Infra-estrutura para o Desenvolvimento de Aplicações Cientes de Contexto em Ambientes Pervasivos

Frederico Moreira Bublitz

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

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Modelos Computacionais e Cognitivos

Angelo Perkusich

(Orientador)

Evandro de Barros Costa

(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

F383i Bublitz, Frederico Moreira
2006 Infra-estrutura para o Desenvolvimento de

Aplicações Cientes de Contexto em Ambientes Pervasivos/ Frederico Moreira Bublitz

– Campina Grande, 2007
79fs.: il.

Referências
Dissertação (Mestrado em Informática) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Orientadores: Angelo Perkusich e Evandro de Barros Costa

1– Computação Pervasiva – Contexto – Ontologia

CDU 004.382.75

Resumo

O desenvolvimento tecnológico tem proporcionado que os computadores se tornem cada vez mais compactos e mais poderosos, levando a computação a um novo paradigma: o de *Computação Pervasiva*. Neste paradigma, os computadores estão embutidos em objetos comuns ao cotidiano das pessoas (e.g., roupas, telefones, automóveis, canetas, óculos e eletrodomésticos em geral), permitindo-os estar acessíveis em qualquer lugar e a qualquer momento, integrando-se à vida das pessoas, agindo em benefício delas.

Tornar estes dispositivos integrados aos seres humanos, requer que as aplicações se adaptem em tempo de execução às alterações no ambiente e às necessidades dos usuários. Isto exige que o desenvolvedor de aplicações voltadas a ambientes pervasivos lide com questões que fogem à lógica de negócio de sua aplicação, principalmente no que diz respeito à obtenção de contexto, que é fundamental para obtenção desta adaptabilidade desejada.

Neste trabalho, aborda-se a problemática supra mencionada, propondo-se uma infra-estrutura capaz de fornecer mecanismos para a obtenção, representação e inferência sobre a informação contextual, tornando-as disponíveis a qualquer aplicação que possa fazer uso delas. Mais especificamente, esta infra-estrutura denominada Lotus, contempla em uma abordagem integrada todos os aspectos relacionados com a provisão de informação contextual.

A viabilidade da infra-estrutura foi demonstrada por meio do desenvolvimento de duas aplicações, uma no domínio de comunidades virtuais móveis e outra no domínio de ambientes pervasivos, onde foi possível constatar que a *Lotus* torna a informação contextual disponível para as aplicações, permitindo que o desenvolvedor foque na lógica de negócio da aplicação.

Abstract

Enabling applications to use any contextual information available in pervasive environment is a hard task from the perspective of the developers. In this work is described an approach to make that an easier task: The Lotus, which consists of an infra-structure for developing context-aware applications, providing mechanisms for acquiring, modeling, and delivering contextual information, enabling it to be shared by different applications. The feasibility of this infra-structure was established through development of a case study where was possible verify that *Lotus* made the contextual information available to the applications, enabling the developer to focuses on application business logic.

Agradecimentos

Agradeço a Deus.

Agradeço aos que contribuíram diretamente para este trabalho. Meus orientadores Angelo e Evandro. Meus co-orientadores Memesso, Hyggo e Glauber =P.

Agradeço aos meus amigos que estiveram comigo durante esta jornada. Os companheiros de AP Xambis, Milena e Romulo onde pudemos aprender a conviver com o passar dos anos =). Os colegas de laboratório Glauber, Milena, Leandro (cabelo), Loreno, Memesso, Leandro (bill). E por fim, aos amigos (quase) sempre presentes Elthon, Memesso, Glauber, Malungo, Carol, Daniel, Luana, Forrageira, Mario, Bill, Vigia, Hyggo, Milena, Xambis, Romulo, Eanes. Aos sempre presentes aos rodízios de pizza dos cariri, Parque do Povo e outras comemorações =D

Agradeço aos meus familiares, em especial aos meus pais (Elmo e Magnólia) e irmãos (Camila e Robinson).

Agradeço em especial a Alice (a.k.a. BB) que sempre esteve comigo nesta etapa e espero que continue por muitas outras.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Desde os tempos mais remotos, a humanidade tem criado mecanismos e ferramentas que visam tornar sua vida mais cômoda. Mecanismos estes, que facilitem a execução de tarefas e, preferencialmente, as façam com mais eficiência e/ou agilidade. Neste sentido, a computação pode ser considerada como uma ferramenta para tornar a vida humana mais confortável, auxiliando as pessoas a executarem suas tarefas de forma mais ágil e eficiente. Apesar de relativamente nova, a computação tem passado por mudanças significativas, conseqüência de uma constante evolução, que a tem aprimorado cada vez mais, levando a computação a um novo paradigma: o de *Computação Pervasiva*. Embora os conceitos de computação pervasiva sejam ainda uma novidade para muitos, suas idéias foram introduzidas em 1991, por Mark Weiser [Wei91]. Em seu artigo, Weiser idealizou um mundo em que a computação pudesse fazer parte do cotidiano das pessoas, em que os computadores pudessem ser usados em qualquer lugar e a qualquer momento. Esse mundo idealizado por Weiser requer uma mudança no modelo de computação que é conhecido atualmente, principalmente na forma como as pessoas interagem com os computadores.

O paradigma de computação pervasiva difere do paradigma de computação pessoal (PC) em dois aspectos principais: complexidade associada ao uso dos computadores e número de computadores por pessoa. No que diz respeito à complexidade associada ao uso dos computadores, a interação com os mesmos passa a ser algo que requer pouca atenção dos usuários para executar as tarefas. Neste paradigma, ao contrário do paradigma de PCs, onde o usuário precisa “aprender” a usar determinada aplicação, necessitando configurá-las para que funcionem corretamente, as aplicações precisam auto configurar-se para atender as ne-

cessidades dos usuários ou exigir o mínimo de esforço para isso. No que diz respeito ao número de computadores por pessoa, esse paradigma viabiliza o uso de vários computadores por uma única pessoa, os quais estão embutidos em objetos comuns ao dia-a-dia das pessoas (e.g., roupas, telefones, automóveis, canetas, óculos e eletrodomésticos em geral), além dos já conhecidos computadores pessoais.

Uma das razões para embutir computação em tais objetos é permitir que eles possam adquirir algum nível de "inteligência", agindo em favor dos usuários. Por exemplo: um *smart phone* deveria ser capaz de reconhecer quando está inserido em determinado ambiente que exija silêncio (uma biblioteca, por exemplo) e repassar automaticamente as ligações recebidas para a secretária eletrônica, evitando assim incomodar as pessoas ao seu redor. Todavia, para que estes dispositivos possam realizar tarefas como esta é necessário que eles sejam pró-ativos [LOA+05], evitando ao máximo a necessidade de interação entre o usuário e seus dispositivos. É importante também preservar a característica de mobilidade dos dispositivos. Para isso, estes dispositivos são dotados de capacidade de comunicação através de redes sem fio, permitindo-os interagir entre si enquanto se movem de um lugar para outro. Juntas, essas características permitem a criação de ambientes pervasivos, que podem ser definidos como ambientes saturados com capacidade de computação e comunicação [Sat01].

Dentre as principais características de ambientes pervasivos, destacam-se a heterogeneidade e a dinamicidade. A primeira é marcada pela diversidade de: dispositivos que podem estar presente num ambiente (e.g.., celulares, *handhelds*, *notebooks*, computadores pessoais); interfaces de rede (e.g., bluetooh, UPnP, Jini); sistemas operacionais; poder de processamento; entre outras. A segunda pode ser facilmente associada à: capacidade de mobilidade de tais dispositivos, permitindo que estes possam entrar/sair do ambiente a qualquer momento; capacidade de um dispositivo passar a fornecer ou deixar de fornecer algum serviço; etc.

A idéia de dispositivos transparentemente integrados aos seres humanos aliada à necessidade de lidar com as características de ambientes pervasivos, requer que as aplicações se adaptem em tempo de execução às alterações no ambiente e às necessidades dos usuários. Dentro do escopo de computação pervasiva, esta adaptabilidade é guiada por dois elementos chave: as noções de *contexto* e *ciência de contexto*. *Contexto* pode ser definido como sendo a situação na qual alguma coisa ocorre ou existe e que pode ajudar a descrevê-la. Dessa

forma, pode-se dizer que uma aplicação está *ciente de contexto* se ela usa a informação presente no ambiente para suprir as necessidades de adaptação às alterações no ambiente ou às necessidades do usuário [BLA+07].

Por exemplo, suponha que em um ambiente pervasivo, mais precisamente uma sala de aula, exista uma aplicação responsável por alterar o modo de atuação do dispositivo do aluno para o "modo silencioso" ao início da aula. A fim de que essa aplicação possa funcionar corretamente, é necessário que a mesma esteja ciente da presença do professor em sala de aula, marcando o início da mesma e possa então fazer com que o dispositivo passe a atuar em modo silencioso. Entretanto, para que isso ocorra é necessário que:

1. a presença do professor seja percebida. Isto normalmente é obtido por meio de sensores que conseguem detectar a presença de dispositivos no ambiente;

2. após o seu dispositivo ser reconhecido, é necessário que a presença do mesmo reflita a presença do professor. Isso requer que a informação obtida pelo sensor seja *representada* de forma que a aplicação possa "entender" o seu significado, ou seja, que a presença desse dispositivo corresponde à presença do professor em sala de aula;

3. agora que a informação já pode ser interpretada, a aplicação pode inferir que a "presença" do professor *implica* no início da aula e então mude a forma de atuação do dispositivo para o "modo silencioso".

Como pode ser visto neste exemplo, fazer com que as aplicações possam estar cientes do contexto no qual estão inseridas é fundamental para que elas possam agir pró-ativamente e lidar com as características de ambientes pervasivos (i.e., heterogeneidade e dinamicidade). Neste sentido, existem três principais etapas para tornar uma aplicação ciente de contexto: *Obtenção da Informação Contextual*, que está ligada à obtenção da informação dos dispositivos e serviços presentes no ambiente; *Representação da Informação Contextual*, que consiste em fornecer uma abstração à informação obtida de forma que possa ser usada pelas aplicações; *Raciocínio sobre a Informação Contextual*, que consiste em fazer com que, a partir da informação que está disponível, algum significado possa ser obtido pela aplicação.

1.1 Problemática

Como descrito anteriormente, fazer com que aplicações percebam o contexto no qual estão inseridas e usem essa informação para tomar decisões em benefício do usuário é fundamental para a computação pervasiva. Porém, isto requer que o desenvolvedor lide com questões não triviais: lidar com a diversidade de formas de comunicação sem fio, representar a informação de forma extensível e permitir que a informação possa ser usada por outras aplicações. Com todos esses requisitos, fazer com que a aplicação use a informação contextual acaba tornando-se uma tarefa mais complicada que a própria lógica de negócio da aplicação em si.

Tentando contornar esse problema, algumas abordagens têm sido propostas. Entre as que mais se destacam, estão o uso de *middlewares* (e.g., [NBB05], [Bar05]) e ontologias (e.g., [CFJ03]). Os principais problemas com estas abordagens é que elas focam em determinado aspecto e acabam deixando lacunas em outros. Por exemplo, a maioria dos *middlewares* focam na etapa de aquisição da informação contextual, levando em conta questões como descoberta de nós e serviços em redes heterogêneas, porém não abordam a forma como a informação é representada e como é feita a inferência sobre a mesma. Já as abordagens que usam ontologias, lidam bem com a parte de representação da informação, porém deixam a desejar quanto à forma de obtenção da informação, muitas vezes não provendo nenhum suporte para isso. Além disso, as soluções estudadas estão, de um modo geral, acopladas ou a um determinado domínio de aplicação (principalmente as soluções baseadas em ontologias) ou à determinada tecnologia de rede como *bluetooth* (mais comum nos *middlewares*).

1.2 Objetivos

Neste trabalho, tem-se como objetivo o desenvolvimento de uma infra-estrutura capaz de fornecer mecanismos para a obtenção, representação e inferência sobre as informações de contexto, tornando-as disponíveis a qualquer aplicação que possa fazer uso delas. Mais especificamente, esta infra-estrutura denominada *Lotus*, deve ser capaz de contemplar em uma abordagem integrada todos os aspectos relacionados com a provisão de informação contextual.

Além disso, faz parte deste trabalho fornecer os mecanismos necessários para o desa-

coplamento da informação contextual, tanto a nível de aquisição, quanto a nível de representação. Sendo assim, deverão ser fornecidos mecanismos para possibilitar a aquisição e comunicação por diferentes tecnologias de rede (e.g., *wi-fi, bluetooth*), bem como possibilitar que a informação esteja representada de tal modo que possa ser usada em diferentes contextos.

Para isso, ter-se-á como base a implementação do *middleware Wings* [LBB+06] que fornece os mecanismos necessários para descoberta de nós e serviços em redes heterogêneas. Já para lidar com a questão da representação da informação e raciocínio será usada uma abordagem baseadas em ontologias. Finalmente, para disponibilização da informação, será adotada uma solução baseada em uma arquitetura cliente-servidor, mais precisamente *Web-services*.

1.3 Relevância

Devido ao grande interesse da comunidade de computação pervasiva na área de ciência de contexto, alguns trabalhos têm sido desenvolvidos nos últimos anos. Porém, a maioria deles trata apenas com aplicações específicas e não existe ainda nenhuma ferramenta de propósito mais geral [MPRB04]. Este trabalho, portanto, surge como uma solução mais abrangente para o campo de computação pervasiva no que se refere à informação de contexto.

Mais precisamente, o *Lotus* permite que os desenvolvedores de aplicações voltadas a ambientes pervasivos, possam acessar a informação contextual como um serviço. Dessa forma, todo o esforço demandado para tornar a aplicação ciente do contexto passa a ser responsabilidade do *Lotus*, permitindo assim, que o desenvolvedor possa contornar as dificuldades encontradas para tornar sua aplicação ciente de contexto.

Além disso, a abordagem adotada, permite que a informação contextual seja compartilhada por diferentes aplicações. Isso causa um grande impacto positivo no desenvolvimento de ambientes pervasivos, por permitir que a informação seja usada por qualquer aplicação, independente do domínio ao qual pertença. Dessa forma é possível que a criação de ambientes pervasivos passe a se tornar uma realidade cada vez mais presente no cotidiano das pessoas.

Por fim, contribui-se diretamente para o projeto *Percomp*[1], que visa desenvolver méto-

[1] http://percomp.org

dos, ferramentas e aplicações para o desenvolvimento de ambientes pervasivos. Este projeto está sendo desenvolvido no Laboratório de Sistemas Embarcados e Computação Pervasiva - *Embedded*[2], da Universidade Federal de Campina Grande.

1.4 Estrutura da Dissertação

O restante deste trabalho está organizado da seguinte forma:

- **No Capítulo 2**, é apresentada uma visão geral da computação pervasiva, focando nos aspectos de contexto e ciência de contexto em ambientes pervasivos.
- **No Capítulo 3**, são descritos os principais conceitos relacionados a ontologias, descrevendo desde a definição do próprio conceito de ontologia até as principais linguagens para representá-las.
- **No Capítulo 4**, é descrito o *Lotus*, mais especificamente, são detalhadas sua arquitetura e implementação.
- **No Capítulo 5**, são apresentadas as aplicações que foram usadas como estudo de caso para validar o *Lotus*.
- **No Capítulo 6**, são descritas algumas soluções que, de algum modo, possuem algo em comum com o *Lotus*.
- **No Capítulo 7**, são apresentadas as considerações finais.

[2]http://embedded.ufcg.edu.br/

Capítulo 2

Computação Pervasiva

Visando facilitar o entendimento do tema de computação pervasiva, neste capítulo são descritos os principais aspectos relacionados ao tema. Mais especificamente, os que estão relacionados à ciência de contexto em ambientes pervasivos.

2.1 Uma Visão Intuitiva de Computação Pervasiva

Para um melhor entendimento sobre o significado de computação pervasiva, considere uma Sala de Aula. Esta sala de aula consiste de uma sala de aula normal, equipada com alguns dispositivos eletrônicos. Na Figura 2.1 podem ser observados alguns desses dispositivos, tais como um computador ①, um projetor ② e um ar-condicionado ③. Além disso, cada aluno, assim como o professor, possui seu próprio dispositivo móvel, como um PDA ou um *notebook* ④. Todos esses dispositivos conseguem se comunicar uns com os outros, seja por meio de redes cabeadas ou através de redes sem fio ⑤.

Poucos minutos antes do inicio de cada aula, uma aplicação controla o nível de iluminação e temperatura da sala (ligando/regulando o ar-condicionado e as luzes), tornando o ambiente agradável e propício a uma boa aula. Além disso, caso haja alguma apresentação preparada para a aula, o computador e projetor são ligados e passam a exibir a apresentação planejada. Ao início da aula, o sistema automaticamente assinala a lista de presença dos alunos, através da percepção dos dispositivos pessoais dos alunos, isto é, como cada aluno possui um dispositivo pessoal, a presença de um dispositivo indica que o aluno está presente na sala de aula. Para evitar que algum aluno traga o dispositivo de um colega e este tenha sua

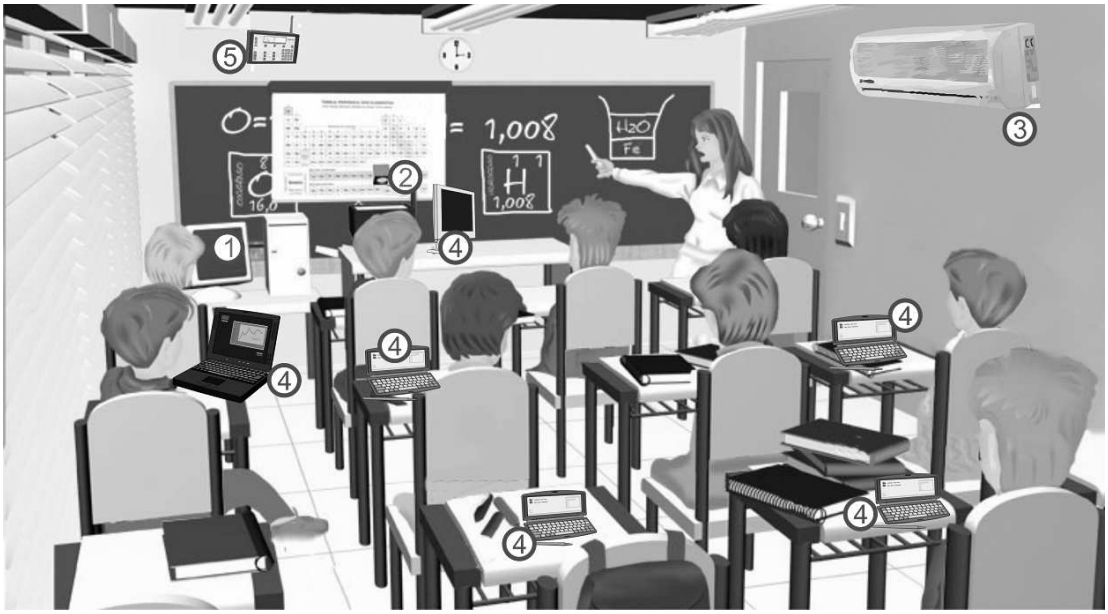

Figura 2.1: Exemplo de uma sala de aula pervasiva

presença marcada indevidamente, é enviado ao dispositivo do professor a lista dos alunos para que este a confira rapidamente.

A aula prossegue normalmente, sendo possível ainda que a apresentação da aula seja também exibida nos dispositivos dos alunos. Alguns minutos antes do fim da aula, o professor faz uma avaliação, um mini-teste, o qual é enviado aos dispositivos dos alunos. Dessa forma, os estudantes respondem as questões em seus próprios dispositivos e enviam as respostas a um servidor que analisa as respostas e gera um relatório com as notas dos alunos. De acordo com as preferências do professor, esse relatório deve ser impresso. Sendo assim, o sistema busca por um serviço de impressão que atenda as necessidades do professor e imprime a lista de presença juntamente com os resultados dos testes.

Embora cenários como este sejam certamente fascinantes, eles ainda são um tanto quanto futuristas. Mesmo que já existam as condições de *hardware* necessárias para conceber ambientes como este (e.g., redes sem fio, dispositivos móveis), a implementação de ambientes pervasivos como o descrito acima ainda é uma tarefa complexa em termos de *software*.

2.2 Princípios da Computação Pervasiva

Segue-se uma visão com mais detalhes sobre o que é a computação pervasiva, onde são apresentados a seguir quatro princípios fundamentais, quais sejam [HMNS03]: *Descentralização*, *Diversificação*, *Conectividade* e *Simplicidade*.

Descentralização

Quando a computação surgiu, havia o predomínio de uma visão centralizada, representada pelo modelo de *mainframes*. Neste modelo, poderosos computadores proviam sua capacidade de processamento para terminais incapazes de processar dados, de forma que cada computador era compartilhado por várias pessoas. Após algum tempo, a computação vivenciou o surgimento dos computadores pessoais, que se caracteriza principalmente pela possibilidade de cada pessoa possuir seu próprio computador, o qual possui um menor poder de processamento, se comparado aos *mainframes*.

Atualmente a computação está passando por um novo processo de evolução e descentralização. Este processo está proporcionando o surgimento de um novo modelo computacional: o de computação pervasiva. Este modelo distribui as responsabilidades entre vários dispositivos (e.g., celulares, PDAs), os quais assumem e executam certas tarefas e funções, simples e específicas para a finalidade que o dispositivo foi criado. Para isso uma rede dinâmica de relações é formada, entre os dispositivos e entre dispositivos e servidores do ambiente, caracterizando um sistema distribuído.

Para computação pervasiva, estes diversos dispositivos devem interagir de maneira dinâmica e autônoma, mantendo os dados sempre sincronizados em tempo de execução. Esta interação deve ocorrer entre os mais diferentes tipos de dispositivos, os quais possuem poder computacional diferentes (processamento, memória, etc).

Diversificação

Atualmente, é possível que uma pessoa compre um único computador (PC) que seja capaz de realizar uma vasta gama de funções. Por exemplo, um computador é capaz de ser um dispositivo para digitação de um texto, para navegar na Internet, etc. Ou seja, o usuário geralmente realiza todas as suas tarefas em uma estação de trabalho de propósito geral que

atende várias de suas distintas necessidades.

O paradigma de computação pervasiva introduz uma nova maneira de executar as tarefas desejadas pelos usuários. Nesse novo paradigma, ao invés de se ter um único computador que seja capaz de executar todas as tarefas, tem-se a diversificação de dispositivos que são responsáveis por determinada tarefa. Por exemplo, o *palmtop* é adequado para fazer anotações rápidas, mas não é o melhor dispositivo para navegar na web. Alguns dispositivos são desenvolvidos visando tornar mais fácil o acesso a conteúdo multimídia, outros são mais especializados nas tarefas de digitação de texto, etc. Dessa forma, os usuários terão um grupo de dispositivos que irão servir aos seus propósitos específicos.

Conectividade

Para a computação pervasiva, os diversos dispositivos existentes devem ser capazes de se comunicar de maneira eficiente, mesmo existindo diversas formas de comunicação. Atualmente existem diversas formas de realizar conexão entre dois dispositivos, entre as quais destacam-se: infra-vermelho, *Wireless Fidelity (Wi-Fi), Bluetooth,* apenas como alguns exemplos. Porém, não é viável que um único dispositivo possua todas estas interfaces de comunicação, primeiro porque a qualquer momento pode surgir uma nova forma de comunicação, segundo que isso faria com que estes fossem muito grandes e/ou caros. Outro ponto que deve receber atenção é o fato de que diferentes dispositivos possuem sistemas operacionais distintos, mas estes devem ser capazes de se comunicar de maneira transparente para o usuário.

Simplicidade

O uso de um computador como uma ferramenta para executar todos os tipos de tarefas acaba complicando a vida do usuário comum, pois é necessário que o mesmo seja capaz de instalar, configurar e ter profundos conhecimentos sobre o funcionamento dos softwares que estão sendo usados no computador. Em vez de uma máquina versátil e mais complexa, os dispositivos pervasivos devem ser especializados, o que os tornam menos aptos a um uso geral, porém bem mais simples de serem usados em seu propósito específico. Dispositivos pervasivos devem ser úteis, convenientes e simples de serem usados, de modo que não seja necessária a leitura de um complexo manual para que possam ser utilizados.

2.3 Contexto e Ciência de Contexto

Uma vez definidos os principais conceitos relacionados à computação pervasiva, pode-se agora, focar nas principais características de Ambientes Pervasivos, mais especificamente, nas características de Contexto e Ciência de Contexto em tais ambientes.

Um bom exemplo de um ambiente pervasivo é o da *Sala de Aula Pervasiva* apresentado na Seção 2.1. Neste exemplo ficam claras as principais características de um ambiente pervasivo, que são *dinamicidade* e *heterogeneidade*. Para lidar com estas características e ainda assim agir pró-ativamente em função dos usuários, é necessário que as aplicações se adaptem em tempo de execução às alterações do ambiente (e.g., número de pessoas ao redor e recursos/serviços disponíveis) e às necessidades dos usuários. Dentro do escopo de computação pervasiva, esta adaptabilidade é guiada por dois elementos chave: as noções de *contexto* e *ciência do contexto*, definidos a seguir:

Contexto

Uma das primeiras tentativas de definir contexto com foco em computação pervasiva foi feita em [ST94], onde foram identificados três tipos de contexto: *contexto de computação* (e.g., rede, custo de comunicação, estações de trabalho), *contexto de usuário* (e.g., perfil, localização, pessoas presentes no ambiente) e *contexto físico* (e.g., iluminação, barulho). Chen e Kotz [CK00] adicionaram mais uma característica ao trabalhos de Schilit, citado anteriormente, o *contexto de tempo* (e.g., dia, hora), que é um importante atributo na análise do *histórico* das informações do contexto.

Outros pesquisadores definem contexto através de uma categorização dos tipos de informações presentes no ambiente. Gwizdka [Gwi00], por exemplo, define duas categorias de contexto: *interno* (informações sobre o estado do usuário) e *externo* (informações sobre o ambiente onde o usuário está inserido). No trabalho de Petrel [PNS+00], são identificados dois tipos de contexto: *material*, que está associado à localização, dispositivos ou infra-estrutura disponível e *social*, encapsulando informações sobre o atual estado social do usuário (e.g., em uma reunião ou no cinema).

O problema encontrado em algumas das definições de contexto citadas anteriormente, está no fato de que elas utilizam exemplos para mostrar o que é contexto, ficando difícil

saber se determinada informação faz parte do contexto, caso ela não se ajuste a algum dos exemplos [DA99]. Em outras definições, o problema é que as mesmas estão fortemente associadas ao domínio de aplicação no qual o pesquisador está interessado. Portanto, faz-se necessária uma definição mais genérica e desassociada de exemplos.

Uma das definições de contexto mais aceitas que satisfaz tais necessidades é a que foi apresentada por [Dey01]:

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

Ciência do Contexto

A primeira discussão sobre computação ciente de contexto *(context-aware computing)*, apresentada em [ST94], defende o conceito de um software que "se adapte de acordo com sua localização, grupo de pessoas e objetos de sua vizinhança e alterações sobre estes objetos com o passar do tempo".

Refinando-se um pouco a idéia de ciência de contexto, é possível identificar duas formas de computação ciente de contexto: *ativa*, onde uma aplicação adapta-se automaticamente ao contexto descoberto, alterando seu comportamento; e *passiva*, onde uma aplicação apresenta o contexto, novo ou atualizado, para um usuário ou torna o contexto persistente para o usuário recuperá-lo posteriormente [CK00].

Essas definições estão relacionadas a um único aspecto de computação ciente de contexto, que é a necessidade de adaptação das aplicações ao contexto, não deixando explícito que essa adaptação depende do que é relevante aos interesses do usuário. Contornando este problema, Dey [Dey01] apresenta uma definição mais geral e bastante adequada ao escopo de computação pervasiva.

> "Um sistema está ciente do contexto se ele usa informação contextual para disponibilizar informações e/ou serviços relevantes para o usuário, onde a relevância depende da tarefa do usuário."

2.4 Habilitando Ciência de Contexto

Estar ciente do contexto é uma tarefa natural para os seres humanos. As pessoas geralmente usam o contexto ao seu redor para tomar decisões. Por exemplo, quando uma pessoa entra num teatro para assistir a uma peça ou um show, ela normalmente evita falar alto para não prejudicar a audiência das demais pessoas. Isso mostra que as pessoas normalmente estão cientes do contexto. Entretanto, fazer com que uma aplicação computacional esteja ciente do contexto requer que a aplicação seja capaz de adquirir e interpretar a informação contextual. Neste sentido, podem ser definidas três etapas com a finalidade de fazer com que uma aplicação esteja ciente do contexto: *Obtenção da Informação Contextual*, *Representação da Informação Contextual* e *Raciocínio sobre a Informação Contextual*.

2.4.1 Obtenção da Informação Contextual

Esta etapa consiste em adquirir as informações de contexto presentes no ambiente. De acordo com Mostéfaoui [MPRB04] pode-se classificar a informação contextual, de acordo com a forma como ela é obtida, em:

- *Sentida:* este tipo de informação pode ser adquirido do ambiente por meio de sensores (e.g., temperatura, nível de ruído, dispositivos presentes)
- *Derivada:* este é o tipo de informação que pode ser obtida em tempo de execução. Por exemplo, é possível calcular a idade de uma pessoa baseada na sua data de nascimento.
- *Provida:* informação que é explicitamente fornecida à aplicação. Por exemplo, os dados cadastrais de um usuário que é diretamente fornecido à aplicação por meio de um formulário.

Esta etapa de aquisição, entretanto, não é uma tarefa fácil, principalmente quando a informação é sentida. Isso ocorre devido à grande variedade de sensores. Além disso, informação contextual possui uma natureza dinâmica, sendo necessário que a aplicação gerencie todos esses aspectos.

2.4.2 Representação da Informação Contextual

Esta atividade consiste em prover um alto nível de abstração às informações de contexto, permitindo que as aplicações possam "entender" o que significam tais informações. Devido às características de dinamismo e heterogeneidade de ambientes pervasivos, o formato no qual essa informação deve ser representada precisa contemplar algumas características, de forma que essa informação possa ser [HBS02]:

- *Estruturada:* esta é uma característica importante no sentido de viabilizar a possibilidade de filtrar ou extrair eficientemente a informação do contexto que é relevante para a aplicação. Além disso, reduz a possibilidade de ambiguidade de atributos.

- *Intercambiável:* muitas vezes, informação contextual precisa ser trocada entre as aplicações, bem como entre os diferentes componentes da própria aplicação.

- *Composta/Decomposta:* compor/decompor informação do contexto é muito útil para prover manutenção de forma distribuída. Por exemplo, no caso de uma atualização da informação de contexto, pode ser enviada apenas aquela parte da informação que foi modificada, evitando que seja enviada novamente toda a informação do contexto de diferentes fontes.

- *Extensível:* este é um conceito fundamental para a representação da informação, pois permite que a qualquer momento sejam adicionados novos parâmetros, visto que, não há um conjunto de atributos que seja identificado hoje e sirva para todas as futuras aplicações.

- *Padronizada:* como a informação pode vir de diferentes entidades, é fundamental que a informação seja representada de forma padronizada.

2.4.3 Raciocínio sobre a Informação Contextual

Esta é uma das etapas mais importantes no trato com a informação contextual, consistindo em usar os dados coletados do ambiente para retornar informações relevantes para o usuário. Sem esta etapa, de nada adiantaria as anteriores, pois o grande objetivo da criação de aplicações cientes de contexto é justamente o provimento de informações relevantes ao usuário.

Atualmente, existe uma série de mecanismos que possibilitam o raciocínio sobre a informação contextual. Estes mecanismos podem consistir de simples instruções “se-então-senão”, ou estruturas mais complexas como raciocínio baseado em casos [MKM+05].

Os mecanismos de inferência que serão usados pelas aplicações depende diretamente de suas necessidades e do formato no qual a informação contextual está representado. Neste escopo, existem ainda algumas soluções que levam em conta a qualidade da informação obtida. Isto é necessário, pois, uma vez que a informação é adquirida por meio de sensores, ela pode conter um certo nível de imprecisão. Assim sendo, informações que contenham um alto nível de imprecisão devem ser desconsideradas, ou possuir um fator de ponderação sobre a confiabilidade da mesma. Neste sentido, existem algumas abordagens que se adaptam melhor a estes requisitos, tais como redes bayesianas [GPZ04] e lógica fuzzy [RAMC04].

Capítulo 3

Ontologias

Ontologias têm sido largamente utilizadas em áreas como gerenciamento de conteúdo e conhecimento, comércio eletrônico e Web semântica. Particularmente, a comunidade científica tem apontado o uso de ontologias para lidar com alguns dos principais desafios relacionados à construção de ambientes pervasivos. De um modo geral, ontologias têm sido usadas para representar ambientes pervasivos, descrevendo, comumente, entidades envolvidas e suas respectivas propriedades. Elas definem principalmente os diferentes tipos de aplicações, serviços, dispositivos, usuários, entre outros. Além disso, estas ontologias definem descrições padrões para localização, atividades, informação sobre temperatura, etc. Neste capítulo, são exibidos os principais conceitos relacionados a este assunto, partindo do conceito de ontologia, passando pelos principais tipos de ontologias, benefícios advindos do uso de ontologias e finalmente descrevendo as principais linguagens para ontologias.

3.1 O Conceito de Ontologia

Embora a palavra ‘ontologia’ denote, em sua origem filosófica, uma teoria sobre a natureza do ser, para a Computação, ela vem sendo usada como um conjunto de entidades com suas relações, restrições, axiomas e vocabulário. Segundo Gruber [Gru93], “uma especificação de um vocabulário de representação para um domínio de discurso compartilhado - definições de classes, relações, funções e outros objetos - é uma ontologia”.

O termo ontologia pode também ser definido a partir dos requisitos para possibilitar sua aplicação em Informática. Sendo assim, uma ontologia pode ser definida como “uma espe-

cificação explícita e formal de uma conceitualização compartilhada" [SBF98]. Esclarecendo os requisitos desta definição, tem-se que [Fre03]:

- Por "especificação explícita", pode ser entendida como sendo definições de conceitos, instâncias, relações, restrições e axiomas.
- Por "formal", que é declarativamente definida através de uma linguagem formal, portanto, compreensível para agentes inteligentes e sistemas.
- Por "conceitualização", que se trata de um modelo abstrato de uma área de conhecimento ou de um universo limitado de discurso.
- Por "compartilhada", por tratar-se de um conhecimento consensual, seja uma terminologia comum da área modelada ou acordada entre os desenvolvedores dos agentes que se comunicam.

3.2 Tipos de Ontologias

Por se tratar de uma área da ciência que se aplica a qualquer parte do conhecimento, ontologias podem ser classificadas em uma escala de generalidade [Miz04], de acordo com o propósito para o qual foi designada, como segue:

Ontologias de representação: definem as primitivas de representação - como *frames*, axiomas, atributos e outros - de forma declarativa. Esse tipo de ontologia serve para abstrair os formalismos de representação.

Ontologias gerais (ou de topo): trazem definições abstratas necessárias para a compreensão de aspectos do mundo (e.g., tempo, espaço, seres, coisas). Esses conceitos tipicamente são independentes de um problema particular ou domínio. Sendo assim, é bem razoável ter-se uma ontologia de alto-nível compartilhada por grandes comunidades de usuários.

Ontologias centrais (*core ontologies*) ou genéricas de domínio: definem os ramos de estudo de uma área e/ou conceitos mais genéricos e abstratos desta área. Por exemplo, a ontologia central de direito criada por Andre Valente [VB96], inclui conhecimentos

normativos, de responsabilidade, reativos, de agências legais, comportamentos permitidos, etc. Esses conceitos e conhecimentos foram agrupados nesta ontologia para que ela sirva de base para a construção de ontologias de ramos mais específicos do direito, como direito tributário, de família e outros.

Ontologias de domínio: tratam de um domínio mais específico de uma área genérica de conhecimento, como direito tributário, microbiologia, etc.

Ontologia de aplicação: procura solucionar um problema específico de um domínio, como identificar doenças do coração, a partir de uma ontologia de domínio de cardiologia. Normalmente, esse tipo de ontologia especializa conceitos tanto das ontologias de domínio, como também das de tarefas. Um exemplo disso é uma ontologia para uma aplicação que trabalhe com carros de luxo. Essa ontologia especializará conceitos da ontologia de veículos (que é uma ontologia de domínio).

Ontologias de tarefas: descrevem tarefas de um domínio (como processos, planos, metas, escalonamentos, etc.) com uma visão mais funcional, embora declarativa.

Como pode ser percebido, no que foi descrito acima os tipos de ontologias estão listados em ordem decrescente de generalidade. É importante salientar também que nem todos os tipos são necessários para a construção de uma aplicação, sem mencionar a importância em manter as ontologias reusáveis, ou seja, fazer com que uma ontologia seja elaborada de forma que possa ser usada em diferentes situações.

3.3 Benefícios das Ontologias

Além dos benefícios advindos de uma abordagem declarativa, que descreve fatos e entidades acerca de um determinado domínio (metáfora do “o que”), outros benefícios mais diretos, ligados à prática de construção de sistemas baseados em conhecimento, têm sido gerados. De início, o projeto Knowledge Sharing Effort (KSE) [NFF+91] de 1991 e suas ontologias contribuíram para uma maior cooperação entre os grupos de pesquisa responsáveis por manter as ontologias, da mesma forma como mantêm conhecimento, o que, tornando-se uma tendência, pode vir a provocar uma mudança cultural. Desde que foi criado o KSE, estão sendo

definidas e mantidas ontologias extensíveis, abrangentes, gerais e muito detalhadas, por grupos de pesquisa, abarcando toda a pesquisa da área cujo conhecimento se deseja representar. Esta orientação ontológica trouxe muitos benefícios, alguns dos quais não previstos, e que só vieram frutificar na época de sua implementação. São eles:

- A oportunidade para os desenvolvedores de *reusar ontologias* e bases de conhecimento, mesmo com adaptações e extensões. O impacto sobre o desenvolvimento de sistemas baseados em conhecimento é substancial: a construção de bases de conhecimento redunda na tarefa mais cara e demorada de um projeto de sistemas especialistas e/ou agentes. As ontologias permitem ainda aos usuários efetuarem consultas, comparações, integração e verificação de consistência;

- A disponibilização de uma vasta gama de *"ontologias de prateleira"*, prontas para uso, reúso e comunicação por pessoas e agentes. Hoje as ontologias mais maduras, algumas com mais de 2.000 definições, incluem metadados de imagens de satélites e para integração de bases de dados de genoma, catálogos de produtos, osciloscópios, robótica, semicondutores, terminologia médica, o padrão IEEE para interconexões entre ferramentas, entre outras;

- A possibilidade de tradução entre diversas linguagens e formalismos de representação de conhecimento. A tradução concretiza um ideal perseguido por gerações de pesquisadores de Inteligência Artificial. Ela facilita o reúso de conhecimento e pode vir a permitir comunicação entre agentes em formalismos diferentes, uma vez que este serviço encontra-se disponível para um número cada vez maior de formalismos de representação de conhecimento (para os formalismos tratados pela Ontolingua [FFR96], ver Figura 3.1). Outra forma de alcançar esse intento são editores de ontologias em que pode-se escolher em que linguagem de representação será escrito o código gerado. No editor Protégé-2000 [NFM00], podem ser geradas ontologias em CLIPS, Jess, Prolog, XML, RDF, OIL, DAML-OIL e F-Logic;

- O acesso *on-line* a servidores de ontologias, capazes de armazenar milhares de classes e instâncias, que serviriam a várias empresas ou grupos de pesquisa, e que podem funcionar como ferramentas para manter a integridade do conhecimento compartilhado entre elas, garantindo um vocabulário uniforme;

- O mapeamento entre formalismos de representação de conhecimento, que, inspirado no componente de conectividade para sistemas gerenciadores de bancos de dados ODBC (*Open Database Connectivity*), integra dois formalismos criando uma interface interoperável de acesso comum para eles, permitindo a um agente acessar o conhecimento de outro agente. O pacote gerado para implementar esta facilidade é chamado de OKBC (*Open Knowledge Base Connectivity*).

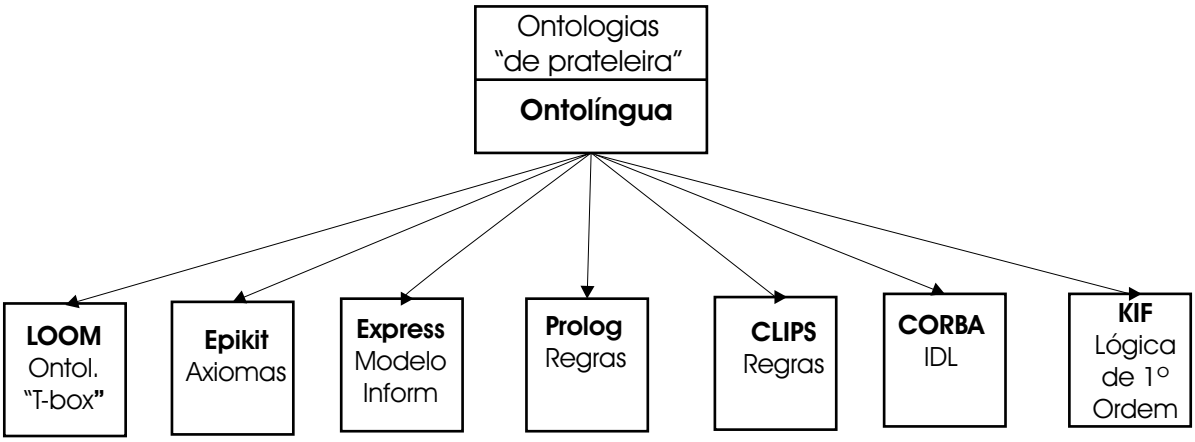

Figura 3.1: A Ontolíngua e os formalismos para os quais podem ser traduzidas as ontologias

3.4 Linguagens para Ontologias

Ontologias estão intimamente relacionadas com a linguagem usada para representá-las. Atualmente, existem algumas linguagens com esse propósito. A seguir, é apresentada uma visão geral sobre as principais linguagens, assim como das ontologias de cada linguagem.

3.4.1 RDF

RDF[1] (*Resource Description Framework*) é uma linguagem de propósito geral para representar informação na Internet que baseia-se na idéia de identificar coisas através identificadores Web: os URIs (*Uniform Resource Identifier*).URIs são cadeias de caracteres utilizadas para

[1]http://www.w3.org/RDF/

identificar recursos na Web, como páginas, serviços, documentos, etc. Além dos identificadores Web (URIs), esta linguagem descreve recursos em termos de simples propriedades e valores. Isto permite que RDF represente recursos sob a forma de expressões sujeito-predicado-objeto:

- *O sujeito:* é o recurso, ou seja, qualquer coisa que pode conter um URI, incluindo as páginas da Web, assim como elementos de um documento XML.
- *O predicado:* é uma característica descritiva ou aspecto do recurso e por vezes expressa uma relação entre o sujeito e o objeto.
- *O objeto:* é o objeto da relação ou o valor da característica descritiva

RDF é um tipo de rede semântica [Sow92], sendo parecida, em termos de linguagem, com o Modelo Relacional [EN94]. Isto implica que RDF é uma forma de representação de conhecimento que possui semântica auto-contida e oferece uma grande liberdade para criação de extensões personalizadas.

3.4.2 RDF Schema

RDF Schema (RDFs) é uma linguagem para representação de conhecimento que baseia-se na idéia de *Frames* [Bub05]. Ela tem sido usada para aumentar a expressividade de RDF, dispondo assim de um melhor suporte à definição e classificação. Este modelo organiza o conhecimento através de herança e de construtores de ontologias (*frames, slots* e facetas). Os *frames* são organizados em rede, significando que quando qualquer um deles for acessado, ligações com outros quaisquer, potencialmente importantes, estarão disponíveis, podendo ser visto como uma "unidade de conhecimento" auto-suficiente.

Um *frame* é uma descrição de um objeto complexo. Ele é identificado por um nome e consiste de um conjunto de *slots*. Cada *slot* possui um nome único ao *frame* em que está definido, consistindo de um conjunto de *facetas* (atributos) de valores particulares. Sistemas baseados em *frames* permitem que os usuários representem o mundo com diferentes níveis de abstração, com ênfase sobre as entidades .

Em adição ao que já é herdado pelo fato de basear-se em frames, RDFs dispõe de construtores de ontologias que tornam as relações menos dependentes de conceitos: usuários podem

definir relações como uma instância de *rdf:Property*, descrever relações de herança como *rdfs:subPropertyOf* e então associar relações definidas com classes usando *rdfs:domain* ou *rdfs:range* [DKD+05].

3.4.3 OWL

A OWL (*Web Ontology Language*) é uma linguagem para definir e instanciar ontologias na Web. Ela foi projetada para disponibilizar uma forma comum para o processamento de conteúdo semântico da informação na Web. Ela foi desenvolvida para aumentar a facilidade de expressar semântica disponível em XML, RDF e RDFs. Conseqüentemente, pode ser considerada uma evolução destas linguagens em termos de sua habilidade de representar conteúdo semântico da Web interpretável por máquinas. Já que a OWL é baseada em XML, a informação pode ser facilmente trocada entre diferentes tipos de computadores usando diferentes sistemas operacionais e linguagens de programação. Por ter sido projetada para ser lida por aplicações computacionais, algumas vezes considera-se que a linguagem não possa ser facilmente lida por humanos, porém esta é uma questão que pode ser resolvida utilizando-se de ferramentas adequadas. OWL vem sendo usada para criar padrões que forneçam um arcabouço para gerenciamento de ativos, integração empresarial e compartilhamento de dados na Web.

OWL atualmente tem três sub-linguagens (algumas vezes também chamadas de "espécies"): OWL Lite, OWL DL e OWL Full. Estas três sublinguagens possuem nível crescente de expressividade, e foram projetadas para uso por comunidades específicas de programadores e usuários[2].

- *OWL Lite* dá suporte aqueles usuários que necessitam principalmente de uma classificação hierárquica e restrições simples. Por exemplo, embora suporte restrições de cardinalidade, ela só permite valores de cardinalidade 0 ou 1. É mais simples fornecer ferramentas que suportem OWL Lite que seus parentes mais expressivos, e ela também permite um caminho de migração mais rápido de dicionários e outras taxonomias.

- *OWL DL* suporta aqueles usuários que querem a máxima expressividade, enquanto mantém a computabilidade (garante-se que todas as conclusões sejam computáveis) e

[2]http://www.w3.org/TR/owl-guide/

decidibilidade (todas as computações terminarão em tempo finito). OWL DL inclui todas as construções da linguagem OWL, porém elas somente podem ser usadas com algumas restrições (por exemplo, embora uma classe possa ser subclasse de muitas classes, uma classe não pode ser instância de outra classe). OWL DL é assim chamada devido a sua correspondência com as lógicas de descrição, um campo de pesquisa que estudou a lógica que forma a base formal da OWL.

- *OWL Full* é direcionada àqueles usuários que querem a máxima expressividade e a liberdade sintática do RDF sem nenhuma garantia computacional. Por exemplo, em OWL Full uma classe pode ser tratada simultaneamente como uma coleção de indivíduos e como um único indivíduo. OWL Full permite que uma ontologia aumente o vocabulário pré-definido de RDF ou OWL.

Capítulo 4

LOTUS

Lotus é uma ferramenta que permite disponibilizar informação contextual existente em ambientes pervasivos às aplicações presentes nos mesmos, fazendo com que estas aplicações possam estar cientes do contexto no qual estão inseridas. Estar ciente do contexto é fundamental para o funcionamento de tais aplicações, dado que ambientes pervasivos são extremamente dinâmicos e heterogêneos, o que requer que as aplicações se adaptem em tempo de execução às necessidades do usuário e alterações no ambiente. Além disso, *Lotus* provê suporte para que a informação contextual seja compartilhada por diferentes aplicações.

Para contemplar estas características, a *Lotus* possui uma arquitetura definida de acordo com as etapas necessárias para tornar uma aplicação ciente do contexto. A arquitetura projetada para a *Lotus* permite em um processo único que a informação contextual seja usada por aplicações em ambientes pervasivos. Na Figura 4.1 é ilustrada a arquitetura da *Lotus*. Nela podem ser observados quatro módulos que são responsáveis por:

- *Aquisição da informação contextual*: efetuada através do módulo de *Aquisição*, que permite que seja feita a descoberta de nós e serviços, mesmo em redes heterogêneas;
- *Representação da informação contextual*: realizada através do módulo *Representação*. Este módulo permite a representação tanto de informações peculiares a ambientes pervasivos (i.e., aquilo que é comum a qualquer ambiente pervasivo) quanto informações referentes a domínios de aplicação específicos;
- *Atualização dinâmica da informação*: para refletir a característica de dinamicidade de ambientes pervasivos, foi desenvolvido um mecanismo que permite que a informação

seja atualizada em tempo de execução através do *Inferência*. Este módulo permite ainda que sejam realizadas inferências sobre o módulo *Representação* através do uso de uma API, a *Pellet*, que permite que sejam feitas consultas através da linguagem *SPARQL* [HMvdSW04].

- *Disponibilização da informação*: realizada pelo módulo *Disponibilização e Compartilhamento*, permite que a informação seja compartilhada por várias aplicações.

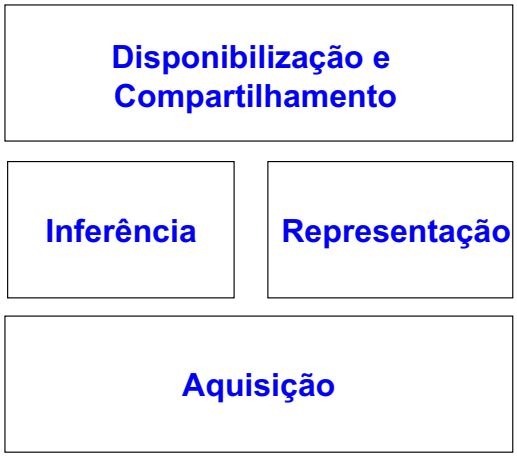

Figura 4.1: Arquitetura da *Lotus*

4.1 Aquisição

Uma das principais características de ambientes pervasivos é a capacidade de interação entre os diversos dispositivos. A própria definição dada para ambientes pervasivos, como sendo ambientes "repletos de capacidade de computação e comunicação" deixa isso claro. Sendo assim, faz-se necessário que os dispositivos presentes em um ambiente pervasivo sejam capazes de interagir entre si, mesmo que esses dispositivos tenham diferentes formas de comunicação (e.g., *bluetooth* e *Wi-Fi*). Isso significa que uma aplicação precisa estar preparada para reconhecer e comunicar-se com qualquer interface de rede, ou, caso contrário, perderá a capacidade de comunicar-se com os demais dispositivos que não possuam a mesma interface de comunicação.

Atualmente existem diversas formas de realizar conexão entre dois dispositivos e possibilitar a comunicação entre eles (e.g., fios, infra-vermelho, *Wi-Fi*, *Bluetooth*). Porém não

é viável que um único dispositivo possua todas estas interfaces de comunicação ao mesmo tempo. Isso ocorre devido aos seguintes fatores: ① acarreta na necessidade de mais espaço em disco e um maior uso de memória, uma vez que cada uma dessas interfaces necessita estar executando no dispositivo (recursos esses ainda escassos em muitos dispoositivos, principalmente dispositivos com porte menor, como celulares e PDAs); ② como a tecnologia está sempre evoluindo, é possível que surjam novas formas de comunicação. Nesse caso, um dispositivo que não tenha sido previamente dotado de tal capacidade de comunicação, não poderá comunicar-se com os dispositivos dotados com esse tipo de comunicação. Isso sem mencionar que as formas de comunicação já existentes em um dispositivo podem se tornar obsoletas, consumindo recursos do mesmo.

Como pode ser facilmente percebido, lidar com a grande diversidade de dispositivos e formas de comunicação que cada um possui é uma tarefa que exige esforço e cautela, uma vez que é necessário não apenas preocupar-se em dispor os dispositivos com capacidade de comunicação, mas também estar atento às limitações dos mesmos. Para contornar esse tipo de problema, a *Lotus* possui um módulo responsável exclusivamente pela comunicação entre os dispositivos. Esse módulo é responsável pelo reconhecimento e descoberta de nós (dispositivos) e serviços (os quais são disponibilizados por esses dispositivos). A implementação desse módulo é baseado no *middleware Wings* descrito a seguir.

4.1.1 wings

O *Wings* foi desenvolvido com propósito de permitir *disponibilização de serviços* e *descoberta de nós* em ambientes pervasivos, através do uso de uma arquitetura baseada em *plug-ins*, que caracterizam-se pela existência de um núcleo funcional, no qual extensões podem ser dinamicamente adicionadas [Bir05]. Para isso, é necessário que sejam implementados os respectivos *plug-ins* de *Disponibilização de Serviços* (PDS) e *Descoberta de Nós (PDN)*.

A principal motivação para encapsular a disponibilização de serviços e descoberta de nós em *plug-ins*, é a possibilidade de adicionar diferentes *plug-ins*, permitindo assim, acessar serviços e nós remotos utilizando diferentes soluções. Como exemplo, pode-se ter dois PDSs inseridos no *Wings*, um implementado sobre *Bluetooth* e outro sobre *UPnP*, permitindo assim, publicar, descobrir e utilizar serviços através dessas duas soluções. Na Figura 4.2, pode ser visto um dispositivo acessando serviços em duas redes diferentes, para isso, basta que o

dispositivo possua os respectivos *plug-ins* instalados.

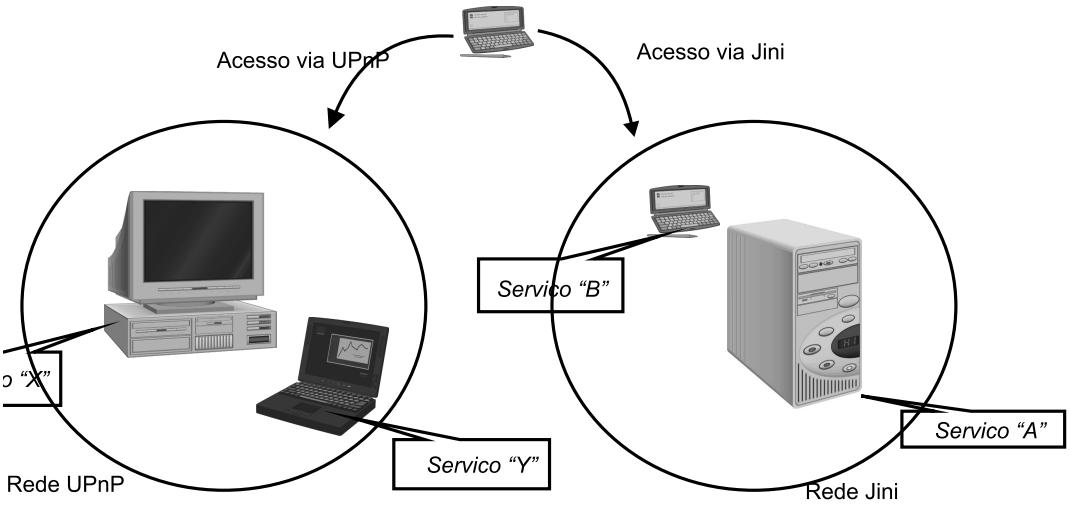

Figura 4.2: Publicação, descoberta e uso de serviços em redes heterogêneas

A fim de que a *Lotus* possa ser notificado da descoberta de nós, o *Wings* conta com um sistema de notificação baseado em eventos. Para isso é necessário que seja implementado o padrão *Observer* para descoberta de nós. Na Figura 4.3, ilustra-se o diagrama de classes deste padrão, detalhado a seguir:

- A classe **Wings** comunica-se com o wings através da *Fachada* disponibilizada pelo mesmo. A inicialização da descoberta de nós é realizada através do método `discoverPeers`. Este método recebe como parâmetro uma instância do tipo `DeviceSearchListener`; dessa forma sempre que um novo dispositivo é encontrado, este *listener* é informado.

- Um `DeviceSearchListener` é o responsável por lidar com a informação de que um novo dispositivo foi encontrado através do método `deviceFound`. Este método deve ser implementado pelas classes que implementam a interface `DeviceSearchListener`, mais especificamente a classe `UpdateDevices` que se comunica com o módulo de *Raciocínio* (veja o funcionamento desse módulo na Seção 4.3).

- A classe `MyThread` é uma *Thread* usada para que seja possível efetuar várias buscas em paralelo, de acordo com a necessidade das aplicações. Cada busca é identificada por um `searchID` que é justamente o retorno do método `discoverPeers` da classe

`Wings`. Esse identificador é usado também para que a busca possa ser interrompida e a *Thread* finalizada.

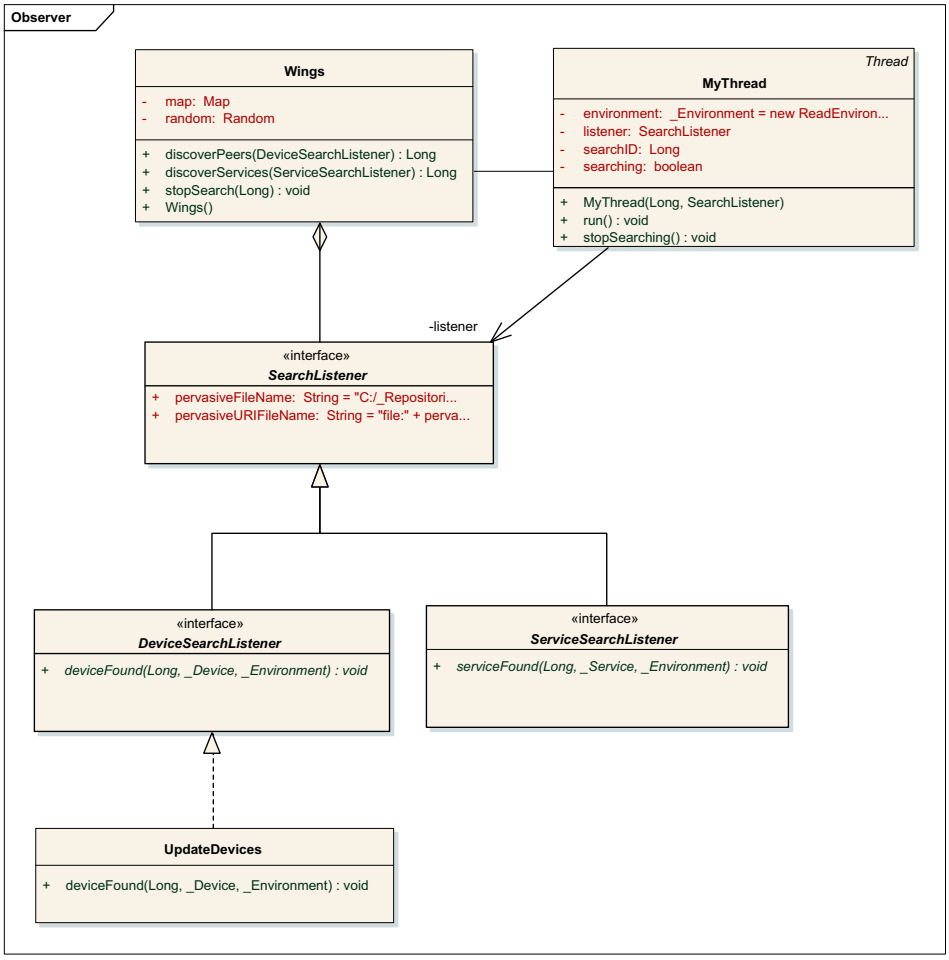

Figura 4.3: *Observer* para a descoberta de nós

4.2 Representação

Agora que é possível efetuar reconhecimento e descoberta de dispositivos e serviços presentes no ambiente (através do módulo de *Aquisição*), é necessário agora fornecer os mecanismos necessários para que essas informações sejam repassadas para as aplicações. Porém, para habilitar essa informação a fim de seja utilizada por qualquer aplicação, é necessário que a mesma esteja o mais desacoplada possível de uma linguagem de programação, ou do protocolo de comunicação pelo qual a informação foi descoberta. Essa etapa, é também

chamada de *representação da informação* e consiste em prover um alto nível de abstração à informação contextual, permitindo que as aplicações possam "entender" o significado delas.

Aproveitando-se dos resultados oriundos da pesquisa realizada para Web semântica, pôde-se fazer um apanhado das principais formas de representação da informação levando-se em conta vários aspectos, tais como *expressividade* e *semântica auto-contida* destas formas de representação. Na Tabela 4.1 [DKD+05], apresenta-se um resumo dos principais resultados obtidos dos estudos para Web semântica. Esta tabela agrupa os principais formalismos para representação de conhecimento, entre os quais destacam-se formalismos baseados em conhecimento (knowledge-based formalisms - KB), modelos formais de Banco de Dados (database - DB) e os principais construtores de ontologia (RDF/RDFS e OWL).

Considerando-se conjuntamente a necessidade de semântica auto-contida e expressividade (fatores essenciais para possibilitar que seja feita uma melhor inferência sobre a informação) e ainda tendo em mente as características necessárias à uma boa representação (Seção 2.4.2), chega-se à conclusão de que o uso de ontologias, mais precisamente através de OWL-DL mostra-se como uma boa escolha para a representação do conhecimento, sendo assim, adotado na *Lotus*.

Uma vez definida a forma como a informação vai ser representada, precisa-se estar atento ao fato de que cada aplicação pode necessitar de alguma informação que não seja necessária por outra aplicação. Dessa forma, faz-se necessário agora definir que tipo de informação precisa estar disponível para as aplicações, pois, pela própria característica de contexto, o que é importante para uma aplicação, pode não ser necessário para outra.

Para contornar esse problema, *Lotus* possui dois tipos de ontologias: *ontologia pervasiva* e *ontologias de domínio*. Juntas estas ontologias compõem o contexto necessário às aplicações. A primeira diz respeito à informação que pode ser adquirida diretamente do ambiente e que é comum a qualquer aplicação executando em um ambiente pervasivo. A segunda, por outro lado, está relacionada com informações que pertencem a um determinado domínio de aplicação.

Para definir a *ontologia pervasiva*, foi adotado o critério de reunir a maior quantidade de informação possível que seja útil a qualquer aplicação pervasiva. Em outras palavras, esta abordagem consiste de reunir a maior quantidade de informação do ambiente, mas de tal forma que essa informação seja comum a qualquer ambiente pervasivo e que essa informa-

CAT-1	CAT-2	Construtores	RDF	RDFS	OWL	DB Rel*	DB OO**	KB Frame	KB DL
classe	definição	Class		X	E		X	X	X
		Enumerated Class			X				O
		Restriction			X			O	X
		intersectionOf			X			O	X
		complementOf			X				
	axioma	subclassOf		X	H		X	X	X
		Equality			X			O	O
		disjointWith			X			O	
relação	definição	Property	X	H	E	X	X	O	
		domain, range		X	H			O	
		subPropertyOf		X	H				
	axioma	(inverse) Functional			X	X			
		Equality, inverseOf			X				
instância	definição	Type	X	H	H		X	X	X
	axioma	(In)Equality			R		O	O	O
* DB-Rel = Banco de Dados Relacional									
** DB-OO = Banco de Dados Orientado a Objetos									
R - Suporta c/ Restrições			X - Suportta				O - Opcional		
H - Herdado (RDF/RDFS)			E - Extendido (RDF/RDFS)						

Tabela 4.1: Comparação entre as principais formas de representação de conhecimento

ção seja útil a qualquer domínio de aplicação, isto é, uma interseção entre os domínios de aplicação, resultando assim no domínio de ambiente pervasivo (veja a Figura 4.4)

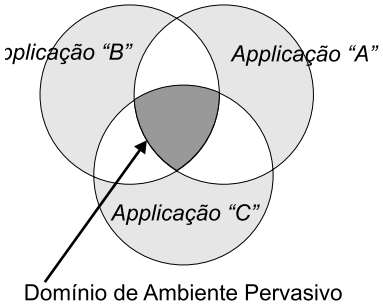

Figura 4.4: Interseção entre vários domínios de aplicação

A *ontologia pervasiva* é composta de quatro classes principais (Figura 4.5). A seguir detalha-se essa ontologia.

- *Environment:* Esta classe é a mais abstrata da *ontologia pervasiva*. Ela representa um *ambiente* pervasivo (e.g., uma sala de aula, um automóvel, um laboratório). Um ambiente, na *ontologia pervasiva*, é representado pelas seguintes propriedades:

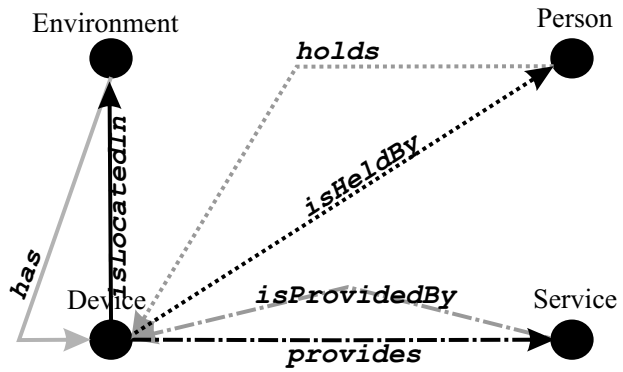

Figura 4.5: Lotus Ontologia Pervasiva

- *environment name:* esta é uma propriedade *funcional* usada para que um ambiente seja identificado por um nome. Isso é garantido pelo uso da propriedade funcional, que é definida como sendo aquela que pode ter apenas um único valor Y para cada instância X, i.e., não pode haver dois valores distintos Y_1 e Y_2 tal que os pares (X, Y_1) e (X, Y_2) sejam instâncias desta propriedade[1].
- *description:* propriedade *funcional* usada para descrever o ambiente.
- *has*: esta é uma propriedade especial, pois se trata de propriedade de objeto. Este tipo de propriedade é usado para relacionar um objeto (classe) a outro. Nesse caso específico, ela está sendo usada para relacionar um *Environment*(ambiente) a vários *Devices* (dispositivos), indicando que um ambiente *tem* vários dispositivos. Estes dispositivos, obviamente, podem mudar de ambiente, uma vez que eles possuem a característica de mobilidade. Para retratar isso, essa propriedade possui uma propriedade *inversa*: *isLocatedIn* de *Device*, que indica qual ambiente (*Environment*) o dispositivo está localizado no momento.

- *Person:* Esta classe é usada para representar uma *pessoa*. Cada pessoa pode possuir vários dispositivos, sendo representada pelas seguintes propriedades:
  - *person_name*: o nome da pessoa;
  - *login* e *password*: estas propriedades *funcionais* servem para identificar um usuário.

[1] http://www.w3.org/TR/owl-ref/

  - *holds*: esta propriedade de objeto indica que uma pessoa possui um (ou vários) dispositivo(s). Esta propriedade é inversa à propriedade *isHoldBy* de *Device*

- *Device:* Esta classe é usada para representar um dispositivo (e.g., telefone, computador pessoal, PDA). Cada dispositivo é associado a uma pessoa (*Person*), que é o individuo que o utiliza, e com vários serviços (*Services*) que o mesmo provê.

  - *CCPP* (*Composite Capability/ Preference Profiles*): um *CC/PP profile* é identificado por uma URI (Seção 3.4.1) e usado para prover informações sobre características específicas do dispositivo (e.g., tamanho da tela).
  - *isLocatedIn*: esta propriedade serve para indicar em qual ambiente o dispositivo está localizado no momento.
  - *provides*: relaciona todos os serviços (*Services*) que o dispositivo disponibiliza (inversa de *isProvidedBy* de *Service*).

- *Service:* Esta classe representa um serviço disponibilizado por algum dispositivo no ambiente. Note que um "mesmo" serviço pode ser disponibilizado por mais de um dispositivo. Por exemplo, um serviço de impressão pode ser disponibilizado por mais de uma impressora. As seguintes propriedades são usadas para caracterizar um serviço:

  - *service_name*: o nome do serviço;
  - *description*: descrição do serviço. É usado principalmente para que a aplicação possa ter uma maior quantidade de detalhes sobre o funcionamento do serviço e escolher aquele que mais se adapta à tarefa que precisa ser efetuada.
  - *isProvidedBy*: esta propriedade relaciona o serviço com os dispositivos (*Devices*) que o fornece.

As *ontologias de domínio* são necessárias para prover um melhor entendimento da informação contextual, de acordo com o domínio ao qual a aplicação pertença. Por exemplo, considere a sala de aula pervasiva, descrita da Seção 2.1. Nesse exemplo, foram apresentadas várias aplicações que estavam executando no ambiente. Uma dessas aplicações era a aplicação que assinalava a lista de presença. Sabe-se que numa sala de aula existem basicamente dois tipos de pessoas, que são os professores e os alunos (para simplificar o entendimento

não se considera aqui a presença de diretor, pais de alunos, funcionários da administração, entre outros). Para que essa aplicação possa agir corretamente, faz-se necessário então que a aplicação possa distinguir se uma *pessoa* é um *aluno* ou um *professor*, para isso é necessário que essa informação esteja presente em algum lugar. Na *Lotus* esse tipo de informação pode ser provida através das *ontologias de domínio*.

Lotus permite que as informações de domínio sejam criadas de duas maneiras distintas:

1. Estendendo a *ontologia pervasiva*: por exemplo, no caso da aplicação que assinala a lista de presença da sala de aula pervasiva, as novas classes *Student* (Aluno) e *Teacher* (Professor) poderiam estender da classe *Person* (Pessoa) da *ontologia pervasiva* (veja a Figura 4.6);

2. Criando novas *ontologias de domínio*: por exemplo, uma aplicação que execute num ambiente de *shopping pervasivo*, mais especificamente num restaurante e que seja responsável por recomendar vinhos de acordo com as preferências do usuário e do prato selecionado para refeição, pode necessitar de uma ontologia de vinhos. Nesse caso, o desenvolvedor fica responsável por tal ontologia e a mesma pode ser usada por outras aplicações que também necessitem de tal ontologia.

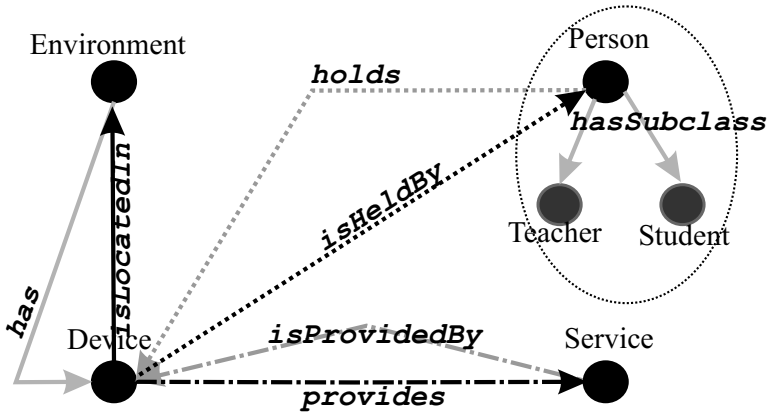

Figura 4.6: Estendendo a Ontologia Pervasiva

4.3 Raciocínio

Uma das principais características de ambientes pervasivos é sua dinamicidade, característica ligada principalmente à mobilidade de grande parte dos dispositivos presentes num ambiente

pervasivo, que permite que um dispositivo possa chegar/deixar um ambiente a qualquer momento. Além disso, um dispositivo pode simplesmente deixar de prover algum serviço, passar a oferecer outro, etc. Essas informações precisam de algum modo serem repassadas às aplicações. Na *Lotus*, estas informações ficam presentes no módulo *Representação* e podem ser acessadas diretamente pelas aplicações.

Entretanto, para que a informação presente no módulo*Representação* retrate o estado real do ambiente, é preciso que ela seja atualizada dinamicamente. Por exemplo, quando uma pessoa entra em determinado ambiente (veja o exemplo da sala de aula pervasiva, Seção 2.1), é necessário que essa informação seja captada pelo módulo de *Aquisição* e repassada para o repositório de ontologias, passando a estar disponível para as aplicações executando no ambiente. Nesse sentido, foi criado o módulo de *Raciocínio*, que é responsável por receber a informação enviada pelo módulo de *Aquisição*, interpretar essa informação, fazer com que essa informação esteja presente no módulo *Representação* e passe a estar acessível às aplicações.

Para manipular as informações presentes no módulo *Representação*, o módulo de *Raciocício* utiliza-se também da *Protege-OWL API* [2]. Através des0pelo módulo de *Aquisição*. O processo de atualização das ontologias acontece da seguinte maneira:

1. O *listener* `SearchListener` (veja Figura 4.3), uma interface que pode ser um `DeviceSearchListener` ou um `ServiceSearchListener`, recebe eventos informando quando ocorre alguma alteração no ambiente (mudança do contexto de serviços ou dispositivos). A partir de agora será descrito como funciona a atualização de dispositivos - o processo de atualização de serviços é similar.

2. A classe `UpdateDevices` recebe a informação (através do disparo de eventos) que houve uma alteração no contexto de dispositivos. Através do método `deviceFound` que recebe como parâmetros um dispositivo (representado pela classe `_Device`) e o ambiente (representado pela classe `_Environment`) no qual o dispositivo está inserido.

3. Através da *fachada* `PervasiveFacade`, que possui um *OWLModel* (necessário para que o Protege possa acessar as ontologias) a classe `UpdateDevices` consegue acessar

[2]http://protege.stanford.edu/plugins/owl/api/guide.html

o repositório de ontologias e atualizar as informações nas ontologias através do uso da *Protege-OWL API* (Veja Figura 4.7).

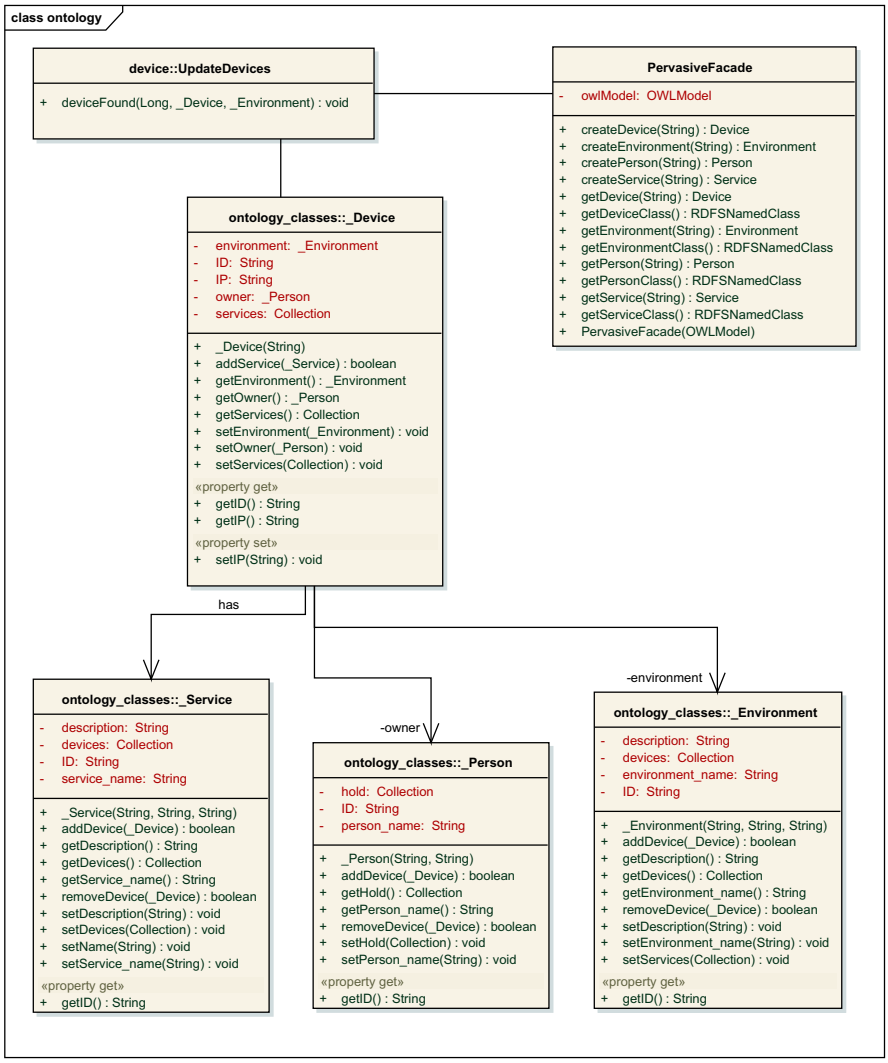

Figura 4.7: Diagrama de classes do módulo de Raciocícnio

Para acessar as ontologias, a *fachada* `PervasiveFacade` faz uso de um `OWLModel`, fornecido pela *Protege-OWL API*, que permite manipular qualquer uma das ontologias da *Lotus*. Para isso, cada uma das classes que fazem referência a uma determinada classe da ontologia estende uma classe da *Protege-OWL API* chamada `OWLIndividual`. Na *Lotus* para cada classe da *ontologia pervasiva* existe uma interface, em Java, correspondente. Estas interfaces definem os mecanismos de acesso às ontologias e suas implementações como ocorrerão esse acesso. Na *Lotus*, cada uma das interfaces que acessa a *ontologia pervasiva* já possui uma implementação padrão. Por exemplo, a interface `Device`, possui uma implementação

padrão `DefaultDevice`, como pode ser visto na Figura 4.8[3].

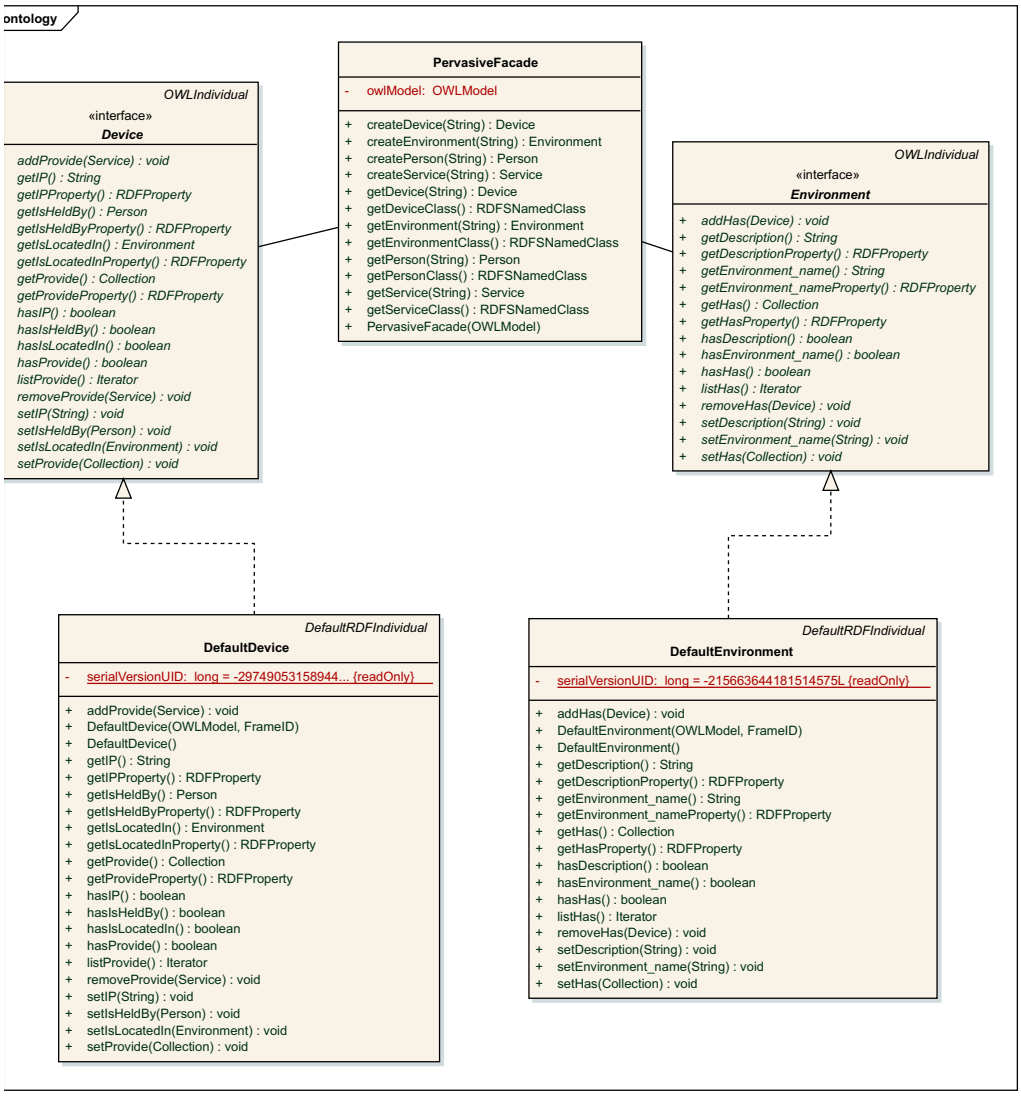

Figura 4.8: Diagrama de classes da implementação padrão para acesso à ontologia pervasiva.

É através da implementação dessas interfaces que é possível atualizar as ontologias dinamicamente. Através do uso da *Protege-OWL API*, é possível que um arquivo .owl (que representa a ontologia) seja gerado e que as alterações detectadas pelo módulo de *Aquisição* sejam repassadas para esse arquivo. Na listagem 4.1 é descrito um pequeno trecho do arquivo gerado por este módulo. Observe que as classes *Person*, *Device*, *Environment* e *Service* são disjuntas (cláusula *disjointWith*) indicando que uma instância de uma dessas classes não pode ser instância de outra ao mesmo tempo. Note ainda a existência de propriedades

[3]Neste diagrama estão sendo ilustradas apenas as classes referentes a dispositivos (*Device*) e ambiente (*Environment*) a fim de simplificar a visualização

objeto (*owl:ObjectProperty*) como *isProvidedBy*.

Listing 4.1: Arquivo gerado pelo Lotus

```
<owl:Class rdf:ID="Person">
 <rdfs:commentrdf:datatype=
          "http://www.w3.org/2001/XMLSchema#string">
          This class represents a person,
          which can be viewed as a user.
  </rdfs:comment>
  <owl:disjointWith>
    <owl:Class rdf:ID="Device"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:ID="Environment"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:ID="Service"/>
  </owl:disjointWith>
</owl:Class>
        .
        .
        .
<owl:ObjectProperty rdf:ID="provide">
  <rdfs:range rdf:resource="#Service"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="isProvidedBy"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#Device"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >This property means that a device provide services. </rdfs:comment>
</owl:ObjectProperty>
       .
       .
       .
```

4.4 Disponibilização e Compartilhamento

Até o momento foi descrito como a arquitetura da *Lotus* procede para recuperar a informação do ambiente e como fazer com que essa informação possa ser representada dinamicamente. Mas é preciso ainda fazer com que a informação esteja disponível para as aplicações. Na *Lotus*, devido ao fato dessa informação ser representada através de ontologias, isso poderia

ser feito facilmente, necessitando apenas tornar acessível tais ontologias. Porém, devido às limitações impostas pelos dispositivos móveis, tanto em nível de processamento, quanto de memória e espaço para armazenamento, essa abordagem não se mostra muito eficiente, pois faz-se necessário que as aplicações possuam algum mecanismo para manipular as ontologias.

Para contornar esse problema, a *Lotus* faz uso de uma arquitetura cliente-servidor para tornar a informação contextual disponível para as aplicações. Dessa forma, a informação contextual e todo o processo de aquisição da informação são efetuados no lado servidor, que torna essa informação disponível para as aplicações executando nos demais dispositivos do ambiente, os quais atuam como clientes dessa aplicação. Essa abordagem elimina um série de gargalos de execução e armazenamento, prevenindo uma série de complicações e limitações impostas pelos dispositivos móveis. Mais especificamente, a *Lotus* disponibiliza a informação por meio de uma arquitetura de *Web Service* [4]. Uma arquitetura de *Web Service* é composta de três elementos principais (conforme pode ser visto na Figura 4.9):

- *Publish (Publicação):* para que um serviço possa estar disponível é necessário que o mesmo seja publicado pelo *Service Provider* para um *Service Broker*. No caso da *Lotus*, o próprio lotus é que funciona como o provedor de serviço, mais especificamente, pode-se dizer que ele disponibiliza o serviço de informação contextual. Essa etapa de publicação é feita através do protocolo UDDI (protocolo desenvolvido para a organização e registro de *web services* [5]).

- *Find (Busca):* essa operação consiste na busca por *Web services*, onde *services requesters* buscam pelos serviços desejados através de um *service broker*. Note que a especificação dos serviços é feita por meio de um documento *WSDL*. Esse documento possui um formato XML que permite que as interfaces do sistema sejam descritas, assim como detalhes de protocolos. É através do arquivo *WSDL* que são especificadas as operações implementadas pelo servidor e através do qual são criados os clientes.

- *Bind (Ligação):* Após a descoberta do serviço desejado, o lado cliente liga-se ao lado servidor e passa a fazer uso do serviço por meio do protocolo *SOAP*, no qual chamadas às operações, incluindo os parâmetros de entrada/saída, são codificadas.

[4] http://www.w3.org/TR/ws-arch/#introduction

[5] http://www.uddi.org/

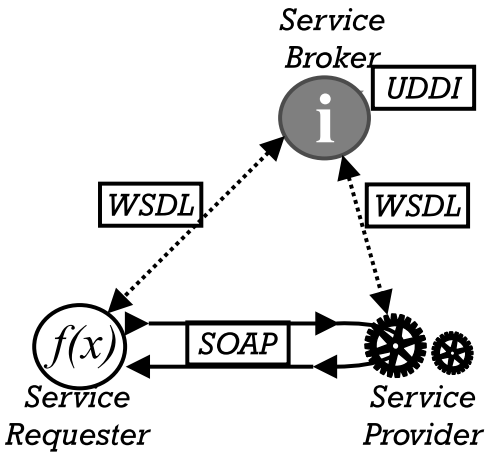

Figura 4.9: Arquitetura de um Web Service.

Para disponibilizar a informação contextual presente nas ontologias, a *Lotus* faz uso de um *Web service*, que oferece algumas vantagens:

- Por se tratar de uma arquitetura cliente-servidor, o web-service permite que a informação seja trabalhada no lado servidor, minimizando os problemas decorrentes das limitações dos dispositivos móveis;

- Uma vez que o serviço está disponibilizado na Web, é possível acessá-lo de qualquer lugar com acesso à Internet, permitindo que o dispositivo possa ter acesso até mesmo a informações remotas. Por exemplo: uma aplicação com o propósito de fornecer a localização de pessoas, pode saber que determinado usuário está localizado no ambiente *X*, mesmo que esteja sendo executada num ambiente *Y*;

- Para ter acesso a informação contextual, tudo o que a aplicação precisa fazer é implementar o lado cliente do *Web service*. Dessa forma, a aplicação fica desacoplada da implementação da *Lotus*, podendo ser implementada em qualquer linguagem de programação que a permita implementar o lado cliente da arquitetura.

Para disponibilizar a informação contextual por meio de um *Web service*, o primeiro passo foi definir uma *fachada* de acesso às informações. Para isso, foi necessário definir quais seriam os métodos principais dessa *fachada*, em outras palavras, quais seriam as informações que seriam disponibilizadas pelo módulo de *Disponibilização e Compartilhamento*. Para implementar essa *fachada*, a *Lotus* possui uma implementação padrão: a classe

`DefaultDeliveImpl` que comunica-se com a *fachada* `PervasiveFacade` e pode acessar o conteúdo das ontologias (veja Figura 4.10).

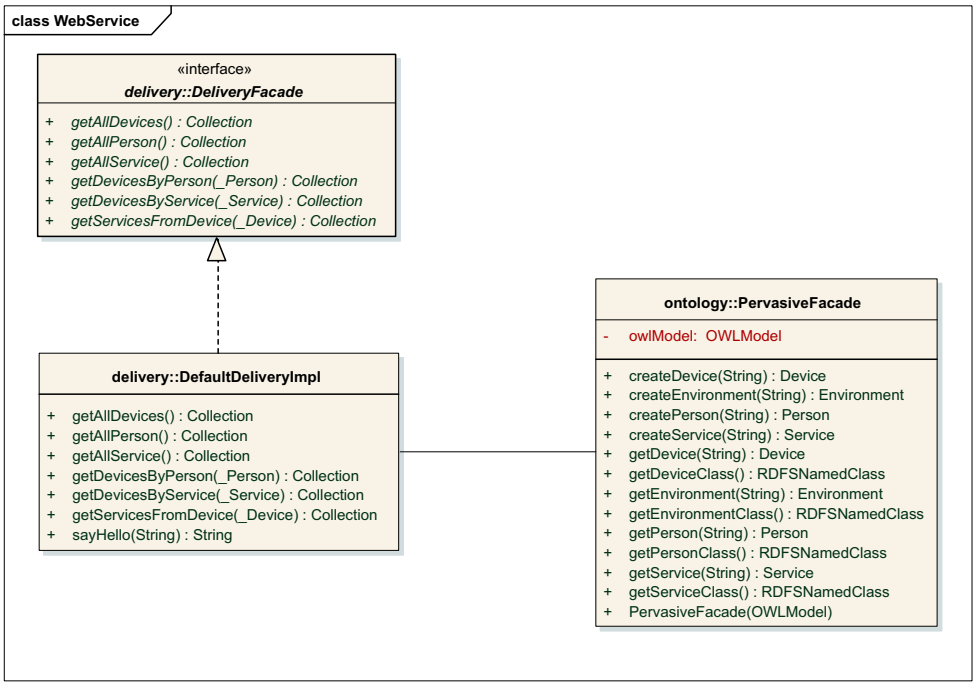

Figura 4.10: Módulo de disponibilização da informação contextual

O módulo de *Disponibilização e Compartilhamento* da *Lotus* atua como o *Service Provider* na arquitetura de *Web Service*. Desta forma, é possível que qualquer aplicação possa acessar a informação contextual. Para que a *Lotus* possa executar como servidor, é necessário que seja usado um servidor de aplicação, que neste caso foi usado o *Apache Tomcat*. Além disso, as funcionalidades que seriam disponibilizadas pela *Lotus* foram todas implementadas pela fachada `DeliveryFacade`.

Na listagem 4.2 tem-se um trecho do arquivo *LotusWebServiceFacade.wsdl* do módulo de *Disponibilização e Compartilhamento*. Pode ser observado neste trecho a descrição de alguns métodos, e.g., o método `getAllPerson` que retorna um array contendo todas as pessoas descrito na *tag* `getAllPersonResponse`.

Listing 4.2: Trecho do arquivo *LotusWebServiceFacade.wsdl*

```
        .
        .
<element name="getAllPerson">
    <complexType/>
</element>
<element name="getAllPersonResponse">
    <complexType>
    <sequence>
    <element name="getAllPersonReturn" type="impl:ArrayOf_xsd_anyType"/>
    </sequence>
    </complexType>
</element>
<complexType name="ArrayOf_xsd_anyType">
    <element maxOccurs="unbounded" minOccurs="0" name="item" type="xsd:anyType"/>
</complexType>
<element name="getAllDevices">
    <complexType/>
</element>
<element name="getAllDevicesResponse">
    <complexType>
    <sequence>
    <element name="getAllDevicesReturn" type="impl:ArrayOf_xsd_anyType"/>
    </sequence>
    </complexType>
</element>
        .
        .
```

Para que uma aplicação faça uso do *Web service* da *Lotus*, basta apenas que a aplicação implemente o lado cliente da arquitetura (*Service Requester*) de acordo com as especificações do arquivo “LotusWebServiceFacade.wsdl”. Uma vez que uma aplicação implemente o lado cliente do *Web service* é possível que ela acesse toda informação contextual disponibilizada pelo ambiente. Dessa forma, para que a aplicação esteja ciente do contexto, é necessário apenas requisitar ao *Web service* a informação que ela precisa e a *Lotus* provê automaticamente os mecanismos necessários para tornar a informação disponível à aplicação. Note que o desenvolvedor não precisa se preocupar com questões como aquisição da informação, forma de representar a informação obtida, entre outras.

Capítulo 5

Estudo de Caso

Com o intuito de validar a infra-estrutura do *Lotus*, duas aplicações foram desenvolvidas. Para a escolha dessas aplicações, foram observadas situações comuns ao cotidiano dos membros do Laboratório de Sistemas Embarcados e Computação Pervasiva da Universidade Federal de Campina Grande (*Embedded*)[1]. Dentre as possíveis aplicações que surgiram como sendo úteis para diversas situações, duas foram selecionadas:

- *Sistema de Localização*: através desse sistema é possível saber quais pessoas estão em determinado ambiente em certo momento. Com isso, pode-se, por exemplo, saber se a sala de reuniões está sendo utilizada, ou ainda, caso um aluno queria falar com o seu orientador, é possível saber se ele está em sua sala e com quem ele está.

- *Sistema de Comunidades Virtuais*: esse sistema tem por objetivo fazer com que pessoas que possuam interesses em comum possam formar comunidades virtuais [Rhe00]. Esse aplicação permite ainda que comunidades possam ser sugeridas aos usuários, de acordo com o seu perfil.

No decorrer deste capítulo são descritos detalhes sobre a implementação e funcionamento destes sistemas. A seguir, é dada uma visão geral do ambiente de desenvolvimento e do ambiente pervasivo no qual essas aplicações executam.

As aplicações deste estudo de caso foram desenvolvidas para executarem na plataforma *Symbian - NOKIA*, mais especificamente para dispositivos da série *80*. Foi também desenvolvida uma versão da aplicação de localização para dispositivos da série *60*. Para o desen-

[1] www.embedded.ufcg.edu.br

volvimento de aplicações da série *S60* é usado o *S60 3rd Edition Software Development Kit for Symbian OS, Supporting Feature Pack 1*[2]. Através do uso deste SDK é possível que sejam desenvolvidas aplicações para dispositivos da série *S60*, uma vez que ele inclui todas as funcionalidades necessárias para isso (e.g., máquina virtual, emulador). Para os dispositivos da série 80, é usado o *Series 80 Developer Platform 2.0 SDK for Symbian OS*.

5.1 Configuração do Ambiente Pervasivo

Para um melhor entendimento da atuação do *Lotus* em um ambiente pervasivo, assim como da participação das aplicações de nosso caso de uso, tem-se uma descrição da configuração do ambiente ilustrada na Figura 5.1.

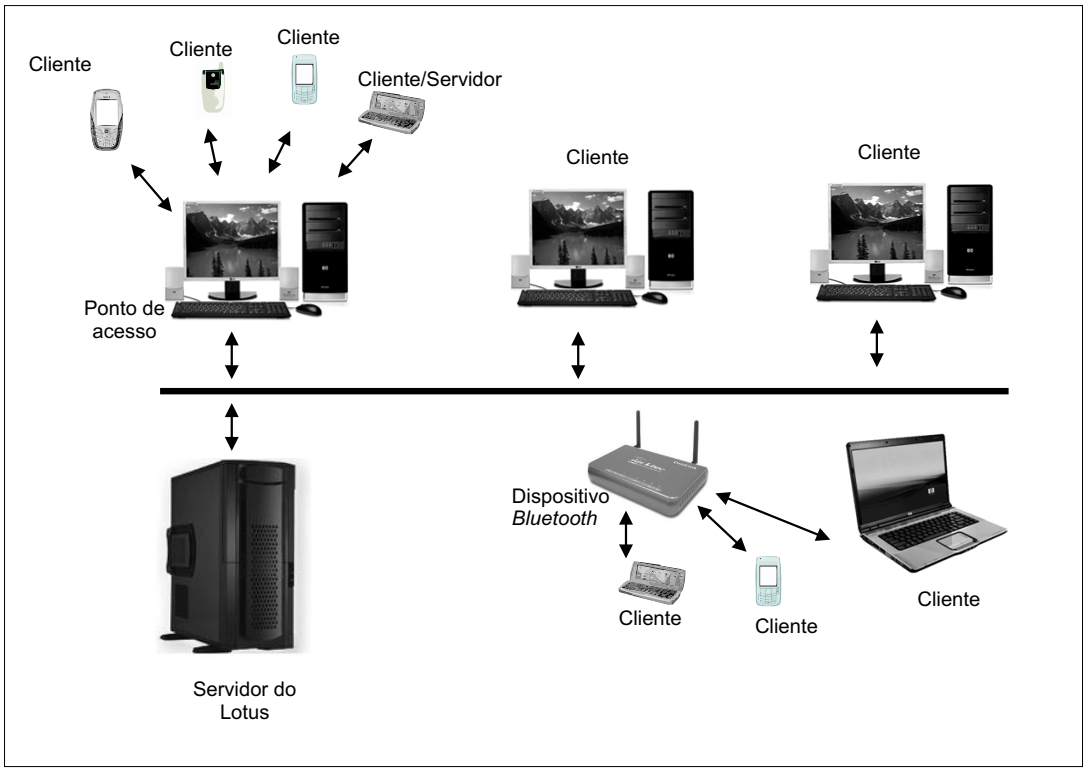

Figura 5.1: Configuração do ambiente

Neste ambiente podem ser encontrados três tipos de entidades: Clientes, Servidores e pontos de acesso. Cada uma destas entidades é descrita a seguir:

- *Cliente*: representa qualquer nó que faça uso dos serviços presentes no ambiente. Num

[2]http://www.forum.nokia.com/main/resources/tools_and_sdks/index.html

ambiente pervasivo, clientes podem estabelecer conexão com os servidores tanto por meio de conexões cabedas quanto por meio de interfaces de rede sem fio (e.g., *Wi-Fi, Bluetooth*);

- *Servidor*: pode ser representado por qualquer nó que diponibilize algum serviço. Assim como acontece com os clientes, um servidor pode estabelecer conexão tanto por meio de interfaces de rede sem fio como por cabeadas. Uma característica importante de ambientes pervasivos é que um dispositivo que atua como servidor de determinado serviço, pode atuar como um cliente em outro. No caso da configuração do ambiente em específico, existe um servidor no qual o *Lotus* é executado e qualquer outro dispositivo é um cliente em potencial, incluindo-o;

- *Ponto de acesso*: a principal funcionalidade de um ponto de acesso é permitir aos dispositivos comunicação sem fio. Isso pode ser feito por meio de dispositivos específicos (como o dispositivo *bluetooth* da Figura 5.1) ou por meio de computadores dotados de tal capacidade. Por exemplo, pode ser adicionada um receptor *bluetooth* na porta *usb* de um computador comum, para que funcione como um ponto de acesso à rede;

5.2 Sistema de Localização

Conforme mencionado no início deste capítulo esta aplicação tem por funcionalidade exibir quais pessoas encontram-se no ambiente em um determinado momento. Para isso, é feito uso da infra-estrutura oferecida pelo *Lotus*. Foi mencionado também que esta aplicação foi desenvolvida tanto para dispositivos da série 60, quanto para dispositivos da série 80. Para o desenvolvimento dessas aplicações, o ponto fundamental é a implementação do lado cliente do *Web service* disponibilizado pelo *Lotus*.

Uma vez que este cliente esteja implementado, a aplicação deverá se comportar da seguinte maneira (veja Figura 5.2):

- Uma vez iniciada a aplicação, uma requisição solicitando uma lista de ambientes que possam ser acessíveis ao usuário é enviada ao *Lotus Web Service*;

- A informação contendo a lista de todos os ambientes disponíveis ao usuário é retornada para a aplicação. A aplicação então exibe esta informação ao usuário;

- O usuário tem a opção de escolher um ambiente, dentre aqueles da lista disponível. Após a seleção do ambiente, a aplicação faz outra requisição ao *Lotus* que retorna a lista de pessoas presentes no ambiente selecionado à aplicação, que por sua vez, a exibe ao usuário. Na Figura 5.4 são exibidas algumas telas desta aplicação.

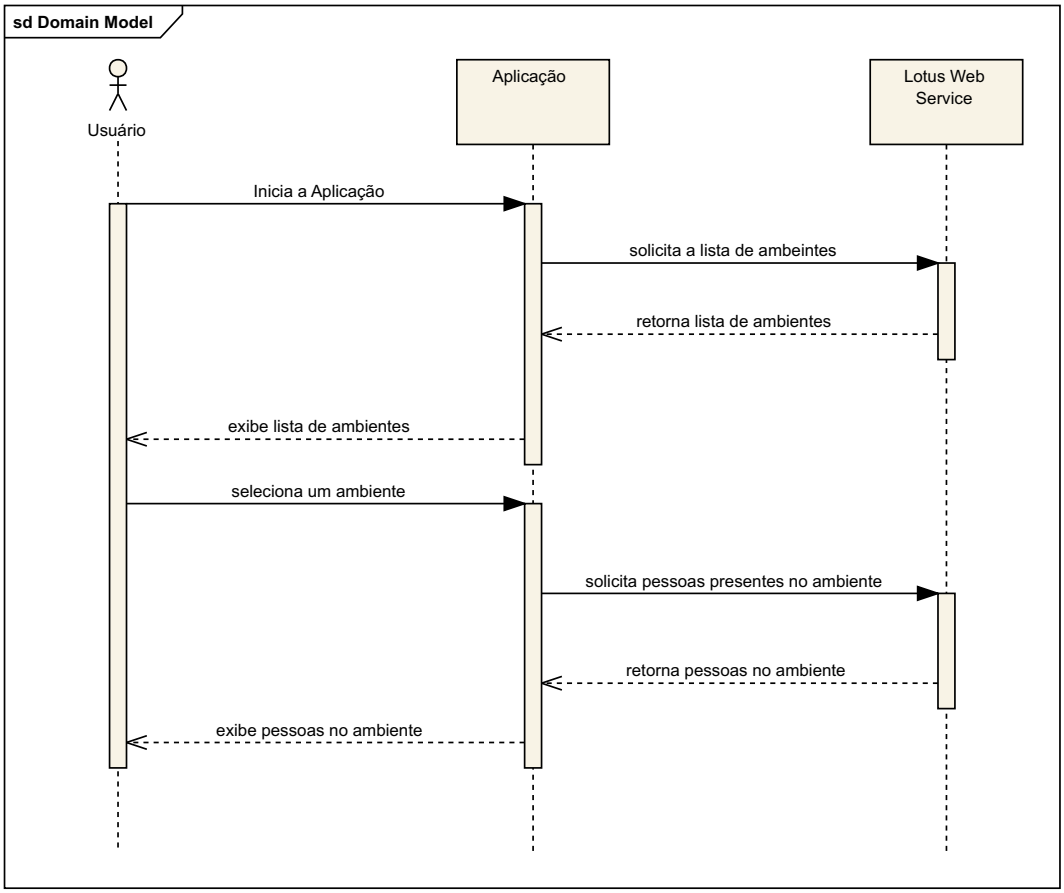

Figura 5.2: Diagrama de Interação entre a aplicação e a *Lotus*

A implementação do cliente é baseada no arquivo *LotusWebServiceFacade.wsdl* disponibilizado pelo *Lotus*. Este arquivo contém todas as informações necessárias para o desenvolvimento da aplicação como cliente. Existe ainda a possibilidade de usar o *plug-in Carbide.j* para gerar automaticamente as classes necessárias para gerar o lado cliente da arquitetura *Web service*. Dessa forma, o desenvolvedor pode focar na lógica de negócio da sua aplicação, uma vez que não necessita mais se preocupar com o processo de aquisição da informação contextual, ficando esta etapa a cargo da *Lotus*. Na Figura 5.3, a *interface* `Delivery` possui os mesmos métodos disponibilizados pela fachada do *Lotus*. Assim fica transparente para o desenvolvedor que as consultas ao *Lotus* são feitas por meio de chamadas *SOAP*, as quais

são implementadas pela classe `DeliverySoapBinding_Stub`.

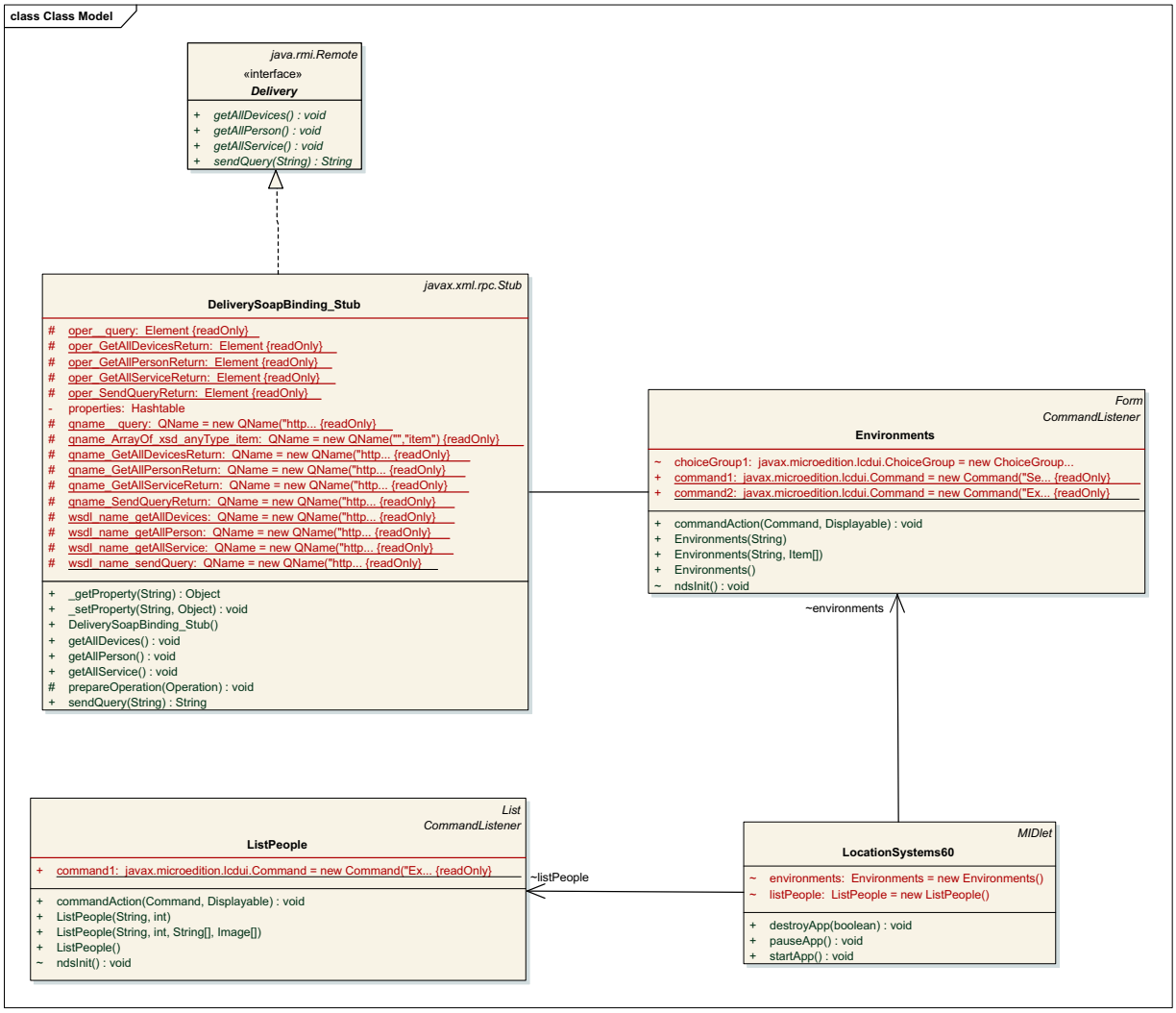

Figura 5.3: Diagrama de Classes do Sistema de Localização

5.3 Sistema de Comunidades Virtuais Móveis

Comunidades virtuais podem ser caracterizadas por grupos de pessoas que podem interagir por meio de computadores. A evolução dos computadores, onde estes ganharam capacidade de mobilidade e comunicação enquanto se deslocam, facilitou a capacidade de interação entre as pessoas, possibilitando a evolução das Comunidades Virtuais para as Comunidades Virtuais Móveis.

O uso de informação contextual é fundamental para o desenvolvimento de sistemas de

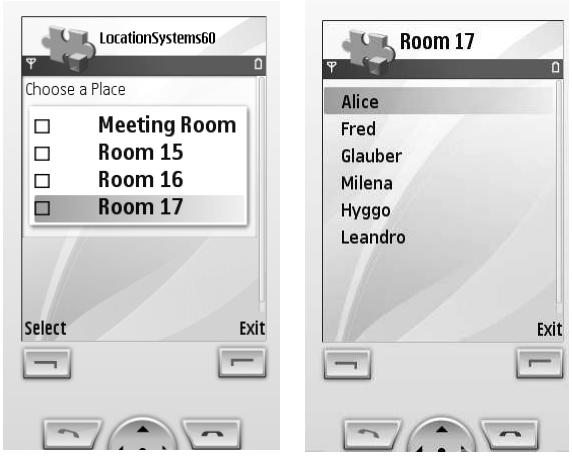

Figura 5.4: Algumas telas do Sistema de Localização

comunidades virtuais móveis. "Através dessas informações, as aplicações podem ser adaptadas com base nas preferências do usuário, nas configurações específicas do dispositivo utilizado e nas informações sobre a localização do usuário" [dMF06]. Dessa forma, no decorrer desta seção será apresentado como o *Lotus* pode ser usado para o desenvolvimento de tal aplicação. Antes disso, descreve-se a definição do sistema que será desenvolvido. O foco desse sistema está na criação de comunidades virtuais móveis e na formação de grupos a partir de análise do perfil dos usuários com duas funcionalidades principais:

1. Permitir que um usuário seja informado de alguma comunidade existente que se adequa ao seu perfil para que ele possa optar por fazer parte dela.

2. Buscar por usuários que tenham interesses em comum para que esses possam ser informados da existência dos demais usuários e possam formar novas comunidades.

Para que esta aplicação possa funcionar corretamente, é necessário que sejam modeladas algumas informações específicas para o domínio de comunidades virtuais. O primeiro ponto que precisa ser modelado é o perfil do usuário, que consiste de:

- *Interesses*. Aqui o usuário deverá listar coisas que ele julga interessante, e.g., futebol, música;

- *Amigos*. Cada usuário poderá ter vários amigos associados a eles;

- *Comunidades*. Cada usuário pode se associar a comunidades virtuais. Neste sistema, cada interesse represente uma comunidade.

O próximo passo é fazer com que essa informação esteja presente nas ontologias de domínio do *Lotus*. Para isso, foi preciso apenas estender a *ontologia pervasiva*, uma vez que tudo pode ser feito sem a necessidade de criar uma nova ontologia de domínio. Dessa forma, a ontologia foi acrescida de:

- Um novo atributo para a classe pessoa (*Person*), o atributo *Interests* que consiste de uma lista de interesses/comunidades do usuário.
- Um novo atributo *visited*, ligando a classe pessoa (*Person*) à classe ambiente (*Environment*), que marca quantas vezes determinada pessoa já esteve no ambiente. Este atributo será usado para fornecer a lista de lugares freqüentados pelo usuário.
- Um atributo *friend* que liga uma pessoa a várias outras pessoas.

A seguir são descritas as principais funcionalidades do sistema.

5.3.1 Edição dos Interesses

É possível que o usuário edite seus interesses através de uma interface gráfica fornecida pela aplicação. Lembre-se que cada interesse representa uma comunidade (interesse = comunidade), dessa forma, para cada interesse, estão listados os membros da comunidade associados ao mesmo. Dessa forma, sempre que um novo interesse é criado, é verificado se existe uma comunidade para esse interesse e, caso exista, o usuário é adicionado à mesma. Na Figura 5.5 pode ser vista a lista de interesses de determinado usuário e, ao selecionar uma delas (*Inglês*), são listados os membros da comunidade.

Figura 5.5: Tela exibindo a lista de membros da comunidade *Inglês*

5.3.2 Edição dos Amigos

Da mesma maneira que é possível editar os interesses de um usuário, é possível também editar a lista de amigos do mesmo. Além disso, é possível que novos amigos sejam adicionados à lista de contatos de acordo com o nível de similaridade do mesmo. A análise de similaridade é feita de acordo com três critérios: Amigos, Comunidades e Lugares em comum. O nível de similaridade pode ser definido pelo usuário através de uma barra de configuração (veja Figura 5.7). Quando um novo amigo é encontrado, uma mensagem de confirmação é exibida mostrando os dados do possível amigo e qual a similaridade entre os perfis.

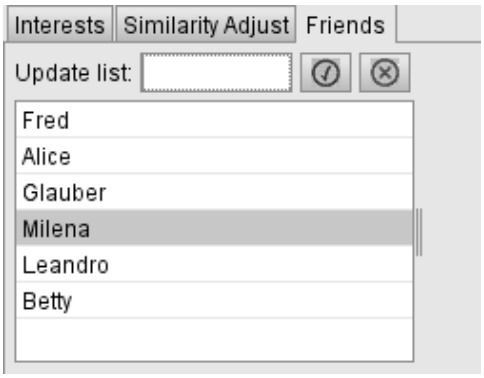

Figura 5.6: Lista de amigos do usuário

5.3.3 Ajuste do Nível de Similaridade

Esta aplicação permite ainda que o usuário defina o nível de similaridade. O nível de similaridade consiste de um cálculo que mede a semelhança entre o perfil de dois usuários. Nesta aplicação é possível medir a semelhança entre os perfis baseados em três critérios (na Figura 5.7 ilustra-se a tela para ajuste do nível de similaridade):

- *Amigos*: permite que novos amigos sejam adicionados a lista de amigos do usuário. Por exemplo, o usuário *Fred* possui muitos amigos em comum com o usuário *Paulo*, logo, existe uma grande possibilidade de *Fred* e *Paulo* serem amigos (veja Figura 5.8);
- *Comunidades*: possibilita que usuários que possuam comunidades em comum possam ser indicados como amigos. Por exemplo, *Alice* e *Betty* possuem várias comunidades em comum (Inglês, Culinária, etc.). Sendo assim, é interessante que elas possam ser indicadas como amigas;

- *Lugares* permite que usuários que freqüentam os mesmos lugares sejam apontados como amigos.

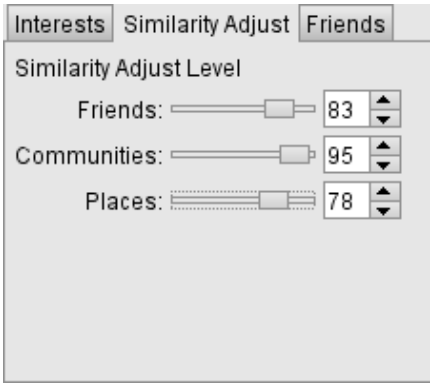

Figura 5.7: Tela para ajuste do nível de similaridade.

A fórmula utilizada para calcular a similaridade entre os usuários é apresentada a seguir:

Sejam A e B conjuntos de amigos, comunidades ou lugares de dois usuários distintos, a similaridade S entre os usuários é dada por [dMF06]:

$$S(A,B) = \frac{A \cap B}{A \cup B}$$

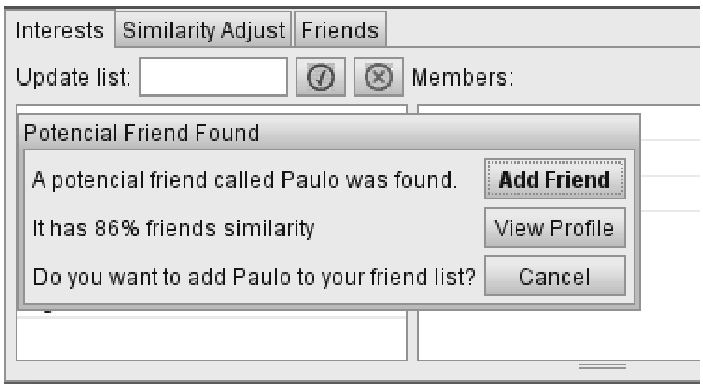

Figura 5.8: Tela para notificação de similaridade de usuários

5.4 Considerações Sobre o Estudo de Caso

Este estudo de caso foi desenvolvido com o intuito de validar a infra-estrutura do *Lotus*. Para isso, duas aplicações foram desenvolvidas. Estas aplicações foram selecionadas de acordo

com as necessidades dos usuários do Laboratório de Sistemas Embarcados e Computação Pervasiva, da Universidade Federal de Campina Grande. As aplicações desenvolvidas foram respectivamente: um *Sistema de Localização* e um *Sistema de Comunidades Virtuais*. Através do primeiro sistema, foi possível validar a infra-estrutura no que diz respeito à descoberta de nós sobre redes heterogêneas, onde é possível determinar quais pessoas estão disponíveis no ambiente. Neste primeiro sistema foi possível usar a *ontologia pervasiva* sem precisar fazer nenhuma extensão sobre a mesma, o que indica que esta ontologia está bem definida.

Para o desenvolvimento da segunda aplicação, por ser um pouco mais complexa, foi necessário estender a *ontologia pervasiva* para que ela se adequasse às regras de negócio da mesma. Note que para fazer essa extensão o desenvolvedor lidou apenas com questões referentes ao seu domínio de aplicação. Através dessa aplicação foi possível mostrar o quanto o *Lotus* é extensível para ser usado por qualquer aplicação independente do domínio a que pertença. Em ambas as aplicações fica claro que usar o *Lotus*, para desenvolver aplicações cientes de contexto, permite que o desenvolvedor foque na sua lógica de negócio. Além disso, as aplicações desenvolvidas validam o trabalho como um todo, uma vez que a informação contextual é realmente compartilhada entre as aplicações, permitindo que a mesma informação esteja disponível para várias aplicações, independente da lógica de negócio.

Capítulo 6

Trabalhos Relacionados

Devido a importância de contexto para a computação pervasiva, muitos trabalhos têm sido propostos com o intuito de facilitar o desenvolvimento de aplicações cientes de contexto. Dentre os trabalhos que focam em facilitar o desenvolvimento de aplicações cientes de contexto, destacam-se as soluções baseadas em ontologias e os *middlewares* para computação pervasiva. Mais especificamente, nesta seção são descritos os trabalhos que abordam de alguma maneira algum dos seguintes problemas:

- *Descoberta de Nós*: indica se a solução provê os mecanismos necessário para a descoberta de nós;
- *Descoberta de Serviços*: mostra se a solução provê os mecanismos necessário para a descoberta de serviços;
- *Independência de Protocolo*: indica se a solução consegue efetuar a descoberta de nós e/ou serviços independentemente do protocolo de rede;
- *Modela Ambiente Pervasivo*: indica se a solução provê algum mecanismo para modelagem de ambientes pervasivos;
- *Modelagem Domínio*: indica se a solução permite que a informação referente ao domínio da aplicação seja modelada;
- *Independente de Domínio*: indica se a solução permite o desenvolvimento de aplicações independente do domínio a que pertence.

- *Suporte a Raciocínio*: indica se a solução provê algum mecanismo que permita que seja realizado algum raciocínio sobre a informação adquirida.

- *Suporte à Disponibilização*: indica se a solução permite que a informação seja disponibilizada às aplicações;

- *Compartilhamento*: indica se a solução permite que a informação contextual seja compartilhada pelas aplicações.

6.1 CAMidO

CAMidO [NBB05] é um *middleware* baseado em meta-modelo ontológico para descrição de contexto. Este meta-modelo é escrito em OWL e permite que os desenvolvedores reúsem vocabulários definidos. Dessa forma, é possível que seja feita uma descrição do contexto e dos sensores pelos quais os dados são coletados.

A arquitetura do *middleware* é baseada em componentes e para adaptar uma aplicação para as alterações do contexto, faz uso dos seguintes componentes:

- *Collection Manager* é responsável por receber as informações obtidas dos sensores. Para cada sensor existe um agente, que permite interação com um tipo de sensor.

- *Context Analyzer* é responsável por filtrar a informação contextual e determinar mudanças relevantes. A filtragem de contexto consiste em encontrar mudanças de contexto através da comparação do valor da nova informação coletada com a antiga.

- *Context Interpreter* é responsável por interpretar a informação enviada pelo *Collection Manager*.

Uma vez que essa informação esteja disponível no *Context Analyzer* é possível que seja criado um serviço para disponibilizar essa informação às aplicações que necessitem dela. Dessa forma, quando uma aplicação precisa fazer uso de determinada informação contextual, o desenvolvedor implementa um serviço que é responsável por receber informações sobre a informação contextual em que o mesmo possui interesse.

6.2 JCAF

Baseado em Java, JCAF (*Java Context Awareness Framework*) [Bar05], é um *middleware* para computação pervasiva baseado em eventos e serviços. Basicamente, ele é constituído de duas partes: *Context-Awareness Programming Framework* e *Context-Awareness Runtime Infrastructure*.

O *Context-Awareness Programming Framework* provê um conjunto de classes e interfaces Java para o desenvolvimento de aplicações pervasivas e serviços de contexto (*Context Services*) que constituem a *Context-Awareness Runtime Infrastructure*.

Mais precisamente, cada serviço de contexto deve lidar com as informações de um ambiente em particular, por exemplo, uma sala ou um quarto, estando os mesmos dispostos em uma topologia ponto a ponto, para que possam assim se comunicar e trocar informações de contexto entre si. As informações providas por cada serviço de contexto podem ser recuperadas através de requisições e respostas (i.e., *request-response*) ou registrando-se como ouvintes de mudanças no contexto.

Um aspecto interessante a ser observado sobre a modelagem de contexto no JCAF é que novas entidades e items de contexto podem ser adicionados a um serviço de contexto sempre que necessário, mesmo em tempo execução. Porém, toda a modelagem da informação contextual fica a cargo do desenvolvedor da aplicação.

6.3 CORTEX

O projeto CORTEX propõe o uso de um modelo baseado em objetos sensíveis (*sentient object model*). Este modelo é uma abstração para o modelo de componentes que usa objetos sensíveis para permitir o desenvolvimento de aplicações distribuídas. Ele permite a construção do sistema baseado nestes objetos sensíveis.Um objeto sensível é componente de software capaz de sentir o ambiente através do consumo de eventos via canais de eventos ou outros objetos sensíveis.

O projeto CORTEX desenvolveu um *middleware* para prover o suporte necessário ao desenvolvimento de aplicações baseadas em objetos sensíveis. O *middleware* consiste de vários *components frameworks* (CF) onde cada CF corresponde a uma área de pesquisa.

Existem dois CFs que são definidos pelo projeto, o *Context CF* e o *Publish-Subscribe CF*. O primeiro registra-se para eventos dos sensores e possivelmente de outros objetos sensíveis. O último implementa um modelo de comunicação entre os objetos [SWS+04].

6.4 SOCAM

O SOCAM [GPZ05] (*Service-Oriented Context-Aware Middleware*), como o próprio nome já diz, é uma arquitetura para o desenvolvimento de serviços cientes de contexto. Para isso o *middleware* propõe uma arquitetura baseada em ontologias e serviços. O mesmo possui uma modelagem baseada em ontologias OWL. Segundo o autor, uma das vantagens da utilização de OWL é o fato desta facilitar o compartilhamento de informações de contexto, por ser baseada em XML, e a inferência sobre as mesmas (i.e., raciocínio de contexto), dado que já existem APIs com esse propósito.

A arquitetura do *middleware* é composta de: *Context providers*, componentes responsáveis por adquirir informações de contexto, dividindo-se em *Internal Context Providers* e *External Context Providers*. Os *External Context Providers* são responsáveis por obter informações a partir de fontes externas, como servidores remotos. Já os *Internal Context Providers* são responsáveis por recuperam as informações através de sensores. Essas informações são enviadas para o *Context Interpreter* que provê serviços de raciocínio lógico para processar informação de contexto e finalmente são armazenadas em um *Context Database*, podendo ser recuperadas e utilizadas posteriormente.

Todos esses componentes servem de base para a execução dos chamados *Context-Aware services*, ou seja, aplicações, agentes ou serviços que fazem uso de informações de contexto e reagem às mudanças no mesmo.

6.5 CoBrA

CoBrA [CFJ03] é uma arquitetura baseada em agentes para apoiar computação com conhecimento de contexto em espaços inteligentes. A principal característica da arquitetura é a presença de um *context broker* inteligente, que é responsável pela manutenção e gerenciamento de um modelo compartilhado de contextos para o benefício da comunidade de agentes.

O *Broker* centralizado foi projetado para possibilitar duas importantes características que são chave para a realização da computação pervasiva, que são o suporte ao recurso limitado dos dispositivos móveis e preocupações com privacidade e segurança do usuário.

Através do uso de uma representação explícita de ontologia, CoBrA permite que o conhecimento de contexto possa ser capaz de derivar informações adicionais. Este conhecimento pode também ser facilmente compartilhado por agentes distribuídos usando linguagens e protocolos padrões de comunicação (e.g., FIPA-ACL, KQML e SOAP/XML-RPC). A ontologia CoBrA define um conjunto de vocabulários para descrever pessoas, agentes, lugares e uma apresentação de eventos para dar suporte a um sistema de uma sala de reunião sendo agrupada em quatro temas distintos, mas relacionados: 1) Conceitos que definem lugares físicos e os relacionamentos especiais associados; 2) Conceitos que definem agentes (humanos ou de software) e seus atributos; 3) Conceitos que descrevem o contexto da localização de um agente em um campus universitário; 4) Conceitos que descrevem as atividade de contexto de um agente, incluindo as funções dos palestrantes e espectadores e seus desejos e intenções em um evento de apresentação.

6.6 SOUPA

A ontologia SOUPA [CPFJ04] (*Standard Ontology for Ubiquitous and Pervasive Applications*) foi projetada para modelar e apoiar aplicações pervasivas, de modo a facilitar o compartilhamento de conhecimento. SOUPA contém representações sobre agentes, tempo, espaço, ações e políticas de segurança e privacidade. Esta ontologia foi criada utilizando a linguagem OWL que é usada para representação de conhecimento. SOUPA é composta de um conjunto de ontologias, porém estas não são completamente importadas, apenas parte do vocabulário destas é que é utilizado. Esta combinação de ontologias é um dos pontos fortes existentes, pois desta maneira SOUPA pode fornecer uma ontologia consensual para os desenvolvedores de sistemas pervasivos.

As ontologias existentes no SOUPA estão divididas em dois diferentes conjuntos: 1) *Core*, que define conceitos comuns a diversas aplicações pervasivas, estes conceitos são representados em nove ontologias diferentes; 2) *Extension*, que estendem as ontologias presentes no *Core*, definindo conceitos mais específicos, os quais podem não ser comuns a diversas

aplicações pervasivas. O principal objetivo do projeto SOUPA é definir uma padronização de uma ontologia compartilhada para aplicações voltadas à computação pervasiva.

6.7 Nexus

Nexus aparece como uma solução que usa uma abordagem orientada a serviços para facilitar o desenvolvimento de aplicações em ambientes pervasivos. Mais precisamente, o *middleware* Nexus [KH04] utiliza uma abordagem de plug-ins para contornar a heterogeneidade de protocolos de serviços existentes. Sua arquitetura é composta por três camadas. A primeira delas, *Discovery Layer*, é responsável pela publicação e descoberta de serviços, independente de protocolo. Dessa forma, cada protocolo a ser utilizado nessa camada é encapsulado em um plug-in, podendo ser adicionado ou mesmo removido do *middleware* em tempo de execução.

A próxima camada, *Agent Layer*, tem como propósito adicionar funcionalidades de busca de serviços mais inteligentes ao *middleware*. Mais precisamente, a idéia é adicionar agentes a essa camada, para que estes realizem tarefas específicas em favor do usuário. A Camada de Agente (*Agent Layer*), no entanto, é opcional. Por fim, na camada de mais alto nível do *middleware* , Camada de Serviços (*Service Layer*), residem os serviços a serem publicados na rede. Dessa forma, serviços de diferentes tecnologias podem executar nessa camada, podendo assim ser publicados através de diferentes protocolos, possivelmente ao mesmo tempo.

6.8 Considerações Sobre os Trabalhos Relacionados

Devido à grande quantidade de trabalhos que possuem o propósito de facilitar o desenvolvimento de aplicações cientes de contexto em ambientes pervasivos, não foi possível citar todos aqui. Porém, foram apresentadas algumas das principais soluções neste sentido. Na Tabela 6.8, ilustra-se um comparativo entre as principais soluções. Como pode ser observado nesta tabela, para todas as características que o *Lotus* possui, já existe pelo menos uma solução que aborde alguma dessas características. Porém, nenhuma das soluções lida com todas elas ao mesmo tempo. É possível observar nesta tabela que uma boa alternativa seria

	Descoberta de Nós	*Descoberta de Serviços*	*Independência de Protocolo*	*Modela Ambiente Pervasivo*	*Modelagem Domínio*	*Independente de Domínio*	*Suporte a Raciocínio*	*Suporte a Disponibilização*	*Compartilhamento*
CAMidO	✓				✓	✓	✓	✓	
JCAF					✓			✓	✓
CORTEX	✓	✓			✓		✓	✓	
SOCAM				✓		✓	✓		
CoBrA					✓		✓	✓	✓
SOUPA				✓	✓		✓	✓	✓
Nexus	✓	✓	✓						
Lotus	✓	✓	✓	✓	✓	✓	✓	✓	✓

Tabela 6.1: Comparativo entre as soluções para provisão de contexto em ambientes pervasivos

adotar uma abordagem que fundisse algumas dessas alternativas. A principio essa foi a solução que parecia mais viável, porém, devido a dificuldade em obter os fontes destes trabalhos, executá-los e devido até mesmo a bugs encontrados, optou-se pelo desenvolvimento da *Lotus* fazendo uso apenas do *middleware Wings*.

Capítulo 7

Considerações Finais

Neste trabalho foi apresentada uma infra-estrutura para o desenvolvimento de aplicações cientes de contexto em ambientes pervasivos. Neste capítulo serão descritas as contribuições e os trabalhos decorrentes do desenvolvimento desta infra-estrutura.

7.1 Contribuições

A infra-estrutura desenvolvida, denominada *Lotus*, foi projetada com o objetivo de abordar o problema da disponibilização da informação contextual às aplicações em uma abordagem integrada. Para isso foi adotada uma solução que reúne o melhor das soluções baseadas em ontologias com as soluções baseadas em *middlewares*. Das soluções baseadas em *middlewares* foram aproveitados mecanismos que permitem que seja feita a descoberta de nós e serviços mesmo em redes heterogêneas. Com relação às ontologias, foi desenvolvido um módulo que permite a representação da informação contextual. Além disso, foram desenvolvidos mecanismos para permitir que a informação presente nas ontologias seja atualizada dinamicamente e um mecanismo para que a informação seja disponibilizada às aplicações.

Foi demonstrado ainda, por meio do estudo de caso, que o desenvolvimento de aplicações cientes de contexto usando o *Lotus* é completamente factível. Neste estudo de caso, foi possível constatar a viabilidade a infra-estrutura, uma vez que a informação contextual é disponibilizada para o desenvolvedor, podendo este focar na lógica de negócio da aplicação. Além disso, a informação contextual é realmente compartilhada entre as aplicações, permitindo que a mesma informação esteja disponível para várias aplicações, independentemente

da lógica de negócio.

Um outro ponto forte da infra-estrutura é que a mesma contempla todos os aspectos necessários ao provimento de informação contextual, em uma abordagem única. Isso foi demonstrado através dos trabalhos relacionados, onde foi feito um comparativo entre as principais soluções, existentes atualmente, que abordam o problema da provisão contextual de alguma maneira. Neste comparativo foi possível mostrar que a *Lotus* aparece como uma solução mais abrangente, contemplando todos os aspectos. Embora para cada um desses aspectos já exista, pelo menos, uma solução que o aborde, nenhuma solução lida com todos.

7.2 Trabalhos Futuros

Até o momento foi visto que a *Lotus* aparece como uma solução que facilita o desenvolvimento de aplicações cientes de contexto para ambientes pervasivos. Conseqüentemente, ela vem a viabilizar também o surgimento de ambientes pervasivos. Porém, para que isso possa se tornar uma realidade, é preciso que a *Lotus* contemple mais algumas características, cedendo espaço a alguns trabalhos futuros.

O primeiro aspecto que precisa ser contemplado é a realização de testes de desempenho. Estes testes devem ser capazes de averiguar o tempo de resposta da *Lotus* se comparado a outras abordagens e qual o impacto da *Lotus* no tráfego de rede. O segundo aspecto que precisa ser incorporado à *Lotus* é o desenvolvimento de um *Modelo de Confiança*, utilizado para possibilitar o estabelecimento de relações de confiança entre entidades. Por fim, é interessante também que seja desenvolvido um módulo que permita que o sistema venha a ser autônomo, contemplando as características de auto-configuração, permitindo, por exemplo, que ao detectar uma nova

Bibliografia

[Bar05] Jakob E. Bardram. The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. In *Proceedings of the 3rd International Conference on Pervasive Computing (Pervasive 2005)*, pages 98–115, Munique, Alemanha, 2005. Springer Verlag. Lecture Notes in Computer Science.

[Bir05] D. Birsan. On Plug-ins and Extensible Architectures. *ACM Queue*, 3(2):40–46, 2005.

[BLA+07] Frederico Bublitz, Emerson Loureiro, Hyggo Almeida, Angelo Perkusich, and Evandro de Barros Costa. Context-Awareness in Pervasive Environments. In Maria Manuela Cunha, editor, *Encyclopedia of Networked and Virtual Organizations*, volume 1. Idea Group Publishing, Hershey, Pensilvânia, EUA, 2007. (No Prelo).

[Bub05] Frederico Moreira Bublitz. FrOnt - Frame-based Ontology System: Uma Ferramenta para Criação e Edição de Ontologias, 2005. Trabalho de Conclusão de Curso, Universidade Federal de Alagoas.

[CFJ03] Harry Chen, Tim Finin, and Anupam Joshi. An Ontology for Context-Aware Pervasive Computing Environments. *Knowledge Engineering Review - Special Issue on Ontologies for Distributed Systems*, 18(3):197–207, 2003.

[CK00] Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dartmouth College, Computer Science, Hanover, EUA, 2000.

[CPFJ04] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, Boston, USA, August 2004.

[DA99] Anind K. Dey and Gregory D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, pages 304–307, Karlsruhe, Alemanha, 1999. Springer-Verlag.

[Dey01] Anind K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[DKD+05] Li Ding, Pranam Kolari, Zhongli Ding, Sasikanth Avancha, Tim Finin, and Anupam Joshi. Using Ontologies in the Semantic Web: A Survey. Technical report, UMBC, Julho 2005.

[dMF06] Glauber Vinícius Ventura de Melo Ferreira. Infra-estrutura de Software Baseada em Componentes para a Construção de Aplicações para Comunidades Virtuais Móveis. Master's thesis, Universidade Federal de Campina Grande, 2006.

[EN94] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, 2nd Edition*. Benjamin/Cummings, 1994.

[FFR96] A. Farquhar, R. Fikes, and J. Rice. The Ontolingua Server: a Tool for Collaborative Ontology Construction. Technical report, Knowledge Systems Laboratory, Stanford University, Knowledge Systems Laboratory, Universidade de Stanford, EUA, 1996.

[Fre03] Fred Freitas. Ontologias e a Web Semântica. In Renata Vieira; Fernando Osório, editor, *Anais do XXIII Congresso da Sociedade Brasileira de Computação*, volume 8, pages 1–52, 2003. Jornada de Mini-Cursos em Inteligência Artificial.

[GPZ04] T. Gu, H. K. Pung, and D. Q. Zhang. A Bayesian Approach for Dealing with Uncertain Contexts. In G. Kotsis, editor, *Proceedings of the 2nd International Conference on Pervasive Computing*, volume 176, Viena, Áustria, 2004. Austrian Computer Society.

[GPZ05] Tao Gu, Hung K. Pung, and Da Q. Zhang. A Service-Oriented Middleware for Building Context-Aware Services. *Journal of Network and Computer Applications*, 28(1):1–18, January 2005.

[Gru93] Thomas R. Gruber. A Translation Approach to Portable Ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[Gwi00] J. Gwizdka. What's in the context. In *Proceedings of the Computer Human Interaction 2000 (CHI2000), Workshop on "The What, Who, Where, When, Why and How of Context-Awareness"*, Hague, Holanda, 2000.

[HBS02] A. Held, S. Buchholz, and A. Schill. Modeling of Context Information for Pervasive Computing Applications. In *Procceding of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI2002)*, Orlando, EUA, 2002.

[HMNS03] Uwe Hansmann, Lothar Merk, Martin S. Nicklous, and Thomas Stober. *Pervasive Computing*. Springer, 2 edition, 2003.

[HMvdSW04] V. Haarslev, R. Möller, R. van der Straeten, and M. Wessel. Extended Query Facilities for Racer and an Application to Software-Engineering Problems. In *Proceedings of the 2004 International Workshop on Description Logics (DL-2004), Canadá*, pages 148–157, 2004.

[KH04] N. Kaveh and R. Ghanea Hercock. NEXUS Resilient Intelligent Middleware. *BT Technology Journal*, 22(3):209–215, 2004.

[LBB+06] Emerson Loureiro, Frederico Bublitz, Nadia Barbosa, Hyggo Almeida, Angelo Perkusich, and Glauber Ferreira. A Flexible Middleware for Service Provision Over Heterogeneous Pervasive Networks. In *4th International*

Workshop on Mobile and Distributed Computing, pages 609–614, Cascatas do Niágara, Nova York, EUA, Junho 2006. IEEE Computer Society.

[LOA+05] Emerson Loureiro, Loreno Oliveira, Hyggo Almeida, Glauber Vinicius, and Angelo Perkusich. Improving Flexibility on Host Discovery for Pervasive Computing Middlewares. In *MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–8, Grenoble, França, 2005. ACM Press.

[Miz04] Riichiro Mizoguchi. Tutorial on Ontological Engineering: Advanced Course of Ontological Engineering. *New Gen. Comput.*, 22(2):198–220, 2004.

[MKM+05] Tinghuai Ma, Yong-Deak Kim, Qiang Ma, Meili Tang, and Weican Zhou. Context-aware implementation based on CBR for smart home. In *Proceedings of IEEE International Conference on Wireless and Mobile Computing, Network and*, volume 4, pages 112–115, Montreal, Canadá: IEEE Computer Society, 2005.

[MPRB04] Ghita Kouadri Mostéfaoui, Jacques Pasquier-Rocha, and Patrick Brézillon. Context-Aware Computing: A Guide for the Pervasive Computing Community. In *Proceedings of The IEEE/ACS International Conference on Pervasive Services (ICPS'04)*, pages 39–48, Beirut, Líbano, 2004.

[NBB05] Chantal Taconet Nabiha Belhanafi and Guy Bernard. CAMidO, A Context-Aware Middleware Based on Ontology Meta-Model. In *Workshop on Context Awareness for Proactive Systems (CAPS2005)*, pages 93–103, 2005.

[NFF+91] Robert Neches, Richard Fikes, Tim Finin, Tom Gruber, Ramesh Patil, Ted Senator, and William R. Swartout. Enabling Technology for Knowledge Sharing. *AI Mag.*, 12(3):36–56, 1991.

[NFM00] Natalya Fridman Noy, Ray W. Fergerson, and Mark A. Musen. The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility. In *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, pages 17–32. Springer-Verlag, 2000.

[PNS+00] D. Petrell, E. Not, C. Strapparava, O. Stock, and M. Zancanaro. Modeling Context is Like Taking Pictures. In *Proceedings of the Computer Human Interaction 2000 (CHI2000), Workshop on "The What, Who, Where, When, Why and How of Context-Awareness"*, Hague, Holanda, 2000.

[RAMC04] A. Ranganathan, J. Al-Muhtadi, and RH. Campbell. Reasoning about Uncertain Contexts in Pervasive Computing Environments. *IEEE Pervasive Computing*, 3(2):62–70, 2004.

[Rhe00] Howard Rheingold. *The Virtual Community: Homesteading on the Electronic Frontier, revised edition*. The MIT Press, Boston, EUA, 2000.

[Sat01] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4):10–17, 2001.

[SBF98] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge Engineering: Principles and Methods. *Data Knowledge Engineering*, 25(1-2):161–197, 1998.

[Sow92] J. Sowa. Semantic Networks. In *Encyclopedia of Artificial Intelligence*. S. Shapiro, 1992.

[ST94] Bill Schilit and M. Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5):22–32, 1994.

[SWS+04] Carl-Fredrik Sorensen, Maomao Wu, Thirunavukkarasu Sivaharan, Gordon S. Blair, Paul Okanda, Adrian Friday, and Hector Duran-Limon. A Context-aware Middleware for Applications in Mobile Ad Hoc Environments. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 107–110, Nova York, EUA, 2004. ACM Press.

[VB96] A. Valente and J. Breuker. Towards Principled Core Ontologies. Em B.R. Gaines and M. Mussen, editores, Proceedings of the KAW-96, Banff, Ca, 1996.

[Wei91] Marc Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):66–75, Setembro 1991.