe *Produto* pode conter um conjunto de propriedades e cada propriedade pode conter um conjunto de restrições. Essas restrições são usadas no momento em que o cliente descreve as restrições do produto que esteja procurando. Por exemplo, o cliente pode informar ao *Advertisement Service* que deseja comprar um carro com preço abaixo de R$ 15.000, da marca Ford e ano acima de 2002. Neste caso, as propriedades *preço*, *marca* e *ano* terão as restrições: *menor*, *igual* e *maior* respectivamente. Um produto é registrado como a instância de um subproduto na ontologia. Todas as subclasses herdam as propriedades das superclasses.

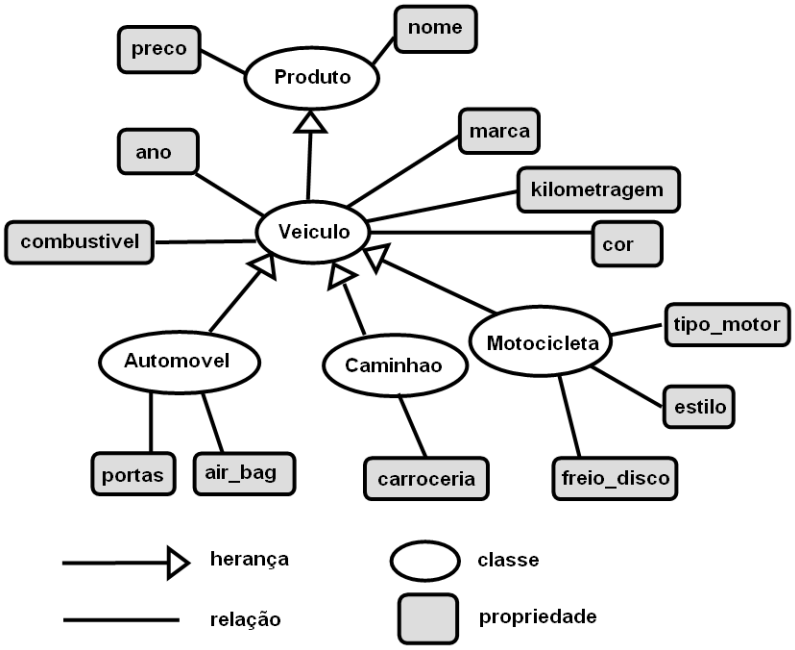

Figura 5.18 Ontologia representando a categoria de produtos

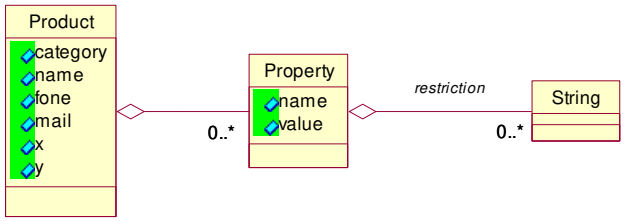

Figura 5.19: Diagrama de classe que descreve um produto e suas propriedades

As operações disponíveis na interface do *Advertisement Service* são:

- addCategory() – adiciona uma nova categoria de produtos neste Web service. Para adicionar uma nova categoria, o usuário informa a supercategoria através do atributo *category* da classe *Product* e o nome da nova categoria através da propriedade *name*. Ele também precisa informar que propriedades esse novo produto deve possuir;
- getCategories() – retorna todas as categorias que são subclasses imediata da classe *Produto* na ontologia;

- getSubCategories(String category) – recebe como parâmetro o nome de alguma categoria e retorna todas as subcategorias que são filhas imediatas desta;
- registerSell(String login, Product p) – através dessa função, um usuário informa um tipo de produto que deseja anunciar;
- registerBuy(String login, Product p) – com esta operação, o usuário registra um pedido de compra. Ele descreve o produto com seus atributos e restrições. O pedido fica registrado no serviço e quando o usuário estiver próximo a alguém que esteja vendendo o mesmo tipo de produto ou um semelhante, ele irá receber a notificação. Essa função retorna um array contendo todos os produtos encontrados;
- searchProduct(Product) – semelhante a função *registerBuy()* com a diferença de que o pedido de compra não fica registrado no serviço. A pesquisa por produtos próximos é realizada no mesmo momento em que a função é invocada;
- buy(String login) – esta função é invocada para que o Web service verifique se existe algum produto de interesse próximo.

À medida que o usuário se movimenta, os produtos que estão próximos são comparados com os produtos que o usuário deseja. A freqüência de comparação não é a mesma da freqüência de atualização da posição geográfica. Isso pode ser configurado no dispositivo móvel. Essa medida é para diminuir a sobrecarga no sistema. Se o usuário está parado, então a freqüência com que a função *buy()* é invocada é menor. Porém, se o usuário estiver em movimento, a freqüência deve ser maior, pois a área de busca está mudando constantemente.

Advertisement Service usa um algoritmo que faz o casamento entre duas instâncias baseado em um dos modelos clássicos da recuperação da informação, que é o modelo vetorial [AUB06]. O algoritmo representa o produto solicitado pelo usuário e os que estão sendo procurados como vetores. O grau de similaridade entre os mesmos é representado através do co-seno do ângulo formado por estes dois vetores. Quando o usuário descreve um produto que deseja comprar e suas restrições, a ferramenta cria um vetor cujo número de dimensões é igual ao número de restrições impostas pelo usuário. Cada uma destas dimensões representa uma restrição e é inicializada com o valor máximo, que no nosso caso é 1. Por exemplo, o usuário que deseja um apartamento cuja área total seja superior a 80 m^2 , que tenha dois quartos e que a taxa de condomínio seja inferior a R$ 200 reais. O vetor R que representa o produto desejado pelo usuário é representado da forma R = (1, 1, 1), no qual a primeira dimensão representa a condição

"área superior a 80 m^2", a segunda dimensão indica que ele deve ter dois quartos e a terceira indica que a taxa de condomínio deve ser inferior a R$ 200.

Um vetor é criado para cada tipo de produto que esteja sendo vendido e esteja próximo ao comprador. Este vetor tem o mesmo número de dimensões do vetor que representa o produto desejado pelo usuário. O valor de cada dimensão varia de acordo com as características do produto. Se o produto atende à restrição associada à dimensão, o sistema atribui a ela o valor 1. Caso contrário, o sistema associa a ela o valor 0. Por exemplo, se um apartamento disponível para venda atende apenas as duas primeiras restrições, ele é representado pelo vetor P1 = (1, 1, 0). Caso um outro apartamento satisfaça as três restrições, ele é representado da forma P2 = (1, 1, 1), e assim sucessivamente. O grau de similaridade entre os dois produtos é calculado através do co-seno do ângulo formado pelos dois vetores. Por exemplo, a fórmula da Figura 5.20 mostra o cálculo efetuado entre a consulta R e o produto P1.

$$\cos(R, P1) = \frac{\vec{R} \bullet \vec{P1}}{|\vec{R}| \times |\vec{P1}|}$$

Figura 5.20 Calculando o co-seno entre a consulta R e o produto P1

Se o grau de similaridade for superior a um *threshold* mínimo especificado no momento da descrição do perfil, o produto é adicionado ao resultado, caso contrário, ele é ignorado. No entanto, antes de aplicar o algoritmo de *matchmaking*, o *Advertisement Service* precisa fazer uma pré-seleção dos produtos. Somente aqueles produtos próximos ao cliente e que pertençam a mesma categoria são interessantes para o usuário. Essa pré-seleção é realizada pela seguinte regra de inferência:

```
[sellerNear: (?s ont#seller_near ?b) -> (?s ont#located_at ?l1) (?b ont#located_at
?l2)(?l1 ont#x ?x1) (?l1 ont#y ?y1) (?l2 ont#x ?x2)(?l2 ont#y ?y2) (?b whats_near
?near) nearEuclidian(?x1, ?y1, ?x2, ?y2, ?near) (?s ont#sell ?p1)(?b ont#buy ?p2) (?p1
rdf:type ?t) (?p2 rdf:type ?t)]
```

Essa regra informa quem são os vendedores próximos a um cliente. Alguém é considerado um vendedor próximo se a distância entre os dois for menor que o permitido. Isso é calculado pela função *nearEuclidian()*, que informa se a distância euclidiana entre o vendedor e o cliente é menor que o permitido. Em seguida, é

verificado se existe algum produto que o vendedor esteja oferecendo e que é do mesmo tipo de algum produto desejado pelo cliente.

Serviço de Roteamento (*Routing Service*)

O *Routing Service* [ABS+06], segue a especificação OpenLS da OpenGeoSapatial. Este serviço de roteamento é responsável por prover rotas entre dois pontos de um mapa. O cliente envia uma consulta para este serviço através do fornecimento de dois pontos. Dessa forma, o usuário seleciona dois pontos quaisquer no mapa (seja um contato ou um ponto de interesse) e consulta o *Routing Service* para calcular a rota entre eles. Em seguida, o serviço retorna a rota no formato SVG.

O Web service de roteamento tem armazenado os dados das ruas e suas interseções. O caminho entre cada dois pares de pontos é previamente calculado e armazenado num banco de dados, usando a estratégia chamada materialização total [SFG97]. O custo desse armazenamento é alto, mas irrelevante tendo em vista o pequeno custo financeiro de memória secundária atualmente, e o ganho em processamento é diferencial. Para amenizar o custo de armazenamento, o *Routing Service* utiliza uma técnica chamada de hierarquização do grafo das ruas da cidade [JHR96]. O grafo é particionado em subgrafos e cada partição é representado por um nodo em um supergrafo. Existem nodos que fazem parte de mais de um subgrafo. Eles são chamados de nodos de borda e fazem parte do supergrafo.

Os caminhos são calculados para cada fragmento e para o supergrafo, resumindo a busca do melhor caminho em cinco possibilidades:

- Os nodos fonte e destino fazem parte do supergrafo. Nesse caso, basta retornar o caminho entre eles;
- O nodo fonte não faz parte do supergrafo, mas o destino faz. Então um nodo intermediário do supergrafo deve ser encontrado, nodo esse que deve pertencer ao mesmo fragmento do nodo fonte. Então, o caminho da fonte ao intermediário somado ao caminho do intermediário ao destino é retornado;
- O nodo fonte faz parte do supergrafo, mas o destino não. Basta inverter a situação anterior e proceder de forma semelhante;
- Nenhum dos nodos fazem parte do supergrafo, mas fazem parte do mesmo fragmento. Basta retornar o melhor caminho entre eles;

- Nenhum dos nodos fazem parte do supergrafo e estão em fragmentos diferentes. Então um nodo-borda do fragmento de origem e um do fragmento de destino deve ser encontrado, e o melhor caminho vai ser a junção do melhor caminho do nodo-fonte ao nodo-borda escolhido do seu fragmento com o melhor caminho entre os nodo-borda e o melhor caminho do nodo-borda do fragmento de destino ao nodo final.

O algoritmo de busca do melhor caminho utilizado para calcular os custos das rotas foi o Dijkstra [Dij59]. Escolhido por sua simplicidade e tendo em vista que a performance não é diferencial, já que esse cálculo é realizado previamente.

A principal operação do *Routing Service* é *getRoute()*. Além dos pontos de origem e destino, ele recebe também como parâmetro o tipo de rota a ser calculada. De acordo com o padrão OpenLS, os tipos são:

- *Fastest* – a rota mais rápida para se chegar ao destino;
- *Shortest* – a menor rota para se chegar ao destino;
- *Pedestrian* – o menor caminho para se chegar ao destino a pé.

Por enquanto, o único tipo de rota oferecido pelo serviço é o *Shortest*.

5.5 Considerações Finais

Este capítulo tratou da descrição da arquitetura de um sistema para aplicações cientes de contexto, o Omnipresent. Foi exibida a arquitetura geral do sistema, apresentando todos os componentes envolvidas. A seguir, cada parte da arquitetura foi explicada com mais detalhes, começando pela camada cliente, depois a aplicação Web e, por fim, a camada de Web services. No próximo capítulo, serão abordados alguns testes realizados para avaliar o funcionamento da arquitetura Omnipresent.

Capítulo 6 . Estudo de Caso

6.1 Introdução

Este capítulo descreve um cenário de testes usado para avaliar as funcionalidades oferecidas pelos serviços. Este cenário consiste de alguns usuários fictícios com relacionamentos entre si, alguns produtos e serviços cadastrados como imóveis, veículos e serviços de saúde. O mapa utilizado foi o da cidade de Campina Grande na Paraíba com alguns pontos de interesses fictícios.

A seção a seguir irá abordar o cenário de testes e os resultados obtidos para cada caso. Por fim, temos as considerações finais na Seção 6.3.

6.2 Cenários de Testes

Para testar a arquitetura, foi montado um cenário com dez usuários pertencentes a dois grupos de famílias e amigos. As instâncias da ontologia mostrada na Figura 6.1 mostram os relacionamentos entre os usuários utilizados nesse teste. O usuário Francisco é casado com Maria, tem um filho chamado João, um amigo chamado Alberto e uma amiga chamada Joelma. A usuária Carla é casada com José, tem uma filha chamada Ana, uma amiga chamada Patrícia e um amigo chamado André.

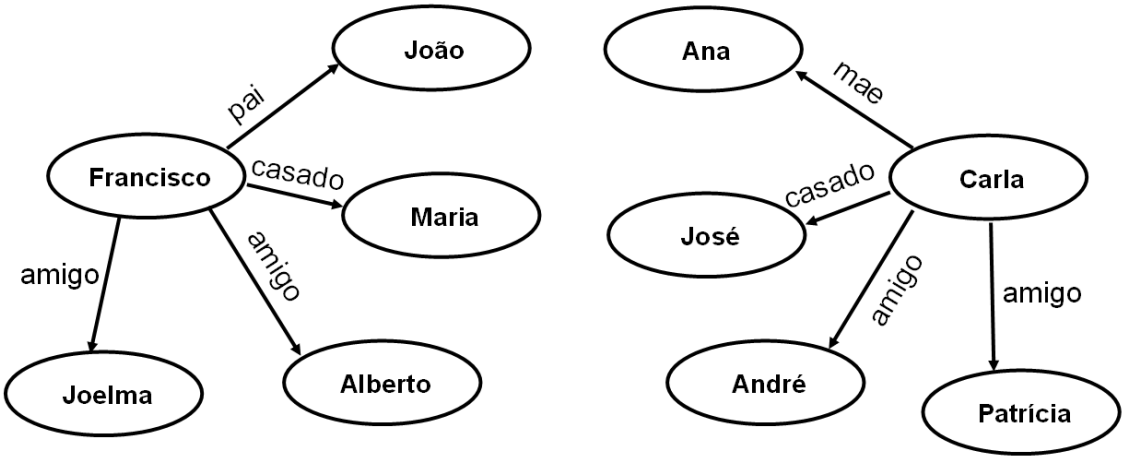

Figura 6.1 Relacionamento entre os usuários do cenário montado

O mapa do cenário é o de Campina Grande, mas as camadas são fictícias, apenas para testar a construção de mapas personalizados. Alguns produtos e algumas lembranças ativadas por contextos também foram inseridos e os detalhes da avaliação serão discutidos nas seções seguintes. Esses testes, além de ajudar a encontrar erros no programa, auxiliaram a encontrar pontos nos quais o desempenho podia ser melhorado e a descobrir novos requisitos para trabalhos futuros.

A seguir são descritos alguns exemplos:

Monitorando o estado fisiológico

O usuário Francisco deseja saber quando sua esposa, Maria, estiver com menos de 50 batimentos por minutos e a temperatura estiver acima de 38 ºC. O usuário Francisco, através de uma aplicação Web, descreve essa regra da seguinte forma:

```
<rule >
  <user name='Maria'>
    <physiologic>
      <heartBeat op='&lt;' threshold='50'/>
      <temperature op='&gt;' threshold='38'/>
    </physiologic>
  </user>
</rule>
```

Quando o contexto de Maria se enquadrar no contexto descrito acima, Francisco receberá um aviso no dispositivo.

Figura 6.2: Usuário recebendo um alerta no seu dispositivo móvel

Monitorando o Status

Alberto deseja ser lembrado de ligar para Francisco quando o status dele for desocupado. Este contexto é descrito da seguinte forma:

```
<rule>
  <user name='Francisco'>
    <status type="idle"/>
  </user>
</rule>
```

O status de um usuário pode ser definido a partir de sua agenda. Ele pode descrever suas tarefas diárias em um calendário e o sistema pode extrair essas informações de sua agenda. Esse sistema ainda não foi implementado ficando para trabalhos futuros. Para efeito de testes, o cliente atualiza seu status no dispositivo móvel.

Monitorando a Emoção

Maria deseja saber quando João está nervoso. Atualmente, esse contexto é informado diretamente pelo usuário através do dispositivo móvel ou pela interface de um browser. Esta foi uma forma rápida de capturar esse tipo de informação. Um modo totalmente pervasivo de obter esse contexto seria através de sensores, como por exemplo, através da análise da fisionomia obtida através de câmeras ou alterações do estado fisiológico como batimento cardíaco ou pressão sanguínea. Porém, esses métodos são bastante complexos de serem analisados necessitando de mais tempo para sua implementação.

```
<rule>
  <user name="Joao">
    <emotion type= "nervous" />
  </user>
</rule>
```

Monitorando a Posição

Este é um dos mais importantes e também o mais complexo contexto para ser monitorado. Como descrito no Capítulo 5, o *LBS Service* permite três formas de monitorar a localização de um usuário: monitorar contatos próximos, monitorar lugares próximos e monitorar a proximidade de um usuário em um ponto específico no mapa. Quando o usuário entra no sistema, a sua lista de contatos é exibida no browser para

auxiliar na construção da regra. As três formas de monitoramento da localização são descritas através dos seguintes exemplos:

- Francisco deseja ser lembrado quando seu amigo, Alberto, estiver próximo. Neste caso, próximo para Francisco corresponde a distância de 1 Km. Quando Alberto estiver a uma distância menor ou igual a 1 Km de Francisco, este receberá no seu dispositivo um alerta e a posição onde Alberto está localizado. O XML abaixo mostra como Francisco deve descrever esta regra. Mais contatos podem ser adicionados ao elemento *near* caso o usuário deseje monitorar a localização de vários usuários. Por questões de simplicidade, para cada regra só pode existir apenas um elemento *near*, ou seja, cada regra calcula apenas a proximidade do cliente em relação a vários usuários. Esse método é mais fácil de calcular, pois existe apenas um raio ao redor do cliente e o sistema verifica se existem outros usuários dentre desse raio. Se fosse possível especificar um raio para cada usuário, o algoritmo seria bem mais complexo e conseqüentemente levaria mais tempo para ser calculado.

```
<rule>
  <near distance="1">
    <user name="Alberto" />
  </near>
</rule>
```

Figura 6.3 Usuário visualizando um contato próximo

- Francisco deseja ser lembrado que precisa pagar uma conta quando estiver próximo a um banco. Neste caso, o usuário também deve especificar a distância mínima entre ele e o local. Neste exemplo, a distância é de 500 metros entre o usuário e o banco. Diferente do elemento *near* descrito na regra anterior, com o *nearPlace* é possível monitorar a proximidade em relação a vários lugares diferentes em uma mesma regra. Isso porque quem calcula essa proximidade é o WFS, porém quanto mais lugares a serem monitorados maior será o tempo de resposta. A Figura 6.4 mostra o usuário recebendo o alerta. Quando ele confirma que recebeu a mensagem a posição do banco é mostrada no mapa.

```
<rule>
  <user name='Francisco'>
    <nearPlace name='bank' r='0.5' />
  </user>
</rule>
```

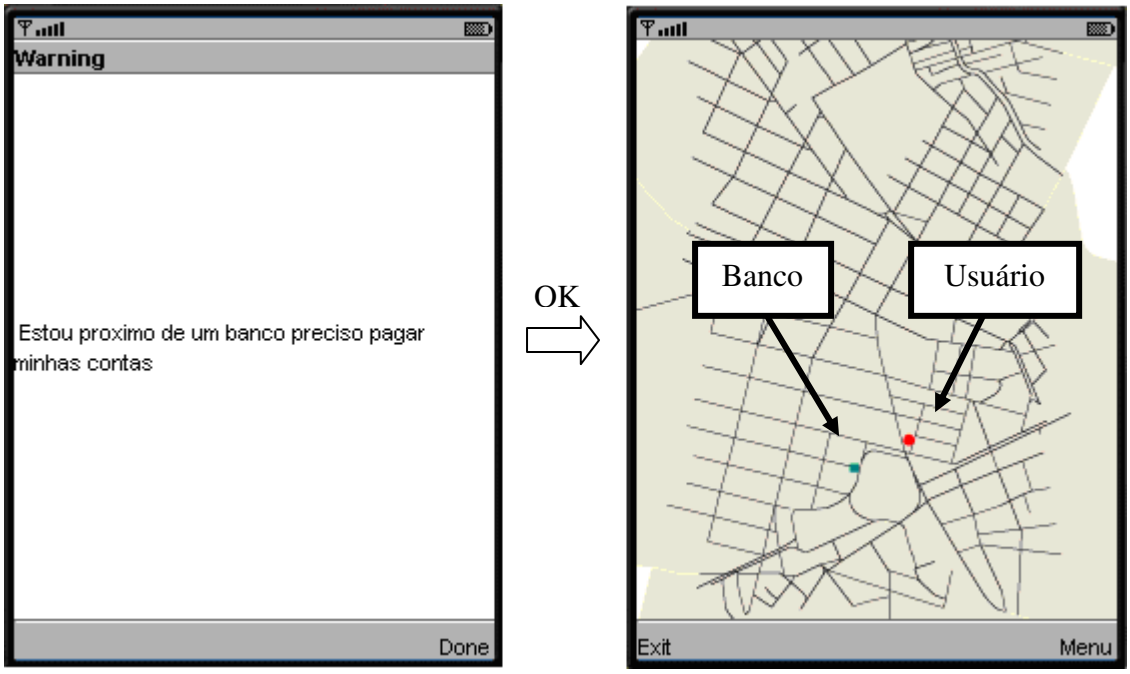

Figura 6.4 Usuário recebendo um alerta de um banco próximo

- O último caso é quando o usuário deseja ser avisado quando algum contato estiver próximo a um ponto específico. Ele constrói a regra especificando o ponto e um raio, onde o *LBS Service* irá avisá-lo quando o seu contato estiver dentro ou fora da área descrita. Esta área pode representar qualquer local, por exemplo, a regra abaixo descreve que Francisco quer saber quando seu filho, João, estiver fora da escola. A representação da escola é aproximada para um ponto central de coordenadas (181200.095, 9201290.480) e raio de 1 Km. Quando João estiver fora dessa área,

Francisco irá receber um alerta no seu dispositivo avisando que seu filho não se encontra na área da escola. Como esse tipo de monitoração é realizado no dispositivo móvel, a geometria do local é aproximada para um círculo ao invés de um polígono, pois, dessa forma, torna-se mais fácil e rápido para um dispositivo com pouca capacidade calcular a distância entre o usuário e o circulo.

```
<rule>
  <user name=" Joao">
    <nearPoint x="181200.095" y="9201290.480" r="1" type="outside"/>
  </user>
</rule>
```

Construindo situações de contexto mais complexas

Os casos anteriores são situações simples que analisam informações de contextos isolados. O sistema de regras do *LBS Service* permite construir situações mais complexas através da combinação dos vários tipos de regras. Vejamos alguns exemplos:
Exemplo 1: Francisco deseja telefonar para sua amiga, mas não quer importuná-la em um momento impróprio, então ele registra uma regra para lembrá-lo de ligar para sua amiga quando ela estiver livre e seu estado emocional é feliz.

```
<rule>
  <user name="Joelma">
    <emotion type="happy"/>
    <status type="idle"/>
  </user>
</rule>
```

Exemplo 2: Se o filho de Francisco estiver doente e sua mãe, Maria, estiver trabalhando, Francisco deseja entrar em contato para saber como ele está.

```
<rule>
  <user name="Joao">
    <status type="ill"/>
  </user>
  <user name="Maria">
    <status type="working"/>
  </user>
</rule>
```

Neste caso, o estado *ill* é deduzido a partir do estado fisiológico do usuário através da regra de inferência:

[ill: (?u ont#has_status ?s) <- (?u ont#hasPhysiologic ?t) (?u myOnt#hasPhysiologic ?p) (?t rdf:type ont#Temperature) (?p rdf:type ont#Pressure) (?t myOnt#value ?vt) (?p myOnt#value ?vp) greaterEqual(?vt, 38) greatEquals(?vp, 140x100)]

Essa regra diz que um usuário é considerado doente se sua temperatura for maior que 38 ºC e pressão sangüínea maior que 14 por 10, ou seja, 140x100 mmHg.

As figuras 6.5 e 6.6 mostram como descrever uma regra de monitoração de contexto usando o browser de um computador. Primeiramente, como mostra a Figura 6.5, o usuário descreve a regra em linguagem XML e clica no botão "Next".

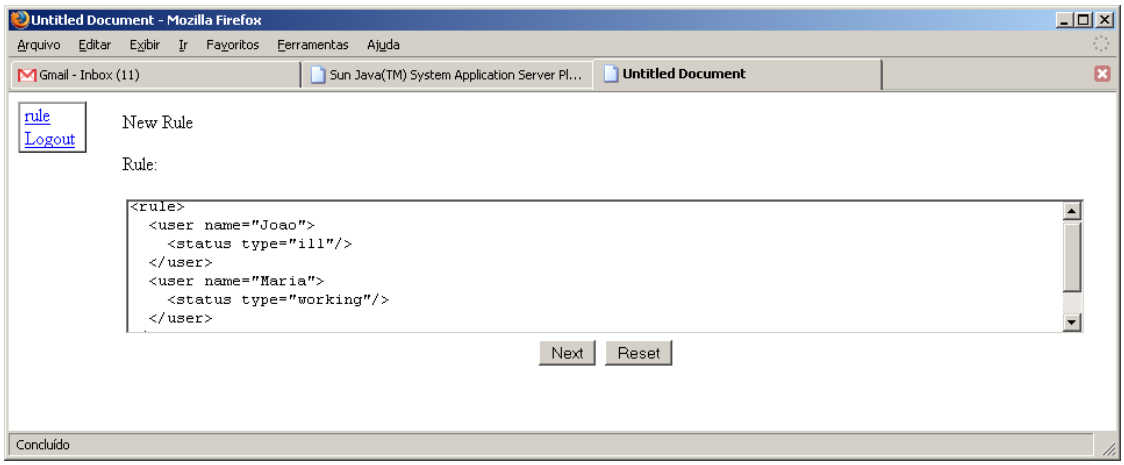

Figura 6.5 Criando uma regra de monitoração de contexto

Depois de descrever a regra, o usuário deve preencher um formulário, como mostra a Figura 6.6, semelhante a um e-mail, com destinatários, assunto e mensagem, além de especificar como deseja receber a mensagem (e-mail ou alerta no dispositivo).

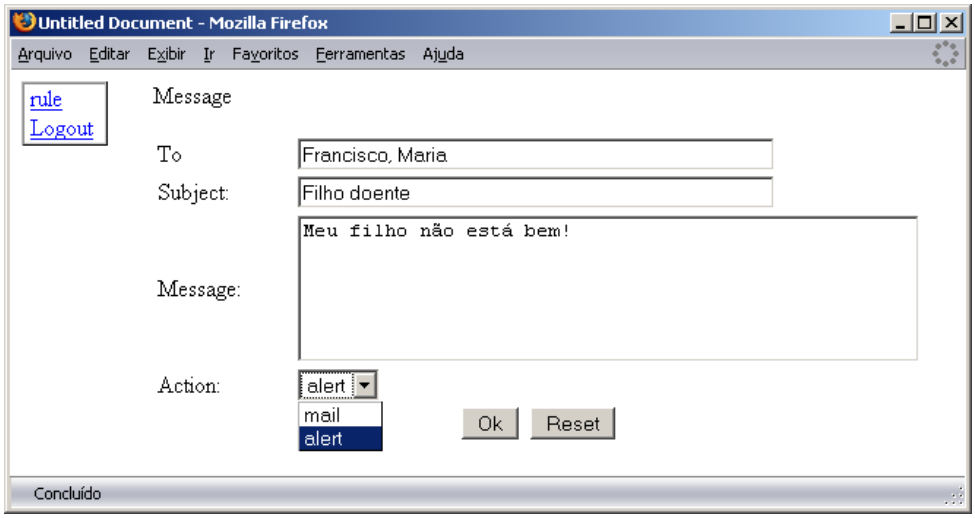

Figura 6.6 Descrevendo a mensagem que será ativada quando o contexto for satisfeito

Para finalizar, o sistema lista todas as regras registradas pelo usuário. Nessa lista, como mostra a Figura 6.7, o usuário pode visualizar a mensagem clicando no título no campo *Subject*, visualizar a regra clicando em *see rule*, parar ou reiniciar a regra clicando em *start*, ou remover a regra do sistema clicando em *delete*.

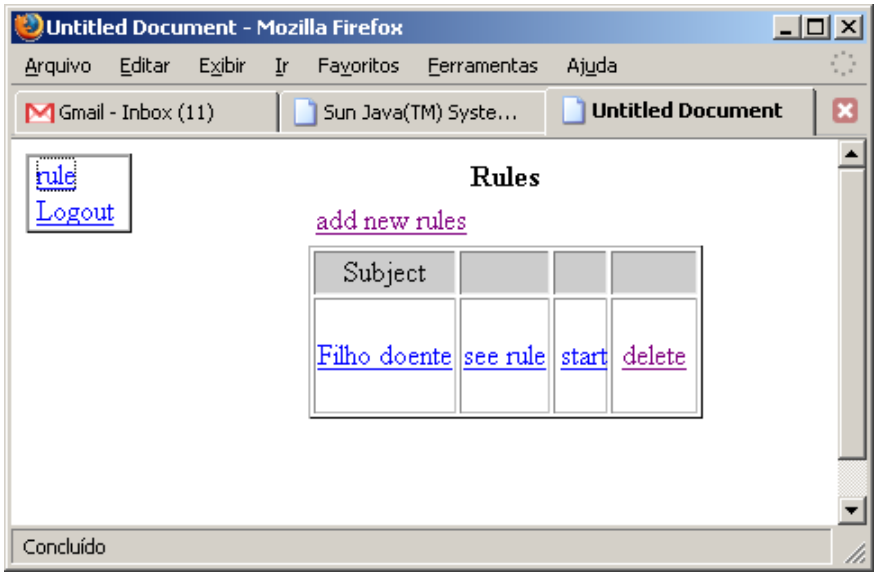

Figura 6.7 Tela principal que lista todas as regras

6.2.1 Pesquisa por produtos

O *Advertisement Service* permite que usuários cadastrem e consultem produtos de interesse para que o serviço encontre aqueles que mais se aproximam de suas necessidades. O cliente pode fazer a busca por produtos através de um browser ou deixar o seu pedido registrado no serviço, dessa forma, quando estiver próximo de alguém que esteja vendendo o produto de interesse, ele receberá um aviso no seu dispositivo mostrando a localização e a forma de contato com o vendedor. Além de produtos, o *Advertisement Service* permite consultar por serviços como: clínicas, cabeleireiros, oficinas de conserto de imóveis, entre outros. Porém, essas categorias de produtos e serviços precisam ser adicionadas à ontologia para que o serviço possa fazer a busca desejada.

Para os exemplos desta seção, a ontologia de produtos e serviços utilizados é a descrita na Figura 6.8. Neste caso, temos dois tipos de categorias de produtos (imóveis e automóveis) e uma categoria de serviço (serviço médico). Neste trabalho, o termo "produto" se referente tanto para produtos quanto para serviços oferecidos. Quando um produto é adicionado como um subproduto de outro, ele herdará todas as propriedades do produto pai.

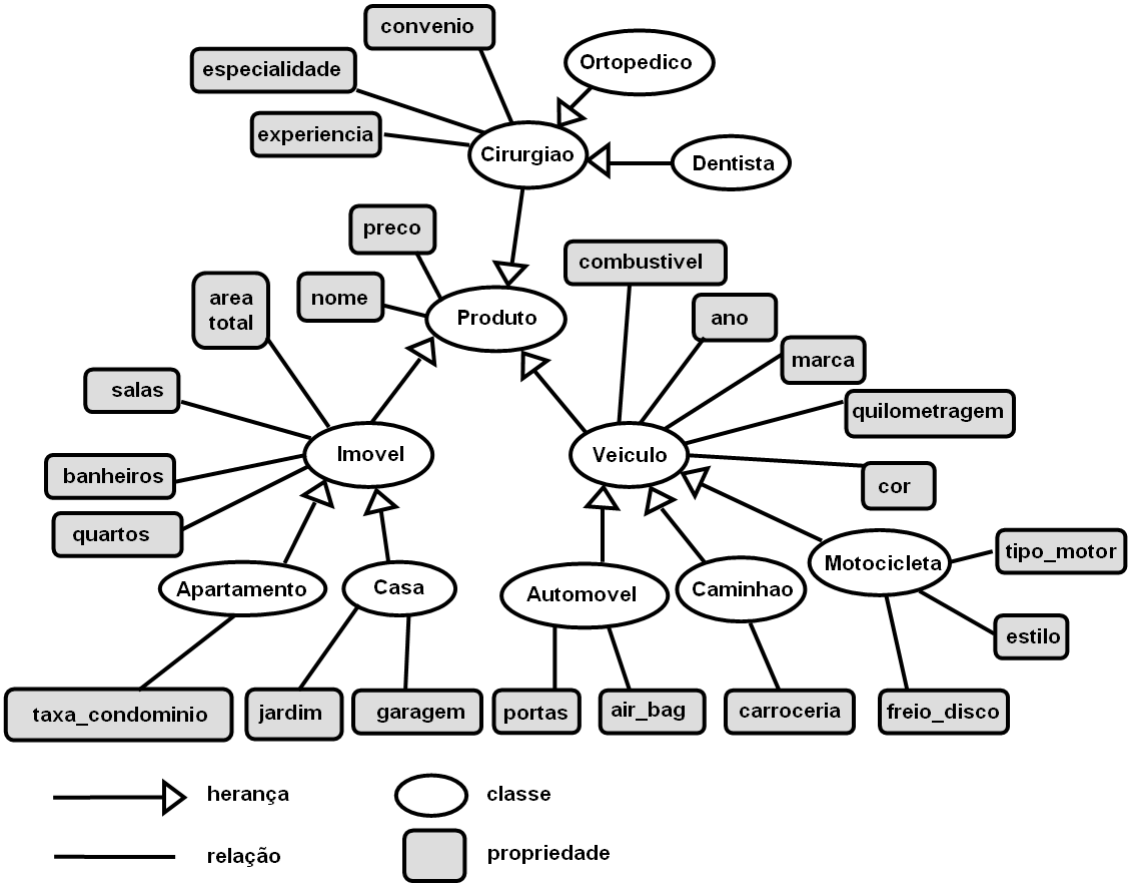

Figura 6.8 Ontologia de produtos

Os testes a seguir avaliam a capacidade do serviço de pesquisar por produtos com um certo grau de compatibilidade.

Exemplo1: Francisco deseja comprar um carro. A Tabela 6.1 descreve as características desejáveis para o carro de Francisco:

Tabela 6.1: Os valores da consulta por um produto no serviço

Atributo	Relação	Valor
Preço	<	R$ 15.000
combustível	=	gasolina
Ano	>=	2003
Marca	=	BMW
quilometragem	<=	40000 Km
Cor	=	Prata
nº portas	=	4
tem air bag	=	Sim

No Web service, os tipos de carros que estão a venda são descritos na Tabela 6.2.

Tabela 6.2: Produtos do tipo carro oferecidos no *Advertisement Service*

Usuário	Propriedades do carro (preço, combustível, ano, marca, quilometragem, cor, nº portas, tem air bag)
Carla	(16.000, gasolina, 2004, BMW, 35.000, preto, 4, sim)
José	(10.000, gasolina, 2002, Ford, 65.000, prata, 4, não)
André	(14.500, gasolina, 2003, BMW, 36.000, prata, 4, sim)
Patrícia	(14.000, gasolina, 2002, BMW, 37.000, branco, 4, sim)

Francisco registra sua consulta no *Advertisement Service*, informando as características do seu produto e o *threshold* que irá limitar o número de resultados da busca. Depois de algumas atualizações da posição geográfica no *LBS Service*, o dispositivo consulta o *Advertisement Service* para verificar se alguns dos pedidos de busca de produtos foram encontrados. No exemplo acima, Francisco informa que o *threshold* de sua busca é de 60%, ou seja, o maior ângulo formado entre o vetor de busca e o de produtos pesquisados não pode ultrapassar 0.6. Na Tabela 6.2, o único produto que satisfaz a consulta em 100% é o produto oferecido por André, porém como o *threshold* é 60%, os resultados retornados pelo serviço são os carros de André, Carla e Patrícia, pois estes dois últimos só possuem 2 atributos que não satisfazem as requisições de Francisco.

Figura 6.9: Produtos pesquisados pelo *Advertisement Service*

Se o usuário desejar, ele pode também solicitar uma busca por produtos parecidos, caso nenhum resultado for encontrado. Por exemplo, Francisco deseja comprar um apartamento, mas na área desejada não existe nenhum apartamento a venda, então o sistema realiza uma busca por produtos semelhantes. No caso do apartamento, o sistema sobe na hierarquia da ontologia e pesquisa por outros tipos de imóveis. Consideremos que Francisco deseje comprar um apartamento com 3 quartos, 1 sala e preço inferior a R$ 45.000. Na Figura 6.10, temos que o sistema não encontrou nenhum apartamento a venda, então os resultados apresentados são casas com características semelhantes a da consulta pelo apartamento.

Figura 6.10 Resultado da consulta por aproximação

6.2.2 Mapas personalizados

Cada cliente pode ter o mapa personalizado de acordo com as suas preferências no sistema. Ele escolhe as camadas para serem exibidas quando o mapa for carregado no dispositivo móvel. Por exemplo: supondo que o usuário tenha preferência por restaurantes japoneses, obras de arte e shows, então o mapa no seu dispositivo poderá aparecer com essas camadas.

Semelhante à categorização de produtos, as camadas também são organizadas em categorias, dessa forma, o sistema pode pesquisar por um ponto de interesse semelhante ao desejado. No exemplo da Figura 6.11, o usuário Francisco tem preferência por comida japonesa e o sistema localiza para ele os restaurantes japoneses

mais próximos. À medida que Francisco caminha, ele chega a uma área que não será possível encontrar nenhum restaurante japonês próximo. Porém, o sistema, através da ontologia, verifica que restaurante japonês é um subtipo de restaurante oriental. Neste caso, como mostra a Figura 6.12, o sistema encontrou um restaurante de comida chinesa, que conforme a ontologia é um tipo de restaurante oriental.

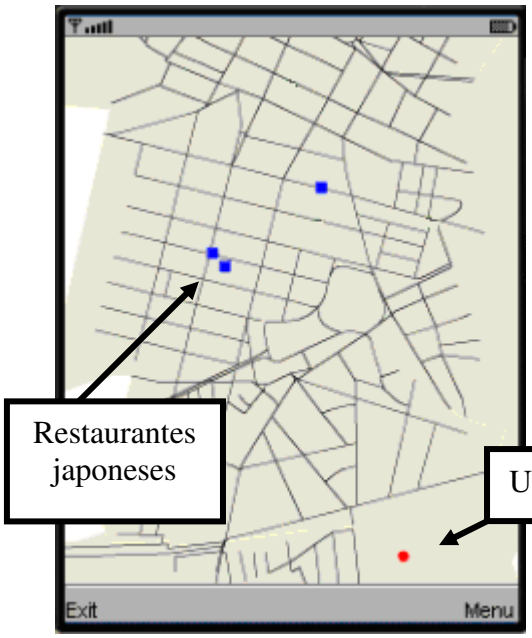

Figura 6.11. Mapa com restaurantes japoneses

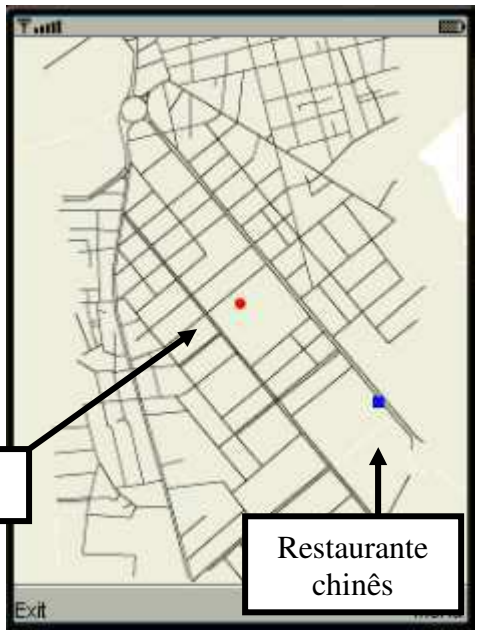

Figura 6.12. Mapa com um restaurante chinês

6.3 Considerações Finais

Este capítulo descreveu aspectos de implementação da arquitetura Omnipresent. Foram listados os requisitos funcionais e não-funcionas, bem como um cenário de teste para avaliar o sistema. Os testes apresentaram o funcionamento do sistema de regras do *LBS Service*, a busca por produtos no *Advertisement Service* e consulta de mapas personalizados. O próximo capítulo apresenta as conclusões desta dissertação e aponta futuras pesquisas a serem realizadas.

Capítulo 7 . Conclusão

Graças ao avanço da tecnologia dos dispositivos móveis e dos sensores, dois paradigmas da computação estão crescendo rapidamente. Uma delas é a computação ubíqua, na qual recursos computacionais estão em toda parte, monitorando ou executando tarefas de interesse do usuário. O outro paradigma é a computação ciente de contexto que adapta as suas funcionalidades de acordo com a situação do usuário e suas preferências.

Construir um serviço que analise todas as informações de contexto é bastante complexo. Neste trabalho, os principais contextos (dentre eles, a localização) foram levados em conta para montar uma arquitetura baseada em dispositivos móveis que possa analisar uma determinada situação e alertar o usuário sobre o estado do contexto.

7.1 Principais Contribuições

A principal contribuição desta dissertação é o desenvolvimento do Omnipresent, uma arquitetura baseada em Web services que proporciona várias funcionalidades na área de computação ciente de contexto. Os serviços oferecidos por essa arquitetura são:

- um serviço de mapas, com os pontos de interesse de acordo com o perfil do usuário;
- um serviço de rota, com o qual o usuário pode encontrar o menor caminho entre dois pontos no mapa;
- um Web service de anúncio de produtos e serviços que permitem que usuários possam vender ou comprar de forma que o anúncio ou a busca pelo produto é realizada de forma dinâmica enquanto ele se movimenta pela cidade;
- um serviço que monitora o contexto do usuário e do ambiente ao seu redor, permitindo que lembranças e alertas sejam ativados de acordo com o contexto e não apenas com o tempo, como em agendas eletrônicas;
- um modelo de representação do contexto usando ontologias, baseado nas principais características do contexto do usuário e do ambiente.

Como se trata de uma arquitetura orientada a serviço, estes serviços podem ser acessados independentemente. Dessa forma, usuários que só necessitam visualizar

mapas, por exemplo, irão consultar apenas um serviço. Outros serviços podem ser adicionados à arquitetura desde que sigam o padrão Web service da W3C. Por exemplo, seria possível acrescentar um serviço que informasse as condições de tráfego para auxiliar o *Routing Service* a calcular a melhor rota entre um ponto de origem e um destino. Outra vantagem em desenvolver a arquitetura em Web services é que ela permite que outros tipos de clientes, além do browser, possam acessar os serviços de forma padronizada, sem que seja necessário acrescentar novos tipos de interfaces para os vários tipos de clientes. Porém, devido às limitações dos dispositivos móveis, nem todas as funcionalidades oferecidas pelos serviços podem ser processadas por estes, como é o caso da edição de regras de análise de contexto.

As especificações da OGC (como WMS, WFS e OpenLS) são especificações que possuem sua própria interface de comunicação, ou seja, não possuem uma interface padronizada de acordo com os Web services. Porém, no Omnipresent, esses padrões foram estendidos para que tivessem uma interface Web service e facilitasse a comunicação com outros serviços.

Outra característica importante da arquitetura é a facilidade de acréscimo de outras informações de contexto. Novas classes na ontologia podem ser acrescentadas em tempo de execução, permitindo uma rápida expansão de informações, como novas categorias de produtos, novos tipos de estado fisiológico e novas categorias de interesses. Além disso, as ontologias permitem deduzir novas informações que não estão explícitas, mas que podem ser obtidas a partir de regras de inferência.

Em relação à monitoração do contexto, o Omnipresent possibilita a construção de regras complexas que podem ser adicionadas ou removidas a qualquer momento. Em outras aplicações, como o SOCAM, as regras devem ser pré-carregadas no sistema para que passem a monitorar o contexto. No Flame2008, o usuário escolhe situações pré-definidas no sistema, no qual, os anúncios de produtos devem ser apresentados ao cliente através do seu dispositivo móvel. No Omnipresent, o usuário pode criar regras para encontrar pontos de interesse quando certas situações acontecem, por exemplo, “quando estiver com fome, para listar as lanchonetes num raio de 1 Km”. Além disso, o Omnipresent permite fazer uma busca de produtos ou serviços de acordo com as restrições especificadas pelo usuário.

Embora essas ferramentas contenham algumas características que não existem no Omnipresent, como por exemplo, o Flame 2008 analisa o histórico do usuário para deduzir novas informações sobre o seu perfil, as comparações realizadas foram baseadas

na análise do contexto, segundo a classificação da Seção 2.2.1. Finalizando, o Omnipresent foi construído para que apresentasse as características usadas na comparação das ferramentas na Seção 3.11. Essas características são:

1. Monitoração em tempo real;
2. Monitoração de vários tipos de contexto;
3. Capaz de monitorar o contexto de terceiros;
4. Sistema SIG onipresente;
5. Análise do perfil do usuário para o fornecimento de mapas personalizados;
6. Anúncio de produtos ou serviços;
7. Orientado a serviço.

Adicionando o Omnipresent à tabela de comparação, temos:

Tabela 7.1: Comparativo entre os sistemas analisados.

Características / Sistemas	1	2	3	4	5	6	7
SOCAM	X	X	X				
Cybre Minder	X	X	X				
AROUND	X			X			
Online Aalborg Guide	X	Apenas localização e perfil		X	X		
Flame 2008	X	Apenas localização e perfil		X	X	Apenas por tipo do produto	X
Nexus	X	Apenas o contexto do ambiente	X	X			
ICAMS		Apenas localização e agenda		X			
FieldMap		Apenas a localização		X			
AMS	X	Apenas a localização e o batimento cardíaco					
Omnipresent	X	Todos, menos o contexto computacional	X	X	X	X	X

Os resultados mostram que a arquitetura é bastante viável e que ela atende ao objetivo almejado. Porém, alguns pontos precisam ser melhorados, mas nada que possa invalidar o sistema. A seguir, alguns desses pontos são descritos juntamente com outras funcionalidades que seriam interessantes para o sistema.

7.2 Trabalhos Futuros

A área de computação ciente de contexto é bastante vasta, possuindo margem para a implementação de uma série de trabalhos. A partir dos resultados obtidos com Omnipresent, pode-se identificar algumas questões importantes para o aperfeiçoamento e extensão deste trabalho. Seguem algumas sugestões de funcionalidades ainda incompletas ou não implementadas que podem ser realizadas em trabalhos futuros:

- a interface gráfica, principalmente na parte de edição das regras de análise de contexto no *LBS Service*, é bastante complexa para que um usuário possa utilizá-la. Um estudo na área de interface homem-máquina seria interessante para descobrir a melhor forma para especificar essas regras de contexto. A interface do programa do dispositivo móvel também necessita ser aperfeiçoada, como por exemplo, seria mais atraente que os pontos de interesse apresentassem algum ícone que facilitasse a identificação, como colocar um $ (cifrão) onde houvesse um banco no mapa;
- o Omnipresent trabalha com a localização do usuário na forma de coordenadas geográficas obtidas através de um GPS conectado ao dispositivo móvel. Essa característica limita o monitoramento desse contexto em ambientes *outdoors*. Para monitorar a localização em ambientes *indoors*, a arquitetura poderia trabalhar com outros métodos de localização como *Bluetooth*;
- o uso de simuladores é importante quando se dispõe de poucos recursos, porém, para uma melhor avaliação do sistema, seria interessante o uso de sensores reais. Dessa forma, novas questões deverão ser avaliadas, como por exemplo, a melhor disposição dos sensores no usuário para que ele não se sinta desconfortável com os aparelhos no seu corpo;
- um ponto importante para a arquitetura é a questão da composição de serviços. No Omnipresent, a composição de serviços ajudaria a selecionar, dinamicamente, serviços que enriqueceriam a informação para o cliente. Por exemplo, se um usuário deseja visualizar um mapa com os pontos de ônibus próximos, juntamente com os seus horários. Dificilmente o sistema irá encontrar um Web service que ofereça os dois serviços. No entanto, um sistema de composição poderia procurar por um serviço de

mapas e um outro serviço de paradas de ônibus e apresentá-los ao usuário, de forma que este não saberia que no fundo se trata de dois Web services diferentes;

- a parte Web service da ferramenta J2ME funciona apenas como cliente, por isso a comunicação é unilateral, ou seja, apenas o dispositivo móvel pode enviar dados para os Web services. Se algum serviço precisa transmitir alguma informação para o dispositivo, ele deve aguardar até a próxima vez que o dispositivo estabeleça conexão. Isso produz um atraso de alguns segundos que pode ser de grande importância, principalmente nos casos nos quais se monitora o estado fisiológico dos usuários. Essa constante troca de informação entre o dispositivo e o servidor provoca uma sensação de "falso *push*", ou seja, para o usuário do dispositivo, o serviço é *push* porque a informação lhe é apresentada sem o seu consentimento, porém, o dispositivo necessita consultar constantemente (característica de serviços *pull*) os serviços para obter informações atualizadas. Para uma melhor rapidez na entrega da informação, os serviços devem possuir alguma forma de transmitir os resultados para os dispositivos;
- os resultados da pesquisa por produtos ou serviços, apenas apresentam aqueles que foram encontrados de acordo com um limite (*threshold*) especificado, como se todos pertencessem ao mesmo nível. Mas se um usuário especifica um limite de 0,6, por exemplo, haverá produtos quer irão casar totalmente (100%) ou parcialmente com o que foi descrito pelo cliente. Poderia haver uma forma do usuário visualizar no mapa o quanto o produto ou serviço encontrado está de acordo com o que ele descreveu. Um trabalho semelhante é apresentado por Burigat e Chittaro [BC05], no qual os pontos de interesse encontrados possuem uma barra de status indicando o quanto aquele ponto de interesse satisfaz a consulta;
- às vezes, os usuários não desejam ser monitorados todo o tempo, ou então desejam que apenas algumas informações estejam visíveis para outros usuários. Configurar que informações podem ser visíveis para outros usuários é uma funcionalidade a ser acrescentada no Omnipresent;
- um outro ponto importante é a questão do desempenho. Apesar dos artifícios utilizados para diminuir a carga de processamento no servidor, as constantes atualizações necessárias para manter os usuários o mais atualizado possível acerca do ambiente ao seu redor ainda é um fator a ser melhorado. Foram realizados testes apenas com poucos usuários. À medida que o número de usuários aumenta, aumenta também o número de conexões e a carga do sistema. Davies, Friday e Storz [DFS04]

apresentam uma sugestão em Grids para solucionar os problemas da computação ubíqua referentes à larga escala e gerenciamento de recursos.

Referências Bibliográficas

[ABR04] Arbter, B.; Bitzer, F.; Ressel, W. *Modeling and Simulation of Mobility in Ubiquitous Computing*. In: Möhlenbrink, W. (ed.); Englmann, F.C.; Friedrich, M.; Martin, U.; Hangleiter, U.: FOVUS - Networks for Mobility; Stuttgart, Alemanha, 2004.

[ABS+05] Almeida, D. R.; Baptista, C.; Silva, E.; Campelo, C.; Nunes, C.; Costa, B.; Andrade, W.; Cabral, J. *Using Service-Oriented Architecture in Context-Aware Applications*. SBC GeoInfo 2005 , Campos do Jordão, Brasil, novembro, 2005.

[ABS+06] Almeida, D. R.; Baptista, C. S.; Silva, E. R.; Campelo, C. E. C.; Figueiredo, H. F.; Lacerda, Y. A. *A Context-Aware System Based on Service-Oriented Architecture*. Proceedings of the International Conference on Advanced Information Networking and Applications (AINA) , IEEE CS Press, Vienna, Austria, Abril 2006.

[ACK03] Andersen, K. V. B.; Cheng, M.; Klitgaard-Nielsen, R. *Online Aalborg Guide – Development of a Location-Based Service*. Technical report, Aalborg Universitet, 2003.

[ACKM04] Alonso, G.; Casati, F.; Kuno, H.; Machiraju, V. *Web Services: Concepts, Architectures and Applications*, Springer-Verlag, Berlin Heidelberg New York, 2004.

[AUB06] Andrade, F. G.; Urich, S.; Baptista, C. S. *Using Constraints to Improve Service Discovery and Composition*. Submetido para publicação, 2006.

[Aug04] Augustin, I. *Abstrações para uma Linguagem de Programação Visando Aplicações Móveis em um Ambiente da Pervasive Computing*. Universidade Federal do Rio Grande do Sul – Porto Alegre: Programa de Pós-Graduação em Computação, 2004.

[AYS04] Augustin, I.; Yamin, A. C.; Silva, L. C.; Real, R. A.; Geyer, C. F. R. *ISAMadapt - um Ambiente de Desenvolvimento de Aplicações para a Computação Pervasiva*, In: Simpósio Brasileiro de Linguagens de Programação, 2004, Niterói.

[BB02] Banavar, G.; Bernstein, A. *Software Infrastructure and Design Challenges for Ubiquitous Computing*. Communications of the ACM, New York, v.45, n.12, Dec. 2002.

[BBC97] Brown, P.J.; Bovey, J.D.; Chen, X. *Context-Aware Applications: From the Laboratory to the Marketplace*. IEEE Personal Communications, 1997, páginas 58-64.

[BC05] Burigat, S.; Chittaro, L. *Visualizing the Results of Interactive Queries for Geographic Data on Mobile Devices*. In: Proc. of the 13th Int. Symp. on Advances in Geographic Information Systems (ACM GIS 2005), ACM Press, pp. 277-284.

[BHL02] Berners-Lee, T.; Hendler, J.; Lassila, O. *The semantic web – a new form of the Web content that is meaningful to computer will unleash a revolvution of new possibilities*. Scientific American, May 17, 2001. Disponível em: <http://www.sciam.com/print_version.cfm?articleID=00048144-10D2-C70-84A9809EC588EF21>. Acessado em: julho de 2002.

[CCH+96] Câmara, G.; Casanova, M.A.; Hemerly, A.; Magalhães, G.C.; Medeiros, C.M.B. *Anatomia de Sistemas de Informação Geográfica*. UNICAMP, 10a. Escola de Computação, 1996.

[CK00] Chen, G.; Kotz, D. *A Survey of Context-Aware Mobile Computing Research*. Technical Report TR2000-381, Dartmouth College, Novembro de 2000. Disponível em <http://www.cs.dartmouth.edu/~dfk/papers/chen:survey-tr.pdf>.

[DA99] Dey, A. K.; Abowd, G. D. *Towards a Better Understanding of context and context-awareness*. Technical Report GIT-GVU-99-22, Georgia Institute of Technology, College of Computing, Junho 1999.

[DA00a] Dey, A. K.; Abowd, G. D. *Towards a Better Understanding of Context and ContextAwareness*. In: CHI 2000 Workshop on the What, Who, Where, When, and How of ContextAwareness, Abril 2000.

[DA00b] Dey, A. K.; Abowd, G. D.; *CybreMinder: A Context-Aware System for Supporting Reminders*, In the Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K), pp. 172-186, Bristol, UK, Springer-Verlag. Setembro 25-27, 2000.

[DFH03] Davies, J.; Fensel, D.; Harmelen, F. van. *Towards the Semantic Web*, John Wiley & Sons, 2003

[DFS04] Davies, N.; Friday, A.; Storz, O. *Exploring the Grid's Potential for Ubiquitous Computing*, IEEE Pervasive Computing, Vol 3, No 2, 2004.

[DHN+04] Dürr, F.; Höhnle, N.; Nicklas, D.; Becker, C.; Rothermel, K. *Nexus - A Platform for Context-aware Applications.* In Proceedings GI-Fachgespräch "Ortsbezogene Dienste", Hagen, Alemanha, 2004.

[Dij59] Dijkstra, E.W. *A Note on Two Problems in Connec-tion With Graphs*, Numerische Mathematik, 1959.

[FAJ04] Feng, L.; Apers, P.M.G.; Jonker, W. *Towards Context-Aware Data Management for Ambient Intelligence*, In*:* Proc. of the 15th Intl. Conf. on Database and Expert Systems Applications, Zaragoza, Espanha, 2004.

[FM04] FieldMap. FieldMap 3.3.6 Manual. Disponível em: <http://www.piranesi.dyndns.org/FieldMap/manual/>. Acesso em: dezembro de 2004.

[For82] Forgy, C.L. *RETE: a fast algorithm for the many pattern/many object pattern match problem,* Artificial Intelligence, volume 19, number 1, 1982.

[GPS04] GPSWorld. Gps Development Timeline. Disponível em: <http://www.gpsworld.com/gpsworld/static/staticHtml.jsp?id=7956>. Acesso em: dezembro de 2004.

[GPZ04] Gu, T.; Pung, H. K.; Zhang, D. Q. *A Middleware for Building Context-Aware Mobile Services*, In Proceedings of IEEE Vehicular Technology Conference (VTC 2004), Milan, Italy, May 2004.

[HHS+99] Harter, A.; Hopper, A.; Steggles, P.; Ward, A.; Webster, P. *The anatomy of a context-aware application*, In Proc. 5th ACM MobiCom Conf., 1999.

[HP06] Hewlett-Packard Development Company. Jena Framework. Disponível em: <http://jena.sourceforge.net/ inference/index.html>. Acesso em: março de 2006.

[JHR96] Jing, N.; Huang, W.; Rundensteiner, E. A. *Hierarchi-cal Optimization of Optimal Path Finding for Transporta-tion Applications*, ACM Conference on Information and Knowledge Management, 1996, pp. 269-276.

[JMRD03] Jose, R.; Moreira, A.; Rodrigues, H.; Davies, N. *The AROUND Architecture for Dynamic Location-Based Services*. Mobile Networks and Applications 8, 2003, páginas 377-387.

[KMK+03] Korpipaa, P.; Mantyjarvi, J.; Kela, J.; Keranen, H.; Malm, E. J. *Managing context information in mobile devices*. IEEE Pervasive Computing, Julho-Setembro 2003, páginas 42-51.

[LML+04] Liszka, K. J.; Mackin, M. A.; Lichter, M. J.; York, D. W.; Pillai, D.; Rosenbaum, D. S., *Keeping a Beat on the Heart*, IEEE Pervasive Computing, vol. 03, no. 4, (Oct-Dec. 2004), pp. 42-49.

[NTT+04] Nakamishi, Y.; Takahashi, K.; Tsuji, T.; Hakozoki, K. *ICAMS: A mobile communication tool using location and schedule information*, In IEEE Pervasive Computing, 3, 1, (Jan.-Mar. 2004), páginas 82-88.

[OGC06] OpenGIS Consortium. OpenGeoSpatial Location Services. Disponível em: <http://www.opengeospatial.org>. Acesso em: março de 2006.

[OMG06] Object Management Group. Common Object Request Broker Architecture. Disponível em: <http://www.omg.org/corba/>. Acesso em: março 2006.

[SAT+99] Schmidt, A.; Aidoo, K. A.; Takaluoma, A.; Tuomela, U.; Laerhoven, K. V.; Velde, W. V. de. *Advanced interaction in context*. In Proceedings of First International Symposium on Handheld and Ubiquitous Computing, HUC'99, páginas 89-101, Karlsruhe, Alemanha, Setembro 1999. Springer Verlag.

[SAW94] Schilit, B.; Adams, N.; Want, R. *Context-Aware Computing Applications*. 1st International Workshop on Mobile Computing Systems and Applications, 1994, páginas 85-90.

[Sch95] Schilit, W. N.. *A system architecture for context-aware mobile computing*. PhD thesis, Columbia University, Maio 1995.

[Sch04] Schiller, J.; Voisard, A. *Location-Based Services*. Morgan Kaufmann, São Francisco, 2004.

[SDA99] Salber, D.; Dey, A.K.; Abowd, G.D. *The Context Toolkit: Aiding the Development of Context-Enabled Applications*. In Proceedings of CHI'99, 1999, páginas 434–441.

[SFG97] Shekhar, S.; Fetterer. A.; Goyal, B. *Materialization Trade-Offs in Hierarchical Shortest Path Algorithms*, Symposium on Large Spatial Databases, Springer Verlag, 1997, pp. 94-111.

[ST94] Schilit, B.; Theimer, M. *Disseminating Active Map Information to Mobile Hosts*. IEEE Network, 1994, páginas 22-32.

[Sun05] Sun Microsystem. Sun Java System Application Server. Disponível em: <http://www.sun.com/software/products/appsrvr/index.xml>. Acesso em: março de 2005.

[W3C04] World Wide Web Consortium. Web Ontology Language. Disponível em: <http://www.w3.org/2004/OWL//>. Acesso em: dezembro de 2004.

[W3C05] World Wide Web Consortium. Mobile SVG Profiles: SVG Tiny and SVG Basic. Disponível em: <http://www.w3.org/TR/SVGMobile>. Acesso em: março de 2005.

[W3C06a] World Wide Web Consortium. Resource Description Framework. Disponível em: < http://www.w3.org/rdf>. Acesso em: março de 2006.

[W3C06b] World Wide Web Consortium. RDF Schema. Disponível em: < http://www.w3.org/TR/rdf-schema/>. Acesso em: março de 2006.

[Wei06] Weiser, M.; Ubiquitous Computing. Disponível em: <http://www.ubiq.com/weiser/>. Acesso em: março de 2006.

[WVG04] Weißenberg, N.; Voisard, A.; Gartmann, R. *Using Ontologies in Personalized Mobile Applications*, In Pro-ceedings Intl. ACM GIS Conference, ACM Press, pp. 2-11, 2004.