

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

MARCIO PEREIRA DE SÁ

**ConBus: Uma Plataforma de
Middleware de Integração de Sensores
para o Desenvolvimento de Aplicações
Móveis Sensíveis ao Contexto**

Goiânia
2010

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

**AUTORIZAÇÃO PARA PUBLICAÇÃO DE DISSERTAÇÃO
EM FORMATO ELETRÔNICO**

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos “reproduzir” e “publicar” conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: ConBus: Uma Plataforma de Middleware de Integração de Sensores para o Desenvolvimento de Aplicações Móveis Sensíveis ao Contexto

Autor(a): Marcio Pereira de Sá

Goiânia, 26 de Abril de 2010.

Marcio Pereira de Sá – Autor

Vagner José do Sacramento Rodrigues – Orientador

Ricardo Couto Antunes da Rocha – Co-Orientador

MARCIO PEREIRA DE SÁ

ConBus: Uma Plataforma de Middleware de Integração de Sensores para o Desenvolvimento de Aplicações Móveis Sensíveis ao Contexto

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

Área de concentração: Redes de Computadores e Sistemas Distribuídos.

Orientador: Prof. Vagner José do Sacramento Rodrigues

Co-Orientador: Prof. Ricardo Couto Antunes da Rocha

Goiânia
2010

MARCIO PEREIRA DE SÁ

ConBus: Uma Plataforma de Middleware de Integração de Sensores para o Desenvolvimento de Aplicações Móveis Sensíveis ao Contexto

Dissertação defendida no Programa de Pós-Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Computação, aprovada em 26 de Abril de 2010, pela Banca Examinadora constituída pelos professores:

Prof. Vagner José do Sacramento Rodrigues
Instituto de Informática – UFG
Presidente da Banca

Prof. Ricardo Couto Antunes da Rocha
Instituto de Informática – UFG

Prof. Markus Endler
Departamento de Informática – PUC-Rio

Prof. Fábio Moreira Costa
Instituto de Informática – UFG

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador(a).

Marcio Pereira de Sá

Graduou-se em Ciência da Computação na UFG - Universidade Federal de Goiás (2007). Durante o Mestrado, atuou como Gerente de Projetos no Grupo de Sistemas Baseados em Localização (LBS) do INF-UFG e coordenou, juntamente com o Professor Vagner Sacramento, o Grupo de Computação Móvel do INF-UFG. Tem experiência na área de Ciência da Computação, com ênfase em Redes de Computadores e Sistemas Distribuídos, atuando principalmente nas subáreas Computação Móvel e Computação Sensível ao Contexto.

Dedico este trabalho à minha esposa, Elizângela, e a todos os meus familiares, em especial a meus pais, Jordelino e Marina.

Agradecimentos

Durante o período do Mestrado, várias pessoas me auxiliaram, direta ou indiretamente. Por isso, gostaria de agradecer primeiramente a Deus por me permitir finalizar mais esta etapa de minha vida e à minha esposa, Elizângela, que sempre esteve comigo nos momentos felizes e também em minhas dificuldades. Agradeço também a meus pais, Jordelino e Marina, pelo apoio incondicional e também a meus irmãos, Marcelo e Fernando pelo incentivo. Agradeço ainda a todos os meus familiares e amigos que souberam compreender os momentos em que necessitava deixar outros afazeres para me dedicar à minha pesquisa.

Gostaria de agradecer ao meu orientador, o Professor Vagner Sacramento, pela compreensão e auxílio durante todo o tempo de pesquisa.

Agradeço também ao meu coorientador, o Professor Ricardo Rocha, que me ajudou a enxergar muitas deficiências em meu trabalho e corrigi-las a tempo.

Agradeço também à CAPES pelo apoio financeiro.

Finalmente, agradeço aos amigos, Brucy e Leonardo, do grupo de pesquisa em Computação Móvel do INF/UFG, pela ajuda no desenvolvimento de *ConUnits* e aplicações sensíveis ao contexto.

“O perigo real não é de computadores começarem a pensar como homens,
mas de homens começarem a pensar como computadores.”

Sydney J. Harris,

.

Resumo

Pereira de Sá, Marcio. **ConBus: Uma Plataforma de Middleware de Integração de Sensores para o Desenvolvimento de Aplicações Móveis Sensíveis ao Contexto**. Goiânia, 2010. 87p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Apesar da grande evolução e disseminação dos dispositivos móveis e sensores acoplados, desenvolver aplicações ubíquas ainda é uma tarefa complexa, principalmente, devido à grande diversidade de informações contextuais e à abundância de tecnologias de sensoriamento. Nesse cenário, sistemas de *middleware* assumem a responsabilidade de intermediar a comunicação entre as aplicações sensíveis ao contexto e os sensores que são as fontes de informações contextuais. Essa responsabilidade envolve diversos serviços, como implementar protocolos de comunicação com sensores heterogêneos, disponibilizar a comunicação assíncrona, possibilitar a inferência de informações contextuais, além da manutenção de modelos de contexto de alto nível. Entretanto, o desenvolvimento de plataformas de *middleware* para a provisão de contexto também é uma tarefa muito complexa, especialmente com relação à integração de módulos de sensoriamento a tais infraestruturas. Esses módulos de sensoriamento são os componentes de software das aplicações responsáveis pelo acesso aos dados de contexto coletados pelos sensores. Dentre os principais problemas relativos à essa integração estão: *i*) a complexidade inerente ao desenvolvimento de módulos de sensoriamento, que usualmente envolvem chamadas de baixo nível ao sistema operacional ou exigem a implementação de protocolos de comunicação para acesso a sensores remotos; *ii*) dificuldade de reutilização dos módulos de sensoriamento devido à falta de mecanismos que facilitem a disponibilização e a manutenção de tais módulos; e *iii*) o gerenciamento do ciclo de vida de módulos de sensoriamento acoplados à plataforma. Com o propósito de lidar com tais desafios, este trabalho propõe uma arquitetura de *middleware* para provisão de contexto em dispositivos móveis, denominada ConBus (**Context Bus**), que implementa estratégias de desenvolvimento, reutilização, implantação e ativação dinâmica de módulos de sensoriamento, fazendo uso racional dos recursos computacionais do dispositivo.

Palavras-chave

dispositivos móveis, middleware de provisão de contexto, sensores, módulos de sensoriamento, sensibilidade ao contexto.

Abstract

Pereira de Sá, Marcio. **ConBus: A Sensor Integration Middleware Platform for Mobile Context-Aware Application Development**. Goiânia, 2010. 87p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

In spite of the great evolution and dissemination of mobile devices and embedded sensors, development of ubiquitous applications is still a complex task mainly due to the great diversity of context information and the abundance of sensor technologies. In this scenario, middleware systems are responsible mediating communication between context-aware applications and sensors. This responsibility involves many services such as sensor communication protocols, asynchronous communication, context information reasoning. In spite of their importance for mobile context-aware applications, the development of middleware platforms for context provisioning is also a very complex task, specially in terms of sensor module integration to these platforms. This happens due to many factors, such as: *i)* huge complexity to develop sensor modules; *ii)* difficulties of reuse of sensor modules; and *iii)* sensor module life cycle management. This work proposes a context provisioning middleware architecture for mobile devices named ConBus (***Context Bus***) that implements development, reuse, deployment and dynamic activation strategies for sensor modules.

Keywords

mobile devices, context provisioning middleware, sensors, sensor module, context-aware.

Sumário

Lista de Figuras	11
Lista de Tabelas	13
Lista de Algoritmos	14
Lista de Códigos de Programas	15
1 Introdução	16
1.1 Objetivos	17
1.2 Resumo das Contribuições	18
1.3 Organização da dissertação	18
2 Integração de Sensores em Plataformas de <i>Middleware</i> de Provisão de Contexto	19
2.1 Principais problemas relativos à integração de sensores em plataformas de provisão de contexto	19
2.2 Requisitos para <i>middleware</i> de integração de módulos de sensoriamento	23
2.2.1 Desenvolvimento e reutilização de módulos de sensoriamento	23
2.2.2 Implantação e Localização de módulos de sensoriamento	24
2.2.3 Gerenciamento do ciclo de vida de módulos de sensoriamento	25
2.3 Conclusões	26
3 Arquitetura ConBus	28
3.1 Visão Geral	28
3.2 <i>Middleware</i> ConBus	30
3.2.1 ConUnit	30
3.2.2 Gerenciador de Contexto	33
3.2.3 Controlador de Comunicações	34
3.3 Modelagem de Contexto	35
4 Implementação	39
4.1 Visão Geral	39
4.2 ConUnits	41
4.3 Gerenciador de Contexto	42
4.4 Bundles OSGi	43
4.5 Controlador de Comunicações	45
4.6 Comunicação entre Aplicações e ConBus	46
4.7 Desenvolvimento de <i>ConUnits</i>	50
4.8 Reutilização de <i>ConUnits</i>	53

4.9	Limitações	54
5	Estudos de Caso	56
5.1	SmartTravel	57
	Seleção e Interação com <i>ConUnits</i>	58
5.2	AutoRinger	61
5.3	Conclusões	63
6	Trabalhos Relacionados	64
6.1	Context Toolkit	64
6.2	Hydrogen	66
6.3	Contory	68
6.4	Comparação	69
7	Conclusões	74
7.1	Contribuições	75
7.2	Trabalhos Futuros	76
	Referências Bibliográficas	78
A	API das Aplicações	80
A.1	API das Aplicações	80

Lista de Figuras

2.1	Os diversos tipos de sensores.	20
2.2	Interface entre sensores e aplicações sensíveis ao contexto/middleware.	21
3.1	Visão Geral da Arquitetura ConBus.	29
3.2	Integração de ConUnits e outros componentes da arquitetura ConBus ao framework OSGi.	32
3.3	Arquitetura de um ConUnit.	33
3.4	Modelo de Contexto.	36
3.5	Um exemplo de uso do modelo de contexto da arquitetura ConBus.	37
4.1	Os componentes da arquitetura ConBus e como se comunicam entre si.	40
4.2	Diagrama Arquitetural de um ConUnit, com o módulo de sensoriamento interno.	42
4.3	Diagrama Arquitetural de um ConUnit, com o módulo de sensoriamento externo.	43
4.4	Interface ContextDataConversion	44
4.5	Funcionamento da comunicação síncrona entre aplicações sensíveis ao contexto e ConBus.	44
4.6	Funcionamento da comunicação assíncrona entre aplicações sensíveis ao contexto e ConBus.	45
4.7	Interação entre aplicações e middleware ConBus, através dos comandos da API das Aplicações.	47
4.8	Implementando a interface ContextDataConversion para criar um ConUnit que coleta dados sobre a localização de um determinado usuário.	51
4.9	Implementando a interface ContextDataConversion para criar um ConUnit que coleta dados sobre a temperatura de uma determinada sala.	52
4.10	Desenvolvimento de um ConUnit	53
5.1	Interação entre aplicações móveis, ConBus e sensores.	56
5.2	a) Tela de Iniciação. b) Tela de cadastro de pontos turísticos.	58
5.3	Chegada de uma notificação sobre ponto turístico encontrado.	59
5.4	a) Opções para receber uma notificação. b) Mapa indicando a localidade encontrada.	60
5.5	Funcionamento e uso do AutoRinger.	62
5.6	a) Criando uma condição de uso. b) Alterando o tipo de toque de campanha de acordo com o contexto corrente.	63
6.1	Diagrama de objetos para as abstrações do Context Toolkit.	66
6.2	Diagrama arquitetural do Hydrogen .	67

Lista de Tabelas

2.1	Problemas e seus respectivos requisitos relativos ao desenvolvimento e reutilização de módulos de sensoriamento.	27
2.2	Problemas e seus respectivos requisitos relativos à implantação e localização de módulos de sensoriamento.	27
2.3	Problemas e seus respectivos requisitos relativos ao gerenciamento do ciclo de vida de módulos de sensoriamento.	27
3.1	Problemas, requisitos e soluções implementadas pelo ConBus relativos ao desenvolvimento e reutilização de módulos de sensoriamento.	37
3.2	Problemas, requisitos e soluções implementadas pelo ConBus relativos à implantação e localização de módulos de sensoriamento.	37
3.3	Problemas, requisitos e soluções implementadas pelo ConBus relativos ao gerenciamento do ciclo de vida de módulos de sensoriamento.	38
5.1	Esforço de implementação dos ConUnits e uso nas aplicações	61
6.1	Comparação do ConBus/ConUnit com outros sistemas de <i>middleware</i>	70

Lista de Algoritmos

Lista de Códigos de Programas

Introdução

A computação ubíqua, idealizada por Mark Weiser [20], ainda na década de 1980, caracteriza-se por permitir que seus usuários tenham acesso a serviços computacionais praticamente em qualquer lugar e a qualquer momento. Atualmente, muitos dos objetivos da computação ubíqua podem ser alcançados através da união de duas outras áreas da Computação, a *computação móvel*, responsável pelas pesquisas e desenvolvimento de tecnologias, como as redes sem fio e os dispositivos móveis, e a *computação sensível ao contexto*.

O objetivo da computação sensível ao contexto é permitir o desenvolvimento de aplicações que percebam e reajam a mudanças do ambiente no qual estão inseridas como, por exemplo, alterações na localização de um usuário, na temperatura de uma determinada sala e no estado dos recursos computacionais de um dispositivo. Todas essas informações são denominadas *informações de contexto* e são providas, direta ou indiretamente, por sensores. Esses sensores podem ser físicos (hardware de sensoriamento, como GPS) ou virtuais (elementos de software capazes de coletar informações relevantes a aplicações ou outros componentes de software que não são coletadas por sensores físicos, como os dados de uma agenda de compromissos). A diversidade e o barateamento das tecnologias de sensoriamento tornam a computação sensível a contexto um paradigma com grande potencial a ser explorado, principalmente, em aplicações ubíquas, onde a união da computação móvel com a computação sensível ao contexto permite desenvolver aplicações inovadoras. Porém, desenvolver tais aplicações ainda é uma atividade complexa por causa da imensa quantidade de informações contextuais e de tecnologias de sensoriamento.

Nesse cenário, sistemas de *middleware* assumem a responsabilidade de intermediar a comunicação entre as aplicações sensíveis ao contexto e os sensores, considerados as fontes de informações contextuais. Essa responsabilidade envolve diversos serviços, como implementar protocolos de comunicação com sensores heterogêneos, disponibilizar a comunicação assíncrona, possibilitar a inferência de informações contextuais, além da manutenção de modelos de contexto de alto nível e a incorporação dos dados provenientes dos sensores. Neste trabalho, o termo *módulo de sensoriamento* é empregado para descrever o componente de software que se conecta a um sensor, seja ele físico ou virtual,

e coleta o dado de contexto propriamente dito, além de meta-informações, como a data e hora da coleta, a precisão e o tipo do sensor correspondente. Os módulos de sensoramento são, portanto, os intermediadores entre os sensores propriamente ditos (representados, por exemplo, pelos seus *device drivers*, no caso de sensores físicos) e os sistemas que utilizam seus dados. Deve-se observar que um módulo de sensoramento é um componente a serviço das aplicações, desenvolvido, geralmente, pelos desenvolvedores dessas aplicações. Ele é responsável pela obtenção dos dados coletados por um sensor e enviados à aplicação correspondente e não deve ser confundido, por exemplo, com um *device driver* ou uma API para coleta de dados de sensores disponibilizada pelo sistema operacional.

Apesar de serem muito importantes para a construção de aplicações móveis sensíveis ao contexto, o desenvolvimento de plataformas de *middleware* para a provisão de contexto é uma tarefa muito complexa, especialmente com relação à integração de módulos de sensoramento a tais infraestruturas. Isso ocorre devido a vários fatores:

- a dificuldade envolvida no próprio desenvolvimento dos módulos de sensoramento, exigindo, muitas vezes, a implementação de protocolos de comunicação para acesso a sensores remotos ou envolvendo chamadas de baixo nível ao sistema operacional;
- desafios relativos à transparência de localização de sensores ou à possibilidade de se identificar onde estão instalados;
- dificuldade de reutilização dos módulos de sensoramento devido à falta de mecanismos que facilitem tanto a disponibilização a outros desenvolvedores quanto a manutenção de tais módulos; e
- o gerenciamento do ciclo de vida de módulos de sensoramento acoplados à plataforma.

1.1 Objetivos

Dentre os principais objetivos deste trabalho, destacam-se:

- o estudo dos principais problemas relativos à integração de sensores a infraestruturas de provisão de contexto em dispositivos móveis;
- a identificação de requisitos desejáveis em plataformas de provisão de contexto capazes de auxiliar na resolução dos problemas identificados no item anterior;
- o desenvolvimento de uma infraestrutura de provisão de contexto em dispositivos móveis, denominada ConBus (*Context Bus*), que oferece facilidades em relação à integração de sensores à essa infraestrutura, facilitando o desenvolvimento de aplicações móveis sensíveis ao contexto.

1.2 Resumo das Contribuições

Apresentamos, nesta seção, um resumo das contribuições deste trabalho, as quais são descritas detalhadamente na Seção 7.1.

- Projeto e implementação de uma arquitetura de provisão de contexto para dispositivos móveis que oferece facilidades para a integração de sensores/módulos de sensoriamento à plataforma de *middleware*.
- Implementação do *framework ConUnit*, que oferece facilidades, tanto para o desenvolvimento quanto para a reutilização e implantação de módulos de sensoriamento.
- Projeto e implementação de mecanismos que permitem à arquitetura ConBus acoplar dinamicamente módulos de sensoriamento.
- Projeto e implementação de mecanismos básicos capazes de gerenciar dinamicamente o ciclo de vida de cada módulo de sensoriamento separadamente.
- Análise dos principais problemas relativos à integração de módulos de sensoriamento em plataformas de *middleware* de provisão de contexto, bem como a identificação dos requisitos relacionados a tais problemas.

1.3 Organização da dissertação

Esta dissertação está organizada da seguinte forma. O Capítulo 2 discute o problema da integração de sensores em sistemas de *middleware* para provisão de contexto e apresenta alguns requisitos para auxiliar na resolução dos principais desafios relacionados. O Capítulo 3 apresenta a arquitetura ConBus. Os aspectos relativos à implementação dessa arquitetura são discutidos no Capítulo 4. O Capítulo 5 apresenta os estudos de caso de implementação de duas aplicações baseadas no *middleware* e discute como a abordagem de integração de sensores facilitou o desenvolvimento dessas aplicações. O Capítulo 6 discute trabalhos na literatura relacionados com a abordagem discutida nesta dissertação. O Capítulo 7 apresenta as conclusões deste trabalho e futuras direções de pesquisa.

Integração de Sensores em Plataformas de *Middleware* de Provisão de Contexto

Este Capítulo discute, inicialmente, os principais problemas relativos à integração de sensores em plataformas de provisão de contexto. Em seguida, apresenta alguns requisitos que auxiliam na resolução de tais problemas.

2.1 Principais problemas relativos à integração de sensores em plataformas de provisão de contexto

Sistemas de *middleware* têm importância fundamental para a computação sensível ao contexto por promoverem o desacoplamento entre aplicações sensíveis ao contexto e as fontes de informação contextual, além de oferecerem transparência com relação aos mecanismos de provisão de contexto. Diversos sistemas de *middleware* com este propósito têm sido propostos, desde o trabalho original do Context Toolkit [6], passando por trabalhos focados em aspectos específicos, como privacidade [11], modelagem complexa de informações contextuais [3], cenários particulares como *smart rooms* [15] [12], mecanismos de adaptação de aplicações [8], execução em *smartphones* [14] e transparência dos mecanismos de provisão de contexto [14]. Entretanto, poucos trabalhos têm focado na facilidade de integração de sensores ao *middleware* de provisão de contexto. Segundo o modelo de camadas, proposto por Henriksen e Indulska [8], essa integração deve ser implementada nas camadas de recepção e coleta de contexto.

Neste trabalho, o termo *sensor* é empregado com um significado mais amplo, como exibido pela Figura 2.1, indicando qualquer dispositivo físico (*hardware*) ou virtual (*software*) capaz de coletar alguma informação contextual relevante para uma aplicação ou usuário [2]. Desse modo, são considerados sensores os dispositivos que coletam dados do ambiente (temperatura, pressão, umidade, luminosidade, etc), do usuário (dados de agendas de compromissos, preferências, localização, pessoas co-localizadas, dentre outras) e, ainda, dados relativos ao próprio ambiente computacional (como nível de

bateria, uso de CPU, memória e qualidade da rede sem fio). A Figura 2.2 ilustra a relação entre os sensores, seus módulos de sensoriamento e as aplicações ou os sistemas de *middleware*.

Deve-se observar que um *device driver* de um sensor não é considerado um módulo de sensoriamento, visto que um módulo de sensoriamento é um componente de software localizado logicamente nas aplicações sensíveis ao contexto. Dessa forma, um módulo de sensoriamento é um componente que se comunica, por exemplo, com um *device driver* ou com o sistema operacional, através de suas APIs específicas, para obter as informações de contexto desejadas pelas aplicações que fazem uso de tais módulos de sensoriamento. As funções básicas de um módulo de sensoriamento são, portanto, abstrair do restante da aplicação os detalhes referentes à comunicação com os sensores (através de seus *device drivers*, por exemplo), conversão de formato e representação dos dados coletados, inclusão de metadados importantes, como tipo de contexto, precisão das informações obtidas e tipo do sensor correspondente, dentre outras.

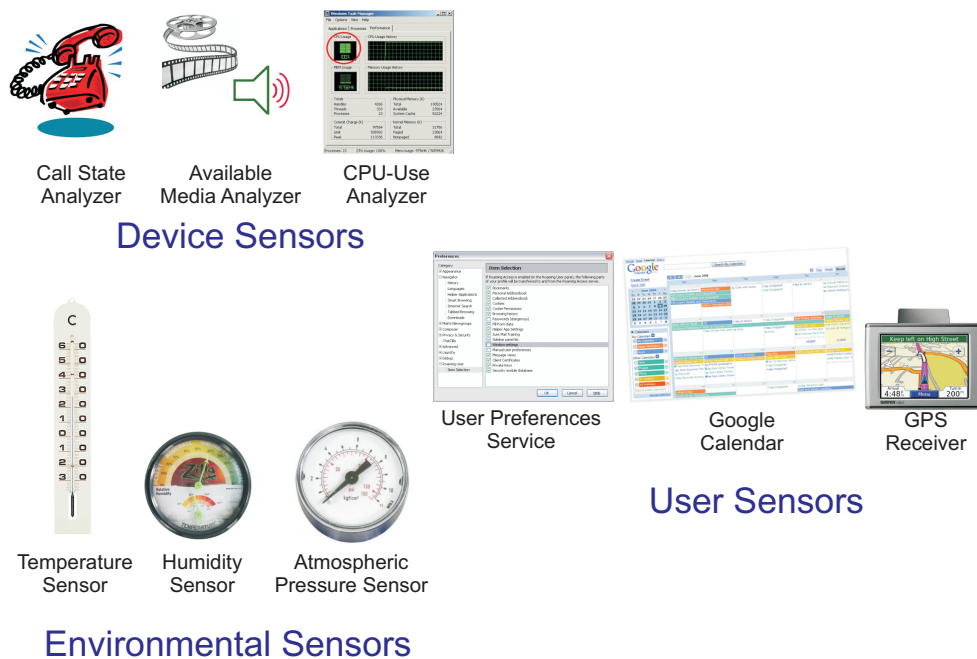


Figura 2.1: Os diversos tipos de sensores.

Os desafios para a integração de tais módulos são especialmente interessantes em sistemas de *middleware* projetados para dispositivos móveis. Em tais dispositivos, a diversidade de tecnologias de sensoriamento motivam a implementação de cenários mais amplos de computação sensível ao contexto. Por exemplo, a maioria dos *smartphones* vendidos atualmente já oferecem sensores como GPS, acelerômetro e bússola digital. Além disso, a mobilidade provida por tais dispositivos permite a elaboração de cenários dinâmicos, nem sempre previsíveis, nos quais uma mudança de localização pode ocasionar

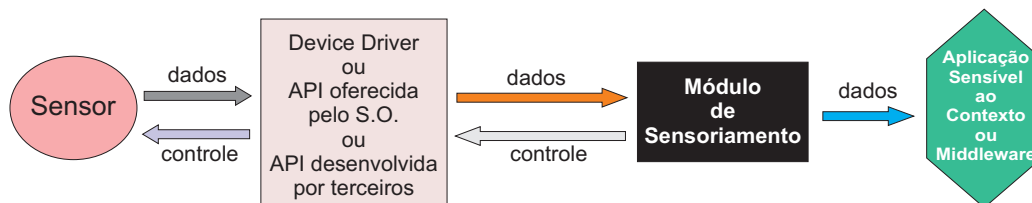


Figura 2.2: Interface entre sensores e aplicações sensíveis ao contexto/middleware.

o acesso a outro sensor remoto responsável por uma mesma informação contextual (e.g. sensor de temperatura do ambiente). Por outro lado, a limitação de recursos computacionais exige a adoção de estratégias para minimizar o uso de memória, CPU e comunicação pela rede. De fato, há vários aspectos que dificultam a integração de sensores em sistemas de *middleware* para provisão de contexto em dispositivos móveis e o desenvolvimento de módulos de sensoriamento nesses ambientes.

A primeira dificuldade é a *complexidade inerente ao desenvolvimento dos módulos de sensoriamento*, que usualmente envolve chamadas de baixo nível ao sistema operacional ou exige a implementação de protocolos de comunicação para acesso a sensores remotos. Esses protocolos necessitam, muitas vezes, de técnicas específicas para tratar questões, como problemas de comunicação entre os sensores e seus módulos de sensoriamento, além da segurança e privacidade dos dados coletados.

Além da complexidade do código *per se*, as chamadas aos módulos de sensoriamento devem ser controladas de maneira que o múltiplo acesso a sensores por diversas aplicações otimize o uso de recursos computacionais e mantenha a consistência do dado obtido do sensor¹. Embora muitos sistemas operacionais para dispositivos móveis ofereçam interfaces para acesso a sensores locais, tais interfaces são *fortemente acopladas aos tipos de sensores estaticamente definidos pela plataforma*. Consequentemente, elas não permitem a integração de sensores alternativos que ofereçam a mesma informação coletada por sensores locais ou a inclusão de sensores externos, tais como TelosB² ou MICA2³. Assim, ainda que o sistema operacional gerencie adequadamente o acesso ao sensor local, ele não exime o *middleware* de implementar mecanismos de gerenciamento adicionais quando um novo módulo de sensoriamento precisa ser utilizado. Interfaces

¹Por exemplo, varreduras em sinais de placas IEEE 802.11 devem levar em conta um tempo mínimo para obtenção de respostas consistentes e o acesso concorrente à placa geralmente não é permitido

²<http://www.xbow.com/Products/productdetails.aspx?sid=252>

³<http://www.xbow.com/Products/productdetails.aspx?sid=174>

de caráter mais geral e que permitem independência de sensores como Java Location API [18], para obtenção de dados sobre a localização de usuários ou dispositivos móveis, são limitadas a tipos de contexto particulares.

Um segundo desafio está relacionado com as questões sobre a *seleção* e o *uso* de módulos de sensoriamento pelas aplicações ou outros componentes da arquitetura nas plataformas de provisão de contexto. Dentre elas, algumas podem ser bastante complexas, como no caso de existirem mais de um módulo de sensoriamento que forneça o mesmo tipo de contexto relacionado a uma mesma entidade. Um exemplo disso é a coleta de dados sobre os compromissos de um mesmo usuário gerenciados por diferentes sistemas de agendas *online*, como o Google Calendar⁴ e o Yahoo Calendar⁵. Nesse caso, a escolha dos critérios que devem ser empregados para realizar a seleção do módulo de sensoriamento mais adequado a cada necessidade de uma aplicação ou outros clientes pode depender de vários fatores, como a precisão e o formato dos dados coletados, o tipo de sensor empregado e a taxa de atualização da informação contextual.

Uma terceira dificuldade ocorre porque, em geral, a plataforma de *middleware* deve prover às aplicações ou outros clientes transparência de localização dos sensores acoplados, visando evitar que uma aplicação dependa unicamente de uma fonte de contexto específica, favorecendo assim a manutenção e reutilização de módulos de sensoriamento.

Porém, muitas vezes, pode ser importante para a aplicação conhecer a localização (local ou remota ao dispositivo onde a aplicação está sendo executada) do sensor correspondente. Isso pode ser útil quando houver, por exemplo, mais de uma fonte de contexto fornecendo a mesma informação contextual para uma mesma entidade. Além disso, também pode ser importante em situações críticas onde a localização do sensor puder influenciar na qualidade e confiabilidade dos dados coletados. Por exemplo, devido a problemas na comunicação entre um sensor de temperatura instalado em uma sala e o dispositivo móvel que está executando a aplicação, essa aplicação escolhe, então, obter as informações sobre a temperatura ambiente coletadas por um sensor local ao dispositivo móvel, mesmo sabendo que este sensor local oferece uma precisão menor que a fornecida pelo sensor remoto instalado no ambiente.

Outro problema comum às arquiteturas é a *dificuldade de reutilização dos módulos de sensoriamento*. Isso ocorre devido à carência de mecanismos que facilitem a disponibilização e a manutenção desses módulos de sensoriamento. Um exemplo disso é a falta de padronização tanto das interfaces de comunicação, quanto dos formatos e representações dos dados e metadados disponibilizados pelos módulos de sensoriamento.

⁴<https://www.google.com/calendar>

⁵<http://calendar.yahoo.com>

Além disso, devido à carência de recursos computacionais nos dispositivos móveis, como memória, processamento e energia, o gerenciamento do ciclo de vida de módulos de sensoriamento torna-se um dos grandes desafios para as arquiteturas de *middleware*. Esse gerenciamento diz respeito ao controle de cada módulo de sensoriamento durante as diversas fases do ciclo de vida (instalação, ativação, desativação, atualização e desinstalação).

2.2 Requisitos para *middleware* de integração de módulos de sensoriamento

A integração dos módulos de sensoriamento a plataformas de *middleware* compreende os aspectos de *desenvolvimento*, *implantação* e *gerenciamento do ciclo de vida* de tais componentes. A partir das dificuldades e problemas discutidos anteriormente, uma lista de requisitos para possibilitar essa integração foi identificada com o objetivo de auxiliar na solução de tais dificuldades. Grande parte desses requisitos foi implementada na plataforma ConBus através de funções específicas de seus componentes internos, discutidos no Capítulo 3. Inicialmente, esses requisitos foram divididos em três grupos principais, relativos a: *i) desenvolvimento e reutilização* de módulos de sensoriamento; *ii) implantação e localização* dos sensores e módulos; e, *iii) gerenciamento do ciclo de vida* de módulos de sensoriamento em execução em dispositivos móveis. Entretanto, muitos desses requisitos poderiam ser listados em mais de um grupo principal, sendo, portanto, essa classificação apenas uma das muitas formas possíveis de organizá-los.

2.2.1 Desenvolvimento e reutilização de módulos de sensoriamento

Os requisitos indicados a seguir tratam dos aspectos relacionados à criação e à reutilização de módulos de sensoriamento.

Inicialmente, é desejável que plataformas de provisão de contexto forneçam abstrações que permitam isolar do *middleware* e das aplicações a complexidade inerente à coleta e disponibilização eficientes dos dados de contexto feita pelos diversos sensores acoplados. Esse requisito visa tratar os problemas relativos à comunicação entre sensores e seus módulos de sensoriamento, tais como possíveis desconexões ou comunicação intermitente. Além disso, lida com questões sobre a privacidade e a segurança dos dados coletados, visto que, dependendo dos mecanismos oferecidos para gerenciar essa comunicação entre sensores e módulos de sensoriamento, muitos desafios relacionados ao controle de acesso aos dados coletados, bem como o uso de técnicas criptográficas e armazenamento seguro desses mesmos dados podem ter que ser solucionados. Por exemplo, uma das funções de um módulo de sensoriamento que obtém dados oriundos

de um sensor remoto que coleta informações sobre os compromissos de um determinado usuário pode ser a de garantir que usuários ou sistemas computacionais não autorizados não consigam acesso a tais informações.

Um segundo requisito importante é a disponibilização de mecanismos que facilitem a criação de módulos de sensoriamento coesos, fracamente acoplados a outros componentes e cujo encapsulamento permita a troca de informações com o meio exterior apenas através de interfaces bem definidas, facilitando a manutenção e, até mesmo, a substituição de um módulo por outro equivalente. Como exemplo, o uso de *bundles* OSGi como estrutura para a criação de módulos de sensoriamento, permite que sejam desenvolvidos módulos independentes e altamente coesos, permitindo ainda que toda a comunicação com outros componentes da arquitetura seja feita através de interfaces e serviços bem definidos pelo desenvolvedor da plataforma e gerenciados pelo *framework* OSGi.

Outro importante requisito é o desenvolvimento, disponibilização e uso de repositórios de módulos de sensoriamento. Esses repositórios poderiam, por exemplo, ser disponibilizados na Internet, de modo a facilitar a reutilização de módulos desenvolvidos por terceiros. Isso favorece a manutenção, a descoberta e a correção de erros, além de facilitar a troca de informações sobre o funcionamento de cada módulo, diminuindo assim o tempo de desenvolvimento de aplicações sensíveis ao contexto.

Além disso, a disponibilização de mecanismos para facilitar a conversão dos dados específicos de cada sensor para um formato padronizado é sempre um desafio, visto que, devido à grande diversidade de informações contextuais e de sensores, existe uma enorme quantidade de formatos específicos. Porém, oferecer um modelo de dados que permita descrever dados de diferentes sensores viabiliza a criação mais rápida e segura dos respectivos módulos de sensoriamento. Por exemplo, para coletar dados sobre a localização de um usuário, um receptor GPS fornece tais informações através de três coordenadas ou três valores distintos (latitude, longitude e altitude); porém, se for necessário obter dados relativos à temperatura de uma determinada sala, será preciso utilizar apenas um único valor (o próprio valor da temperatura). Por isso, o modelo de dados de contexto deve proporcionar flexibilidade para permitir que tanto dados sobre localização, temperatura ou, até mesmo, dos compromissos de um usuário armazenados por uma agenda eletrônica possam ser armazenados e disponibilizados através dessas estruturas.

2.2.2 Implantação e Localização de módulos de sensoriamento

As questões a respeito da implantação e localização dos módulos de sensoriamento são tratadas através de requisitos descritos a seguir.

Uma moderna plataforma de provisão de contexto em dispositivos móveis, deveria proporcionar mecanismos que viabilizem a implantação dinâmica dos módulos de sensoriamento junto ao *middleware*. Tais funcionalidades facilitam o acoplamento de novos módulos de sensoriamento, na medida em que não se necessita reiniciar todo o sistema de provisão de contexto para que esses novos módulos sejam conectados a ele. Isso ajuda a evitar possíveis problemas em aplicações sensíveis ao contexto que já se encontram em execução, pois, em alguns casos, se o *middleware* parar, mesmo que por alguns instantes, de enviar informações de contexto a elas, a correta execução dessas aplicações poderia ser comprometida.

Esse fato poderia comprometer a confiabilidade e a eficiência dessas aplicações e poderiam levar seus usuários a desistir de usá-las em casos extremos. Por exemplo, se uma aplicação, que configura o tipo de toque de campainha de um aparelho móvel, deixar de executar essa função adequadamente porque o *middleware* de provisão de contexto ficou por alguns momentos inoperante devido a uma reinicialização. Esse fato poderia gerar situações desagradáveis a um usuário que estivesse em uma reunião ou em sala de aula, onde um barulho ocasionado por uma chamada seria muito inconveniente e constrangedor.

Outro requisito necessário é a capacidade dessas infraestruturas oferecerem às aplicações transparência de localização dos sensores (local ou remoto) que coletam informações de contexto. Isso facilita a manutenção das aplicações sensíveis ao contexto, evitando a dependência de sensores e/ou módulos específicos. Entretanto, conforme discutido anteriormente, a infraestrutura deve sempre fornecer a possibilidade, caso desejado pelas aplicações, de se obter informações sobre os sensores ou módulos de sensoriamento responsáveis pela coleta e disponibilização dessas informações contextuais, disponibilizando dados, como a localização dos mesmos (local ou remota), precisão das informações coletadas, modelo do sensor e periodicidade de atualização dos dados contextuais.

2.2.3 Gerenciamento do ciclo de vida de módulos de sensoriamento

O ciclo de vida de um módulo de sensoriamento em um sistema de *middleware* para provisão de contexto compreende todas as fases desde a instalação, passando pela ativação, desativação, atualização e desinstalação desse módulo dentro do sistema. Portanto, seu gerenciamento deve levar em consideração todas essas etapas.

A plataforma de *middleware* deve prover meios para que tanto ativações, quanto desativações aconteçam dinamicamente, sem a necessidade de parar todo o sistema sempre que essas ações forem necessárias. Ativar e desativar fontes de contexto dinamicamente pode ser um importante aliado no gerenciamento dos escassos recursos computacionais disponíveis nos dispositivos móveis atuais.

Além de ativar e desativar módulos de sensoriamento dinamicamente, é impor-

tante que o *middleware* de provisão de contexto ofereça mecanismos para permitir a atualização de tais módulos sempre que necessário sem que o sistema todo tenha que ser reiniciado. Através desses mecanismos de atualização dinâmica e do uso de repositórios remotos, pode ser possível a atualização automatizada de módulos de sensoriamento. Nesse caso, sempre que uma nova versão de um módulo de sensoriamento for inserida em um repositório acessível pelo *middleware*, o sistema poderá, de acordo com a vontade do usuário, atualizar automaticamente o módulo de sensoriamento, corrigindo possíveis erros em versões anteriores ou agregando novas funcionalidades.

2.3 Conclusões

Com o propósito de lidar com esses problemas mencionados na Seção anterior, este trabalho descreve uma arquitetura de *middleware* para provisão de contexto em dispositivos móveis denominada ConBus (**Context Bus**), apresentada no Capítulo 3. Ela oferece facilidades para o desenvolvimento, implantação e gerenciamento do ciclo de vida de módulos de sensoriamento, favorecendo a reutilização de tais módulos em outros ambientes, em uma abordagem que limita o uso dos recursos computacionais do dispositivo.

A versão corrente do ConBus ainda não contempla todos os requisitos citados anteriormente, como o uso de repositórios de módulos de sensoriamento e a disponibilização de abstrações que permitam isolar do *middleware* a complexidade inerente à coleta e disponibilização eficientes dos dados de contexto, especialmente em relação à comunicação entre sensores e módulos de sensoriamento. Por exemplo, mecanismos que evitem a paralização das atividades do *middleware* quando um sensor não estiver ativo ou a comunicação entre ele e seu o módulo de sensoriamento não for possível em um determinado momento.

As Tabelas 2.1, 2.2 e 2.3 resumem os problemas e requisitos discutidos neste Capítulo, organizando-os em três aspectos ou fases diferentes da integração dos módulos de sensoriamento a plataformas de *middleware*: *desenvolvimento*, *implantação* e *gerenciamento do ciclo de vida*.

Tabela 2.1: *Problemas e seus respectivos requisitos relativos ao desenvolvimento e reutilização de módulos de sensoriamento.*

Problemas	Requisitos
Complexidade inerente ao desenvolvimento dos módulos de sensoriamento	Mecanismos para tratar problemas de comunicação entre sensores e módulos.
	Mecanismos para facilitar a conversão dos dados específicos de sensor para um formato padronizado.
	Mecanismos para a criação de módulos coesos, fracamente acoplados e encapsulados.
Seleção e uso de módulos de sensoriamento	Metadados para facilitar a escolha do sensor mais adequado às necessidades das aplicações a cada instante, como a precisão do sensor e outros parâmetros de QoS.
	Algoritmos e/ou heurísticas para a seleção de módulos quando houver mais de um fornecendo o mesmo tipo de contexto para uma mesma entidade.
Dificuldade de reutilização	Repositórios de módulos de sensoriamento.
	Facilidades para documentar e exibir documentação de módulos de sensoriamento.

Tabela 2.2: *Problemas e seus respectivos requisitos relativos à implantação e localização de módulos de sensoriamento.*

Problemas	Requisitos
Problemas relativos à implantação de módulos de sensoriamento	Mecanismos para possibilitar a implantação dinâmica desses módulos.
Transparência de localização	Abstrações para permitir transparência de localização tanto de sensores quanto dos módulos de sensoriamento.
	Metadados que informem, sempre que necessário, a localização do sensor (local ou remoto), tipo e modelo.

Tabela 2.3: *Problemas e seus respectivos requisitos relativos ao gerenciamento do ciclo de vida de módulos de sensoriamento.*

Problemas	Requisitos
Problemas relativos à ativação, atualização e desativação de módulos de sensoriamento	Mecanismos para ativar e desativar dinamicamente os módulos de sensoriamento.
	Mecanismos para atualizações dinâmicas e automatizadas.

Arquitetura ConBus

Este Capítulo tem por objetivo apresentar o projeto arquitetural do ConBus, seus componentes e suas principais características, identificando, ainda, as soluções propostas por essa arquitetura para atender aos requisitos identificados no Capítulo 2. Pelo fato deste Capítulo discutir apenas as questões de projeto da infraestrutura, os aspectos relativos à implementação dessas soluções aqui descritas serão tratados no Capítulo 4.

3.1 Visão Geral

O ConBus é uma infraestrutura desenvolvida para prover informações de contexto a aplicações móveis. Através de seu projeto arquitetural, baseado na interação entre três camadas (Camada de Aplicação, *Middleware* ConBus e Camada de Sensores), ilustradas na Figura 3.1, essa plataforma de *middleware* oferece uma arquitetura flexível em relação ao gerenciamento dos módulos de sensoramento de contexto acoplados a ela. Por isso, o ConBus é responsável por gerenciar o ciclo de vida dos módulos de sensoramento integrados à sua implementação, oferecer um *framework* de integração de novos módulos de sensoramento (*ConUnit framework*), e prover às aplicações móveis sensíveis ao contexto uma interface uniforme que padroniza o acesso aos dados de contexto providos por diferentes sensores integrados à arquitetura através dos módulos de sensoramento.

O ConBus (Context Bus) atua como um barramento de contexto através do qual as aplicações sensíveis ao contexto e os sensores se interconectam e interagem. Desse modo, uma aplicação pode obter dados de um ou vários sensores e um sensor pode prover informações contextuais para uma ou mais aplicações.

A Camada de Aplicação (*Application Layer*) compreende todas as aplicações sensíveis ao contexto que utilizam as informações contextuais fornecidas por uma única instância do *middleware* ConBus. O fato do ConBus centralizar a comunicação com os módulos de sensoramento proporciona uma melhor gerência dos recursos computacionais do dispositivo móvel, pois permite que a instância de um *driver* de comunicação com um sensor específico seja compartilhada entre diferentes aplicações. Um outro benefício direto é serializar o acesso a fontes de informações de contexto que só podem ser acessa-

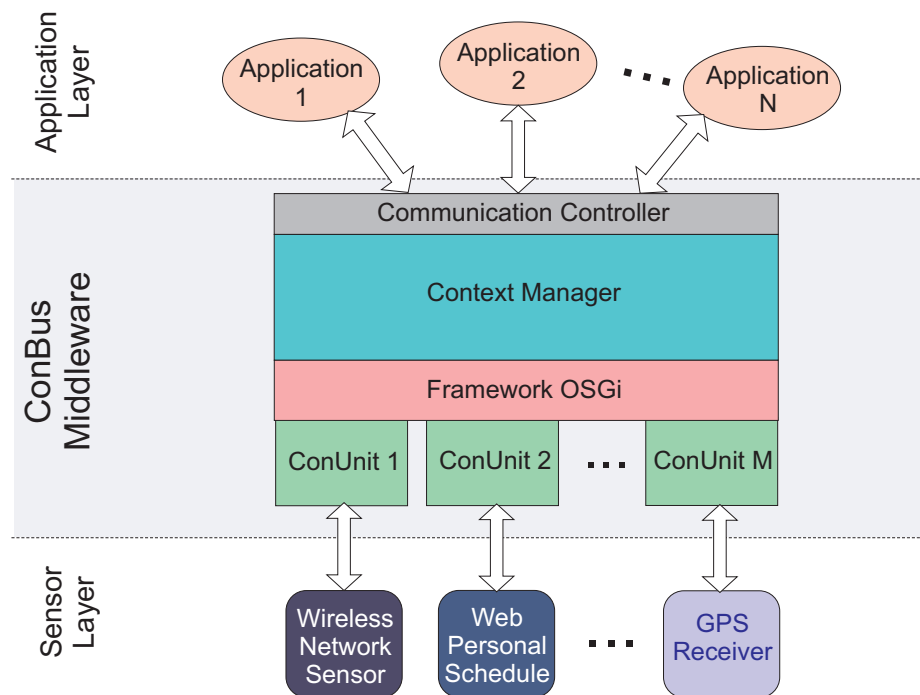


Figura 3.1: Visão Geral da Arquitetura ConBus.

das por uma aplicação a cada instante, como, por exemplo, a operação de varredura em interfaces de rede sem fio IEEE 802.11.

As aplicações podem acessar as informações providas pelos sensores através da *API das Aplicações* oferecida pelo *middleware* ConBus. Essa API disponibiliza os comandos através dos quais as informações de contexto podem ser requisitadas pelas aplicações, estabelecendo o formato, a sintaxe e a codificação tanto das requisições quanto das respostas, bem como os tipos de erros mais comuns para cada requisição feita ao *middleware*.

Há duas formas básicas de comunicação possíveis disponibilizadas pela *API das aplicações*: (i) através de *interface síncrona*, em que é possível consultar qualquer informação contextual gerenciada pelo ConBus e coletada pelos sensores acoplados; ou ainda, (ii) por meio de *interface assíncrona*, baseada em eventos, em que se pode registrar interesse em obter determinado tipo de informação de contexto sempre que condições específicas ocorrerem. A Seção 4.6 discute todos os comandos da *API das Aplicações*, mostrando, ainda, exemplos de uso de cada comando disponibilizado.

Os módulos de sensoriamento são acoplados ao restante do *middleware* através dos componentes conhecidos por *ConUnits*, que, de acordo com a Figura 3.1, pertencem à camada intermediária, denominada *middleware* ConBus. Esses *ConUnits* intermedeiam a comunicação entre os sensores propriamente ditos e o restante da arquitetura ConBus para que, finalmente, suas informações contextuais possam ser utilizadas pelas aplicações sensíveis ao contexto.

A Camada de sensores (*Sensor Layer*), ilustrada na Figura 3.1, é constituída por sensores que fornecem informações contextuais ao *middleware ConBus*. Conceitualmente, um sensor pode ser local ou remoto em relação ao dispositivo móvel. Conforme discutido no Capítulo 2, os sensores podem ser vistos como: *i*) um software que provê informações do sistema local, tais como, uso da CPU, memória, tipo e versão do sistema operacional, *codecs* de vídeo e áudio instalados; *ii*) um sensor físico específico embutido no dispositivo móvel, como um GPS, acelerômetro, câmera de vídeo, interfaces de rede sem fio; *iii*) um software que coleta dados de contexto de um provedor externo, como, por exemplo, um serviço de agenda de compromissos remoto (e.g., Google Calendar, Yahoo Calendar); ou *iv*) um sensor físico instalado no ambiente que coleta informações de temperatura, umidade, ruído (e.g., um sensor TelosB, Mica2).

3.2 Middleware ConBus

O *middleware ConBus* é a camada intermediária, na arquitetura proposta, e tem a função de gerenciar todas as informações contextuais coletadas pelos sensores que constituem a *Camada de Sensores* e que, posteriormente, são fornecidas a todas as aplicações sensíveis ao contexto. Para realizar esse trabalho, o ConBus implementa e utiliza os seguintes componentes: *ConUnits*, *Gerenciador de Contexto* e *Controlador de Comunicações*. Além deles, o *middleware ConBus* utiliza os serviços providos pelo *framework OSGi*.

O *framework OSGi* [1] é o componente que permite o acoplamento, tanto estático quanto dinâmico, de todas as fontes de contexto (sensores) junto ao *middleware ConBus* e também a interligação de todos os outros componentes internos do *middleware*. Dessa forma, o *OSGi* atua como um barramento de comunicação que permite ao *Gerenciador de Contexto*, por exemplo, controlar o uso de todos os *ConUnits* instalados, além de possibilitar o acesso dos outros componentes da arquitetura aos dados disponibilizados por essas fontes.

3.2.1 ConUnit

O *framework ConUnit* (*Context Unit*) é responsável por definir como um sensor deve ser integrado ao ConBus. Através de uma instância de *ConUnit*, o ConBus poderá coletar dados de contexto de um sensor específico, de acordo com o interesse das aplicações. Em outras palavras, cada instância de *ConUnit* incorpora um determinado módulo de sensoriamento à arquitetura ConBus. Esse módulo de sensoriamento incorporado pelo *ConUnit* pode ser implementado dentro ou fora desse *ConUnit*. Isto acontece porque,

muitas vezes, pode não ser viável implementá-lo dentro do *ConUnit*¹. Essa escolha dá ao desenvolvedor maior flexibilidade para integrar um módulo de sensoriamento ao ConBus. Porém, sempre que possível, é desejável que o módulo de sensoriamento seja implementado dentro de um *ConUnit*, pois isso facilita o gerenciamento da execução de todo o código desse módulo, possibilitando ao *middleware* parar e reiniciar adequadamente todas as suas “*threads*” de execução, fechar e reabrir adequadamente possíveis conexões de rede necessárias, parar e reiniciar o funcionamento dos próprios sensores, que, em geral, consomem a maior parte da energia do dispositivo móvel, possibilitando assim um maior controle pelo *middleware* sobre o consumo de recursos computacionais, como memória, processamento e energia pelos módulos de sensoriamento e sensores acoplados.

Deve-se ressaltar que o desenvolvedor responsável por criar e acoplar um sensor ao ConBus é quem deve se preocupar com todos os detalhes referentes à obtenção dos dados de contexto. Por exemplo, o desenvolvedor deve lidar com a complexidade de comunicação de baixo nível de um *driver* de um sensor físico instalado no dispositivo móvel ou remoto (e.g., acelerômetro, sensor de temperatura instalado em uma sala), implementar os protocolos de autenticação e comunicação (e.g., *web services*) necessários para interagir com provedores de contexto externos, tais como Serviços de Agenda, Rede Social, *Geocoding*.

Os *ConUnits* são implementados em *bundles OSGi*², como ilustra a Figura 3.2. Dessa maneira, o ConBus utiliza as facilidades do *OSGi* para instalar, remover, ativar e desativar dinamicamente *ConUnits* conforme a demanda das aplicações interessadas em suas informações contextuais.

Conforme ilustrado pela Figura 3.3, um *ConUnit* é constituído por duas partes: uma representando a parte do *ConUnit* que é dependente de um sensor específico (*Sensor Integration*), composta pelo módulo de sensoriamento (*Sensor Module*) propriamente dito e pelo *Componente de Integração com o Sensor* (*Sensor Integration Component*) que implementa a interface *ContextDataConversion*. A implementação dessa interface permite ao desenvolvedor converter os dados coletados em um formato específico por cada sensor em um formato padronizado de acordo com a modelagem de contexto, discutida na Seção 3.3. Essa conversão é necessária porque permite que os outros componentes da arquitetura manipulem, de forma uniforme, os dados de contexto coletados por qualquer sensor acoplado.

Na parte do *ConUnit* independente de sensor (*ConBus Integration*), o *ConUnit*

¹Por exemplo, no sistema operacional Android, muitas vezes não é possível implementar um módulo de sensoriamento dentro de um *ConUnit*, porque, para obter certas informações do próprio dispositivo móvel, é necessário utilizar classes oferecidas pelo Android que não podem ser executadas dentro do *OSGi*.

²Um *bundle OSGi* é a menor unidade executável em um container *OSGi*. Cada *bundle*, nessa arquitetura, é um componente modularizado cuja comunicação com os outros componentes do sistema se dá através de interfaces bem definidas e/ou serviços, controlados pelo *framework OSGi*.

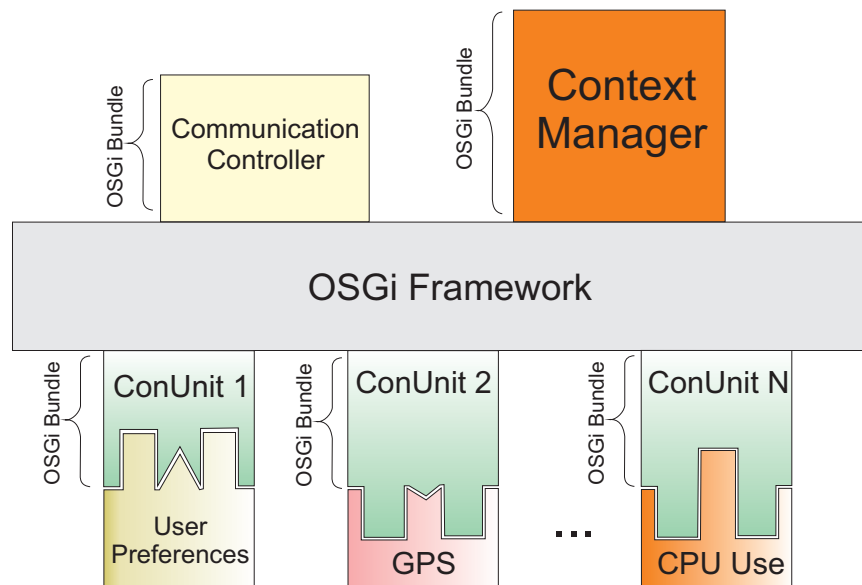


Figura 3.2: Integração de ConUnits e outros componentes da arquitetura ConBus ao framework OSGi.

Manager é responsável por gerenciar todo o acesso aos dados providos pela fonte de contexto à qual o *ConUnit* está associado. Por exemplo, quando uma aplicação envia uma requisição ao ConBus desejando obter uma consulta a alguma informação contextual gerenciada por ele, essa aplicação deve informar o tipo de contexto desejado, à qual entidade (pessoa, local, objeto, etc.) essa informação está relacionada e, opcionalmente, o sensor propriamente dito (por exemplo, receptor GPS ou PlaceLab para coletar dados sobre a localização de um usuário). Ao chegar ao *Context Manager*, essa requisição é repassada ao *ConUnit Manager* da instância do *ConUnit* que faz a integração com a fonte de contexto informada, conforme descrito na Seção 4.7. Em seguida, o *ConUnit Manager* faz uma requisição chamando o método apropriado do *Sensor Integration Component* e aguarda uma resposta, que será representada segundo o modelo de contexto do ConBus (ver Seção 3.3). O componente *Synchronous Controller*, também presente na *ConBus Integration* de cada *ConUnit*, apenas controla os detalhes de comunicação entre o *Gerenciador de Contexto* e o *ConUnit* correspondente, como o registro da interface de comunicação entre o *ConUnit* e o *Gerenciador de Contexto* junto ao OSGi toda vez que o *ConUnit* for ativado, para que, posteriormente, eles possam se comunicar.

Pelo fato dos *ConUnits* serem elementos essenciais na arquitetura ConBus, as Seções 4.7 e 4.8 descrevem, detalhadamente, o processo de desenvolvimento e uso de um novo *ConUnit*, bem como a reutilização de *ConUnits* desenvolvidos por terceiros em

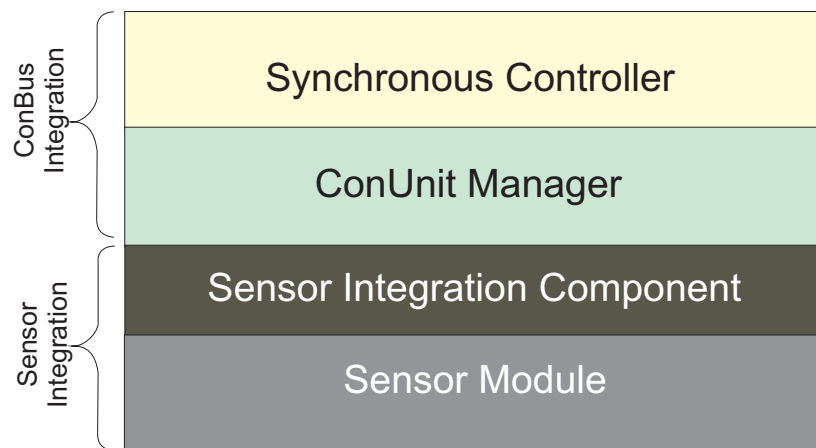


Figura 3.3: Arquitetura de um *ConUnit*.

novas aplicações sensíveis ao contexto.

3.2.2 Gerenciador de Contexto

O *Gerenciador de Contexto* (*Context Manager*) é o componente do *middleware* ConBus que, de fato, atua como intermediário entre as aplicações sensíveis ao contexto e os *ConUnits*. Ele é o responsável pelo gerenciamento do ciclo de vida de cada *ConUnit* acoplado ao *middleware*. O ciclo de vida de um *ConUnit* é constituído pelas seguintes etapas: instalação, ativação, desativação e desinstalação de cada instância desse *framework*. Por isso, as tarefas do *Gerenciador de Contexto* consistem em instalar, ativar, desativar e desinstalar dinamicamente *ConUnits*, bem como implementar o acesso síncrono e assíncrono às informações contextuais providas por essas fontes de contexto.

O *Gerenciador de Contexto* deve ativar dinamicamente, através do *framework* *OSGi*, os *ConUnits* quando os dados fornecidos por eles forem requisitados pelas aplicações, desativando-os, também em tempo de execução, sempre que nenhuma aplicação ou outro componente da arquitetura estiver interessado em suas informações contextuais. Essa função satisfaz o requisito sobre ativação e desativação dinâmicas de módulos de sensoriamento, discutido na Seção 2.2. Para ser ativado, cada *ConUnit* deve ser primeiramente identificado. Isso é feito através de metadados contidos no próprio *ConUnit*, como o tipo de contexto (i.e., localização, temperatura, dados da agenda de compromissos, uso de CPU, etc.) provido por ele, a entidade (por exemplo, LOJA_101) à qual essa informação contextual está associada e, opcionalmente, o tipo de sensor responsável pela coleta dos dados contextuais.

Quando o *middleware* ConBus é iniciado, todos os *ConUnits* acoplados (previamente instalados) a ele encontram-se desativados. Cada módulo é carregado dinamicamente assim que recebe, pela primeira vez, uma requisição (síncrona ou assíncrona)

de uma aplicação através do *Gerenciador de Contexto*. Essa funcionalidade visa gerenciar melhor os recursos computacionais do dispositivo móvel, principalmente memória. A desativação acontece depois de transcorrido um certo intervalo de tempo, que pode ser configurado pelo usuário do *middleware*, sem que nenhuma aplicação tenha requisitado dados provenientes desse *ConUnit*. Essa abordagem é bastante simplista, pois não leva em conta questões, como os custos envolvidos para ativar e desativar tais módulos, além de não analisar as reais necessidades que o dispositivo móvel tem de poupar recursos computacionais a cada momento. Outros trabalhos usam estratégias diferentes e mais complexas para gerenciar dinamicamente o ciclo de vida de componentes/recursos, tanto em plataformas móveis [14] quanto em sistemas de *middleware* para outras áreas da computação [19].

Além de ativar e desativar *ConUnits*, o *Gerenciador de Contexto* também implementa o acesso síncrono e assíncrono às informações contextuais providas por esses componentes, através de seus módulos de sensoriamento. No caso síncrono, o *Gerenciador de Contexto* encaminha cada requisição feita por uma aplicação ao respectivo *ConUnit*, devolvendo a resposta à aplicação correspondente. Para o caso assíncrono, as aplicações inicialmente devem registrar interesse (*subscription*) em receber uma determinada informação contextual sempre que uma condição desejada for satisfeita. Após o registro, o *Gerenciador de Contexto* ficará acessando, em intervalos de tempo determinados comum a todos os *ConUnits*, o respectivo *ConUnit*, comparando os valores coletados com os registros (*subscriptions*) feitos pelas aplicações sensíveis ao contexto. As interfaces de comunicação síncrona e assíncrona são discutidas na Seção 4.6.

Caso haja mais de um sensor/módulo de sensoriamento acoplado que coleta uma mesma informação contextual associada a uma mesma entidade, o *Gerenciador de Contexto* irá escolher o primeiro que encontrar. Esse algoritmo é muito simples e deverá ser melhorado em trabalhos futuros, visando principalmente melhorar, quando possível, a qualidade (precisão, confiabilidade, etc) das informações disponibilizadas pelo ConBus às aplicações sensíveis ao contexto.

Através da combinação dos *frameworks OSGi* e *ConUnit*, os módulos de sensoriamento desenvolvidos para serem utilizados pelo ConBus podem ser criados atendendo aos requisitos citados na Seção 2.2 relacionados à criação de módulos coesos, fracamente acoplados a outros componentes e cuja comunicação com o meio externo seja realizada apenas através de interfaces bem definidas e/ou serviços.

3.2.3 Controlador de Comunicações

O *Controlador de Comunicações* (*Communication Controller*), conforme ilustrado na Figura 3.1, é o componente da arquitetura responsável por receber requisições

das aplicações e enviar às aplicações as respostas correspondentes. Sua função é receber as requisições, extrair as informações necessárias (comando enviado, bem como os seus parâmetros, quando houverem), verificando possíveis erros na mensagem, enviando, em seguida, ao *Gerenciador de Contexto*, os dados necessários para que este possa processar a resposta que será enviada à aplicação que solicitou a informação.

3.3 Modelagem de Contexto

Muitos conceitos do modelo de contexto adotado pela arquitetura ConBus foram baseados em [9]. Conforme ilustrado na Figura 3.4, para o ConBus, a classe *Context* representa toda informação de contexto gerenciada pelo *middleware*. Ela é constituída por uma ou mais Informações de Contexto (*ContextInformation*). Cada *ContextInformation* possui um e apenas um Tipo de Contexto (*ContextType*). Uma *ContextInformation* contém todos os dados e metadados relativos a um tipo de contexto coletado pelo sensor correspondente, tais como valores, unidades, precisão dos dados, data e hora da coleta.

Entretanto, deve-se observar que o modelo implementado, de fato, não é orientado a objetos. Ele é essencialmente um modelo par/valor. O diagrama da Figura 3.4 deve ser entendido apenas como um modelo conceitual, capaz de ilustrar as relações entre os diversos componentes do modelo e não como um modelo de implementação, como empregado em [9]. Isso foi feito porque o modelo par/valor supre, de forma simples, as necessidades essenciais requeridas pela modelagem de contexto do ConBus. Além disso, o uso de técnicas mais complexas para a modelagem de contexto não é o foco principal deste trabalho.

Para o ConBus, cada informação de contexto (*ContextInformation*) deve estar relacionada a uma única entidade (*Entity*). Uma entidade pode ser, por exemplo, uma pessoa (*person*), um objeto (*object*), um local (*place*), etc.

Por exemplo, imagine uma situação em que a informação de localização de um usuário, conhecido pelo sistema, como BOB, esteja sendo coletada por um receptor GPS e enviada ao ConBus, através de um *ConUnit* correspondente. Nesse caso, essa informação contextual poderia ser representada, no modelo de contexto empregado pelo ConBus, da maneira ilustrada pelo diagrama de objetos UML, exibido na Figura 3.5. Nela vemos que há um objeto da classe *Entity* para representar a entidade BOB, que é uma pessoa. Além disso, todos os outros metadados, como tipo de sensor, data e hora em que os dados sobre a localização estão sendo coletados, precisão, etc., são armazenados por um objeto da classe *ContextInformation*. Apenas os dados sobre a localização propriamente dita (valores das coordenadas geográficas e seus metadados) são armazenados em objeto especial, pertencente à classe *ContextData*, responsável por armazenar, de forma flexível, os dados enviados pelos sensores adjacentes.

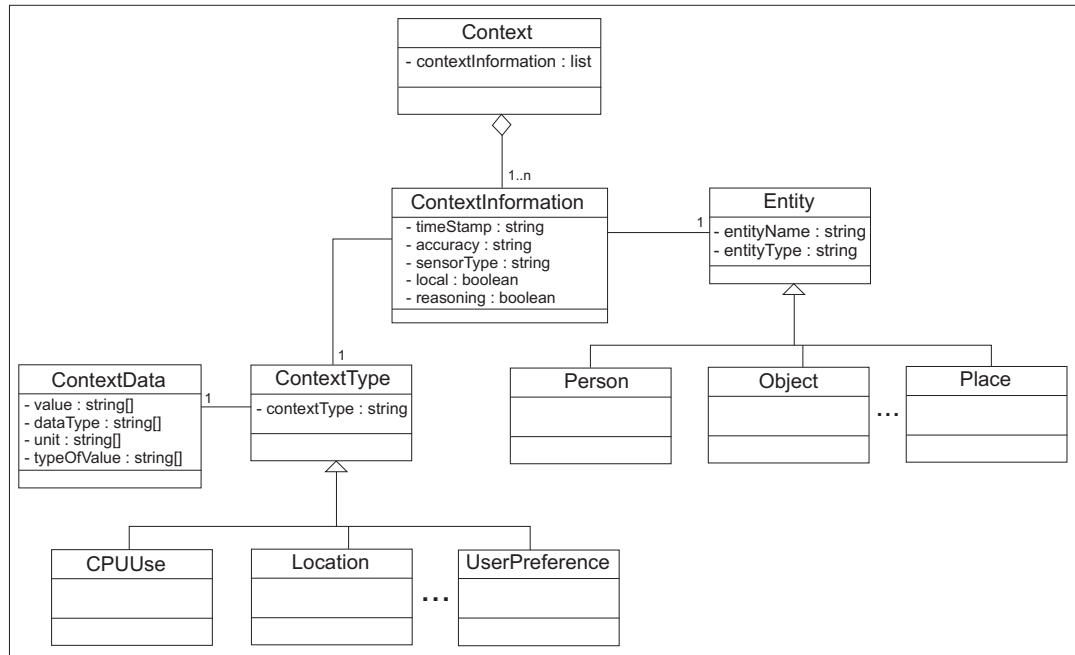


Figura 3.4: Modelo de Contexto.

A flexibilidade da classe *ContextData* está relacionada com o fato de permitir que vários tipos de valores possam ser armazenados por ela. Por exemplo, um objeto *ContextData* pode tanto armazenar uma informação de contexto do tipo LOCALIZAÇÃO, contendo três variáveis diferentes (latitude, longitude e altitude), como pode armazenar também uma informação de contexto do tipo TEMPERATURA, que necessita apenas de uma única variável para representá-la.

A modelagem é baseada no modelo atributo/valor e as expressões usadas para analisar condições sobre esses valores são equivalentes às oferecidas pelo *middleware* MoCA [16].

Para criar um *ConUnit*, o desenvolvedor deverá representar os dados coletados pelo sensor correspondente usando esse modelo de contexto discutido nesta Seção. Exemplos de implementação de *ConUnits* usando o modelo proposto são descritos na Seção 4.7.

As Tabelas 3.1, 3.2 e 3.3 adicionam aos problemas e requisitos, resumidos pelas Tabelas 2.1, 2.2 e 2.3, as soluções implementadas pela arquitetura ConBus para satisfazer as necessidades e desafios relativos à integração (*desenvolvimento, implantação e gerenciamento do ciclo de vida*) dos módulos de sensoramento.

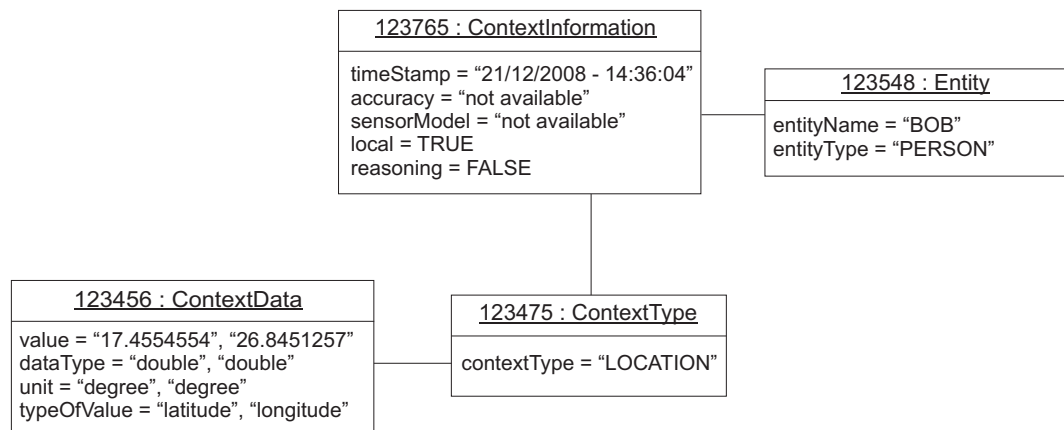


Figura 3.5: Um exemplo de uso do modelo de contexto da arquitetura ConBus.

Tabela 3.1: Problemas, requisitos e soluções implementadas pelo ConBus relativos ao desenvolvimento e reutilização de módulos de sensoriamento.

Problemas	Requisitos	Solução (ConBus)
Complexidade inerente ao desenvolvimento dos módulos de sensoriamento	Mecanismos para tratar problemas de comunicação entre sensores e módulos.	Não implementado!
	Mecanismos para facilitar a conversão dos dados específicos de sensor para um formato padronizado.	Interface <i>ContextDataConversion</i> e Modelagem de Contexto do ConBus.
	Mecanismos para a criação de módulos coesos, fracamente acoplados e encapsulados.	<i>framework ConUnit</i> e <i>framework OSGi</i> .
Seleção e uso de módulos de sensoriamento	Metadados para facilitar a escolha do sensor mais adequado às necessidades das aplicações a cada instante, como a precisão do sensor e outros parâmetros de QoS.	<i>framework OSGi</i> e <i>Gerenciador de Contexto</i> .
	Algoritmos e/ou heurísticas para a seleção de módulos quando houver mais de um fornecendo o mesmo tipo de contexto para uma mesma entidade.	Algoritmo simples: escolha do primeiro módulo da lista que satisfaça às necessidades da aplicação.
Dificuldade de reutilização	Repositórios de módulos de sensoriamento.	Não implementado!
	Facilidades para documentar e exibir documentação de módulos de sensoriamento.	<i>framework ConUnit</i> e <i>API das Aplicações</i> .

Tabela 3.2: Problemas, requisitos e soluções implementadas pelo ConBus relativos à implantação e localização de módulos de sensoriamento.

Problemas	Requisitos	Solução (ConBus)
Problemas relativos à implantação de módulos de sensoriamento	Mecanismos para possibilitar a implantação dinâmica desses módulos.	<i>Gerenciador de Contexto</i> , <i>framework ConUnit</i> e <i>framework OSGi</i> .
Transparência de localização	Abstrações para permitir transparência de localização tanto de sensores quanto dos módulos de sensoriamento.	Modelagem de Contexto do ConBus e <i>API das Aplicações</i> .
	Metadados que informem, sempre que necessário, a localização do sensor/módulo (local ou remoto), tipo e modelo.	Modelagem de Contexto do ConBus e <i>API das Aplicações</i> .

Tabela 3.3: *Problemas, requisitos e soluções implementadas pelo ConBus relativos ao gerenciamento do ciclo de vida de módulos de sensoriamento.*

Problemas	Requisitos	Solução (ConBus)
Problemas relativos à ativação, atualização e desativação de módulos de sensoriamento	Mecanismos para ativar e desativar dinamicamente os módulos de sensoriamento.	<i>Gerenciador de Contexto e framework OSGi.</i>
	Mecanismos para atualizações dinâmicas e automatizadas.	Não implementado!

Implementação

Este Capítulo trata dos aspectos relacionados à implementação do *middleware* ConBus, cuja arquitetura foi descrita no Capítulo 3. Por isso, são analisados aqui o funcionamento das principais estruturas (classes Java, nesse caso) de que é constituído cada componente do ConBus, bem como os relacionamentos entre classes de um mesmo componente ou, ainda, entre classes de componentes distintos.

4.1 Visão Geral

O *middleware* ConBus foi implementado em Java, mais especificamente Java 2 Platform Standard Edition 6 (J2SE 6). O sistema operacional escolhido foi o Android (SDK 1.5)¹, empregado como ambiente de execução. Porém, deve-se ressaltar que o ConBus não é executado diretamente sobre o Android, mas sobre o *framework* OSGi, mais especificamente sobre o *Apache Felix 1.8*², uma implementação da especificação OSGi R4 Service Platform que, por sua vez, pode ser executado por esse sistema operacional para dispositivos móveis.

De fato, o ConBus é executado como um conjunto de componentes/módulos (*bundles*, na terminologia OSGi) gerenciados pelo *Apache Felix OSGi* que, por conseguinte, pode ser executado nos dispositivos móveis com suporte a Java 6, como o sistema operacional Android. Internamente, a implementação do ConBus está organizada em vários pacotes Java, cada um contendo as classes específicas para cada componente do *middleware*. Além de cada pacote específico, representando cada componente do ConBus, há também o pacote *common*, onde estão todas as classes de uso comum a todas as outras classes de todos os outros pacotes da implementação. O restante deste Capítulo discute os principais aspectos relativos à implementação de cada componente do ConBus (*ConUnits*, *Gerenciador de Contexto*, *Controlador de Comunicações*) e a integração deles

¹<http://www.android.com/>

²<http://felix.apache.org/>

com o *framework OSGi*. A Figura 4.1 ilustra todos os componentes do ConBus e a forma como se comunicam.

A comunicação entre as aplicações sensíveis ao contexto e o *middleware* ConBus acontece através de um *webservice* local, empregando o protocolo de comunicação HTTP em conjunto com arquivos de resposta codificados em XML, conforme discutido na Seção 4.6. O uso de um *webservice* permite que aplicações desenvolvidas em diferentes linguagens possam interagir com o ConBus e obter os dados contextuais de que necessitam. Por exemplo, no sistema operacional Android, tanto as aplicações desenvolvidas em Java quanto em C++ podem utilizar os serviços de provisão de contexto oferecidos pelo ConBus. Internamente, a comunicação entre os componentes do *middleware* ConBus ocorre através de referências diretas, que são obtidas por intermédio do *framework OSGi*. A comunicação entre os *ConUnits* e seus sensores correspondentes pode acontecer de várias formas, como através de referência direta via OSGi, chamada local ou uso de algum protocolo proprietário específico.

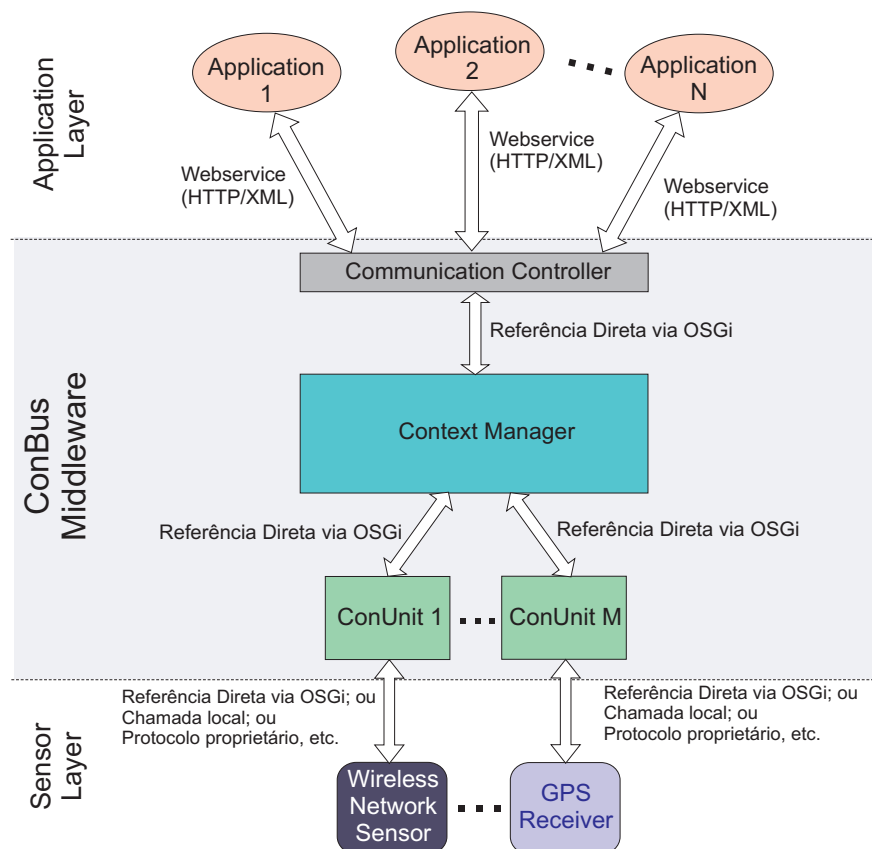


Figura 4.1: Os componentes da arquitetura ConBus e como se comunicam entre si.

4.2 ConUnits

As Figuras 4.2 e 4.3 descrevem a estrutura interna de um *ConUnit*. A diferença fundamental entre as duas figuras é o fato de uma apresentar o módulo de sensoramento implementado dentro do *ConUnit* e a outra apresentá-lo implementado externamente ao *ConUnit* correspondente, conforme discutido na Seção 3.2.1. O *Módulo de sensoramento* (*Sensor Module*) implementa a recuperação dos dados coletados pelo sensor correspondente. Esses dados são convertidos para o modelo de contexto usado pelo ConBus, através da implementação da interface *ContextDataConversion* (ver Figura 4.4), pelo desenvolvedor do *ConUnit*, dentro do *Componente de Integração do Sensor* (*Sensor Integration Component*). O *ConUnitManager* é o responsável pelo acesso direto às informações contextuais já padronizadas fornecidas pela implementação da interface *ContextDataConversion*. Todos os métodos que constituem essa interface são descritos na Figura 4.4. Essa interface fornece os métodos necessários para acessar todos os dados propriamente ditos e os metadados de um *ConUnit*, como o tipo de contexto provido, o instante em que a informação contextual foi coletada, o modelo e a precisão do sensor correspondente e a entidade à qual esse dado de contexto está associado.

Conforme ilustrado nas Figuras 4.2 e 4.3, todo *ConUnit* deve exportar a interface *ConUnitSensorAPI*, com a qual o *Gerenciador de Contexto* pode recuperar as informações de contexto publicadas por esse *ConUnit*. Isso ocorre porque, dentro do *framework OSGi*, um *bundle* só consegue acesso às informações de um outro *bundle* se este, ao ser iniciado, registrar junto ao OSGi, uma determinada interface de acesso. Dessa forma, a partir desse registro, um outro *bundle* poderá obter acesso aos dados providos por este, através dos métodos fornecidos por sua interface. Esse registro é feito durante a invocação do método *start* da classe *ConUnitActivator* que implementa a interface *BundleActivator* fornecida pela implementação OSGi. O método *start* é invocado pelo *framework OSGi* toda vez que o *ConUnit* é iniciado.

Através da tupla (tipo de contexto, nome da entidade e tipo da entidade) é possível identificar *ConUnits* junto ao *middleware* de forma a recuperar os dados providos por esses componentes. O *tipo de contexto*, conforme descrito pelo modelo de contexto, indica o tipo básico de contexto provido pelo *ConUnit* correspondente, como localização, temperatura, uso de CPU, dados da agenda de compromissos. O *nome* e o *tipo da entidade* identificam a entidade à qual a informação contextual está associada. O nome e o tipo de uma entidade podem ser, por exemplo, “SALA113” (tipo da entidade = LUGAR), “PAULO_SILVA” (tipo da entidade = PESSOA), “G1_ANDROID_DEVICE” (tipo da entidade = OBJETO).

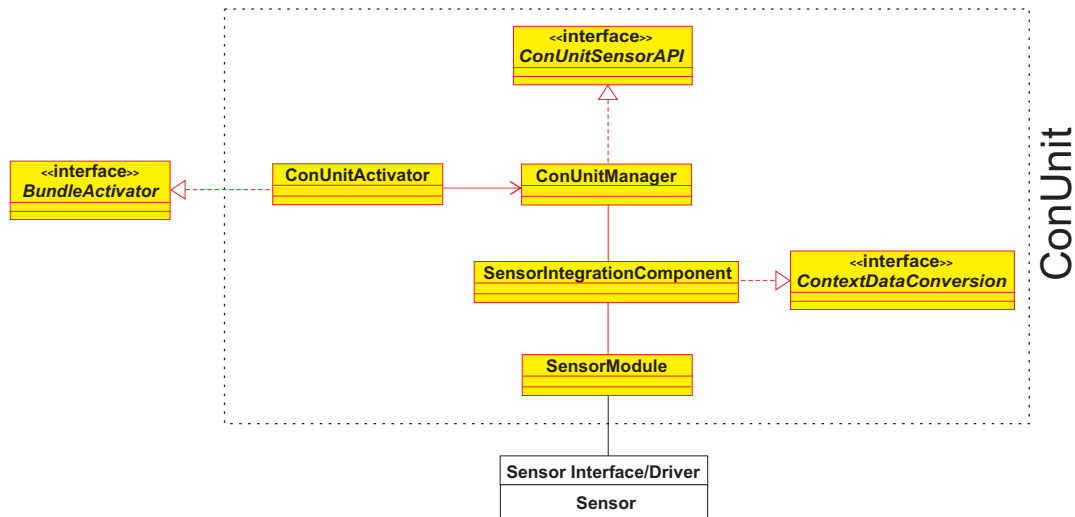


Figura 4.2: Diagrama Arquitetural de um ConUnit, com o módulo de sensoriamento interno.

4.3 Gerenciador de Contexto

Quando o *middleware* é iniciado, todos os *ConUnits* estão desativados. O processo de ativação de um *ConUnit* acontece sempre que uma aplicação requisita uma informação contextual provida por um *ConUnit* desativado. A desativação acontece depois de transcorrido um certo intervalo de tempo, que pode ser configurado pelo usuário do *middleware*, sem que nenhuma aplicação tenha requisitado dados provenientes desse *ConUnit*. Após sua desativação, um *ConUnit* poderá ser ativado a qualquer momento sempre que uma aplicação necessitar obter informações contextuais providas por ele.

Como ilustrado pela Figura 4.5, requisições síncronas são tratadas pelo *Gerenciador de Contexto* quando uma aplicação envia uma requisição HTTP com o comando *getContext*, oferecido pela *API das Aplicações*, ao *middleware ConBus*. Ao receber essa requisição, o *Controlador de Comunicações* a envia ao *Gerenciador de Contexto*, onde é processada. Caso não exista erro na requisição e exista algum *ConUnit* provendo a informação solicitada, essa requisição será enviada ao *ConUnit* correspondente. Ao receber a resposta do *ConUnit*, o *Gerenciador de Contexto* a envia ao *Controlador de Comunicações* para ser encapsulada em um *arquivo XML* que deverá ser enviado à aplicação solicitante.

A comunicação assíncrona começa com uma aplicação registrando interesse, junto ao *middleware ConBus*, em receber uma determinada informação contextual sempre que uma condição específica for satisfeita. A partir desse momento, o *Gerenciador de Contexto* passa a acessar o *ConUnit* correspondente, em intervalos de tempo regulares, analisando os dados recebidos, verificando se a condição desejada pela aplicação é satisfeita. Toda vez que isso acontecer, o *Gerenciador de Contexto* irá enviar uma resposta

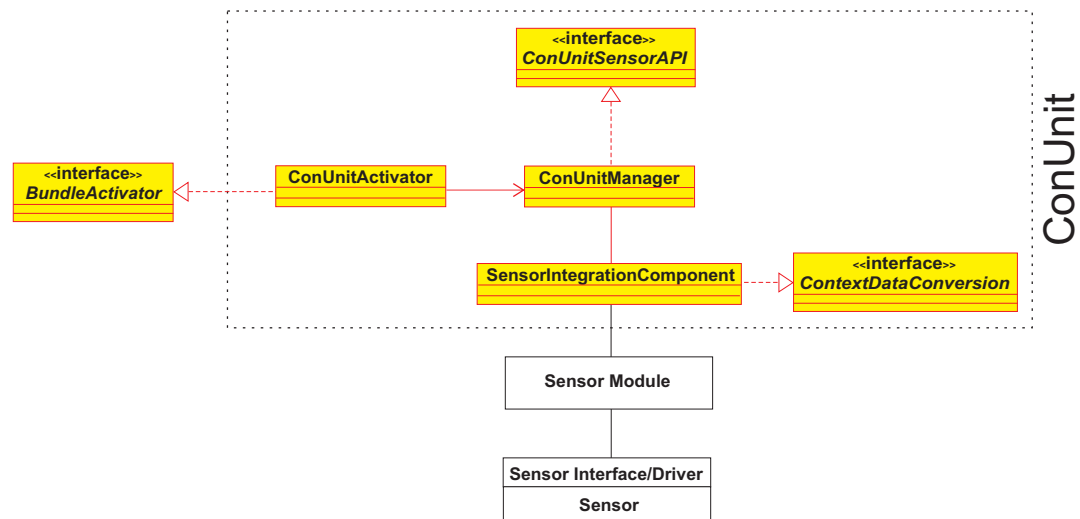


Figura 4.3: Diagrama Arquitetural de um ConUnit, com o módulo de sensoriamento externo.

à aplicação, através do endereço de *end point* informado no registro (*subscription*) feito por essa aplicação. Esse processo continuará até que a aplicação retire o registro de interesse feito anteriormente, conforme indicado pela Figura 4.6. A API das Aplicações é discutida na Seção 4.6.

4.4 Bundles OSGi

O *Framework OSGi* é o componente da arquitetura ConBus que permite o acoplamento, estático ou dinâmico, de todas as fontes de contexto e de todos os outros componentes internos junto ao *middleware*. Por essa razão, este é considerado um componente central para a implementação da arquitetura. Para a versão corrente do ConBus, foi utilizada a implementação Apache Felix 1.8, por ser uma implementação da especificação OSGi R4 Service Platform, contemplando todos os recursos necessários ao desenvolvimento e execução dos demais componentes do *middleware*, além de ser uma das implementações R4 mais compactas³, o que é sempre desejável em ambientes móveis.

Para o OSGi, cada componente é considerado um *bundle*. Um *bundle* possui todas as características de um módulo, como alta coesão, fraco acoplamento, encapsulamento e interfaces de acesso bem definidas. Além disso, cada *bundle* é representado fisicamente por um arquivo JAR. Todo *bundle* possui uma classe Java que implementa a interface *BundleActivator*, por onde o *framework* ativa, desativa e atualiza qualquer *bundle*. Atualmente, a implementação do ConBus é composta pelos seguintes *bundles*: *context_manager.jar*, *comm_controller.jar*, *common.jar* (contendo as classes e interfaces de

³O arquivo felix.jar tem cerca de 515 KB!

```

public interface ContextDataConversion {
    public Date getTimeStamp();
    public String getContextType();
    public String getAccuracy();
    public String getSensorModel();
    public ContextData getData();
    public boolean isLocal();
    public String getEntityName();
    public String getEntityType();
    public boolean isReasoning();
    public ContextInformation getContextInformation();
}

```

Figura 4.4: Interface *ContextDataConversion*

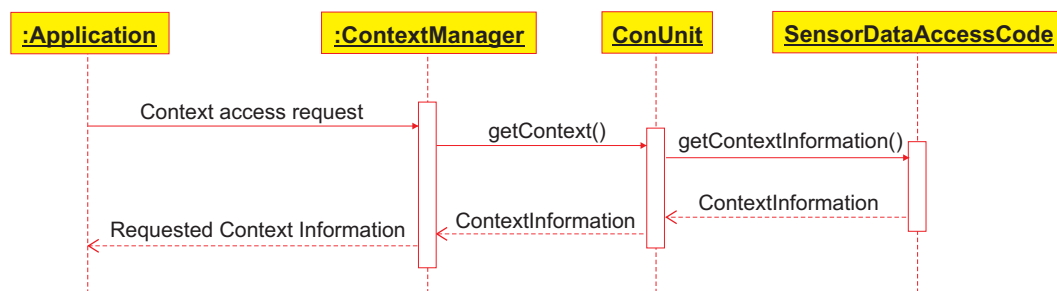


Figura 4.5: Funcionamento da comunicação síncrona entre aplicações sensíveis ao contexto e ConBus.

uso comum a todos os outros *bundles*, como **ContextInformation**, **ContextData**, **Context** e **ContextDataConversion**), além dos *bundles* que implementam as fontes de contexto (*ConUnits*).

Da forma como um *bundle* é criado, ele não pode ser executado pelo OSGi dentro do sistema operacional Android. Por isso, algumas modificações no arquivo JAR inicial devem ser realizadas. Essas modificações visam criar, dentro do JAR correspondente, código executável (*.dex*) para o ambiente Android, pois apesar de se empregar a linguagem Java para desenvolver aplicações para esse sistema operacional, o código compilado disponibilizado pela compilação dessa linguagem (*bytecodes*) não pode ser executado diretamente pelo Android. Felizmente, para essa conversão são disponibilizadas ferramentas apropriadas, facilitando esse processo. Para facilitar ainda mais o processo de criação de *ConUnits*, a implementação do ConBus disponibiliza um pequeno software denominado *ConUnit Builder*, responsável pela criação de *ConUnits* (veja a Seção 4.7). Além disso, esse software faz uso das ferramentas apropriadas disponibilizadas pela implementação

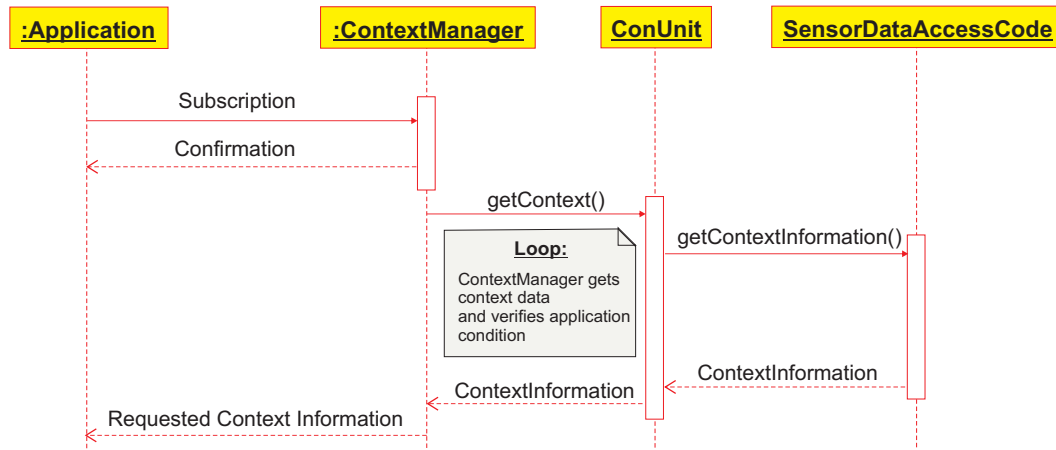


Figura 4.6: Funcionamento da comunicação assíncrona entre aplicações sensíveis ao contexto e ConBus.

Android, de modo que todo *ConUnit* desenvolvido com essa ferramenta está pronto para ser executado no ambiente Android.

É importante notar que também o próprio JAR que representa o *framework OSGi* deve ser modificado para incorporar código executável (.dex) e assim viabilizar sua execução no sistema operacional Android.

Uma outra importante característica de um *bundle OSGi* é que ele permite que ações sejam executadas antes de parar sua execução, através do método *stop*, oferecido pela classe de ativação e desativação (**ConUnitActivator**) próprio *ConUnit*. Por exemplo, no caso de *ConUnits*, é permitido que os módulos de sensoramento incorporados por esses *ConUnits* enviem comandos aos sensores correspondentes e parem sua execução enquanto esses módulos estiverem desativados. Isso auxilia no gerenciamento dos recursos computacionais dos dispositivos móveis, especialmente energia.

4.5 Controlador de Comunicações

A principal classe presente neste componente é a **ConBusServer**, que controla um servidor HTTP, que por sua vez aguarda, em uma dada porta TCP, requisições oriundas das aplicações. Assim que uma requisição chega ao **ConBusServer**, o cabeçalho da mensagem é retirado, analisando possíveis erros de comunicação e enviando, em seguida, a mensagem propriamente dita à classe **ConBusProtocol**, responsável por seu tratamento. A classe **ConBusProtocol** identifica o comando enviado pela aplicação, juntamente com seus parâmetros necessários e invoca o método adequado da classe **CommController**. É a classe **CommController** que tem acesso às classes do componente *Gerenciador de Contexto*, responsável pelo tratamento, de fato, das requisições feitas pelas aplicações sensíveis ao contexto.

A classe **CommControllerActivator** também implementa a interface *BundleActivator* e, por isso, tem a mesma função, neste componente, que a classe **ConUnitActivator** para o *ConUnit*, conforme discutido na Seção 4.2.

4.6 Comunicação entre Aplicações e ConBus

Para obter as informações de contexto providas pelo ConBus, uma aplicação necessita enviar requisições HTTP informando, na própria URL, no mínimo, o tipo de comando (i.e. *getContext*, *getAllContext*, etc.) e o tipo de contexto desejado (LOCATION, ENERGY_LEVEL, TEMPERATURE, etc), assim como, o nome e o tipo da entidade à qual esta informação de contexto está relacionada (i.e. *entityName*=JOHN_SMITH e *entityType*=PERSON). A seguir, são exibidos exemplos de uso do protocolo HTTP para o envio de requisições entre as aplicações e o ConBus.

```
http://localhost:8585/  
getContext?contextType=LOCATION&entityName=GABRIEL_SANTOS&entityType=PERSON
```

Neste exemplo, uma aplicação móvel sensível ao contexto envia à instância local do *middleware* ConBus (representada pelo endereço IP 127.0.0.1 ou *localhost* e pela porta TCP 8585, que é a porta padrão do ConBus), o comando *getContext* para obter informações sobre a localização corrente de uma pessoa registrada no sistema como GABRIEL_SANTOS.

No exemplo a seguir, uma aplicação ou o próprio desenvolvedor deseja obter informações sobre todos os *ConUnits* instalados e iniciados no ConBus e, por conseguinte, sobre todos os sensores acoplados ao *middleware* naquele instante. Essas informações são importantes, por exemplo, quando o desenvolvedor deseja saber dados para acesso a novos módulos/sensores previamente instalados e ativos no *middleware*, como a entidade à qual cada sensor está relacionado, o tipo de contexto oferecido, etc.

```
http://localhost:8585/getAllContext
```

Há duas formas básicas de comunicação possíveis disponibilizadas pela *API das aplicações*: (i) através de *interface síncrona*, em que é possível consultar qualquer informação contextual gerenciada pelo ConBus e coletada pelos sensores acoplados; ou ainda, (ii) por meio de *interface assíncrona*, baseada em eventos, em que se pode registrar interesse em obter determinado tipo de informação de contexto sempre que condições específicas ocorrerem.

Uma breve descrição de cada comando oferecido pela *API das Aplicações* é feita a seguir. Entretanto, uma visão detalhada desta API é disponibilizada no Apêndice A. A Figura 4.7 exibe um diagrama ilustrando o uso de cada comando e suas respectivas respostas.

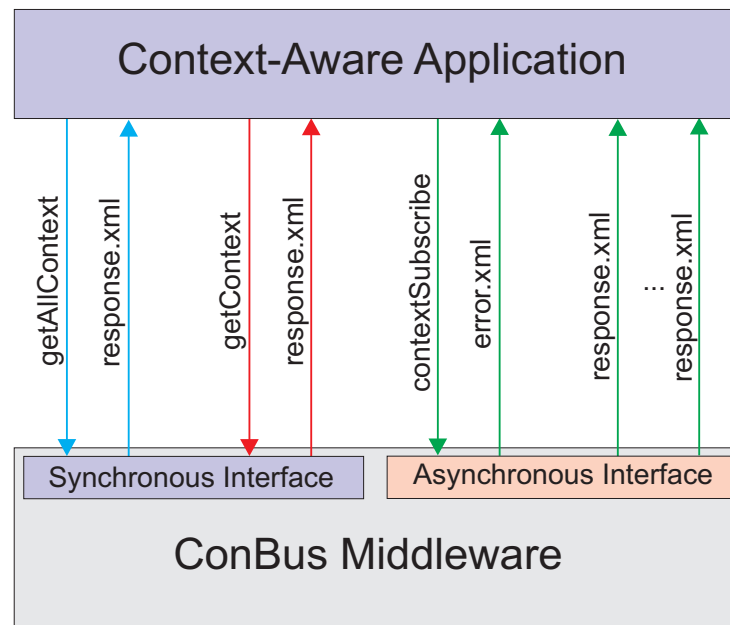


Figura 4.7: Interação entre aplicações e middleware ConBus, através dos comandos da API das Aplicações.

getAllContext Através deste comando, tanto as aplicações quanto os próprios desenvolvedores de aplicações sensíveis ao contexto podem obter informações sobre todas as fontes de contexto (*ConUnits*) acopladas ao ConBus em um dado instante. Isso é importante, por exemplo, quando se deseja utilizar um módulo de sensoriamento desenvolvido por terceiros e não se tem documentação sobre seu uso, como o tipo de contexto coletado, a entidade à qual esse módulo está associado ou se o sensor está local ou remoto ao dispositivo. Para ter ciência de quais módulos estão acoplados ao ConBus e saber como enviar requisições a eles, o desenvolvedor pode enviar o seguinte comando, através de um navegador *Web*:

```
http://localhost:8585/getAllContext
```

Como se pode notar, não há parâmetros de entrada para o comando *getAllContext*. A resposta a esse comando é enviada, pelo ConBus, através do arquivo *response.xml*, mostrado a seguir:

```
<?xml version="1.0"?>
<response>
  <error>
    <errorType>NONE</errorType>
    <errorCode>000</errorCode>
  </error>
  <contextSource>
    <number>00001</number>
    <contextType>LOCATION</contextType>
    <entityName>JOHN_SMITH</entityName>
```

```

    <entityType>PERSON</entityType>
    <isLocal>TRUE</isLocal>
    .
    .
</contextSource>
<contextSource>
    <number>00002</number>
    <contextType>TEMPERATURE</contextType>
    <entityName>ROOM_111</entityName>
    <entityType>PLACE</entityType>
    <isLocal>FALSE</isLocal>
    .
    .
</contextSource>
.
.
</response>

```

getContext Este comando permite às aplicações consultarem, em tempo real, informações de contexto específicas gerenciadas pelo ConBus. Um exemplo de uso é apresentado a seguir:

```

http://localhost:8585/
    getContext?contextType=CPU_USE&entityName=DEV_UFG_001&entityType=OBJECT

```

O primeiro parâmetro indica o tipo de contexto desejado (uso de CPU), o segundo parâmetro indica o nome da entidade à qual esta informação contextual está relacionada (neste caso, um dispositivo móvel denominado DEV_UFG_001) e, finalmente, o terceiro parâmetro mostra que esta entidade é um objeto. Opcionalmente, pode-se indicar também, como um quarto parâmetro, o sensor específico do qual se deseja obter a informação contextual correspondente. Esse fato permite que desenvolvedores e aplicações sensíveis ao contexto escolham se desejam usar a transparência de localização e de sensor oferecida pela plataforma de *middleware* ou se é necessário o uso de um determinado sensor para a coleta dos dados de contexto desejados. Essa possibilidade de escolha permite ao ConBus satisfazer aos requisitos relativos à transparência de localização discutidos na Seção 2.2. A resposta para esse comando é enviada, pelo ConBus, através do arquivo *response.xml*, exemplificado a seguir. Esse arquivo reflete o modelo de contexto empregado pela arquitetura ConBus, discutido na Seção 3.3.

```

<?xml version="1.0"?>
<response>
    <error>
        <errorType>NONE</errorType>
        <errorCode>000</errorCode>
    
```

```

</error>
<contextType>TEMPERATURE</contextType>
<entityName>ROOM_111</entityName>
<entityType>PLACE</entityType>
<accuracy>NOT AVAILABLE</accuracy>
<sensorModel>NOT AVAILABLE</sensorModel>
<reasoning>FALSE</reasoning>
<isLocal>FALSE</isLocal>
<isRemote>TRUE</isRemote>
<contextData>
  <value>
    <value1>27.4534</value1>
  </value>
  <typeOfValue>
    <typeOfValue1>TEMPERATURE</typeOfValue1>
  </typeOfValue>
  <unit>
    <unit1>CELSIUS</unit1>
  </unit>
  <dataType>
    <dataType1>FLOAT</dataType1>
  </dataType>
</contextData>
</response>

```

contextSubscribe Através deste comando, uma aplicação pode registrar interesse em receber determinadas informações contextuais sempre que certos eventos ocorrerem. Além disso, toda vez que o evento desejado acontecer, as aplicações receberão um arquivo *response.xml*, semelhante ao enviado em resposta ao comando *getContext*. Entretanto, toda vez que esse comando for enviado ao ConBus, será enviada à aplicação cliente uma resposta através de um outro arquivo, *error.xml*. Esse arquivo informa à aplicação solicitante que seu registro foi realizado com sucesso ou houve ocorrência de algum erro. A seguir, é mostrado um exemplo de uso, pelas aplicações, do comando *contextSubscribe*:

```

http://localhost:8585/
contextSubscribe?contextType=PROCESS_EXECUTION&entityName=MY_ANDROID_888
&entityType=OBJECT&appPort=12232
&condition=ENERGY_LEVEL<15

```

Neste exemplo, a aplicação cliente está interessada em receber informações sobre os processos que estão sendo executados em um dispositivo denominado MY_ANDROID_888, sempre que o nível de energia desse dispositivo estiver abaixo de 15%. Nesse caso, o arquivo *response.xml*, semelhante ao enviado pelo ConBus em resposta ao comando *getContext*, contendo a informação de contexto

desejada, deverá ser enviado, pelo ConBus, à porta TCP 12232 desse mesmo dispositivo.

No ConBus, uma expressão é descrita conforme sequência indicada a seguir: *operando operador valor*, onde um *operando* é sempre um tipo de contexto, um *operador* pode ser qualquer um dos operadores relacionais (= (igual a), <> (diferente de), > (maior que), < (menor que), >= (maior que ou igual a), <= (menor que ou igual a)) e um *valor* pode ser um valor numérico ou alfanumérico. Por exemplo, uma expressão válida poderia ser: LUMINOSITY >= 21.

Deve-se ressaltar que, para as aplicações desenvolvidas em Java, é disponibilizada, aos desenvolvedores, uma classe (*ConBusCommunication*) que abstrai os detalhes da comunicação entre essas aplicações e o *middleware*. Ao utilizá-la, as aplicações podem obter as informações contextuais gerenciadas pelo ConBus através de chamadas a métodos simples, como *getContext()*, *getAllContext()* e *contextSubscribe()*, especificando os devidos parâmetros. Esse processo simplifica ainda mais o desenvolvimento dessas aplicações. APIs específicas para outras linguagens podem ser providas futuramente.

4.7 Desenvolvimento de *ConUnits*

Para criar um *ConUnit*, o desenvolvedor deverá, inicialmente, implementar a interface *ContextDataConversion*, dando comportamento aos métodos definidos, criando assim o *Componente de Integração com o Sensor* (*Sensor Integration Component*), conforme ilustrado na Figura 3.3. O *Componente de Integração com o Sensor* é responsável pela comunicação direta com o módulo de sensoriamento correspondente. Além disso, essa interface, conforme mostrado na Figura 4.4, espelha o modelo de contexto definido na Seção 3.3. Para ilustrar esse processo, alguns exemplos de implementação da interface *ContextDataConversion* são exibidos pelas Figuras 4.8 e 4.9.

Após implementar a interface de conversão, o desenvolvedor de um *ConUnit* deverá gerar um arquivo JAR (Java Archive) que irá representar o *ConUnit* correspondente e, em seguida, implantá-lo junto ao ConBus. Para gerar o arquivo JAR, o desenvolvedor pode utilizar a aplicação denominada *ConUnit Builder*, disponibilizada pela implementação do ConBus. Essa aplicação, a partir de algumas informações, como o nome do *ConUnit* a ser criado e o local onde se encontra a classe Java que implementa a interface *ContextDataConversion*, compila e gera automaticamente o código necessário para aco- plar o *ConUnit* desenvolvido ao *framework OSGi*, facilitando assim o processo de criação de *ConUnits* e a integração desses componentes ao *middleware*. A Figura 4.10 ilustra esse processo de criação de *ConUnits*.

Após o processo de criação de um *ConUnit* pelo *ConUnit Builder*, vários componentes são criados e/ou incorporados ao JAR que representa tal *ConUnit*. Nesse JAR

```

public class ContextDataConversion_Location
    implements ContextDataConversion {
    public String getContextType() {
        return "LOCATION";
    }
    public Date getTimeStamp() {
        return Date date_time = new Date();
    }
    public String entityName() {
        return "ROBERT_G_W";
    }
    .
    .
    public ContextData getData() {
        double latitude =
            GPSReceiver_implementation.getLatitude();
        double longitude =
            GPSReceiver_implementation.getLongitude();
        int arraySize = 2;
        dataOfContext = new ContextData();
        String[] typeOfValues =
            {"LATITUDE", "LONGITUDE"};
        String[] values = new String[arraySize];
        values[0] = String.valueOf(latitude);
        values[1] = String.valueOf(longitude);
        String[] units = {"DEGREE", "DEGREE"};
        String[] dataType =
            {"double", "double"};
        dataOfContext.setTypeOfValues(typeOfValues);
        dataOfContext.setUnit(units);
        dataOfContext.setValue(values);
        dataOfContext.setDataType(dataType);
        return dataOfContext;
    }
    .
    .
}

```

Figura 4.8: Implementando a interface *ContextDataConversion* para criar um *ConUnit* que coleta dados sobre a localização de um determinado usuário.

estão contidos: *Componente de Integração com o Sensor* (Sensor Integration Component), módulo de sensoriamento (quando é possível inseri-lo dentro do *ConUnit*), *Ativador do ConUnit* junto ao framework OSGi (ConUnit Activator), arquivo *MANIFEST.MF*, *Gerenciador do ConUnit* (ConUnit Manager), *Controlador de Comunicação Síncrona* (Synchronous Controller) e o arquivo *classes.dex*, contendo o código executável no ambiente Android que representa o *ConUnit* correspondente. De fato, o *ConUnit Builder* é responsável por criar apenas os componentes *Ativador do ConUnit* e o arquivo *MANIFEST.MF*. O *Componente de Integração com o Sensor* e o *módulo de sensoriamento* são fornecidos pelo desenvolvedor do *ConUnit*, enquanto o *Gerenciador do ConUnit* e o *Controlador de Comunicação Síncrona* são disponibilizados pelo pacote *common* da implementação do ConBus. O arquivo *classes.dex* é criado pelas ferramentas disponibilizadas pelo SDK do Android e incorporadas pelo *ConUnit Builder*.

Deve-se observar que o desenvolvedor não precisa ter conhecimento da existência do *container* de componentes utilizado internamente pelo ConBus. Para implantar o

```
public class ContextDataConversion Temperature
    implements ContextDataConversion {
    public String getContextType() {
        return "TEMPERATURE";
    }
    public Date getTimeStamp() {
        return Date date_time = new Date();
    }
    public String entityName() {
        return "ROOM_113";
    }
    .
    .
    public ContextData getData() {
        double temperature = thermometer_impl.getTemperature();
        int arraySize = 1;
        dataOfContext = new ContextData();
        String[] typeOfValues = {"TEMPERATURE"};
        String[] values = new String[arraySize];
        values[0] = String.valueOf(temperature);
        String[] units = {"CELSIUS"};
        String[] dataType = {"float"};
        dataOfContext.setTypeOfValues(typeOfValues);
        dataOfContext.setUnit(units);
        dataOfContext.setValue(values);
        dataOfContext.setDataType(dataType);
        return dataOfContext;
    }
    .
    .
}
```

Figura 4.9: Implementando a interface *ContextDataConversion* para criar um *ConUnit* que coleta dados sobre a temperatura de uma determinada sala.

novo *ConUnit* junto ao *middleware*, é necessário apenas que o arquivo JAR resultante do processo anterior seja copiado para a pasta *ConBus-Bundle* da implementação corrente do ConBus no dispositivo móvel. Se o ConBus ainda não estiver iniciado, ao iniciá-lo, o *middleware* irá detectar a existência de um novo *ConUnit* e instalá-lo, automaticamente, no sistema. Essa característica é viabilizada pelos componentes *Gerenciador de Contexto*, *framework ConUnit* e *framework OSGi* e satisfaz o requisito sobre a implantação dinâmica de módulos de sensoriamento, discutida na Seção 2.2.3. Caso o ConBus já esteja em execução no dispositivo móvel, o próprio usuário poderá instalar o novo *ConUnit*, em tempo de execução, através da interface de administração, disponibilizada pela implementação do *middleware*. Desse modo, o novo *ConUnit* estará disponível para as aplicações interessadas. Para obter dados de contexto coletados pelo novo *ConUnit*, a aplicação deverá realizar as chamadas aos comandos disponibilizados pela *API das Aplicações*, conforme descrito na Seção 4.6.

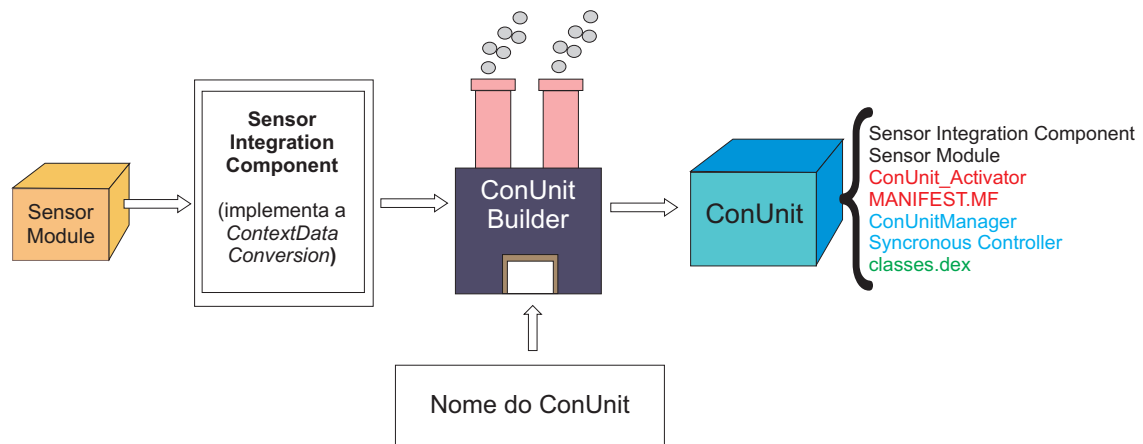


Figura 4.10: *Desenvolvimento de um ConUnit*

4.8 Reutilização de *ConUnits*

O desenvolvedor de uma nova aplicação móvel sensível ao contexto pode reutilizar a implementação de um *ConUnit* desenvolvido por terceiros de acordo com as seguintes etapas. O primeiro passo é conseguir uma cópia do arquivo JAR que representa o módulo de sensoriamento (*ConUnit*) desejado. Em seguida, é necessário apenas que esse arquivo JAR seja copiado para a pasta *ConBus-Bundle* e, finalmente, instalá-lo junto ao ConBus, conforme já descrito na Seção 4.7.

Por exemplo, um desenvolvedor de aplicação sensível ao contexto poderia necessitar de um *ConUnit* para obter informações a respeito dos compromissos de um usuário armazenados no *Google Calendar*⁴. Sabendo que já existe uma implementação desse *ConUnit*, bastaria que o desenvolvedor da aplicação em questão obtivesse o arquivo JAR correspondente e o implantasse no ConBus, conforme os passos já descritos anteriormente. Após iniciar o *ConUnit* correspondente, o mesmo já estaria disponível para que a aplicação interessada fizesse uso das informações fornecidas por esse novo módulo de sensoriamento, desde que todas as condições para que a comunicação entre o sensor (serviço Google Calendar) e seu respectivo módulo de sensoriamento fossem satisfeitas, como autenticação de usuários e comunicação satisfatória.

Se houver um repositório público com implementações de *ConUnits*, os desenvolvedores terão à sua disposição uma maior diversidade de módulos de sensoriamento e poderão, sempre que possível, reutilizar uma implementação existente sem ter que lidar com a complexidade de entender interfaces e protocolos de comunicação de sensores diversos para adaptar suas aplicações às necessidades dos usuários móveis.

⁴disponível em <http://www.google.com/calendar>

Caso o desenvolvedor da aplicação não conheça *a priori* o tipo de contexto, a entidade relacionada ao *ConUnit* em questão e todas as outras informações a respeito do novo *ConUnit*, ele pode consultar essas informações através do comando *getAllContext*, também oferecido pela *API das Aplicações*. Nesse caso, pode-se fazer uma requisição *getAllContext*, através de um navegador Web, para a instância do ConBus em execução no dispositivo móvel correspondente, para receber um arquivo HTML com informações (i.e., tipo de contexto provido, precisão do sensor, a entidade à qual esse sensor está associado) sobre todos os *ConUnits* ativos no ConBus naquele instante. Essa característica do ConBus atende ao requisito sobre facilidades para documentar e exibir documentação de módulos de sensoriamento, discutido na Seção 2.2.

4.9 Limitações

Do modo como foi projetado e implementado o *middleware* ConBus, deve existir um *ConUnit* para cada tipo de sensor associado a uma determinada entidade. Por exemplo, se uma aplicação necessitar obter dados de agendas de compromissos de vários usuários, então deverá existir um *ConUnit* especificamente desenvolvido para coletar os dados sobre os compromissos de cada usuário. Isso ocorre porque tanto o modelo de contexto quanto a interface *ContextDataConversion*, que espelha tal modelo, exigem a associação de uma única entidade para cada *ConUnit*.

Além disso, o *ConUnit* não dispõe de interface assíncrona para a comunicação com o *Gerenciador de Contexto*. Por causa dessa ausência, o *Gerenciador de Contexto* precisa fazer *polling* constante para obter os dados contextuais de que precisa para servir adequadamente às aplicações sensíveis ao contexto. Isso aumenta o consumo de recursos computacionais, como processamento, memória e energia porque obriga o *Gerenciador de Contexto* a fazer chamadas, em intervalos de tempo regulares, aos *ConUnits* para obter os dados contextuais que serão analisados por esse componente, em seu controlador de comunicação assíncrona, visando atender às solicitações das aplicações.

Uma outra limitação é a ausência de mecanismos que permitam ao *ConUnit* gerenciar a execução de módulos de sensoriamento quando esses componentes estiverem implementados externamente aos *ConUnits* correspondentes, conforme discutido na Seção 3.2.1. Por exemplo, em algumas circunstâncias, como quando for necessário poupar recursos computacionais, poderia ser necessária, além da desativação do *ConUnit*, também a desativação de *threads* de execução do próprio módulo de sensoriamento correspondente ou o encerramento de possíveis conexões de rede abertas pela execução desse código. De fato, o ConBus somente gerencia completamente um módulo de sensoriamento quando o mesmo estiver implementado dentro do próprio *ConUnit*.

Em relação ao uso de mais de um tipo de sensor que fornece um tipo de contexto para uma mesma entidade, deve-se observar que a troca por um outro sensor só será possível se eles forem realmente equivalentes, de modo a fornecerem a informação contextual no mesmo formato. Como exemplo, podemos imaginar a informação de localização sendo coletada por dois sensores distintos, um receptor GPS e um sistema de posicionamento que fornece a localização simbólica de um usuário. Nesse caso, como o receptor GPS fornece a informação de localização através de três coordenadas geográficas e o sistema de posicionamento fornece a mesma informação contextual através de um único dado simbólico, como o nome da rua ou do edifício em que um usuário se encontra, não seria possível substituir, diretamente, o receptor GPS por esse sistema de posicionamento simbólico. Isso só seria possível se houvesse, de alguma forma, uma conversão entre os formatos de cada sensor.

Como discutido na Seção 3.2.2, caso haja mais de um sensor/módulo de sensoriamento acoplado que coleta uma mesma informação contextual associada a uma mesma entidade, o *Gerenciador de Contexto* irá escolher o primeiro que encontrar. Esse algoritmo é, de fato, muito simples e deverá ser melhorado em trabalhos futuros. Nesse caso, há vários fatores que poderiam ser levados em consideração por algoritmos mais complexos, como a localização do sensor (se está instalado localmente em relação ao dispositivo móvel correspondente), bem como sua precisão e confiabilidade.

Outro fator limitante é que o ConBus não permite o uso de expressões complexas, como aquelas constituídas pelos operadores lógicos E (AND), OU (OR) e NÃO (NOT).

Estudos de Caso

Este Capítulo discute, a partir de dois estudos de caso, como os serviços de provisão de contexto do ConBus auxiliam o desenvolvimento de aplicações móveis sensíveis ao contexto e a integração de módulos de sensoriamento. O primeiro estudo de caso apresenta a aplicação *SmartTravel* (Seção 5.1), que é capaz de notificar seus usuários sobre a proximidade de pontos turísticos ou de lazer previamente registrados. O segundo estudo de caso discute a implementação da aplicação *AutoRinger* (Seção 5.2), que adapta o tipo de toque do telefone àquele mais adequado ao local e à situação onde o usuário se encontra. A Figura 5.1 ilustra o interrelacionamento das aplicações com o ConBus e os respectivos módulos de sensoriamento, de acordo com as camadas arquiteturais discutidas no Capítulo 3. Ambas as aplicações foram desenvolvidas e testadas na plataforma Android, por meio do emulador da plataforma.

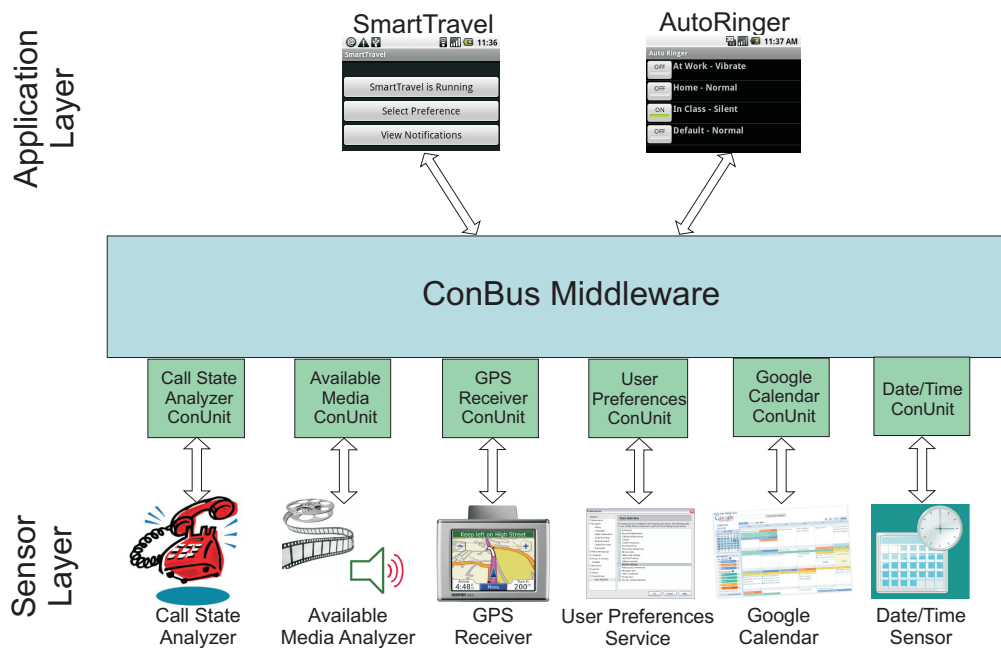


Figura 5.1: Interação entre aplicações móveis, ConBus e sensores.

5.1 SmartTravel

O SmartTravel [5] é uma aplicação que notifica o usuário sempre que ele estiver na proximidade de pontos turísticos ou de lazer previamente registrados como *pontos de interesse*. A Figura 5.2a exibe a tela inicial da aplicação. Na verdade, o SmartTravel é uma aplicação cliente/servidor, cuja parte cliente é executada nos dispositivos móveis e a parte servidora é executada na rede fixa. Neste trabalho, apenas a parte cliente foi implementada, sendo simulada a parte referente ao servidor SmartTravel.

Para ser executado adequadamente, o SmartTravel necessita obter as seguintes informações de contexto: localização, data e hora, dados da agenda de compromissos, tipos de mídia disponíveis no dispositivo móvel, preferências do usuário e se o usuário está ou não realizando uma chamada telefônica. Para registrar um ponto turístico ou de lazer como um ponto de interesse, o usuário deve escolher uma categoria predefinida, como “museus”, “parques”, “hotéis”, dentre outras, ou ainda cadastrar um novo ponto turístico ou de lazer, conforme ilustrado pela Figura 5.2b. Para cada categoria, o usuário pode indicar parâmetros específicos, como “preço máximo das diárias”, para o caso de hotéis e pousadas, “horário de funcionamento”, para o caso de lojas, bares, museus, etc. Após o registro dos pontos de interesse, a aplicação está pronta para ser usada.

Uma vez iniciado, o SmartTravel começa a enviar as informações de localização juntamente com as preferências dos pontos turísticos ou de lazer previamente cadastradas para um servidor remoto. Esse servidor permite que as instâncias do SmartTravel, localizadas nos dispositivos móveis, enviem informações sobre a localização corrente, bem como as preferências de seus usuários (os pontos turísticos previamente registrados). Após receber essas informações, o servidor remoto procurará encontrar os pontos turísticos ou de lazer que estejam nas proximidades do local onde o usuário móvel se encontra naquele momento. Caso encontre algum ponto turístico ou de lazer de interesse do usuário, o servidor remoto enviará uma resposta ao SmartTravel com as informações sobre o ponto turístico ou de lazer encontrado. Em seguida, de posse dessas informações, a aplicação móvel irá analisar o contexto atual do usuário e do dispositivo móvel procurando encontrar a melhor maneira possível de notificá-lo (através de texto, áudio, vídeo, mapas, etc) sobre a existência de algum ponto turístico ou de lazer de seu interesse nas proximidades. Por exemplo, ao consultar a agenda de compromissos do usuário, pode-se descobrir se o usuário tinha ou não uma reunião naquele local e horário e, assim, postergar uma notificação, para após o seu término. As Figuras 5.3(a), 5.3(b) e 5.4(a) ilustram a chegada de uma notificação do SmartTravel, informando ao usuário da proximidade do ponto de referência “Castros Park Hotel”. Na Figura 5.4b, um mapa é exibido indicando o local do ponto turístico ou de lazer encontrado.

Do ponto de vista da provisão de contexto, todas as informações contextuais

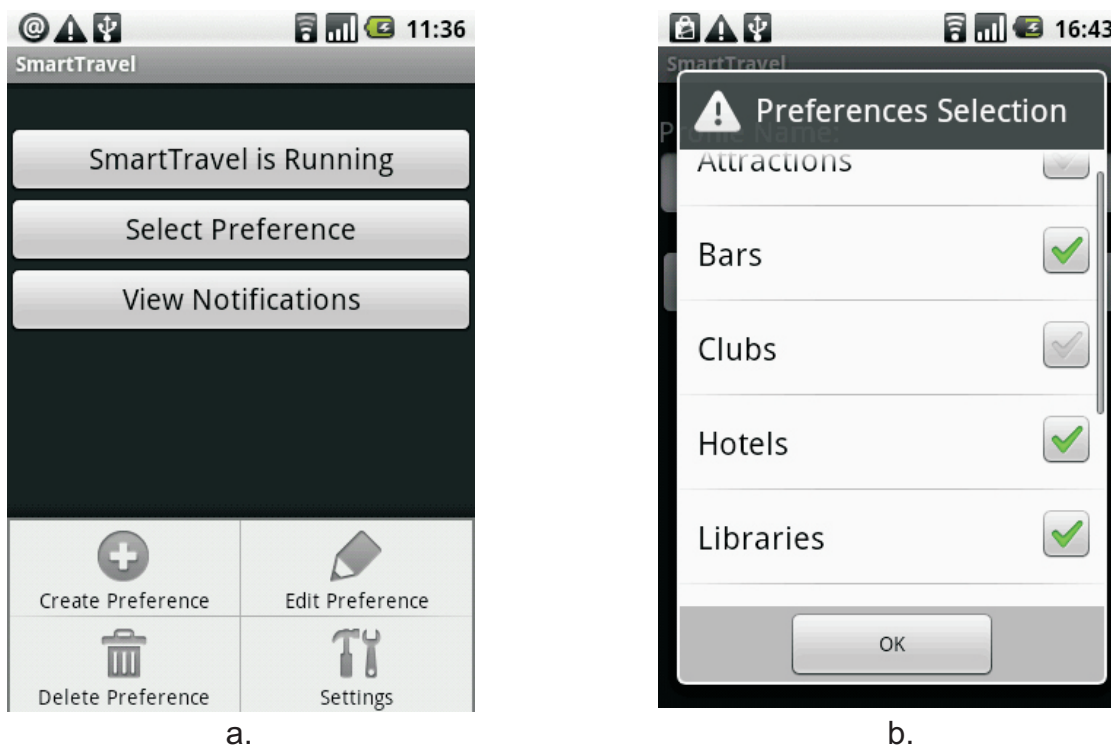


Figura 5.2: a) Tela de Iniciação. b) Tela de cadastro de pontos turísticos.

necessárias são obtidas dos sensores correspondentes através de *ConUnits*. Dentre as informações contextuais necessárias, a informação sobre os tipos de mídia para os quais o dispositivo móvel provê suporte é fornecida através do *Available Media ConUnit*. Isso é importante para que o SmartTravel possa notificar o usuário da melhor maneira possível, de acordo com cada situação. Por exemplo, caso um usuário esteja dirigindo um automóvel e seja necessário notificá-lo sobre um ponto turístico ou de lazer próximo a ele, o SmartTravel poderia escolher, como melhor opção para notificação, nesse caso, o uso de um arquivo de áudio. Porém, mesmo sabendo que uma mensagem de áudio seja mais adequada para essa situação, é necessário saber se o dispositivo móvel correspondente suporta tal formato de arquivo. É nesse momento que o *Available Media ConUnit* seria consultado. Além dessas informações, é importante para o SmartTravel saber se o usuário está ou não em ligação, pois caso esteja, a aplicação poderá postergar o envio de uma notificação. Essa informação contextual é obtida através do *Call State Analyzer ConUnit*.

Seleção e Interação com *ConUnits*

O ConBus permite que o usuário móvel ou uma aplicação sensível ao contexto troque, de acordo com a necessidade, os módulos de sensoriamento utilizados pelo *middleware* por outros equivalentes que coletam o mesmo tipo de contexto de sensores

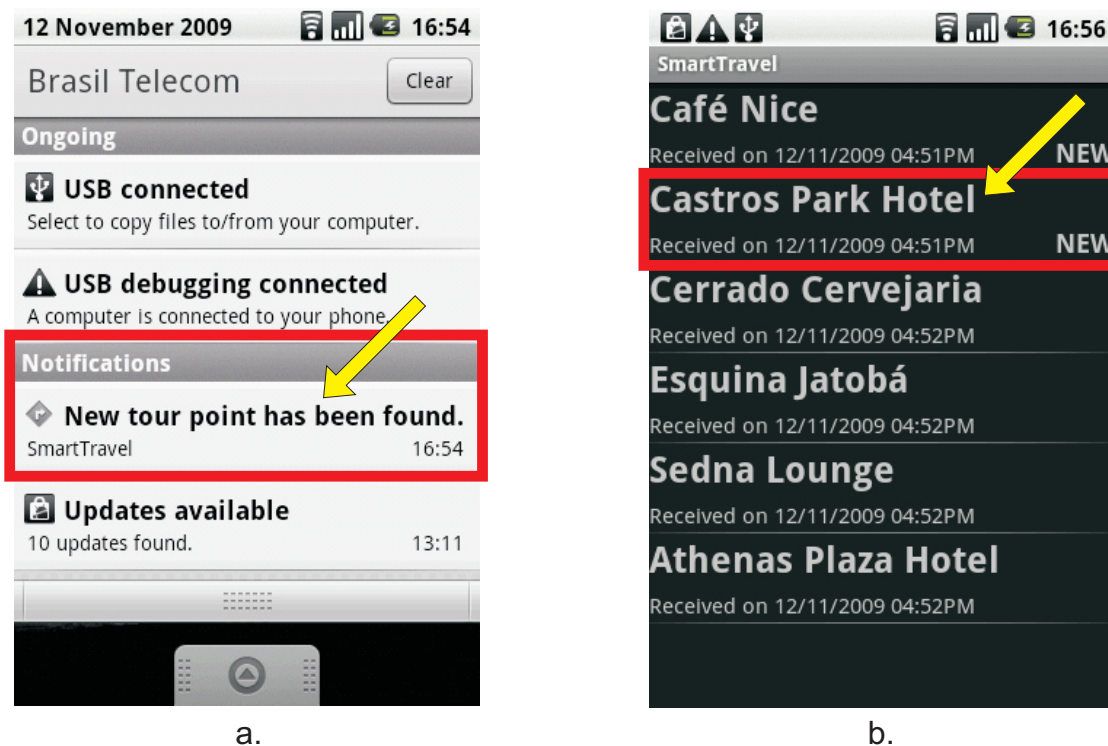


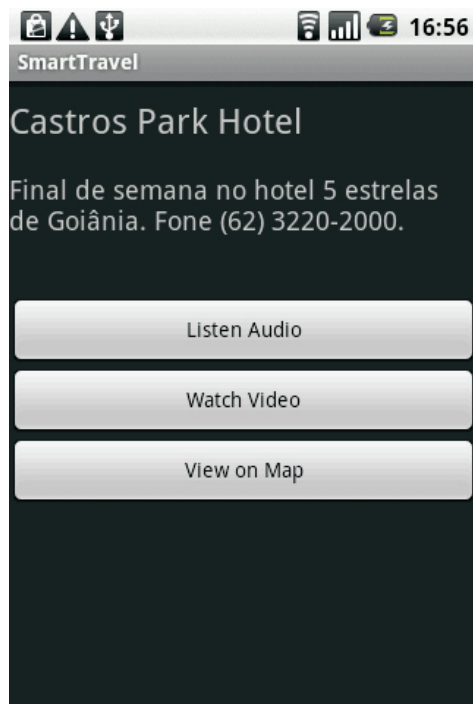
Figura 5.3: *Chegada de uma notificação sobre ponto turístico encontrado.*

diferentes e que utilizam o mesmo formato para armazenar os dados contextuais. Isso pode ser útil quando a aplicação souber que um determinado sensor está fornecendo uma mesma informação contextual fornecida por outro sensor, porém com uma precisão maior ou com dados mais atualizados. Por exemplo, para a implementação do SmartTravel foi utilizado um *ConUnit* que coleta dados da agenda de compromissos do Google (*Google Calendar*). Entretanto, durante a fase de avaliação, foi possível mudar o tipo de sensor virtual que coleta dados da agenda de compromissos, de forma a obter esses dados de um outro serviço equivalente (por exemplo, o Yahoo Calendar). Nesse caso, a única alteração necessária pela aplicação foi mudar, na chamada ao ConBus, o tipo de sensor correspondente. Os trechos de código Java a seguir ilustram as diferenças entre as duas chamadas.

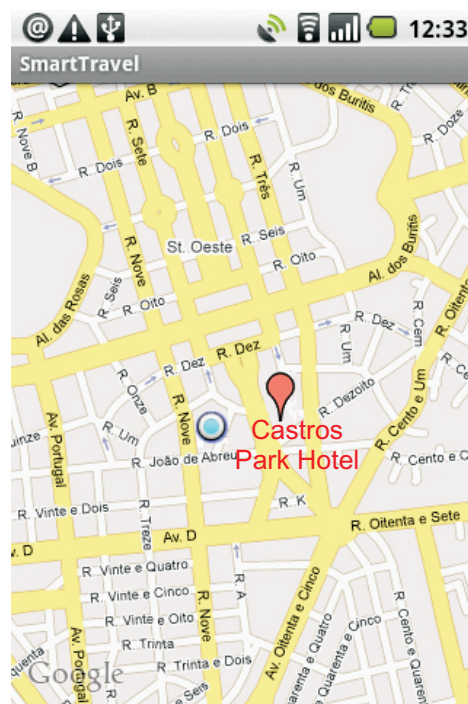
```
ContextInformation ci =
    ConBusCliente.getContext(
        "MEETINGS", "Bob", "PERSON", "GoogleCalendar");

ContextInformation ci =
    ConBusCliente.getContext(
        "MEETINGS", "Bob", "PERSON", "YahooCalendar");
```

É importante notar que, na chamada à *API das Aplicações*, a indicação do sensor que deve enviar as informações de contexto é opcional. Nesse caso, se não for indicado,



a.



b.

Figura 5.4: a) Opções para receber uma notificação. b) Mapa indicando a localidade encontrada.

e caso exista mais de um sensor coletando dados de um mesmo tipo de contexto para uma mesma entidade, o ConBus escolherá o primeiro sensor/módulo de sensoriamento que satisfizer às condições indicadas pela aplicação (i.e., tipo de contexto e entidade associada).

A Tabela 5.1 exibe algumas informações importantes sobre a implementação do SmartTravel, mostrando, para cada tipo de contexto utilizado, o sensor correspondente e a quantidade de linhas de código (SLOC) implementadas para a construção de seu módulo de sensoriamento, excluindo o código já implementado pelo próprio *ConUnit framework*. A aplicação SmartTravel totalizou 2221 linhas de código.

Dentre os benefícios de se usar o ConBus como infraestrutura para prover informações de contexto ao SmartTravel destaca-se a possibilidade de mudar automaticamente os sensores que coletam informações para a aplicação. Essas mudanças permitem, por exemplo, que as próprias aplicações escolham, em tempo de execução, o sensor mais adequado dentre aqueles que coletam o mesmo tipo de contexto para uma mesma entidade. Entretanto, pode ser que, em muitos casos, o uso de *ConUnits* e do próprio ConBus diminua o desempenho do SmartTravel, visto que muitas informações contextuais, como a informação de localização provida pelo GPS embutido nos dispositivos móveis, poderiam ser coletadas mais rapidamente se não passassem primeiramente pelo *middleware* e

Tabela 5.1: *Esforço de implementação dos ConUnits e uso nas aplicações*

ConUnit	SLOC	Tipo de Contexto	Sensor real / fonte	SmartTravel	AutoRinger
GPS Receiver	56	Location	GPS - HTC G1	✓	✓
Google Calendar	172	Meetings	Google Calendar	✓	✓
Yahoo Calendar	188	Meetings	Yahoo Calendar	✓	
Media Analyzer	15	Available Media	sistema operacional	✓	
Call State Analyzer	23	Call State	sistema operacional	✓	
User Preferences	150	User Preferences	software proprietário	✓	✓
Date and Time	150	Date and Time	sistema operacional	✓	✓

fossem conduzidas diretamente à aplicação. Porém, os benefícios da reutilização de *ConUnits* devem ser levados em consideração, bem como o uso de uma interface uniforme de acesso aos dados de contexto, pois podem diminuir o tempo de desenvolvimento dessas aplicações sensíveis ao contexto.

5.2 AutoRinger

O AutoRinger é uma aplicação móvel que utiliza informações de contexto, como a localização e as preferências de seus usuários para, de acordo com cada situação do dia-a-dia, configurar o tipo de toque de campainha mais adequado para o dispositivo móvel. Como exemplo, imagine que todo dia, ao chegar à sala de aula, um professor, manualmente, altere o tipo de toque de campainha de “normal” para “silencioso” ou “alerta vibratório”. Se ele estivesse usando o AutoRinger, precisaria apenas configurar a aplicação uma única vez para que todos os dias, no horário das aulas, o aparelho, ao detectar que o usuário se encontra em sala de aula, altere o tipo de toque para “silencioso” automaticamente e o mantenha assim até que o usuário/aparelho móvel saia do local em que se encontra ou que o próprio usuário realize, manualmente e explicitamente, alguma alteração.

A Figura 5.5a exibe a tela inicial da aplicação. A exemplo do SmartTravel, para que o sistema funcione adequadamente, o usuário deverá indicar ao sistema suas preferências de configuração. Isso pode ser realizado a partir das telas de configuração da aplicação, como é exibido pelas Figuras 5.5b e 5.6a. Após o processo de configuração, o sistema irá constantemente verificar se há a necessidade de alterar o tipo de toque de campainha de acordo com as informações contextuais recebidas pelo ConBus e sempre que houver necessidade, o tipo de toque de campainha será alterado automaticamente pelo AutoRinger, como indicado pela Figura 5.6b.

Como mencionado anteriormente, para funcionar adequadamente, o AutoRinger necessita de várias informações contextuais: dados da agenda de compromissos, data

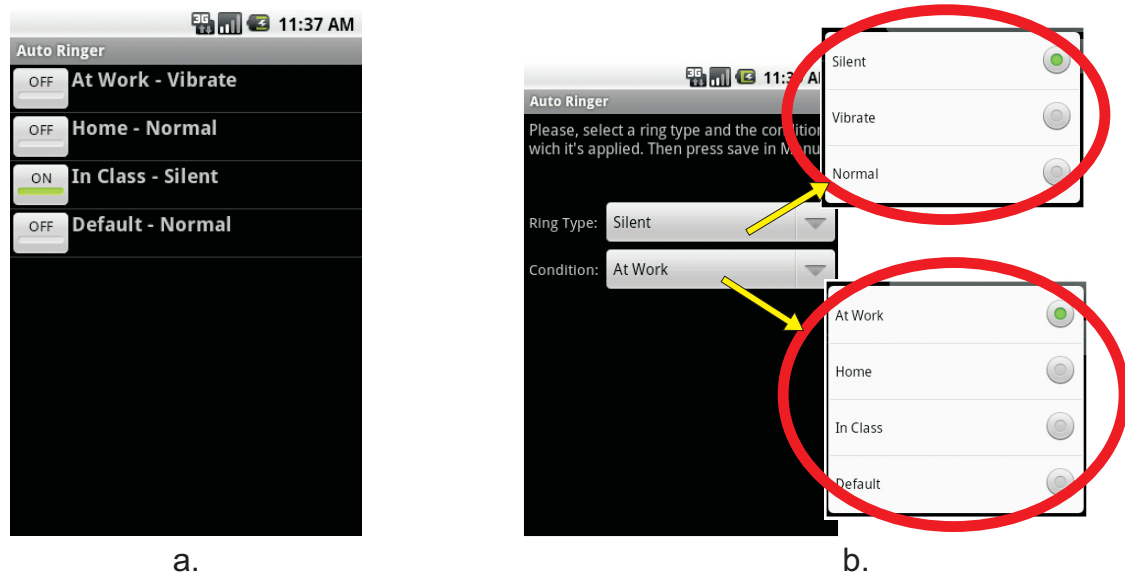


Figura 5.5: *Funcionamento e uso do AutoRinger.*

e hora, localização e preferências do usuário. De modo similar ao ocorrido para o SmartTravel, toda a provisão de contexto é também realizada pelo ConBus, propiciando aos desenvolvedores do AutoRinger uma separação de interesses entre a provisão de informações contextuais e o uso dessas informações pela aplicação propriamente dita.

Além das facilidades já discutidas na construção do SmartTravel, deve-se mencionar uma das características mais importantes do ConBus que é explorada no desenvolvimento do AutoRinger, a reutilização de módulos de sensoriamento. Nesse caso, os módulos de sensoriamento necessários, representados por seus *ConUnits* correspondentes (Date/Time ConUnit, Google Calendar ConUnit, User Preferences ConUnit e GPS Receiver ConUnit) para coletar informações contextuais para o AutoRinger foram reutilizados a partir daqueles desenvolvidos para o SmartTravel. A reutilização juntamente com a disponibilização de uma interface uniforme para acesso aos dados contextuais são alguns dos grandes benefícios que uma plataforma de provisão de contexto oferece aos desenvolvedores de aplicações sensíveis ao contexto. Isso ocorre porque a possibilidade de reaproveitar tanto código quanto instâncias de módulos de sensoriamento, evitando que o desenvolvedor da aplicação tenha que se preocupar com os detalhes de funcionamento e implementação de cada sensor específico, como protocolos de comunicação, chamadas de baixo nível ao sistema operacional, facilita o desenvolvimento e popularização dessas aplicações.

A aplicação AutoRinger totalizou 1512 linhas de código e utilizou os *ConUnits* indicados na tabela 5.1. Todos os *ConUnits* utilizados foram compartilhados com a aplicação SmartTravel.

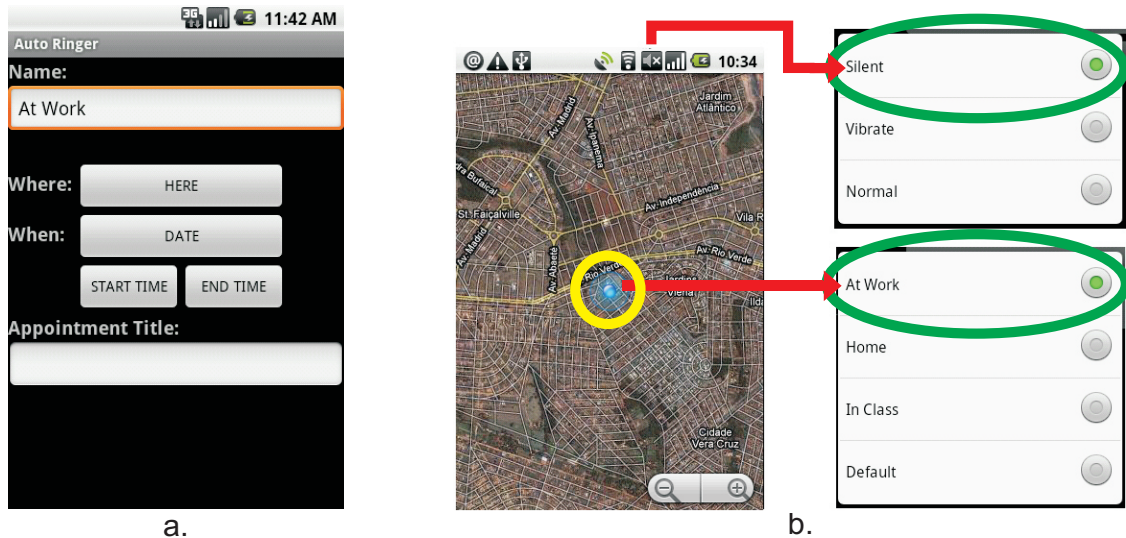


Figura 5.6: a) Criando uma condição de uso. b) Alterando o tipo de toque de campainha de acordo com o contexto corrente.

5.3 Conclusões

A partir dos estudos de caso discutidos neste Capítulo, nota-se que o uso do Con-Bus como infraestrutura de provisão de contexto em dispositivos móveis permite diminuir o tempo de desenvolvimento de aplicações móveis sensíveis ao contexto na medida em que facilita a reutilização de módulos de sensoriamento, proporcionando ainda melhoria na qualidade das aplicações, visto que a reutilização desses módulos pode permitir, na maioria das vezes, o uso de código mais confiável, possivelmente testados por vários grupos de desenvolvimento. Além disso, um benefício direto do uso dessas plataformas de *middleware* de provisão de contexto é a separação de interesses, que permite aos desenvolvedores de aplicações móveis sensíveis ao contexto se preocuparem mais com a lógica das aplicações propriamente ditas do que com a provisão de informações contextuais para tais aplicações.

Trabalhos Relacionados

Módulos de sensoriamento são elementos comuns a qualquer arquitetura de *middleware* para computação sensível ao contexto. Entretanto, poucos sistemas de *middleware* oferecem abstrações específicas para facilitar o desenvolvimento e integração desses módulos. Essa tarefa envolve problemas como distribuição do sensor (local, remoto ou distribuído), descoberta do sensor (dinâmica ou estática), carregamento e desconexão do módulo de sensoriamento. Context Toolkit [6] [17], Hydrogen [9] e Contory [14][13] são alguns dos exemplos de sistemas de *middleware* que oferecem abstrações para tratar da integração de módulos de sensoriamento. Nas Seções seguintes, esses trabalhos serão analisados em maior profundidade. Inicialmente, serão feitas apenas descrições da arquitetura e do funcionamento de cada plataforma. Na Seção 6.4 é feita uma comparação entre cada trabalho discutido, analisando suas principais semelhanças e diferenças em relação ao *middleware* ConBus.

Entretanto, há muitas outras plataformas de *middleware* de provisão de contexto disponíveis, tais como MoCA [16], CoBrA [3], Context Fabric [10], SOCAM [7], ContextGrid [4], Middleware for Multimedia Services [21], porém essas arquiteturas não têm o foco na provisão de contexto especificamente em dispositivos móveis ou não oferecem abstrações específicas para facilitar o desenvolvimento e integração de módulos de sensoriamento.

6.1 Context Toolkit

O Context Toolkit [6] [17] é uma implementação de um *framework* conceitual ou, como os próprios autores dizem, um *kit de ferramentas* para dar suporte ao desenvolvimento rápido de aplicações sensíveis ao contexto. Esse *framework* conceitual é composto de elementos que atuam desde a aquisição até o processamento e disponibilização de informações de contexto às aplicações. Seus elementos são: *context widgets*, interpretadores (*interpreters*), agregadores (*aggregators*), serviços (*services*) e descobridores (*discoverers*). A seguir, cada elemento será examinado em maiores detalhes.

- **Context widgets:** são componentes de software empregados para fornecer às aplicações acesso à informação de contexto. Os *Context widgets* são considerados as estruturas oferecidas pela arquitetura para permitir a integração dos *módulos de sensoramento* a essa plataforma, atuando como os responsáveis pela obtenção das informações contextuais coletadas pelos sensores, disponibilizando-as às aplicações interessadas e a outros componentes do *framework*. Por isso, são os mediadores localizados entre a aplicação e seu ambiente operacional. Suas principais funções são:
 - Fornecer separação de interesses, escondendo da aplicação a complexidade dos sensores reais utilizados;
 - Fornecer informações abstratas de forma a alimentar a aplicação com apenas informações relevantes;
 - Fornecer blocos de aquisição de contexto reusáveis e personalizáveis que possam ser utilizados por uma variedade de aplicações sem a necessidade de serem alterados.
- **Interpretadores (*Interpreters*):** são os responsáveis por elevar o nível de abstração de um certo “bloco” de contexto. Ou seja, um interpretador de contexto recolhe informações de uma ou mais fontes de contexto e produz uma nova unidade de informação contextual. Todos os interpretadores têm uma interface comum facilitando, assim, a identificação das capacidades que um determinado interpretador fornece, além de permitir a comunicação com o ambiente externo.
- **Agregadores (*Aggregators*):** têm a função de reunir múltiplos pedaços de informação de contexto que estão logicamente relacionados e armazená-los em um repositório comum. De acordo com os autores, a necessidade de agregação surge devido principalmente à natureza distribuída da informação de contexto.
- **Serviços (*Services*):** Nota-se que os três primeiros componentes citados (*widgets*, interpretadores e agregadores) são responsáveis pela aquisição de contexto e sua respectiva entrega às aplicações interessadas nesses dados. Os serviços, por outro lado, “são os componentes do *framework* que executam as ações em favor das aplicações” [6]. Para os autores, esses serviços podem ser síncronos ou assíncronos, dependendo ou não da necessidade de se esperar uma resposta para que outras ações sejam executadas subsequentemente. Um exemplo disso seria o uso de um serviço para, a partir da informação sobre a temperatura de um sala obtida por um sensor, ligar o condicionador de ar daquela respectiva sala.
- **Descobridores (*Discoverers*):** destinam-se a manter um registro de quais capacidades existem, em cada momento, no *framework*. Por essa razão, são considerados os componentes finais no *framework* conceitual proposto. Portanto, os descobridores devem conhecer os *widgets*, interpretadores, agregadores e serviços que estão

atualmente disponíveis para uso pelas aplicações. A Figura 6.1 exibe o diagrama de objetos para as abstrações descritas anteriormente para o Context Toolkit.

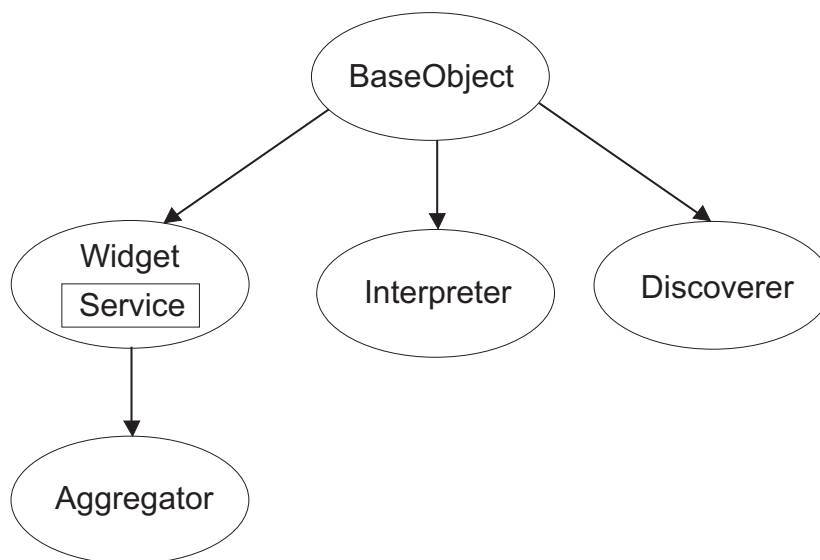


Figura 6.1: Diagrama de objetos para as abstrações do Context Toolkit.

Nota-se, na arquitetura, que *Widgets* podem ser consultados ou usarem notificações para informar a seus clientes sobre mudanças ocorridas. Seus clientes podem ser aplicações, agregadores ou outros *widgets*. Agregadores, normalmente, atuam como pontes entre *widgets* e as aplicações. Interpretadores podem ser solicitados em qualquer estágio e por qualquer *widget*, agregador ou aplicação. Serviços são acionados primeiramente pelas aplicações (embora outros componentes possam também usá-los). E, finalmente, descobridores se comunicam com todos os componentes, adquirindo informação de *widgets*, interpretadores e agregadores e, em seguida, fornecendo essas informações às aplicações via notificações ou consultas.

6.2 Hydrogen

O projeto Hydrogen [9] propõe uma arquitetura e uma implementação de um *framework* para facilitar o desenvolvimento de aplicações móveis sensíveis ao contexto. De forma semelhante ao ConBus, esse *framework* é totalmente executado no próprio dispositivo móvel, diferentemente de outras infraestruturas para provisão de contexto, como o Context Toolkit, visto anteriormente.

O *framework* é definido em uma arquitetura composta por três camadas: *camada de adaptadores*, *camada de gerência* e *camada de aplicação*. A Figura 6.2 exibe o diagrama arquitetural do Hydrogen.

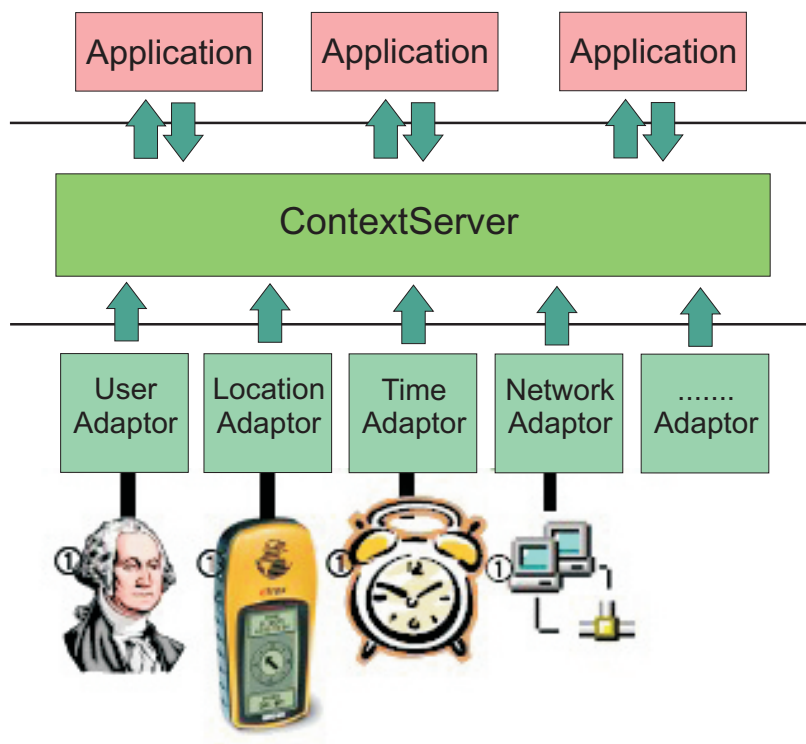


Figura 6.2: Diagrama arquitetural do Hydrogen .

A Camada de Adaptadores (*Adaptor Layer*) é a responsável pela obtenção de informações de contexto dos sensores. Por isso, cada componente *adaptador* dessa camada é considerado como a estrutura dessa arquitetura que permite integrar um *módulo de sensoriamento* ao *framework*, visto que intermedeia a comunicação entre um sensor específico e o restante do *framework*, disponibilizando os dados coletados por esses sensores a todas as aplicações interessadas. Nessa camada é possível ainda enriquecer os dados obtidos dos sensores através de algum tipo de processamento. Por exemplo, a partir de coordenadas GPS (latitude, longitude e altitude), o *adaptador* responsável por obter esses dados poderia ainda inferir um endereço postal, como o nome da rua, bairro e número da localidade indicada pelas coordenadas GPS.

A Camada de Gerenciamento (*Management Layer*) tem a função de armazenar toda a informação de contexto obtida pelos sensores (por intermédio dos *adaptadores*) sobre o ambiente atual do dispositivo móvel e disponibilizá-los às aplicações interessadas. Seu principal componente é o *Servidor de Contexto* (*ContextServer*). O *Servidor de Contexto* é um objeto executável Java e pode ser acessado pelas aplicações através de uma porta de comunicação definida pelo usuário. O *Servidor de Contexto*, após ser iniciado, ficará aguardando comandos enviados pelos adaptadores e aplicações.

6.3 Contory

O **CONTextfactORY** ou simplesmente Contory [14][13] é um *middleware* para provisão de contexto em *smartphones*. O grande diferencial dessa arquitetura é a capacidade de integrar múltiplas estratégias para provisão de contexto nesses dispositivos móveis, empregando o mecanismo de provisão de contexto mais adequado às necessidades atuais e recursos disponíveis.

A primeira estratégia básica para prover contexto é denominada **provisão de contexto interna** (*internal context provisioning*) e consiste na implantação de provedores de contexto instalados no próprio dispositivo móvel. A segunda estratégia consiste na implantação de componentes autônomos que podem ser utilizados por qualquer aplicação, existindo independentemente de aplicação e que são executados em dispositivos remotos. Essa estratégia é conhecida como **provisão de contexto centralizada externa** (*external centralized context provisioning*). A terceira abordagem é a provisão distribuída em redes *ad hoc*, denominada **provisão de contexto distribuída externa** (*external distributed context provisioning*), em que nós equipados com os sensores necessários podem adquirir dados de contexto brutos, processá-los e torná-los disponíveis aos nós vizinhos.

A ideia chave do Contory é prover mecanismos especializados e transparentes para a recuperação de itens de contexto. Um item de contexto é a unidade básica usada para expressar o contexto associado a uma determinada situação. Itens de contexto típicos descrevem informação espacial (localização, velocidade), temporal (data, hora, duração), caracterização de usuário (atividade, estado emocional), ambiental (ruído, temperatura, luminosidade), etc.

O componente principal de toda a arquitetura é o *ContextFactory*, que é baseado no padrão de projeto *Factory Method*. Esse componente permite às aplicações consultarem as informações de contexto disponibilizadas pelo Contory. Além disso, o *ContextFactory* é também um *singleton*, visto que pode existir apenas uma instância dele para cada aplicação. Porém, exigir que cada aplicação utilize uma instância diferente do *ContextFactory* pode comprometer ainda mais os já limitados recursos computacionais dos dispositivos móveis, especialmente quando houver várias aplicações utilizando, ao mesmo tempo, o Contory. A provisão de informações contextuais ocorre através do uso de vários componentes *Facade* e *CxtProviders*; entretanto, o armazenamento e a disponibilização de contexto são gerenciados pelos componentes *QueryManager*, *ContextRepository* e *ContextPublisher*. A Figura 6.3 ilustra o modelo conceitual da arquitetura Contory.

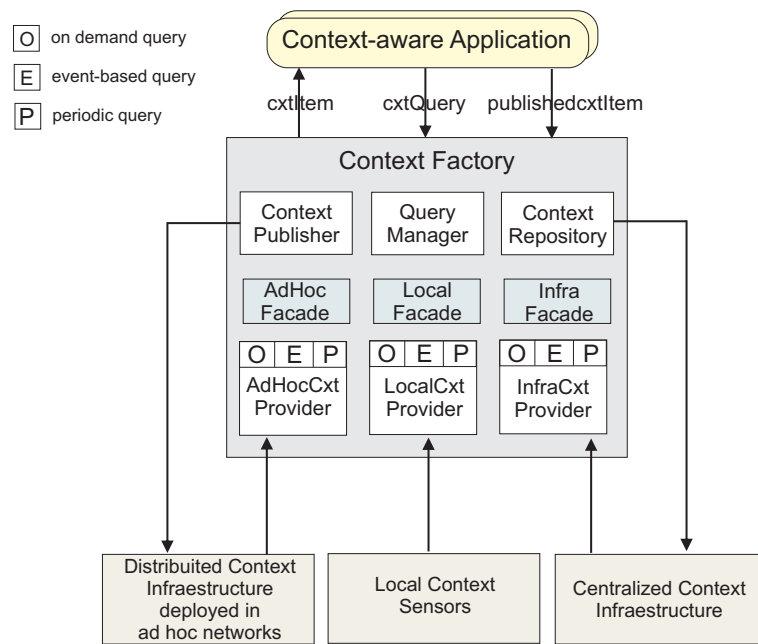


Figura 6.3: Diagrama arquitetural do middleware Contory.

6.4 Comparação

Os trabalhos podem ser comparados em relação a três aspectos principais: *i*) como a infraestrutura de provisão de contexto facilita o desenvolvimento e a reutilização de módulos de sensoriamento; *ii*) como a arquitetura facilita a integração dos módulos com a própria infraestrutura ou com as aplicações cliente; e *iii*) como ocorre o gerenciamento do ciclo de vida dos módulos, conforme resumido pela Tabela 6.1.

Em relação aos aspectos de desenvolvimento, são analisadas, neste texto, questões como as abstrações utilizadas pelas infraestruturas para representar fisicamente um módulo de sensoriamento, bem como as formas de reutilização desses módulos (reutilização apenas de instâncias ou também do código de cada módulo de sensoriamento). Aqui, deve-se notar que, em princípio, todas as infraestruturas analisadas permitem, de alguma forma, reutilizar códigos de seus módulos de sensoriamento para serem reaproveitados em outros projetos. Porém, a questão a ser analisada é quais facilidades a arquitetura oferece para permitir essa reutilização. Por exemplo, o código deve ser recompilado; se há repositórios que disponibilizam tais módulos; como esses módulos são construídos fisicamente e fornecidos aos desenvolvedores. Além disso, também é analisada a modelagem de contexto empregada por cada arquitetura (baseada em ontologias, atributo/valor, orientada a objetos).

O Context Toolkit utiliza o componente denominado *Context Widget* como abstração para incorporar módulos de sensoriamento à sua arquitetura. Devido ao fato

Tabela 6.1: Comparação do ConBus/ConUnit com outros sistemas de middleware

	Desenvolvimento			Implantação			Ciclo de Vida			
	Abstração	Reutilização	Modelo de Dados	Mecanismo	Componentizável	Localização sensor	Ativação dinâmica	Instalação dinâmica	Registro/descoberta	Seleção de módulos
ConBus	ConUnit	Inst/Cód	par/valor	OSGi	✓	Dep. Módulo	✓	✓	OSGi	Expl/Prim
ContextToolkit	Widget	Inst/Cód	par/valor	serviço	✓	Dist./Transp.	—	—	serviço	Expl
Contory	Reference	Inst/Cód	par/valor	?	—	Transp.	✓	?	?	—
Hydrogen	Adaptor	Inst/Cód	O.O.	serviço	✓	Dep. Módulo	×	×	serviço	Expl

Legenda:

—	não se aplica ao sistema	Inst	reutilização de instâncias de sensores ou módulos
?	não foi possível identificar.	Transp.	permite qualquer estratégia, deixando transparente às aplicações
Cód	reutilização de código de módulos	Expl	indicação explícita do sensor/módulo pela aplicação
Dist.	A localização é conhecida pela aplicação	Prim	seleção do primeiro sensor/módulo que atende requisição

dessa infraestrutura ser distribuída, um *context widget* pode ser instalado e executado em qualquer máquina ou sistema operacional pertencente ao sistema computacional correspondente, funcionando independentemente de se ter ou não aplicações ou outros componentes do *framework* interessados em utilizar as informações disponibilizadas por ele. Por outro lado, o Hydrogen oferece o componente *adaptor* como abstração para a incorporação de módulos de sensoriamento ao *framework*. Esse componente, assim como todos os outros dessa arquitetura, é executado integralmente no dispositivo móvel.

O Contory utiliza a abstração denominada *Reference Module*, um componente que agrupa diversos módulos de sensoriamento/sensores com características semelhantes. O Contory inclui em sua implementação quatro tipos de *Reference Module*: (i) *Internal-Reference*, que é especializado em viabilizar a comunicação com sensores integrados ao próprio dispositivo móvel; (ii) o *BTRreference*, componente responsável pela descoberta de dispositivos e serviços oferecidos através da tecnologia *Bluetooth*, além de viabilizar a comunicação com tais serviços e dispositivos; (iii) o *WiFiReference* gerencia a comunicação em redes WiFi, mas também fornece abstrações para roteamento geográfico, roteamento baseado em conteúdo e comunicação *multi-hop* em redes *ad hoc*; (iv) o *2G/3GReference* gerencia a comunicação com entidades remotas que não podem ser descobertas pelas tecnologias de rede discutidas anteriormente e oferece ainda interface baseada em eventos.

Deve-se observar que, de acordo com os trabalhos pesquisados, nem o Context Toolkit [6] [17], Hydrogen[9] ou Contory [14][13] oferecem mecanismos que facilitem especificamente o desenvolvimento de módulos de sensoriamento, como ocorre com o *ConUnit Framework* oferecido pelo ConBus. Além disso, outro benefício importante oferecido por este *middleware* é a facilidade de reutilização dos *ConUnits* em outros projetos, visto que é necessário apenas que se obtenha uma cópia do arquivo JAR que representa o módulo de sensoriamento desejado e que, em seguida, que a mesma

seja inserida dentro da pasta *conbus-bundle* da implementação ConBus do respectivo dispositivo móvel, conforme discutido na Seção 4.8.

Dentre os aspectos relativos à implantação de tais módulos de sensoriamento junto à infraestrutura de provisão de contexto, são analisados os mecanismos de acoplamento desses módulos à arquitetura, como o acoplamento direto, serviços para registro e descoberta de módulos de sensoriamento e sistemas de módulos dinâmicos (i.e., OSGi).

O acoplamento direto se caracteriza pelo união, em tempo de projeto e implementação, dos módulos de sensoriamento com os outros componentes da infraestrutura. Nesse caso, todo o código do sistema (módulos de sensoriamento e o restante da infraestrutura) é compilado junto. Isso simplifica o projeto da infraestrutura de provisão de contexto, além de ser uma opção de comunicação entre módulos e *middleware* bastante eficiente (uso de chamadas a métodos locais). Entretanto, esse mecanismo dificulta a implantação de novos módulos na medida em que não permite o acoplamento dinâmico de tais componentes. Por outro lado, o emprego de serviços especializados para registro e descoberta de módulos de sensoriamento auxilia tanto no acoplamento dinâmico dos módulos quanto na possibilidade de se distribuir esses módulos em várias máquinas e locais do ambiente computacional. Além disso, o uso desses serviços viabiliza a independência de linguagem para o desenvolvimento de módulos de sensoriamento, pois esses serviços permitem isolar, através de protocolos de comunicação, aspectos específicos de cada linguagem particular, como os mecanismos de comunicação e chamadas a métodos locais. Por outro lado, o uso de sistemas de módulos dinâmicos fornece à arquitetura de *middleware* a possibilidade de aliar as vantagens do uso do acoplamento direto com relação à eficiência na comunicação entre módulos e *middleware* com a possibilidade da implantação dinâmica desses módulos, como acontece com o emprego de serviços de registro e descoberta.

Com relação à implantação desses módulos, o Context Toolkit emprega um serviço especializado para registro e descoberta representado na arquitetura pelo componente *Discoverer*. O Hydrogen, através de uma classe Java denominada *ContextClient*, também disponibiliza um serviço responsável por realizar a implantação dos módulos de sensoriamento junto ao *framework*, viabilizando a comunicação entre um *adaptador* e o *Servidor de Contexto* da Camada de Gerenciamento. Nesse caso, tanto o Context Toolkit quanto o Hydrogen disponibilizam facilidades para viabilizar essa comunicação, que é baseada em XML sobre TCP/IP. No caso do Hydrogen, todos os detalhes dessa comunicação, como a abertura de portas e envio de dados de contexto ao Servidor de Contexto são gerenciados por essa classe predefinida. O *middleware* Contory não se preocupa em oferecer facilidades específicas para a implantação de módulos de sensoriamento isolados, como acontece com o ConBus, através do *framework* *ConUnit*. Isto ocorre porque seu foco está na capacidade de integrar múltiplas estratégias de provisão de contexto completas, não sendo

possível identificar, nos trabalhos analisados, como essa implantação acontece, de fato.

O ConBus utiliza, como mecanismo para viabilizar o acoplamento de módulos à infraestrutura, o *framework OSGi*. Esse *framework* permite o acoplamento dinâmico dos módulos, representados pelos *ConUnits*. Esse tipo de acoplamento se torna vantajoso para a arquitetura na medida em que não é necessário reiniciar todo o sistema para inserir ou excluir um módulo de sensoriamento, evitando assim possíveis problemas nas aplicações sensíveis ao contexto que utilizam os serviços do ConBus naquele instante, devido à indisponibilidade de informações contextuais. Porém, esse sistema de módulos dinâmicos é executado essencialmente em Java, não sendo permitindo acoplar ao *middleware* diretamente módulos de sensoriamento desenvolvidos em outra linguagem de programação.

Em relação a questões relativas à localização do sensor, nota-se que no Context Toolkit as aplicações podem escolher ou não lidar explicitamente com a distribuição dos sensores e módulos em diversas máquinas e ambientes, visto que o *discoverer* pode abstrair tal informação. Porém, tanto no Hydrogen como no ConBus cada módulo de sensoriamento é responsável por lidar com os aspectos relativos à localização (local ou remoto) de seus respectivos sensores, evitando que as próprias aplicações ou a infraestrutura correspondente se preocupe com isso. Entretanto, o Contory, através dos *Reference Modules* disponibilizados por sua implementação, deixa transparente às aplicações tais questões, cuidando dos detalhes relativos à localização (local, remoto ou distribuído) de cada sensor correspondente.

Para o gerenciamento do ciclo de vida, são analisados aspectos como instalação e ativação dinâmicas dos módulos de sensoriamento, bem como os mecanismos empregados para realizar tanto o registro quanto a descoberta e a seleção de módulos de sensoriamento. Em geral, esses mecanismos compreendem o uso de sistemas de módulos dinâmicos, como o OSGi ou serviços especializados para registro e descoberta de novos módulos, discutidos anteriormente.

No que diz respeito ao gerenciamento do ciclo de vida de módulos de sensoriamento, no Context Toolkit, por ser uma arquitetura distribuída, todo o gerenciamento de ciclo de vida é viabilizado pelo componente *Discoverer*. Ele é quem se encarrega de disponibilizar aos outros componentes do *framework* as referências de cada *Context Widget* e também notificar cada componente quando um *widget* foi desativado. Porém, a exemplo do *framework* Hydrogen, não há ativação e desativação dinâmicas. Por isso, o *framework* não dispõe de mecanismos para carregar módulos de sensoriamento à medida em que eles são necessários e descarregá-los em outros momentos, visando poupar recursos do sistema, como memória, energia e processamento. Em relação ao Contory, através dos componentes *AccessController* e dos *Reference Modules*, esse *middleware* pode, à medida em que for necessário, ativar e desativar fontes de contexto dinamicamente, de modo a manter, em memória, apenas aquelas mais recentemente utilizadas.

O ConBus permite tanto a instalação e desinstalação, quanto a ativação e desativação dinâmicas dos módulos de sensoriamento, representados pelos *ConUnits*. Como já discutido no Capítulo 3, quando o ConBus é iniciado, nenhum módulo de sensoriamento encontra-se ativo. A ativação de um módulo só acontece quando este é requisitado por alguma aplicação. A partir desse momento, esse módulo permanecerá ativo enquanto for útil. Após um determinado período de tempo de ociosidade, o *Gerenciador de Contexto* irá desativá-lo, visando poupar recursos do dispositivo móvel, como memória, processamento e energia.

Conclusões

É notória a importância, tanto da computação sensível ao contexto quanto da computação móvel, para a evolução da computação ubíqua, visto que o uso da sensibilidade ao contexto permite desenvolver aplicações móveis mais adaptadas ao ambiente ubíquo em que estão inseridas. Essa importância se deve principalmente à popularização e evolução dos dispositivos móveis, tecnologias de redes sem fio e sensores. Entretanto, desenvolver aplicações móveis sensíveis ao contexto ainda é uma tarefa complexa devido, principalmente, à grande quantidade de informações de contexto e de tecnologias de sensoriamento, à falta de padronização das interfaces de acesso aos dados dos sensores e à complexidade envolvida tanto no desenvolvimento e reutilização quanto na implantação e gerenciamento do ciclo de vida de módulos de sensoriamento.

Por isso, o emprego de infraestruturas, como *frameworks*, *toolkits* e *middleware* para provisão de contexto é essencial para diminuir as dificuldades encontradas pelos desenvolvedores de aplicações móveis sensíveis ao contexto. Isso acontece porque essas infraestruturas podem favorecer tanto a criação quanto a reutilização de estruturas e mecanismos empregados para coletar, processar, armazenar e disponibilizar informações contextuais utilizadas por tais aplicações.

Porém, desenvolver essas infraestruturas é uma tarefa complexa, principalmente em relação à integração de módulos de sensoriamento a tais plataformas. Por essa razão, é essencial que as plataformas de provisão de contexto ofereçam mecanismos para facilitar a integração de tais módulos, desde a criação e reutilização desses módulos, passando pela sua implantação junto à própria infraestrutura, até o completo gerenciamento de seu ciclo de vida.

Com o objetivo de auxiliar na solução desses problemas, este trabalho apresentou o ConBus, uma arquitetura de *middleware* para provisão de contexto em dispositivos móveis que implementa estratégias de desenvolvimento, reutilização, implantação e gerenciamento dinâmico do ciclo de vida de módulos de sensoriamento. Para isso, analisamos e identificamos vários requisitos relacionados à integração de módulos de sensoriamento a plataformas de *middleware* de provisão de contexto. A partir das categorias de requisitos discutidas no Capítulo 2, projetamos a arquitetura ConBus, descrita no Capítulo 3, cuja

implementação e estudos de caso foram analisados nos Capítulos 4 e 5, respectivamente.

7.1 Contribuições

Projeto e implementação de uma arquitetura de provisão de contexto para dispositivos móveis. Apesar de ainda não contemplar todos os requisitos discutidos no Capítulo 2, a arquitetura ConBus oferece facilidades concretas para permitir a integração de sensores/módulos de sensoriamento à plataforma de *middleware* e, consequentemente, a posterior disponibilização das informações de contexto fornecidas por essas fontes de contexto às aplicações móveis sensíveis ao contexto.

Projeto arquitetural e implementação do *framework ConUnit*. A partir dos requisitos levantados no Capítulo 2, foi possível desenvolver as bases para o projeto arquitetural do *framework ConUnit*, cujos objetivos principais foram o fornecimento de mecanismos e estruturas que facilitem o desenvolvimento e reutilização de módulos de sensoriamento, bem como sua implantação junto ao *middleware* ConBus. Em relação ao desenvolvimento, foi criado, além do *framework* em si, um software, o **ConUnit Builder**, que facilita o processo de criação de *ConUnits*. A forma como um *ConUnit* é desenvolvido e disponibilizado, através da modularização viabilizada pelo uso de *bundles OSGi* como unidade envoltória de um *ConUnit*, também facilita a reutilização de módulos de sensoriamento, visto que há, para cada *ConUnit*, um arquivo JAR independente que pode ser inserido em qualquer dispositivo móvel que possa executar o *middleware* ConBus. O processo de implantação desses módulos junto ao *middleware* também é simplificado e facilitado, pois, para acoplá-los à plataforma, é necessário apenas copiar os arquivos JAR que os representam para uma determinada pasta no sistema de arquivos do dispositivo móvel correspondente.

Uso de mecanismos que permitem o acoplamento dinâmico de módulos de sensoriamento. O *middleware* ConBus permite que novos componentes sejam acoplados dinamicamente, sem interromper a execução dos outros componentes da plataforma, facilitando tanto as instalações e desinstalações quanto as ativações, desativações e atualizações de módulos de sensoriamento ou qualquer outro componente da arquitetura. Isto é viabilizado, em parte, pelo emprego do *framework OSGi* como sistema de gerenciamento de componentes da arquitetura, mas também pela criação de estruturas auxiliares, como o *framework ConUnit* e o sistema de gerenciamento de contexto, que viabilizam a criação de componentes e controlam sua execução dentro do *middleware*.

Mecanismos para gerenciar dinamicamente o ciclo de vida de módulos de sensoriamento. O gerenciamento dinâmico do ciclo de vida de módulos de sensoriamento é especialmente importante em infraestruturas de provisão de contexto para dispositivos móveis, devido principalmente à carência de recursos computacionais desses dispositivos. A arquitetura ConBus permite tal gerenciamento, controlando a ativação e a desativação de módulos de sensoriamento de acordo com as necessidades das aplicações sensíveis ao contexto. Conforme visto no Capítulo 3, inicialmente, cada módulo de sensoriamento se encontra inativo. O ConBus ativa cada módulo na medida em que eles são necessários às aplicações e os desativa sempre que ficam determinados períodos de tempo ociosos.

Levantamento dos principais problemas e seus respectivos requisitos relativos à integração de módulos de sensoriamento a plataformas de provisão de contexto. A partir desse levantamento dos principais problemas encontrados para se integrar módulos de sensoriamento a plataformas de provisão de contexto, feito no Capítulo 2, foi possível identificar os requisitos que nortearam nossa pesquisa. Essa análise tanto dos problemas quanto dos requisitos se torna um importante ponto de partida para futuras pesquisas nessa área.

7.2 Trabalhos Futuros

Como trabalhos futuros, inicialmente, deve ser projetada e implementada a interface assíncrona do *ConUnit*. A implementação dessa interface no próprio *ConUnit* visa melhorar o desempenho do *middleware*, visto que, com a existência dessa interface, o *Gerenciador de Contexto* não necessitará fazer *pollings* constantes em cada *ConUnit* para coletar os dados de contexto, confrontando-os, em seguida, com as *subscriptions* definidas na solicitação de eventos feita pelas aplicações.

Outro trabalho futuro está relacionado à pesquisa e implementação de algoritmos e/ou heurísticas para selecionar, em tempo de execução, o módulo mais adequado quando houver mais de um sensor coletando o mesmo tipo de contexto para uma mesma entidade. Na implementação corrente, quando há mais de um módulo que coleta o mesmo tipo de contexto para uma mesma entidade, o *Gerenciador de Contexto* escolhe o primeiro módulo de sensoriamento que encontrar em sua lista de módulos que satisfaça às necessidades das aplicações que requisitam tal informação contextual. Esse algoritmo evidentemente não garante a melhor escolha possível, pois não leva em consideração questões, como a precisão de cada sensor, a velocidade de atualização dos dados coletados, o formato das informações armazenadas.

Além disso, o projeto e a implementação de componentes interpretadores de contexto deve ser iniciado futuramente, visando investigar como a implantação desses

componentes dentro do *middleware* favorece o desenvolvimento de aplicações móveis sensíveis ao contexto, facilitando a reutilização de tais componentes em outras aplicações e projetos. Esses componentes tem por objetivo a inferência de novas informações contextuais a partir de outras coletadas diretamente de algum *ConUnit* ou de outro *Interpretador*. Por exemplo, um *Interpretador de Contexto* poderia inferir se, em uma determinada sala de um edifício, há uma reunião, naquele momento, a partir de outras informações de contexto, como quais pessoas estão presentes, informações sobre as agendas de compromissos dessas pessoas, o nível de ruído.

Um outro campo de pesquisa que deve merecer maior atenção é o desenvolvimento de repositórios de *ConUnits*. Esses repositórios poderiam ser os locais para a distribuição e o compartilhamento de módulos de sensoriamento (representados pelos *ConUnits* correspondentes), além de facilitar a manutenção e atualização desses *ConUnits*, auxiliando na melhoria da qualidade de tais módulos pelo fato de estarem sendo utilizados e testados por desenvolvedores e pesquisadores de várias partes do mundo e em diferentes projetos.

Outro aspecto que merece ser tratado é a possibilidade de se usar expressões complexas, como aquelas constituídas pelos operadores lógicos E (AND), OU (OR) e NÃO (NOT), para a formação das condições empregadas pelas aplicações para receberem informações contextuais através da interface assíncrona oferecida pelo ConBus.

Além disso, faz-se necessário pesquisar novos algoritmos para o gerenciamento do ciclo de vida de *ConUnits*, de modo a melhorar a eficiência do *middleware* ConBus em relação à ativação e desativação desses componentes. Isso é necessário porque o algoritmo de ativação e desativação atualmente empregado pela implementação não leva em consideração vários fatores importantes, como os custos envolvidos tanto na ativação quanto na desativação de *ConUnits*, nem a real necessidade de poupar recursos computacionais, como memória, processamento e energia.

Referências Bibliográficas

- [1] ALLIANCE., O. **Osgi - the dynamic module system for java.** disponível em: <http://www.osgi.org>, pesquisado em 11/10/2009, 2009.
- [2] BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. **A survey on context-aware systems.** *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, 2007.
- [3] CHEN, H. L. **An intelligent broker architecture for context-aware systems.** Unpublished phd thesis, University of Maryland, Department of Computer Science and Electrical Engineering, 2005.
- [4] DA SILVA JUNIOR, A. J. **ContextGrid: Uma infra-estrutura de suporte a contexto para integração de dispositivos móveis à computação em grade.** Dissertação de mestrado, Universidade Federal de Goiás/UFG, Instituto de Informática, Julho 2006.
- [5] DELFIACO, L. M. **SmartTravel - Uma aplicação de notificação de pontos turísticos baseada na localização de usuários móveis.** Relatório técnico, Universidade Federal de Goiás/UFG, Instituto de Informática, Dezembro 2009.
- [6] DEY, A.; SALBER, D.; ABOWD, G. **A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications.** *Human-Computer Interaction Journal (HCI)*, 16(2-4):97–166, 2001.
- [7] GU, T.; PUNG, H. K.; ZHANG, D. Q. **A service-oriented middleware for building context-aware services.** *J. Netw. Comput. Appl.*, 28(1):1–18, 2005.
- [8] HENRICKSEN, K.; INDULSKA, J. **Developing context-aware pervasive computing applications: Models and approach.** *Pervasive and Mobile Computing*, 2(1):37–64, February 2006.
- [9] HOFER, T.; SCHWINGER, W.; PICHLER, M.; LEONHARTSBERGER, G.; ALTMANN, J.; RETSCHITZEGGER, W. **Context-awareness on mobile devices - the hydrogen approach.** In: *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, 2003.
- [10] HONG, C. J. **The context fabric: An infrastructure for context-aware**, 2002.

- [11] HONG, J. I.; LANDAY, J. A. **An architecture for privacy-sensitive ubiquitous computing.** In: *MobiSYS '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, p. 177–189. ACM Press, 2004.
- [12] RANGANATHAN, A.; CAMPBELL, R. H. **A middleware for context-aware agents in ubiquitous computing environments.** In: *Middleware '03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, p. 143–161, New York, NY, USA, 2003. Springer-Verlag New York, Inc.
- [13] RIVA, O.; DI FLORA, C. **Contory: A smart phone middleware supporting multiple context provisioning strategies.** In: *ICDCS Workshops*, p. 68, 2006.
- [14] RIVA, O. **Contory: A middleware for the provisioning of context information on smart phones.** In: *ACM/IFIP/USENIX 7th International Middleware Conference (Middleware'06)*, Melbourne (Australia), November 2006.
- [15] ROMÁN, M.; HESS, C.; CERQUEIRA, R.; RANGANATHAN, A.; CAMPBELL, R. H.; NAHRSTEDT, K. **Gaia: a middleware platform for active spaces.** *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):65–67, 2002.
- [16] SACRAMENTO, V.; ENDLER, M.; RUBINSZTEJN, H. K.; LIMA, L. S.; GONCALVES, K.; NASCIMENTO, F. N.; BUENO, G. A. **Moca: A middleware for developing collaborative applications for mobile users.** *IEEE Distributed Systems Online*, 5(10):2, 2004.
- [17] SALBER, D.; DEY, A. K.; ABOWD, G. D. **The context toolkit: aiding the development of context-enabled applications.** In: *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, p. 434–441, New York, NY, USA, 1999. ACM.
- [18] SUN MICROSYSTEMS. **JSR 179 location API for J2ME**, 2009.
- [19] TREADWELL, J.; KUMAR, R.; VITALE, P.; PATHAK, J.; PATHAK, J.; FRATICELLI, O.; FRATICELLI, O. **A framework for dynamic resource management on the grid.** *HP Laboratories Palo Alto*, 2005153, 2005.
- [20] WEISER, M. **The computer for the twenty-first century.** *Scientific American*, p. 94–100, september 1991.
- [21] ZHOU, L.; NAIXUE, X.; SHU, L.; VASILAKOS, A.; YEO, S.-S. **Context-aware middleware for multimedia services in heterogeneous networks.** *IEEE Intelligent Systems*, 99(PrePrints), 2009.

API das Aplicações

A.1 API das Aplicações

A API das Aplicações oferece as formas de comunicação possíveis entre as aplicações sensíveis ao contexto e o *middleware* ConBus. Essa API disponibiliza os comandos através dos quais as informações de contexto podem ser requisitadas pelas aplicações, estabelecendo o formato, a sintaxe e a codificação tanto das requisições quanto das respostas, bem como os tipos de erros mais comuns para cada requisição feita ao *middleware*.

Há duas formas básicas de comunicação possíveis disponibilizadas pela *API das aplicações*: (i) através de *interface síncrona*, em que é possível consultar qualquer informação contextual gerenciada pelo ConBus e coletada pelos sensores acoplados; ou ainda, (ii) por meio de *interface assíncrona*, baseada em eventos, em que se pode registrar interesse em obter determinado tipo de informação de contexto sempre que condições específicas ocorrerem.

A seguir, são descritos todos os comandos disponibilizados pela API, bem como o formato e a sintaxe das requisições e das respostas, além de possíveis mensagens de erro. Para enviar a resposta relativa a qualquer comando disponibilizado pela *API das Aplicações*, o ConBus utilizará um arquivo XML, denominado *response.xml*. Esse arquivo irá variar seu conteúdo de acordo com o tipo de comando enviado, as mensagens de erro encontradas e o tipo de informação desejado pelas aplicações sensíveis ao contexto.

getAllContext. Através deste comando, tanto as aplicações quanto os próprios desenvolvedores de aplicações sensíveis ao contexto podem obter informações sobre todas as fontes de contexto (módulos de sensoriamento (através dos *ConUnits*)) acopladas ao ConBus em um dado instante. Isso é importante, por exemplo, quando se deseja utilizar um módulo de sensoriamento desenvolvido por terceiros e não se tem nenhuma documentação sobre seu uso. A seguir são exibidas todas as informações necessárias para a sua correta utilização, como o formato do arquivo de resposta, principais erros e um exemplo de uso do comando.

Parâmetros: Para este comando, não há parâmetros de entrada.

Resposta: A resposta para este comando é enviada, pelo ConBus, através do arquivo *response.xml*, cujo modelo é exibido em seguida:

```
<?xml version="1.0"?>
  <response>
    <error>
      <errorType>NONE</errorType>
      <errorCode>000</errorCode>
    </error>
    <contextSource>
      <number>00001</number>
      <contextType>LOCATION</contextType>
      <entityName>JOHN_SMITH</entityName>
      <entityType>PERSON</entityType>
      <accuracy>NOT AVAILABLE</accuracy>
      <sensorModel>NOT AVAILABLE</sensorModel>
      <reasoning>FALSE</reasoning>
      <isLocal>TRUE</isLocal>
      <contextData>
        <typeOfValue>
          <typeOfValue>LATITUDE</typeOfValue>
          <typeOfValue>LONGITUDE</typeOfValue>
          <typeOfValue>ALTITUDE</typeOfValue>
        </typeOfValue>
        <unit>
          <unit>DEGREE</unit>
          <unit>DEGREE</unit>
          <unit>METER</unit>
        </unit>
        <dataType>
          <dataType>DOUBLE</dataType>
          <dataType>DOUBLE</dataType>
          <dataType>FLOAT</dataType>
        </dataType>
      </contextData>
    </contextSource>
    .
    .
    .
    <contextSource>
      <number>00002</number>
      <contextType>TEMPERATURE</contextType>
      <entityName>ROOM_111</entityName>
      <entityType>PLACE</entityType>
      <accuracy>NOT AVAILABLE</accuracy>
```

```

<sensorModel>NOT AVAILABLE</sensorModel>
<reasoning>FALSE</reasoning>
<isLocal>FALSE</isLocal>
<contextData>
  <typeOfValue>
    <typeOfValue>TEMPERATURE</typeOfValue>
  </typeOfValue>
  <unit>
    <unit>CELSIUS</unit>
  </unit>
  <dataType>
    <dataType>FLOAT</dataType>
  </dataType>
</contextData>
</contextSource>
.
.
.
</response>

```

Erros mais comuns: Na versão corrente do ConBus, apenas o erro descrito a seguir é tratado para o comando *getAllContext* ¹:

- **Command Not Found** (ERROR CODE = 001): Esta mensagem será enviada, pelo ConBus, através da *tag* <error> do arquivo de resposta *response.xml*, sempre que não for possível identificar o comando *getAllContext* devido a erros léxicos na requisição HTTP.

Exemplo de uso: Um exemplo de uso para o comando *getAllContext* é exibido a seguir:

`http://localhost:8585/getAllContext`

getContext. Este comando permite às aplicações consultarem, em tempo real, informações de contexto específicas gerenciadas pelo ConBus.

Parâmetros de entrada: Para este comando, há tanto parâmetros obrigatórios quanto opcionais.

- **contextType** (*string*) [OBRIGATÓRIO]: indica qual o tipo de contexto (LOCATION, ENERGY_LEVEL, DATE_TIME, TEMPERATURE, MEETINGS, etc.) desejado.
- **entityName** (*string*) [OBRIGATÓRIO]: indica o nome da entidade (JOHN_SMITH, ROOM_113, ANDROID_DEVICE_1218, etc.) à qual esta informação de contexto está relacionada.

¹ Porém, deve-se observar que este é um erro geral e que pode ocorrer com o uso de qualquer comando disponibilizado pela API.

- **entityType** (*string*) [OBRIGATÓRIO]: indica o tipo da entidade (PERSON, OBJECT, PLACE, etc.) à qual esta informação de contexto está relacionada.
- **sensorType** (*string*) [OPCIONAL]: indica de qual sensor específico a informação deve ser coletada. Isso pode ser importante quando há dois sensores diferentes coletando o mesmo tipo de contexto de uma mesma entidade, como, por exemplo, dois sensores de localização, sendo um para ambientes externos (receptor GPS) e outro para ambientes internos (PlaceLab) coletando informações sobre a localização corrente uma determinada pessoa.

Resposta: A resposta para este comando é enviada, pelo ConBus, através do arquivo *response.xml*, cujo modelo é exibido em seguida:

```
<?xml version="1.0"?>
<response>
  <error>
    <errorType>NONE</errorType>
    <errorCode>000</errorCode>
  </error>
  <contextType>LOCATION</contextType>
  <entityName>JOHN_SMITH</entityName>
  <entityType>PERSON</entityType>
  <timeStamp>27/11/2009 13:17:47 GMT</timeStamp>
  <accuracy>NOT AVAILABLE</accuracy>
  <sensorModel>NOT AVAILABLE</sensorModel>
  <reasoning>FALSE</reasoning>
  <isLocal>TRUE</isLocal>
  <contextData>
    <value>
      <value1>16.3234545</value1>
      <value2>27.8545484</value2>
      <value3>548.12</value3>
    </value>
    <typeOfValue>
      <typeOfValue1>LATITUDE</typeOfValue1>
      <typeOfValue2>LONGITUDE</typeOfValue2>
      <typeOfValue3>ALTITUDE</typeOfValue3>
    </typeOfValue>
    <unit>
      <unit1>DEGREE</unit1>
      <unit2>DEGREE</unit2>
      <unit3>METER</unit3>
    </unit>
  </contextData>
</response>
```

```
<dataType1>DOUBLE</dataType1>
<dataType2>DOUBLE</dataType2>
<dataType3>FLOAT</dataType3>
</dataType>
</contextData>
</response>
```

Erros mais comuns: A seguir são indicados alguns dos erros mais comuns que podem ocorrer durante uma requisição *getContext*²:

- *Command Not Found* (ERROR CODE = 001): Esta mensagem será enviada, pelo ConBus, através da tag <error> do arquivo de resposta *response.xml*, sempre que não for possível identificar o comando *getContext* devido a erros léxicos na requisição HTTP.
- *Incorrect Parameter Number* (ERROR CODE = 002): Esta mensagem será enviada, pelo ConBus, através da tag <error> do arquivo de resposta *response.xml*, sempre que a quantidade de parâmetros de entrada estiver incorreta.
- *Context Type Not Indicated* (ERROR CODE = 003): Esta mensagem será enviada pelo ConBus, através da tag <error> do arquivo de resposta *response.xml*, sempre que não houver nenhuma indicação, na mensagem de requisição, a respeito do tipo de contexto desejado.
- *Entity Name Not Indicated* (ERROR CODE = 004): Esta mensagem será enviada pelo ConBus, através da tag <error> do arquivo de resposta *response.xml*, sempre que não houver nenhuma indicação, na mensagem de requisição, a respeito do nome da entidade à qual a informação contextual desejada está relacionada.
- *Entity Type Not Indicated* (ERROR CODE = 005): Esta mensagem será enviada pelo ConBus, através da tag <error> do arquivo de resposta *response.xml*, sempre que não houver nenhuma indicação, na mensagem de requisição, a respeito do tipo da entidade à qual a informação contextual desejada está relacionada.
- *Context Information Not Exists* (ERROR CODE = 006): Esta mensagem será enviada pelo ConBus, através da tag <error> do arquivo de resposta *response.xml*, sempre que a informação contextual desejada, ou seja, o tipo de contexto desejado relacionado à entidade correspondente não estiver sendo fornecida, naquele momento, por nenhuma fonte de contexto (*ConUnit*).

²Deve-se notar que o ConBus enviará, no arquivo de resposta *response.xml*, apenas a indicação do primeiro erro encontrado.

Exemplo de uso: A seguir é exibido um exemplo de uso para o comando *getContext* pelas aplicações:

```
http://localhost:8585/
```

```
getContext?contextType=TEMPERATURE&entityName=ROOM_111&entityType=PLACE
```

contextSubscribe. Através deste comando, uma aplicação pode registrar interesse em receber determinadas informações contextuais sempre que certos eventos ocorrerem.

Parâmetros de entrada: Para este comando, há tanto parâmetros obrigatórios quanto opcionais.

- **contextType** (*string*) [OBRIGATÓRIO]: indica qual o tipo de contexto (LOCATION, ENERGY_LEVEL, DATE_TIME, TEMPERATURE, MEETINGS, etc.) desejado.
- **entityName** (*string*) [OBRIGATÓRIO]: indica o nome da entidade (JOHN_SMITH, ROOM_113, ANDROID_DEVICE_1218, etc.) à qual esta informação de contexto está relacionada.
- **entityType** (*string*) [OBRIGATÓRIO]: indica o tipo da entidade (PERSON, OBJECT, PLACE, etc.) à qual esta informação de contexto está relacionada.
- **sensorType** (*string*) [OPCIONAL]: indica de qual sensor específico a informação deve ser coletada. Isso pode ser importante quando há dois sensores diferentes coletando o mesmo tipo de contexto de uma mesma entidade, como, por exemplo, dois sensores de localização, sendo um para ambientes externos (receptor GPS) e outro para ambientes internos (PlaceLab) coletando informações sobre a localização corrente uma determinada pessoa.
- **appPort** (*string*) [OBRIGATÓRIO]: indica a porta de comunicação TCP para a qual a informação contextual coletada deverá ser enviada (como o *middleware* executa localmente em relação às aplicações móveis sensíveis ao contexto, não há necessidade de indicar o endereço IP da aplicação destinatária).
- **condition** (*string*) [OBRIGATÓRIO]: indica a condição que deve ser satisfeita para que a informação contextual desejada seja enviada à aplicação interessada. Para cada condição, uma aplicação especifica um nome de atributo, um operador (==, !=, <, >, <=, >=) e um valor. Na versão corrente da implementação, apenas são possíveis múltiplas condições usando o operador lógico AND (indicado aqui pelos símbolos &&). Um exemplo de uso é exibido em seguida: ENERGY_LEVEL < 20 && TEMPERATURE > 34.

Resposta: A resposta para este comando é enviada, pelo ConBus, através do arquivo *response.xml*, cujo modelo é exibido em seguida:

```
<?xml version="1.0"?>
<response>
  <error>
    <errorType>NONE</errorType>
    <errorCode>000</errorCode>
  </error>
</response>
```

Caso não haja nenhum erro, a tag `<error>` do arquivo *response.xml* exibirá a indicação NONE para a subtag `<errorType>` e o código 000 para a subtag `<errorCode>`, indicando que o registro para o evento desejado foi realizado com sucesso.

Erros mais comuns: A seguir são indicados alguns dos erros mais comuns que podem ocorrer durante uma requisição *contextSubscribe*:

- *Command Not Found* (ERROR CODE = 001): Esta mensagem será enviada, pelo ConBus, através da tag `<error>` do arquivo de resposta *response.xml*, sempre que não for possível identificar o comando *contextSubscribe* devido a erros léxicos na requisição HTTP.
- *Incorrect Parameter Number* (ERROR CODE = 002): Esta mensagem será enviada, pelo ConBus, através da tag `<error>` do arquivo de resposta *response.xml*, sempre que a quantidade de parâmetros de entrada estiver incorreta.
- *Context Type Not Indicated* (ERROR CODE = 003): Esta mensagem será enviada pelo ConBus, através da tag `<error>` do arquivo de resposta *response.xml*, sempre que não houver nenhuma indicação, na mensagem de requisição, a respeito do tipo de contexto desejado.
- *Entity Name Not Indicated* (ERROR CODE = 004): Esta mensagem será enviada pelo ConBus, através da tag `<error>` do arquivo de resposta *response.xml*, sempre que não houver nenhuma indicação, na mensagem de requisição, a respeito do nome da entidade à qual a informação contextual desejada está relacionada.
- *Entity Type Not Indicated* (ERROR CODE = 005): Esta mensagem será enviada pelo ConBus, através da tag `<error>` do arquivo de resposta *response.xml*, sempre que não houver nenhuma indicação, na mensagem de requisição, a respeito do tipo da entidade à qual a informação contextual desejada está relacionada.
- *Context Information Not Exists* (ERROR CODE = 006): Esta mensagem será enviada pelo ConBus, através da tag `<error>` do arquivo de resposta

response.xml, sempre que a informação contextual desejada, ou seja, o tipo de contexto desejado relacionado à entidade correspondente não estiver sendo fornecida, naquele momento, por nenhuma fonte de contexto (*ConUnit*).

- *Application Port Not Indicated* (ERROR CODE = 007): Esta mensagem será enviada pelo ConBus, através da *tag* <error> do arquivo de resposta *response.xml*, sempre que não houver nenhuma indicação, na mensagem de requisição, a respeito da porta de comunicação TCP para a qual a informação de contexto desejada deverá ser enviada.
- *Condition Not Indicated* (ERROR CODE = 008): Esta mensagem será enviada pelo ConBus, através da *tag* <error> do arquivo de resposta *response.xml*, sempre que não houver nenhuma indicação, na mensagem de requisição, das condições que devem ser satisfeitas para que o ConBus envie à aplicação uma informação contextual.

Exemplo de uso: A seguir é exibido um exemplo de uso para o comando *contextSubscribe* pelas aplicações:

```
http://localhost:8585/  
contextSubscribe?contextType=ENERGY_LEVEL&entityName=MY_ANDROID_888  
&entityType=OBJECT&appPort=12232  
&condition=ENERGY_LEVEL<15&&TEMPERATURE>40
```

Deve-se ressaltar que, para as aplicações desenvolvidas em Java, é disponibilizada, aos desenvolvedores, uma classe (*ConBusCommunication*) que abstrai os detalhes da comunicação entre essas aplicações e o *middleware*. Ao utilizá-la, as aplicações podem obter as informações contextuais gerenciadas pelo ConBus através de chamadas a métodos simples, como *getContext()*, *getAllContext()* e *contextSubscribe()*, indicando os devidos parâmetros. Esse processo simplifica ainda mais o desenvolvimento dessas aplicações.