UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÀTICA

RODRIGO MANTOVANELI PESSOA

Infraware: Um *Middleware* de Suporte a Aplicações Sensíveis ao Contexto

VITÓRIA
2006

Infraware: Um *Middleware* de Suporte a Aplicações Sensíveis ao Contexto

RODRIGO MANTOVANELI PESSOA

Dissertação submetida ao Programa de Pós-Graduação em Informática do Centro Tecnológico da da Universidade Federal do Espírito Santo, como requisito parcial para a obtenção do grau de Mestre em Informática – Sistemas Distribuídos.

Aprovada em 25/10/2006 por:

Prof. José Gonçalves Pereira Filho (D.Sc.)
Departamento de Informática - UFES
Orientador

Prof. Giancarlo Guizzardi (Ph.D)
Departamento de Informática - UFES

Prof. Clever Ricardo Guareis de Farias (D.Sc.)
Departamento de Física e Matemática - USP-Ribeirão Preto

Prof. Markus Endler (D.Sc.)
Departamento de Informática - PUC-Rio

Vitória, 25 de outubro de 2006

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Central da Universidade Federal do Espírito Santo, ES, Brasil)

Pessoa, Rodrigo Mantovaneli, 1980-
P475i Infraware : um middleware de suporte a aplicações sensíveis ao contexto / Rodrigo Mantovaneli Pessoa. – 2006.
140 f. : il.

Orientador: José Gonçalves Pereira Filho.
Dissertação (mestrado) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Middleware. 2. Ontologia. 3. Inferência (Lógica). I. Pereira Filho, José Gonçalves. II. Universidade Federal do Espírito Santo Centro Tecnológico. III. Título.

CDU: 004

"Na sociedade pós-moderna, fundamental haverá de ser o contexto cultural em que as relações humanas estão inseridas. Este, por sua vez, é simbólico do contexto social e subverte a sintática do imaginário coletivo, dos tabus, dos padrões comportamentais generalizados e das crenças e rituais que caracterizam os principais aspectos de uma cultura."

Marcus Haas

AGRADECIMENTOS

A todos que contribuiram para a realização deste trabalho, fica expressa aqui a minha gratidão, especialmente:

- aos meus pais, aos pais de meus pais e a minha irmã, que me ensinaram mais do que eu jamais pude aprender, pelo apoio e paciência irrestritos durante a realização deste trabalho.
- á minha namorada, Alline Berger de Oliveira, pelo amor, amizade e companheirismo irrestritos, demonstrados em tempos tão difíceis e turbulentos.
- ao Prof. Doutor José Gonçalves Pereira Filho, professor e orientador, pela disponibilidade revelada ao longo de sua orientação, pela amizade estabelecida, pelos constantes estímulos e pelas críticas e sugestões feitas durante a elaboração desta dissertação;
- aos membros da banca: Prof. Dr. Giancarlo Guizzardi, Prof. Dr. Clever Ricardo Guareis de Farias e Prof. Dr. Markus Endler, pela presença, atenção e sugestões;
- ao Prof. Dr. Álvaro Cesar Pereira Barbosa, pelas valiosas discussões e sugestões;
- ao Camilo Zardo Calvi, jovem membro do Laboratório de Pesquisa em Redes e Multimídia, pela amizade construída ao longo deste trabalho e pelas reflexões diárias que nortearam boa parte das definições aqui estabelecidas.
- a todos os demais companheiros do Laboratório de Pesquisa em Redes e Multimídia, que tanto ajudaram no desenvolvimento dessa dissertação.
- à Flavia Carpanedo Monteiro, pelas muitas xícaras de café saboreadas no meio da tarde e incondicional amizade.
- à Coordenação de Aperfeiçoamento de Pessoal de Estudo Superior (CAPES), pela concessão da bolsa de estudo.
- à Fundação de Apoio à Ciência e Tecnologia do Espírito Santo (FAPES), pela concessão de auxílio à pesquisa (processo nº 30897041/2005) possibilitando a execução deste projeto.
- ao Programa de Pós-Graduação em Informática (PPGI) e ao Departamento de Informática (DI/UFES), pelo apoio logístico.

SUMÁRIO

RESUMO 10
ABSTRACT 11
1. Introdução 12
1.1. Objetivos 14
1.2. Metodologia 15
1.3. Organização da Dissertação 15
2. Plataformas de Suporte a Aplicações Sensíveis ao Contexto 16
2.1. Context Toolkit 18
2.2. Solar System 21
2.3. SOCAM 24
2.4. GaiaOS 27
2.5. Aura 29
2.6. CoBrA 32
2.7. WASP – Web Architectures for Service Platforms 34
2.8. Conclusão do Capítulo 38
3. Avaliação de Requisitos 39
3.1. Generalidade da Informação Contextual 39
3.2. Linguagem de Subscrição 40
3.3. Modelagem Contextual 41
3.4. Interface Com Sensores 42
3.5. Incerteza e Confiabilidade das Informações Contextuais 43
3.6. Tecnologias de Distribuição e Integração 44
3.7. Segurança e Privacidade 45
3.8. Escalabilidade 46
3.9. Descoberta de Serviços 46
3.10. Personalização dos Serviços Para os Usuários 47
3.11. Coordenação Entre Aplicações 48
3.12. Persistência de Dados Contextuais 49
3.13. Conclusão do Capítulo 50
4. A Plataforma Infraware 51
4.1. Repositórios 53
4.2. Gerente de Subscrição 54
4.3. Controle de Acesso e Privacidade 56
4.4. Interpretador de Contexto 58
4.5. Acesso e Integração de Dados 59
4.6. Gerente de Serviços 60
4.7. Coordenador 63
4.8. Um Cenário de Uso para a Plataforma Infraware 65
4.9. Conclusão do Capítulo 67
5. Modelagem e Interpretação de Contexto 68
5.1. Modelos de Pares Atributo-valor 68
5.2. Modelos Baseados em Esquemas 69
5.3. Modelos Gráficos 70
5.4. Modelos Baseados em Objetos 73
5.5. Modelos Baseados em Lógica 75

5.6. Modelos Baseados em Ontologias 76
5.7. Avaliação dos Modelos Analisados 78
5.8. Conclusão do Capítulo 79
6. O Interpretador de Contexto 80
6.1. Interpretação de Contexto 80
6.2. Inferências e Ontologias 81
6.3. Linguagem para Representação de Ontologias 82
6.4. Modelagem de Contexto Com Ontologias 84
6.5. Arquitetura Conceitual do Interpretador de Contexto 87
6.6. Implementação 89
6.6.1. Máquina de Inferência 91
6.6.2. Generic Rule Language 93
6.6.3. RDF Data Query Language 95
6.6.4. Interface de Programação 96
6.6.5. Interface de Depuração 98
6.7. Cenário de Aplicação 101
6.7.1. Cenário Turístico 102
6.8. Trabalhos Relacionados 107
6.9. Conclusão do Capítulo 109
7. Conclusões e Perspectivas Futuras 110
8. Referências 112
Apêndice A: Código Fonte – Interface de Programação 118
Apêndice B: Código Fonte – Interface de Depuração 123
Apêndice C: Ontologia de Turismo – OWL 137

LISTA DE FIGURAS

Figura 1-1 – Aplicações tradicionais 12
Figura 1-2 – Aplicações sensíveis ao contexto 13
Figura 2-1 - Exemplo de configuração dos componentes do Context Toolkit. 20
Figura 2-2 – Quatro tipos de operadores: Transformador (T), Filtro (F), Merger (M) e Agregador (A) 22
Figura 2-3 – Um exemplo de grafo de operadores. 23
Figura 2-4 – A arquitetura do Solar System. 24
Figura 2-5 – Modelo Ontológico da Arquitetura SOCAM 25
Figura 2-6 – Arquitetura SOCAM. 26
Figura 2-7 - Arquitetura em Camadas do GaioOS. 27
Figura 2-8 - Diagrama esquemático do framework de aplicação do GaiaOS. 28
Figura 2-9 – A Plataforma Aura 30
Figura 2-10 – Arquitetura Interna da Camada Prism 31
Figura 2-11 – Ontologia SOUPA 33
Figura 2-12 – Arquitetura CoBrA 34
Figura 2-13 - Plataforma WASP e seus relacionamentos 35
Figura 2-14 - Arquitetura da plataforma WASP. 36
Figura 2-15 – Arquitetura WASP estendida para a utilização de serviços semânticos 37
Figura 4-1 – Arquitetura Geral da Plataforma Infraware 52
Figura 4-2 – O Gerente de Subscrição 55
Figura 4-3 – Módulo de Controle de Acesso e Privacidade 57
Figura 4-4 – Componente de Acesso e Integração de Dados. 59
Figura 4-5 – O Gerente de Serviços. 61
Figura 4-6 – O Componente Coordenador. 63
Figura 4-7 – Utilização da plataforma Infraware em um cenário médico-hospitalar 66
Figura 5-1 – Exemplo de modelagem de contexto adotada pelo Context kernel 70
Figura 5-2 – Exemplo de modelo gráfico construído a partir de extensões à ORM 72
Figura 5-3 – Trecho de código exemplificando a modelagem baseada em objetos do JCAF. 74
Figura 5-4 – Trecho de código da COBRA-ONT Location Ontology 77
Figura 5-5 – Avaliação dos Modelos de Contexto Analisados 79
Figura 6-1 – Um exemplo simples de ontologia 82
Figura 6-2 - Ontologia de Domínio descrevendo as relações funcionários de um hospital e pacientes 85
Figura 6-3 – Projeto Conceitual do Interpretador de Contexto 88
Figura 6-4 – Exemplo de sentença lógica transformada em regra 89
Figura 6-5 – Diagrama de componentes do Interpretador de Contexto 90
Figura 6-6 – Notação informal e simplificada da Generic Rule Language 94
Figura 6-7 – Exemplo de regra de produção para a introdução de novos fatos. 94
Figura 6-8 – Exemplo de regra de produção para a invocação de um serviço. 95
Figura 6-9 – Exemplo de consulta em RDQL 96
Figura 6-10 – Diagrama de classes utilizadas pelo Interpretador de Contexto 97
Figura 6-11 – Carga das Ontologias de Domínio 99
Figura 6-12 – Carga dos indivíduos instanciados 99
Figura 6-13 – Consulta aos fatos conhecidos e inferidos pelo Interpretador 100
Figura 6-14 – Execução de uma consulta em RDQL 100
Figura 6-15 – Envio de informações contextuais e subscrições 101

Figura 6-16 – Ontologia simplificada para o domínio do turismo 103
Figura 6-17 – Regra para o envio de informações turísticas através do serviço SMS 104
Figura 6-18 – Regra para o envio de informações turísticas através do serviço MMS 104
Figura 6-19 – Recebimento de informações turísticas em diferentes dispositivos..... 105
Figura 6-20 – Consulta aos pontos turísticos de interesse de um usuário 106
Figura 6-21 – Sugestões para visitas a pontos turísticos baseadas nas preferências dos usuários 106

LISTA DE ACRÔNIMOS

API	*Application Programming Interface*
CC/PP	*Composite Capabilities/Preferences Profile*
DAML	*DARPA Agent Markup Language*
GPS	*Global Positioning System*
GRL	*Generic Rule Language*
GRR	*Generic Rule Reasoner*
OCL	*Object Constraint Language*
OIL	*Ontology Inference Layer*
ORM	*Object-Role Modeling*
OWL	*Ontology Web Language*
PDA	*Personal Digital Assistant*
RDF	*Resource Description Framework*
RDFS	*Resource Description Framework-Schema*
RDQL	*Resource Description Query Language*
SGML	*Standard Generalized Markup Language*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
W3C	*World Wide Web Consortium*
XML	*eXtensible Markup Language*

RESUMO

O desenvolvimento de novas tecnologias de computação móvel e o crescente emprego de dispositivos portáteis têm tornado a computação cada vez mais presente na realização de diversas atividades humanas, favorecendo o surgimento de um novo paradigma computacional: a Computação Ubíqua. Nesse novo cenário, destacam-se as aplicações móveis sensíveis ao contexto, que aprimoram a interação com os seus usuários ao se beneficiarem do uso de informações contextuais.

Pesquisas recentes destacam a necessidade do desenvolvimento de infra-estruturas especializadas de *middleware* para o gerenciamento de informações contextuais dinâmicas. Essas novas infra-estruturas fornecem facilidades adequadas ao desenvolvimento de uma ampla variedade de aplicações móveis, sensíveis ao contexto, em domínios diversos.

Este trabalho propõe uma arquitetura de *middleware* para suporte ao desenvolvimento e execução de aplicações móveis sensíveis ao contexto. A arquitetura é definida a partir do levantamento de requisitos funcionais representativos para essa nova classe de aplicações. Aspectos relacionados aos principais componentes da arquitetura são apresentados. O trabalho também apresenta o projeto e implementação de um dos componentes essenciais da arquitetura proposta, responsável pela inferência de novas informações contextuais e pela percepção de contexto.

ABSTRACT

New mobile computing technologies and the increasing use of portable devices make computers more present on human life activities with each passing day. This favors the rising of a new computing paradigm: the Ubiquitous Computing. In this scenario, context-aware mobile applications stand out among others, improving user interaction by supporting new adaptive behavior according to context changes.

Recent research points out the need for developing specialized middleware infrastructure for the management of dynamic contextual information. These new infrastructures provide the adequate facilities for the development of a large range of context-aware mobile applications, in several domains.

This work proposes the design of a middleware to support the development and execution of context-aware applications. The architecture is defined by using relevant functional requirements of this new class of applications. Related aspects of the middleware's main components are presented. This work also presents the design and implementation of an essential component of the proposed middleware, responsable for the inference of new context information and context perception.

1. Introdução

Idealizado como uma mudança no paradigma de interação entre usuários e computadores, o termo Computação Ubíqua (*Ubiquitous Computing*), introduzido por Mark Weiser no início dos anos 90, referencia a incorporação simbiótica e implícita da computação no cotidiano dos usuários. Weiser introduziu o conceito de Computação Ubíqua ao vislumbrar novos sistemas e ambientes acrescidos de recursos computacionais capazes de prover serviços e informações quando e onde sejam desejados pelos usuários ("*everywhere, everytime computing*") [Weiser, 1991]. Weiser propõe, assim, uma integração contínua entre ambiente e tecnologia na tarefa de auxiliar os usuários nas suas mais variadas atividades cotidianas.

Dentre as novas classes de aplicações existentes no cenário da Computação Ubíqua, destacam-se as Aplicações Móveis Sensíveis ao Contexto (*Context-Aware Mobile Applications*). Essas aplicações tentam explorar as mudanças de contexto ocorridas dentro de um domínio dinâmico para aprimorar a interação com seus usuários. Diferentemente das aplicações tradicionais, que recebem entradas explícitas, executam algum processamento a partir do qual são gerados os resultados de suas saídas (como mostra a Figura 1-1), as aplicações sensíveis ao contexto são programadas para levar em consideração em seus processamentos, não apenas as entradas de dados fornecidas explicitamente pelos usuários, mas também entradas implícitas, referentes ao contexto físico e computacional dos usuários e dos ambientes que os cercam. Tal situação é ilustrada através da Figura 1-2.

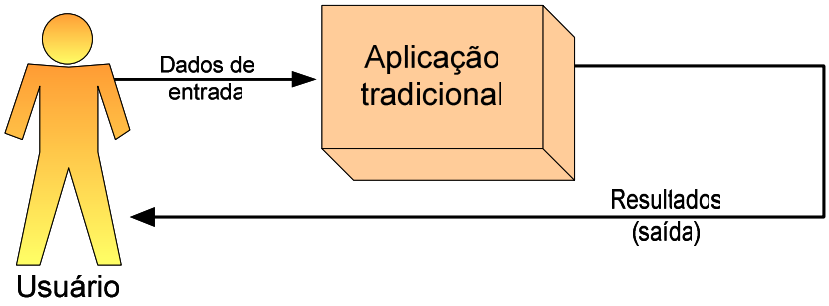

Figura 1-1 – Aplicações tradicionais

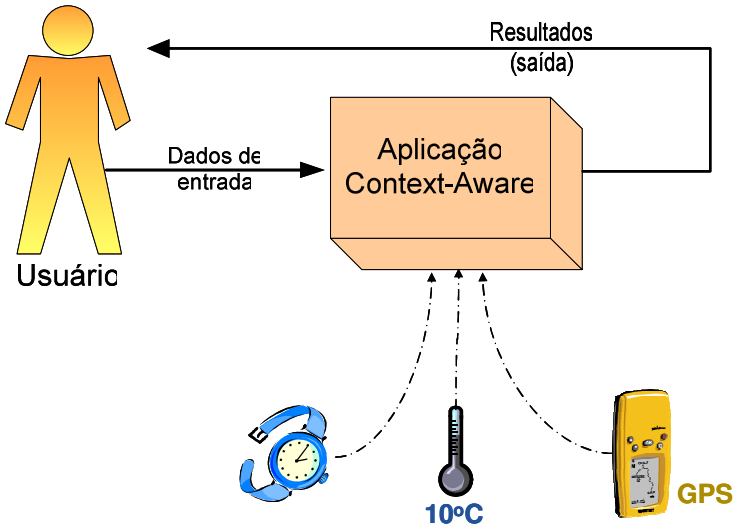

Figura 1-2 – Aplicações sensíveis ao contexto

De acordo com Dey [Dey, 2000], contexto é "*qualquer informação que possa ser utilizada para caracterizar a situação de uma entidade*", onde uma entidade pode ser uma pessoa, lugar ou objeto considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a própria aplicação. Exemplos típicos de contexto são localização, tempo, temperatura e estados de objetos computacionais e físicos. Assim, ao invés de tratar a mobilidade como um problema, as aplicações sensíveis ao contexto exploram a natureza contextual provocada pela mobilidade do usuário, com a clara intenção de produzir serviços mais flexíveis, adaptáveis, ricos em funcionalidade e centrados no usuário.

Entretanto, em que pese os seus benefícios e potencial de uso em domínios dos mais variados (Saúde, Entretenimento, Transportes, Segurança, Turismo, etc.), o desenvolvimento de aplicações móveis sensíveis ao contexto em larga escala esbarra em alguns desafios em conseqüência da natureza complexa das informações contextuais. Uma lista não exaustiva desses desafios inclui: (*i*) a definição de um modelo adequado de contexto; (*ii*) o suporte à adaptabilidade das aplicações; (*iii*) a resolução de conflitos e a suporte à colaboração entre aplicações; (*iv*) a descoberta, seleção e composição dinâmica de serviços; (*v*) o estabelecimento de mecanismos de controle de privacidade e segurança das informações contextuais; (*vi*) a definição de estratégias de aquisição, armazenamento e interpretação de contexto; (*vii*) o acesso e a integração de dados provenientes de sensores e dispositivos distribuídos e heterogêneos; (*xiii*) o apoio à personalização de serviços; (*ix*) a utilização de mecanismos de persistência dos dados contextuais; (*x*) o uso adequado de linguagens de subscrição e de solicitação de serviços; além de muitos outros.

Estes e outros desafios motivaram o projeto Infraware [Infraware, 2005] a propor e desenvolver uma plataforma (ou *middleware*) de serviços para suportar o desenvolvimento e a execução de aplicações móveis sensíveis ao contexto. A arquitetura da plataforma deverá facilitar a concepção dinâmica de novas aplicações, abstraindo a complexidade da manipulação de informações contextuais.

1.1. Objetivos

O presente trabalho tem como um de seus objetivos propor a versão inicial da arquitetura conceitual da plataforma Infraware. Exemplos de funcionalidades desejadas para este *middleware* incluem um modelo de contexto, um mecanismo de interpretação de contexto, um gerente de serviços semânticos, uma política de privacidade e segurança das informações, uma interface uniforme com os sensores e as diversas outras fontes de dados contextuais, e um módulo gerente de subscrições. Tais funcionalidades vão ao encontro da tarefa de se investigar e propor soluções para questões fundamentais que devem ser tratadas por esta plataforma no que se refere ao suporte às aplicações ubíquas reais.

A implementação deste *middleware* constitui uma tarefa multidisciplinar, envolvendo pesquisas e conhecimentos de diversas áreas como: Integração de Dados Distribuídos, Engenharia de Software, Engenharia de Ontologias, Engenharia de Protocolos, Inteligência Artificial, Redes de Sensores e Sistemas Adaptativos. Dada a natureza complexa e multidisciplinar do *middleware* proposto, este trabalho concentra-se inicialmente no projeto e implementação do Interpretador de Contexto, um dos componentes mais essenciais deste *middleware*, cuja função é a interpretação semântica dos dados adquiridos dos sensores e dispositivos dedicados à aquisição das informações contextuais, com a finalidade de torná-los disponíveis para as aplicações, de forma transparente.

Como prova de conceito, este trabalho também se propõe a desenvolver algumas aplicações-piloto sensíveis ao contexto. O protótipo deverá simular as funcionalidades referentes aos módulos ainda não implementados da plataforma Infraware além de validar a estratégia adotada no projeto e implementação do componente responsável pela interpretação de contexto.

1.2. Metodologia

A metodologia empregada no desenvolvimento do trabalho incluiu atividades regulares de estudos e investigação científica na linha principal do projeto (computação sensível ao contexto). Algumas plataformas de serviço sensíveis ao contexto foram analisadas e avaliadas e serviram como base para a atividade de levantamento de requisitos da plataforma Infraware. A partir dos requisitos identificados foi especificada a arquitetura conceitual da plataforma. O componente responsável pela interpretação de contexto foi o módulo da plataforma escolhido para ter sua especificação interna projetada e implementada. Por fim, foram desenvolvidas aplicações piloto como prova de conceito para a validação do trabalho concebido.

1.3. Organização da Dissertação

O restante desta dissertação está organizado da seguinte forma: a Seção 2 apresenta alguns trabalhos relacionados ao desenvolvimento de plataformas sensíveis ao contexto; a Seção 3 introduz alguns requisitos e desafios no desenvolvimento desse tipo de plataforma; a Seção 4 descreve a arquitetura da plataforma Infraware; a Seção 5 apresenta técnicas referenciadas na literatura para a modelagem de contexto; a Seção 6 discute o projeto e implementação do Interpretador de Contexto; e, finalmente, a Seção 7 apresenta as considerações finais e discute perspectivas de trabalhos futuros.

2. Plataformas de Suporte a Aplicações Sensíveis ao Contexto

Esta Seção apresenta algumas iniciativas de projetos de plataformas sensíveis ao contexto referenciadas na literatura da área e utilizadas como referência para o projeto da plataforma Infraware. Essas plataformas visam fornecer suporte arquitetural adequado ao desenvolvimento de aplicações sensíveis ao contexto, fornecendo serviços, mecanismos e interfaces que abstraiam a complexidade da aquisição, manipulação e uso das informações contextuais por parte das aplicações.

Os seguintes projetos são apresentados nas próximas seções:

- ***Context Toolkit*:** resultado de trabalhos desenvolvidos na *Georgia Institute of Technology* o *Context Tollkit* [Dey, 2000] é um *framework* conceitual que utiliza um conjunto de ferramentas de suporte ao desenvolvimento de aplicações sensíveis ao contexto. Este *framework* é um dos primeiros trabalhos citados na literatura que busca aproveitar as informações provenientes do contexto dos usuários no processamento e em decisões de aplicações. O *Context Toolkit* introduz alguns conceitos interessantes sobre reuso de componentes e apresenta algumas estratégias relevantes ao desenvolvimento e suporte à execução de aplicações sensíveis ao contexto;

- ***Solar System:*** desenvolvido no *Dartmouth College*, o *Solar System* [Chen et al., 2002] é uma arquitetura distribuída de componentes, organizada de maneira semelhante ao sistema solar, que oferece apoio à realização da computação sensível ao contexto por parte das aplicações. A arquitetura do *Solar System* foca em características voltadas para a mobilidade do usuário, flexibilidade e escalabilidade, tendo estruturas direcionadas para o paradigma distribuído de computação;

- ***SOCAM:*** desenvolvida na *Computing School of Singapure National University*, a arquitetura *Service-Oriented Context-Aware Middleware* (SOCAM) [Gu et al., 2005] utiliza um modelo formal de contexto com o objetivo de prover prototipação rápida de serviços e aplicações sensíveis ao contexto em ambientes de computação pervasiva. As informações contextuais são descritas em uma linguagem de representação de conhecimento (*OWL* –

Ontology Web Language), permitindo, com isso, o compartilhamento de conhecimento contextual entre diferentes entidades e a realização de inferências sobre essas informações.

- ***GaiaOS:*** desenvolvida na *University of Illinois*, esta plataforma introduz um ambiente de computação composto por um espaço físico qualquer enriquecido com dispositivos eletro-eletrônicos capazes de realizar algum tipo de computação útil aos freqüentadores do ambiente [Roman, 2002]. Além de um *framework* para o desenvolvimento de aplicações sensíveis ao contexto, o GaiaOS também apresenta suporte para configuração e coordenação de aplicações que executam no contexto de um espaço físico. A abordagem utilizada pelo GaiaOS se diferencia das demais plataformas ao propor a criação de um sistema operacional intermediário, capaz de gerenciar os recursos destes ambientes, assim como os sistemas operacionais tradicionais gerenciam os recursos de um computador.

- ***Aura:*** resultado de trabalhos desenvolvidos na *Carnegie Mellon University* [Souza et al., 2002], esta plataforma tem como intuito prover a usuários móveis uma infra-estrutura de computação virtual, particular e repleta de serviços capazes de se adaptar em ambientes propensos a falhas e com variabilidade de recursos. Uma característica particular desta plataforma é a definição de uma infra-estrutura capaz de permitir as suas aplicações moverem-se facilmente de um dispositivo para outro, suportando a heterogeneidade e variabilidade dinâmica de recursos nos ambientes ubíquos.

- ***CoBrA:*** desenvolvida na *University of Maryland* [Chen, 2004], é uma arquitetura com distribuíção híbrida, composta de um componente central denominado C*ontext Broker*, cujas funções são: compartilhar com os outros agentes da arquitetura uma ontologia comum para a representação das informações contextuais, realizar inferências a partir dos dados sensoriais e tratar questões relativas a privacidade dos usuários e segurança do sistema. Além de seu componente central (*broker*), a arquitetura *CoBrA* também incorpora outros agentes configuráveis para notificar às aplicações acerca de mudanças de contexto.

- ***WASP:*** desenvolvida na *University of Twente*, Holanda, e descrita em trabalhos como [Dockhorn, 2003] e [Santos, 2004], a arquitetura da plataforma WASP (*Web Architectures for Service Platforms*) tem como característica particular o uso da tecnologia de distribuição de *Web Services*, e é executada sobre uma rede móvel 3G. Essa arquitetura é tomada como base para a construção da plataforma Infraware.

As plataformas e *frameworks* descritos acima formam um conjunto representativo dos trabalhos desenvolvidos nos últimos anos voltados para a concepção de infra-estruturas de

suporte à aplicações sensíveis ao contexto. Cabe observar que existem ainda outras iniciativas aqui não reportadas, dentre elas [Hong, 2001], [Arruda jr., 2003], [Yamim, 2004] e [Viterbo Filho et al., 2006], que apresentam características interessantes e contribuições importantes no que se refere ao projeto e desenvolvimento de plataformas de suporte a aplicações sensíveis ao contexto.

2.1. Context Toolkit

O *Context Toolkit* é um *framework* conceitual de suporte à construção, ao desenvolvimento, à organização e ao funcionamento de aplicações sensíveis ao contexto. O *toolkit* foi desenvolvido de maneira a atender alguns requisitos de tratamento de contextos e informações contextuais [Dey, 2000]:

- *Separação de preocupações*: separar a aquisição da manipulação e do tratamento de contextos;
- *Interpretação de contexto*: estender mecanismos de busca, notificação, interpretação e inferência de contextos para permitir o uso uniforme e adequado das informações provenientes de um ambiente dinâmico e altamente distribuído;
- *Comunicação transparente e distribuída*: os responsáveis pelo desenvolvimento das aplicações e os projetistas de sensores de informações não devem se preocupar com detalhes de protocolos de comunicação e com os níveis de codificação e implementação da transmissão de informação contextual;
- *Disponibilidade permanente na tarefa de aquisição de contexto*: as aplicações sensíveis ao contexto não devem se relacionar apenas com uma única fonte de informação contextual, e sim acessar aquelas fontes que estão disponíveis no momento. Além disso, múltiplas aplicações podem acessar a mesma informação ao mesmo tempo. Assim, os componentes que realizam a aquisição de contextos devem ser executados independentemente das aplicações clientes que os utilizam;
- *Histórico e armazenagem de contexto*: necessidade constante de se manter e guardar um histórico das informações contextuais lidas e processadas pelas aplicações;

- *Descoberta de recursos*: descoberta de novos e eficientes provedores de contexto (ou pelo menos suas interfaces de *software*). As aplicações precisam saber que tipo de informação contextual um provedor de contexto oferece, onde ele se localiza e qual a forma de comunicação de dados que ele utiliza (tipo de protocolo, linguagem ou mecanismos de comunicação).

O *Context Toolkit* divide os elementos de apoio à construção das aplicações nas seguintes categorias de componentes: *Context Widgets, Interpreters, Aggregators, Services* e *Discoverers*. A combinação desses componentes no desenvolvimento das aplicações busca satisfazer os requisitos anteriormente descritos.

Os *Context Widgets*, ou simplesmente *Widgets*, são componentes de *software* que provém às aplicações acesso às informações contextuais em seus ambientes de operação. Eles abstraem a complexidade de uso e comunicação de sensores e fornecem informações em formatos adequados para o entendimento dos dados pelas aplicações. Os *Widgets* são blocos de componentes que podem ser organizados e reutilizados de acordo com a necessidade das aplicações, fornecendo um mecanismo de acesso uniforme e encapsulado das informações contextuais.

Os *Interpreters* são responsáveis por elevar o nível de abstração de uma informação contextual. Por exemplo, uma informação de localização pode ser expressa tanto na forma de latitude e longitude quanto em níveis de abstração mais elevados como uma cidade ou uma rua. *Interpreters* podem ser compostos em múltiplas camadas, de acordo com o nível de abstração desejado pelas aplicações.

Os *Aggregators* coletam informações de diversas fontes de informações contextuais logicamente relacionadas e as disponibilizam em um repositório único.

Os *Services* são componentes que capazes de executar ações de acordo com o interesse e a necessidade das aplicações. Quando uma combinação de condições for atingida, uma ação pré-definida pode ser tomada a pedido de uma aplicação.

Os *Discoverers* são responsáveis pela manutenção de um registro das informações de todos os componentes instanciados e suas respectivas competências. Quando um *Widget,* um *Interpreter,* um *Aggregator* ou um *Service* é instanciado no *framework, o componente Discovery* é notificado dessa presença, registrando a existência do novo componente instanciado.

A Figura 2.1, extraída de [Dey, 2000], mostra um exemplo possível de configuração dos componentes apresentados.

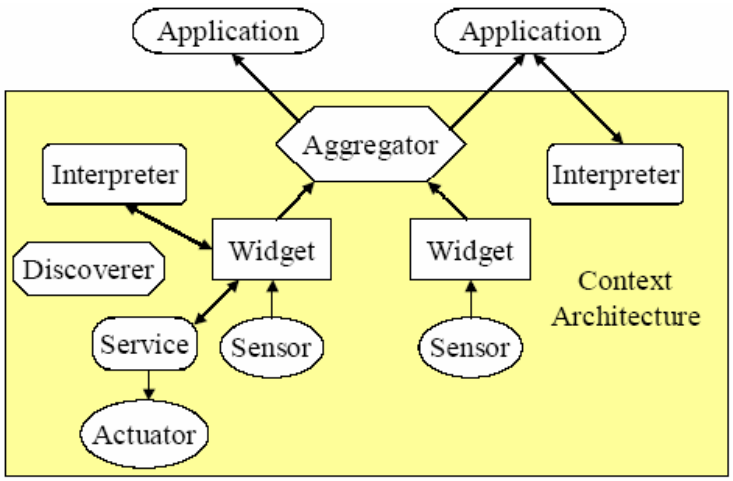

Figura 2-1 - Exemplo de configuração dos componentes do Context Toolkit.

Os componentes apresentados são: dois sensores, dois *Widgets*, um *Aggregator*, dois *Interpreters*, um *Service*, um *Discovery* e duas aplicações quaisquer. Ao ser instanciado, cada componente registra-se ao *Discovery*. Isso permite que os componentes instanciados sejam localizados por outros componentes ou mesmo por aplicações interessadas. Os sensores fornecem dados aos *Context Widgets*, que podem armazenar essas informações ou enviá-las aos *Interpreters*. Estes, por sua vez, podem manipular as informações primitivas provenientes dos *Widgets* para gerar novas informações com maior nível de abstração e disponibilizá-las aos demais componentes do *toolkit* e às aplicações. Um *Aggregator* coleta e armazena informações contextuais obtidas a partir dos *Widgets*. Finalmente, as aplicações podem requisitar aos *Aggregators,* aos *Widgets* ou aos *Interpreters* as informações contextuais de seu interesse.

Algumas das abstrações conceituais de componentes do *Context Toolkit* foram desenvolvidas e concretizadas em [Dey, 2000]. Contudo, as estruturas implementadas foram destinadas a uma classe específica de aplicações, e não fornecem um nível de generalidade expressivo para um conjunto maior de possíveis situações que podem ocorrer em ambientes ubíquos reais.

2.2. Solar System

O *Solar System* [Chen et al., 2002] é uma infra-estrutura de suporte à construção e desenvolvimento de aplicações sensíveis ao contexto. Para a coleta, disseminação, agregação e manipulação das informações contextuais são utilizados *grafos acíclicos dirigidos*, em que cada nó do grafo constitui um operador de funções. Esses grafos são chamados de grafos de operadores e podem ser organizados de maneira a permitir a composição dinâmica de novas funções desejadas pelas aplicações.

O *Solar System* foi desenvolvido com o intuito de atingir aos seguintes objetivos:

- *Flexibilidade*: As aplicações clientes do *Solar* podem acessar tanto informações brutas (provenientes diretamente de sensores e normalmente com baixo nível de abstração) quanto informações contextuais processadas (obtidas a partir de processos de refinamento e inferência). Além disso, também é oferecido para as aplicações suporte a adaptação perante mudanças ocorridas sobre os estados das informações contextuais;
- *Mobilidade*: O *Solar System* deve oferecer suporte a ambientes dinâmicos em que há constante movimentação entre fontes de contexto, sensores, dispositivos e usuários;
- *Escalabilidade*: O *Solar* deve buscar e coletar informações contextuais de várias fontes de contexto e disseminá-las a inúmeros clientes, e não permitir que haja gargalos nos fluxos de informações para as aplicações.

No *Solar System,* as informações contextuais são representadas como eventos (*events*). Os sensores, também denominados de fontes (*sources*), publicam um fluxo de eventos unidirecional. Os elementos que produzem um fluxo de eventos são chamados de *publishers* e aqueles que consomem um determinado fluxo de eventos são chamados *subscribers*. Os *publishers* no grafo do *Solar* são as fontes de dados e os nós operadores, enquanto os *subscribers* são as aplicações e outros nós operadores. As aplicações se inscrevem para receber um fluxo particular de eventos e reagem e se adaptam de acordo com a chegada dos mesmos. As operações fornecidas às aplicações através dos nós operadores dos grafos permitem ainda que as informações contextuais sejam filtradas (*filter*), transformadas (*transform*), combinadas (*merge*) ou agregadas (*aggregate*). Essas operações podem ser ainda combinadas, gerando novas funcionalidades para as aplicações.

Alguns exemplos de operações citados por [Chen et al., 2002] são apresentados a seguir:

- Filtragem (*filtering*): um sensor publica o valor de uma temperatura a cada 10 segundos enquanto uma aplicação precisa ser avisada apenas quando a leitura exceder 90 graus;

- Transformação (*transformation*): um sensor de localização envia apenas as coordenadas geográficas, mas uma aplicação precisa receber um valor mais abstrato como o nome de uma rua;

- Combinação (*merging*): uma aplicação de mapas recebe e mostra as informações de localização de todos os empregados de uma indústria de maneira combinada;

- Agregação (*aggregation*): uma aplicação que envia avisos a um usuário deseja saber, de acordo com a agenda do dia desse usuário, se já está na hora de uma determinada reunião e o avisa caso ele não esteja no local estipulado.

As Figuras 2-2 e 2-3, extraídas de [Chen et al., 2002], mostram, respectivamente, os operadores básicos do *Solar System* e um exemplo de um grafo de operadores:

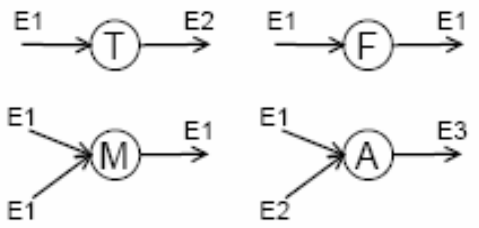

Figura 2-2 – Quatro tipos de operadores: Transformador (T), Filtro (F), Merger (M) e Agregador (A).

Na Figura 2-2, um Filtro (F) coloca em sua saída um subconjunto dos valores de entrada do operador. Um Transformador (T) recebe como entrada eventos do tipo E1 e gera na saída eventos do tipo E2. E2 pode ser do mesmo tipo de E1, dessa forma o Transformador agiria apenas alterando valores de atributos do evento em questão. O operador *Merger* (M) combina em sua saída todos os eventos que recebe em sua entrada. Um *Agregador* (A) gera em sua saída um fluxo de eventos de um tipo arbitrário baseado nos fluxos de eventos de entrada. Um tipo de dados contendo uma tupla com os valores mínimos e máximos de temperaturas em uma dada região é gerado por um operador *Agregador* tendo como entrada dois fluxos de eventos: um com os valores mínimos de temperatura e um outro com os valores máximos. Os fluxos de eventos que saem de cada operador podem ser utilizados por um ou mais operadores subseqüentes no grafo de operadores.

A Figura 2-3, extraída de [Chen et al., 2002], mostra um exemplo de uso dos operadores do *Solar System* e a organização dos mesmos para formar uma infra-estrutura em grafo em que os fluxos de eventos geram informações mais elaboradas às aplicações (*Active Map*, *Bob's Locator*, *Bob's Messenger*, etc...) a partir dos sensores (S) e das combinações de operações ao longo do grafo. Um detalhamento mais aprofundado dos operadores do *Solar System* e exemplos de seu uso podem ser encontrados em [Chen et al., 2002].

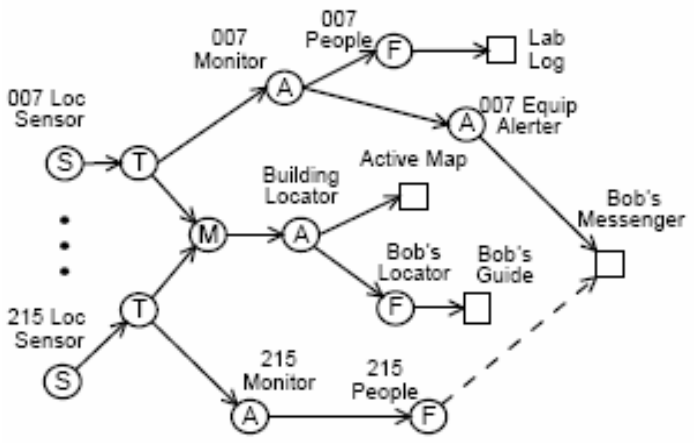

Figura 2-3 – Um exemplo de grafo de operadores.

A organização interna do *Solar System* é baseada em uma metáfora entre o sistema solar e seus componentes. A Estrela (*Star*) é o centro da infra-estrutura do *Solar System*. Ela organiza todas as ações da plataforma, mantendo os grafos de operadores e recebendo requisições e pedidos das aplicações. Os principais componentes da Estrela são um *OPerator Space* (OPS), que é uma estrutura de dados que representa e armazena as informações relativas aos grafos de operadores, e uma *Subscription Engine* (SE), que recebe e realiza um *parser* sobre as requisições e inscrições das aplicações e tomas as ações necessárias para atender a todos esses pedidos. Além da Estrela, o *Solar System* é composto por vários Planetas (*Planets*). Cada Planeta é um conjunto de grafos de execução da plataforma formado por fontes (*sources*) e operadores (*operators*), ou seja, um planeta é um pequeno ambiente de execução de operadores do *Solar System*. A Estrela instancia e reutiliza seus Planetas de acordo com a necessidade e as requisições das aplicações clientes. A Figura 2-4, extraída de [Chen et al., 2002], mostra a organização entre uma Estrela e seus Planetas no *Solar System*.

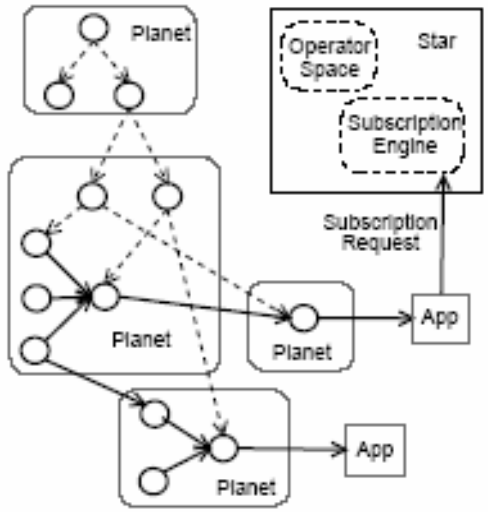

Figura 2-4 – A arquitetura do Solar System.

2.3. SOCAM

A arquitetura *Service-Oriented Context-Aware Middleware* (*SOCAM*) [Gu et al., 2005] tem como um de seus principais objetivos a prototipação rápida de serviços sensíveis ao contexto em ambientes de computação ubíqua. Segundo [Gu et al., 2005], a rápida prototipação pode ser conseguida através da conversão de espaços físicos, de onde as informações contextuais são adquiridas, em espaços semânticos, fazendo com que as informações contextuais sejam facilmente compartilhadas e acessadas por diferentes serviços e aplicações.

Para representar e compartilhar o conhecimento de um domínio de computação e as informações contextuais foi definido um modelo formal de contexto baseado em ontologia. As *Generalized Ontologies* descrevem semanticamente conceitos comuns presentes no modelo e compartilhadas por diferentes domínios. Já as *Domain-Specific Ontologies* são utilizadas para definir conceitos relacionados a domínios específicos. A Figura 2-5, extraída de [Gu et al., 2005], ilustra estes dois níveis de abstração:

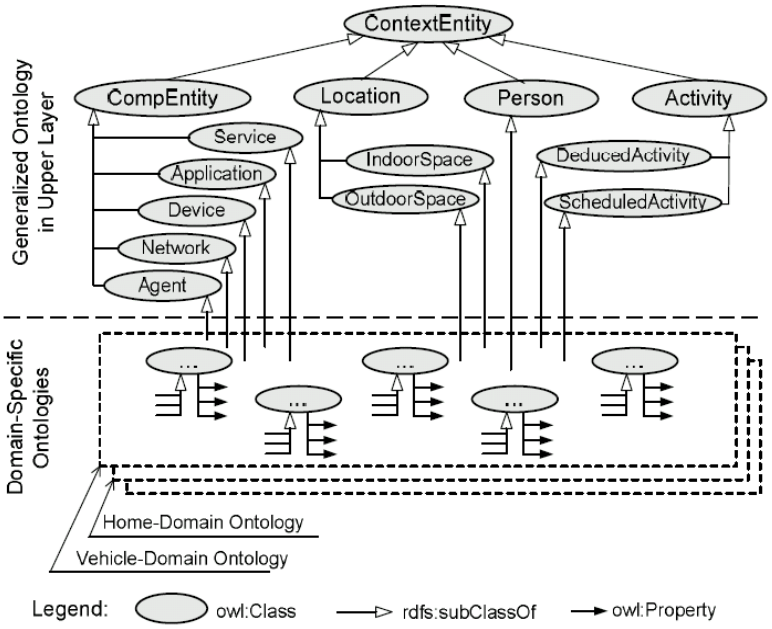

Figura 2-5 – Modelo Ontológico da Arquitetura SOCAM

Generalized Ontology descreve conceitos gerais como entidade computacional, localização, pessoa, atividade. O conceito geral "localização" é sub-dividido em outros conceitos gerais como espaço interno e externo. Esses conceitos são mapeados, por exemplo, nos sub-domínios casa, veículo, escola, escritório do nível *Domain-Specific Ontologies*, que representam ontologias individuais de nível mais baixo de contexto, definindo assim uma ontologia para o sub-domínio casa, outra para o sub-domíno veículo, etc.

Os componentes da arquitetura são definidos como serviços independentes e podem estar distribuídos por entre redes heterogêneas. Podem também ser anunciados, localizados e acessados através do Serviço de Localização de Serviços. São cinco os principais serviços providos pela arquitetura SOCAM:

- Serviços de Provedores de Contexto
- Serviço Interpretador de Contexto
- Serviço de Banco de Dados de Contexto
- Serviço de Localização de Serviço
- Serviços Móveis Sensíveis ao Contexto

A arquitetura SOCAM tem seus principais componentes detalhados na Figura 2-6, extraída de [Gu et al., 2005].

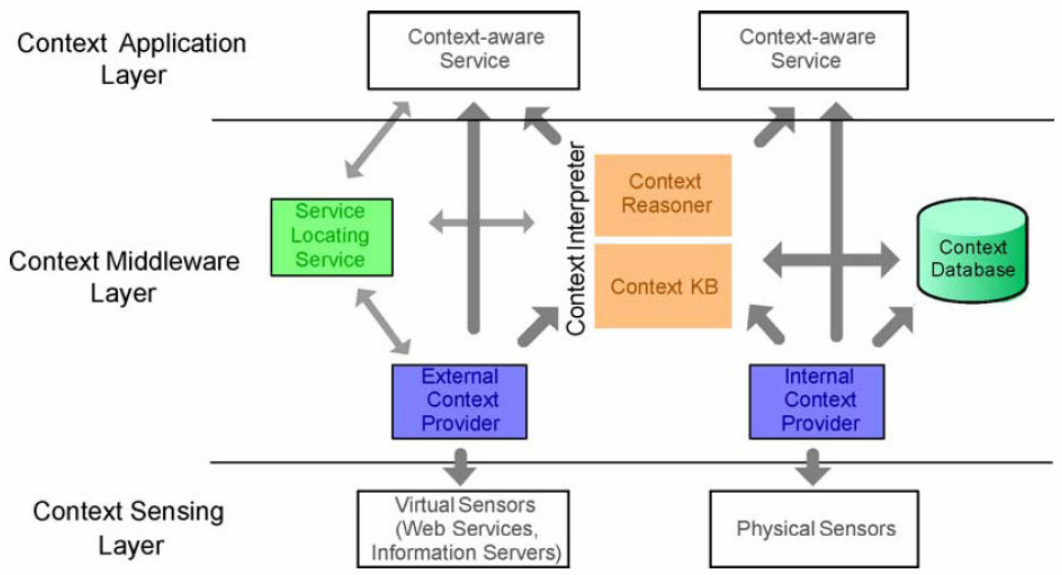

Figura 2-6 – Arquitetura SOCAM.

Os provedores de contexto são responsáveis por fazer a aquisição das informações contextuais. Provedores de contexto externos obtêm informação contextual de fontes externas, por exemplo, um servidor de informação capaz de informar a temperatura de várias cidades. Provedores de contexto internos obtêm informações contextuais de fontes localizadas em um sub-domínio ontológico, por exemplo sensores de localização. Os provedores internos estão diretamente relacionados ao sub-domínio ontológico ao qual pertencem.

O interpretador de contexto permite que informações contextuais de mais alto nível possam ser inferidas a partir de informações contextuais menos abstratas. Este componente é composto por uma máquina de inferência e uma base de dados contendo as informações contextuais. Novas informações contextuais são inferidas a partir da interpretação de regras durante a adição, remoção ou atualização das informações contextuais armazenadas na base de dados. Este módulo também atua como um provedor de contexto.

O serviço de localização permite que usuários, agentes e aplicações localizem provedores de informação contextual de interesse particular. Este serviço é capaz de rastrear e adaptar-se às mudanças dos provedores de contexto.

Serviços móveis são aplicações ou serviços capazes de adaptar seu comportamento de acordo com o contexto corrente. Para isso são definidas regras que especificam ações a serem invocadas quando uma determinada condição é satisfeita.

A arquitetura SOCAM, portanto, define um conjunto de componentes independentes que auxiliam a construção de aplicações sensíveis ao contexto. Esses componentes atuam

como serviços providos pela arquitetura e podem interagir entre si de maneira transparente para as aplicações. Dessa forma, protótipos de novos serviços e aplicações sensíveis ao contexto são rapidamente construídos a partir da identificação dos provedores de informação contextual e da definição das regras que descrevam os eventos monitorados e as ações a serem invocadas na ocorrência dos mesmos.

2.4. GaiaOS

Um outro esforço para o desenvolvimento de aplicações sensíveis ao contexto relacionadas a ambientes móveis de computação ubíqua é o projeto GaiaOS [Roman et al., 2002], da *University of Illinois*. Este projeto apresenta um m*iddleware* que exporta e coordena os recursos contidos em um espaço físico para suportar o desenvolvimento e execução de aplicações sensíveis ao contexto.

Para tanto, são definidos ambientes denominados *Active Spaces* - espaços físicos equipados com dispositivos conectados em rede, coordenados por um sistema operacional, capazes de se adaptar às mudanças relacionadas ao ambiente, usuários e dispositivos. De maneira semelhante a um sistema operacional tradicional, o GaiaOS gerencia a execução das aplicações construídas, oferecendo serviços de identificação, localização e utilização dos recursos existentes em um ambiente. A figura 2-7, extraída de [Roman et al., 2002], apresenta a arquitetura em camadas da plataforma GaiaOS.

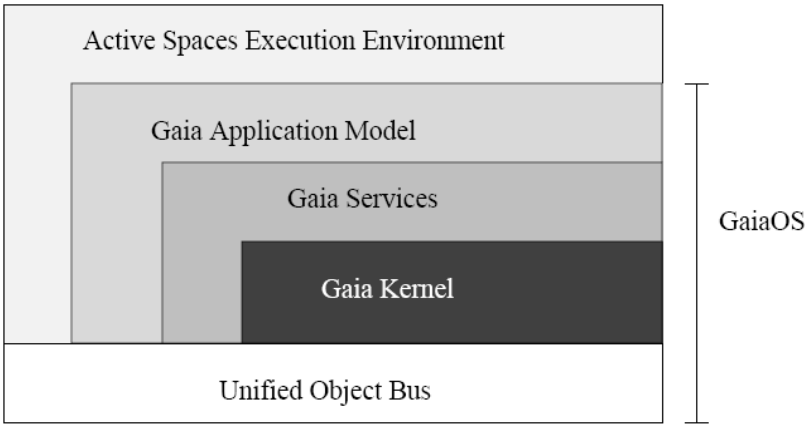

Figura 2-7 - Arquitetura em Camadas do GaioOS.

A natureza heterogênea de um a*ctive space* pode ser constatada pela grande variedade de dispositivos e protocolos presentes em um ambiente ubíquo. Entretanto, toda esta heterogeneidade torna-se transparente para as aplicações através do *Unified Object Bus* (UOB). Este barramento é responsável por prover ferramentas comuns para a manipulação do

ciclo de vida dos componentes executados em um *active space*, sendo responsável pela criação, inspeção, remoção e nomenclatura de tais componentes.

O *kernel* do GaiaOS contém os serviços que implementam o núcleo funcional do sistema, incluindo descoberta de entidades, um repositório de componentes, distribuição de eventos, serviço de nomes, persistência e manipulação de dados e segurança. Os componentes que implementam tais funcionalidades possui sua infra-estrutura de comunicação baseada em CORBA, e provêem alguns serviços básicos:

- Gerenciador de Eventos: permite que as aplicações registrem canais específicos de eventos a serem notificados com informações a respeito de mudanças no ambiente.
- Serviço de Descoberta: utiliza o Gerenciador de Eventos para rastrear componentes de software, pessoas e outras entidades presentes no espaço.
- Repositório do Espaço: determina uma interface para a busca de entidades específicas baseadas em condições como localização, tipo de serviço e disponibilidade de recursos.
- Serviço de Dados: implementa um sistema de arquivos dinamicamente tipado que suporta adaptação de conteúdo, acesso customizado aos dados e consciência de localização.
- Serviço de Autenticação: utiliza credenciais para verificação de identidade de usuários e provê o controle de acesso baseado em regras e perfis.

O modelo de aplicação proposto pelo GaiaOS provê ainda um *framework* padrão baseado no paradigma *Model-Presentation-Adapter-Controller-Coordinator* (MPACC) – Modelo-Apresentação-Adaptador-Controlador. Este *framework*, ilustrado pela figura 2-8, extraída de [Roman et al., 2002], suporta o desenvolvimento de aplicações ubíquas e define cinco componentes principais:

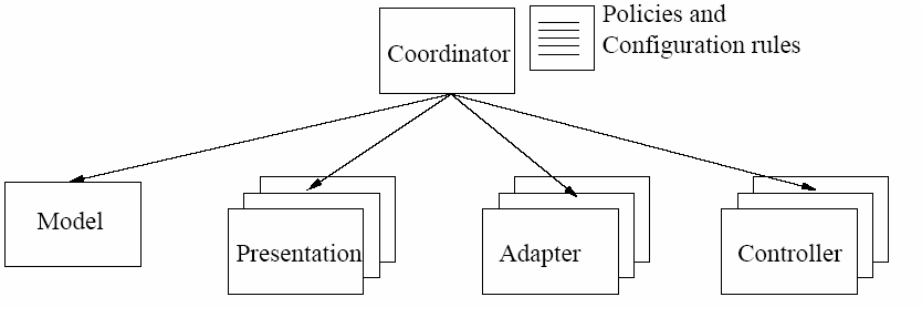

Figura 2-8 - Diagrama esquemático do framework de aplicação do GaiaOS.

- *Model*: implementa a estrutura central da aplicação, que normalmente consiste nos dados e na interface programável disponibilizada para manipulá-los.
- *Presentation*: externalização física do modelo que permite aos usuários percebê-lo através de um ou mais sentidos.
- *Adapter*: responsável por adaptar o formato dos dados do modelo às características de um dispositivo de saída, realizando uma espécie de tradução.
- *Controller*: provê mecanismos para modificar o estado do modelo coordenando não apenas os dispositivos de entrada, mas qualquer contexto possa influenciar a aplicação.
- *Coordinator*: gerencia a composição da aplicação aplicando políticas de adaptação considerando aspectos como personalização e mobilidade da aplicação.

Em suma, a infra-estrutura proposta pelo GaiaOS auxilia no desenvolvimento de aplicações sensíveis ao contexto oferecendo um *middleware* distribuído que coordena os recursos computacionais de um espaço físico, transformando-o em um sistema computacional programável, similar a um sistema operacional tradicional, à medida que gerencia as tarefas comuns a todas as aplicações. Para permitir que seus usuários interajam com diversos dispositivos e serviços simultaneamente, o GaiaOS disponibiliza para seus usuários os *active spaces* - abstrações que compreendem as informações contextuais, as tarefas e os dispositivos associados a cada usuário.

2.5. Aura

Um dos desafios mais importantes da computação sensível ao contexto esta relacionado à livre mobilidade do usuário. O projeto *Aura* [Souza et al., 2002], desenvolvido por pesquisadores da *Carnegie Mellon University*, provê uma infra-estrutura para permitir a suas aplicações moverem-se facilmente de um dispositivo para outro, suportando de maneira robusta a heterogeneidade e variabilidade dinâmica de recursos nos ambientes. No momento em que um usuário move-se de um ambiente para outro, o Aura procura adaptar as tarefas que estão sendo executadas. Como os recursos disponíveis variam em cada ambiente, alguns possíveis ajustes como a reconfiguração de alguns serviços ou mesma a substituição de um serviço por outro deverão ser realizados para que as tarefas do usuário possam continuar a ser realizadas de forma transparente no novo ambiente.

A infra-estrutura proposta pelo *Aura*, apresentada pela Figura 2-9 (extraída de [Souza e Garlan, 2002]), baseia-se na adaptação de sistemas previamente desenvolvidos – como o *Coda* [Satyanarayanan, 2002] e o *Odyssey* [Noble et al. 1997] – modificados e adaptados para prover acessos a arquivos por clientes móveis e para monitorar recursos e adaptabilidade, respectivamente. Além disso, também são investigados os problemas associados ao desenvolvimento de novos sistemas, como o *Spectra* [Flinn et al. 2001], um mecanismo adaptativo para receber chamadas para execução remota de aplicações.

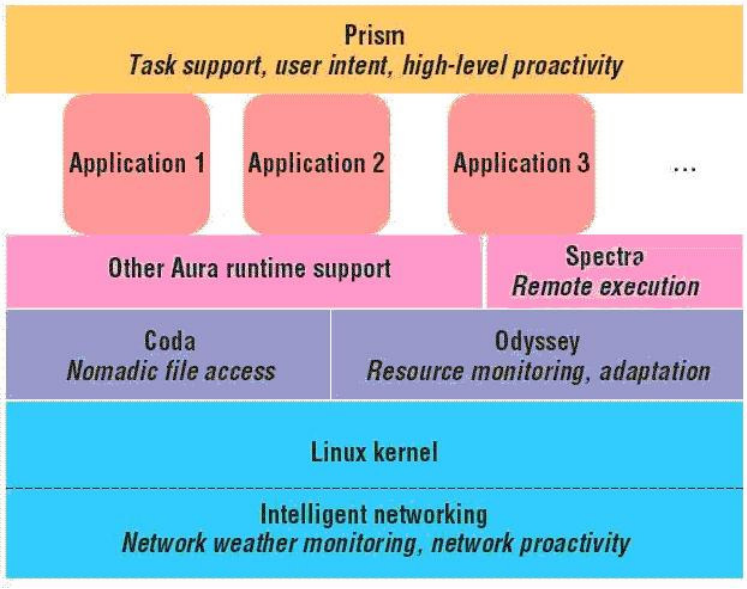

Figura 2-9 – A Plataforma Aura

Para prover adaptabilidade no nível das aplicações, o Aura introduz uma nova abstração em forma de camada de sistema: a camada de tarefas. Esta camada, denominada *Prism*, compõe uma camada superior à camada das aplicações destinada a capturar e gerenciar a intenção do usuário provendo suporte de alto-nível para pró-atividade e auto-ajuste. Com isso, é disponibilizada para os outros componentes do sistema uma base para a adaptação ou antecipação das necessidades dos usuários.

Como pode ser observado na Figura 2-10 (extraída de [Souza e Garlan, 2002]), os principais componentes da camada *Prism* são:

- *Task Manager*: possui representação explícita das tarefas do usuário como combinações de serviços abstratos.
- *Context Observer*: monitora as informações contextuais, permitindo que o Prisma configure as tarefas do modo mais apropriado para o ambiente.

- *Enviroment Manager:* gerencia a infra-estrutura do ambiente, ajudando no monitoramento de recursos e na adaptação.

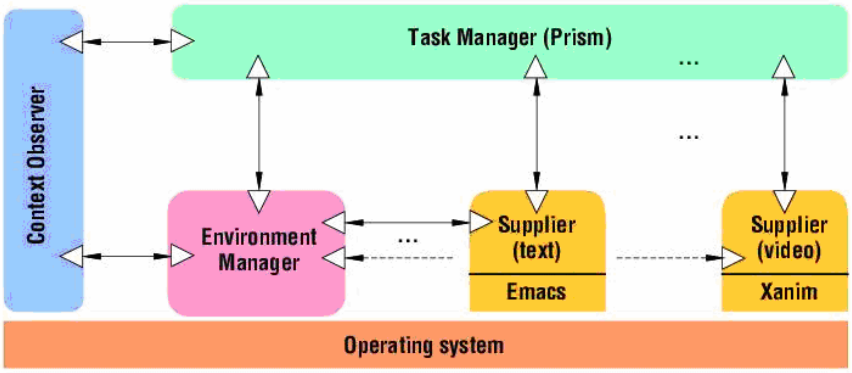

Figura 2-10 – Arquitetura Interna da Camada Prism

O modelo de aplicação do Aura é definido por um *framework* padrão onde as tarefas do usuário tornam-se entidades, sendo definidas explicitamente pelos ambientes e representadas como uma coleção de serviços abstratos. A partir daí, os ambientes também são capazes de se auto-monitorar e negociar o suporte a uma tarefa quando houver variação de recursos. Estas características são providas pelo Aura através de quatro componentes de software principais:

- *Service Supplier:* fornece os serviços abstratos que compõem uma determinada tarefa, permitindo, assim, que aplicações existentes sejam encapsuladas e entrem em conformidade com as APIs do Aura.
- *Context Observer:* monitora e fornece informações contextuais além de reportar a ocorrência de eventos aos demais componentes.
- *Environment Manager:* gerencia os serviços disponíveis no ambiente. Adicionalmente à descoberta de serviços individuais, ele pode avaliar configurações alternativas que melhor se adaptem a uma determinada situação.
- *Task Manager:* coordena a migração da informação relacionada a tarefa do usuário durante as mudanças de ambiente além de identificar alterações nos provedores de serviços e gerenciar a execução das tarefas.

Duas das principais habilidades do Aura são: *(i)* o suporte a mobilidade, e *(ii)* a proteção dada aos usuários contra as variações de disponibilidade de recursos no ambiente. Neste sentido, o Aura atua como um *proxy* para o usuário móvel, de modo que quando esse

adentra um novo ambiente, o *proxy* identifica as capacidades e restrições do contexto e busca (ou adapta) o recurso apropriado para a execução de uma determinada tarefa.

Como benefícios importantes desta proposta podem ser citados: a representação explícita das tarefas do usuário e a introdução de uma entidade responsável por capturar e gerenciar a intenção do usuário, atuando de forma pró-ativa na busca por configurações ideais em cada ambiente. Além disso, a representação de tarefas como combinações de serviços possibilita ainda ao Aura detectar quando os requisitos de uma tarefa não estão sendo contemplados, permitindo a busca por configurações alternativas para atender a uma determinada tarefa.

2.6. CoBrA

Desenvolvida na *University of Maryland*, CoBrA (*Context Broker Architecture*) é uma arquitetura baseada em agentes para suportar a execução de aplicações em ambientes inteligentes. Um exemplo de caso de uso é uma aplicação que gerencia um ambiente para reuniões, sendo capaz de detectar a localização de pessoas interessadas, através de seus PDAs, e tomar decisões baseadas em dados contextuais como sua agenda de horários.

Na visão do projeto, um pré-requisito para a construção de sistemas sensíveis a contexto é possibilitar a interpretação do contexto por máquina. Para isto, é considerado apropriado o uso de tecnologias da Web semântica. A ontologia SOUPA, descrita em OWL, foi criada para a manipulação e inferência de informações contextuais: foram definidas descrições de espaços físicos, dispositivos e tempo, além de contar com uma descrição de conceitos no domínio de ambientes inteligentes, como reuniões. Através de consultas RDQL (*Resource Description Query Language*), o usuário pode fazer as consultas desejadas. São mostrados os diversos módulos da ontologia SOUPA na figura 2-11.

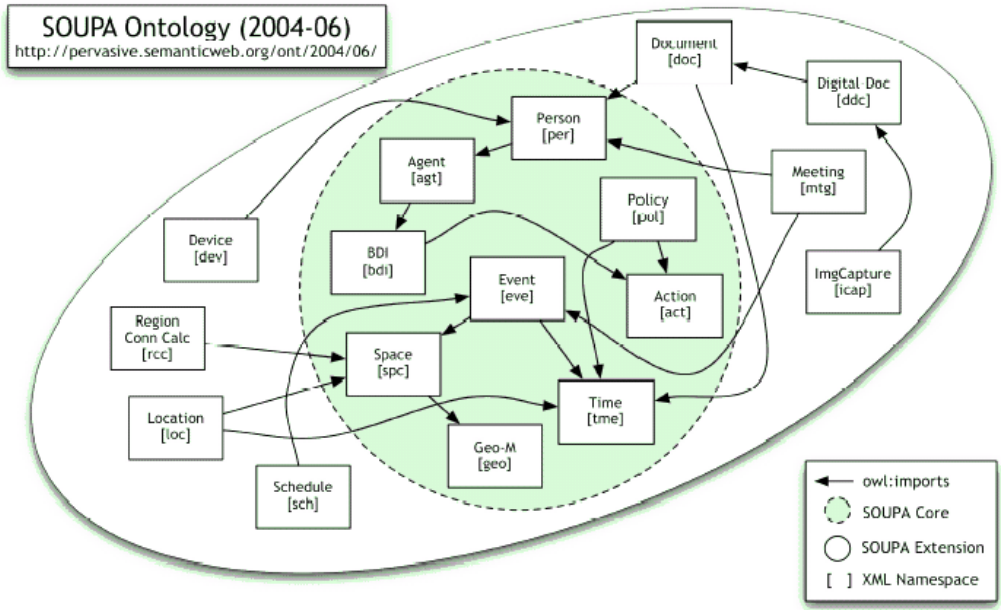

Figura 2-11 – Ontologia SOUPA

A máquina de inferência da arquitetura CoBrA utiliza tanto a API *Jena* para manipulação de ontologias OWL quanto o *Jess*, um *shell* para sistemas especialistas. Além das regras, são utilizadas heurísticas sobre os diversos domínios. Inconsistências, mesmo que originadas de fatos verdadeiros, podem ser detectadas através do *context broker*: por exemplo, quando um usuário possui dois dispositivos, ambos com rastreamento de localização, e esquece um deles em determinado lugar. Pode existir uma regra afirmando que o usuário se encontra no mesmo local de seus dispositivos. Se não houvesse nenhum tratamento, seria apontado que aquele usuário está em dois lugares simultaneamente, o que é obviamente incorreto.

A arquitetura do CoBrA é baseada em um componente central (*broker*), um agente que tem como função manter a definições padrão de contexto que serão utilizadas por todos os dispositivos, fazer inferências a partir dos dados sensoriais e cuidar da privacidade e segurança do sistema, através das ontologias citadas anteriormente. Além do núcleo, outros agentes são configurados para notificar mudanças de contexto. Estes agentes foram implementados utilizando-se a arquitetura JADE, um *framework* para programação de agentes em Java, e seguem o padrão FIPA (*Foundations of Intelligent Physical Agents*) para a troca de mensagens entre eles. Os agentes são modelados seguindo uma estratégia BDI (*belief-desire-intention*). A arquitetura possui, portanto, a habilidade de notificar o usuário quando houver alguma mudança no contexto. A figura 2-12, extraída de [Chen et al., 2003], ilustra com maiores detalhes a arquitetura CoBrA.

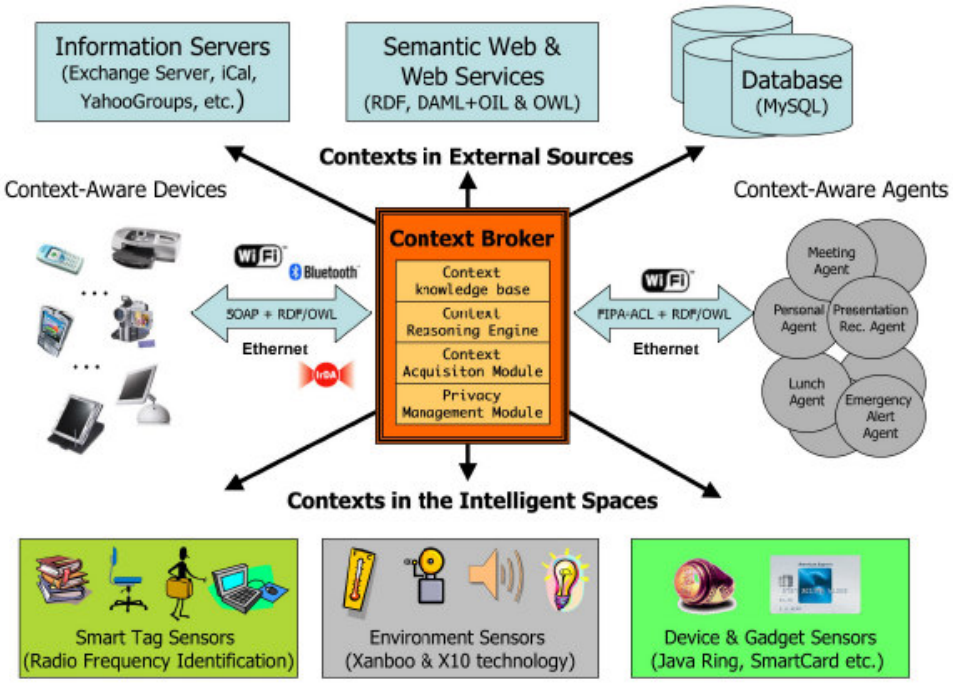

Figura 2-12 – Arquitetura CoBrA

Algumas características particulares destacadas por esta arquitetura são: o uso de um *broker* e a criação de uma ontologia para se modelar as informações contextuais. Além do modelo de contexto, questões relacionadas à realização de inferências de novas informações contextuais também são implementadas, juntamente com a utilização do paradigma de agentes para o desenvolvimento de alguns componentes da arquitetura.

2.7. WASP – Web Architectures for Service Platforms

A arquitetura WASP (*Web Architectures for Service Platforms*) ([Dockhorn, 2003], [Rios, 2003], [Santos, 2004]) é resultado de trabalhos desenvolvidos na *University of Twente*, na Holanda, juntamente com a contribuição de alguns pesquisadores do LPRM/DI/UFES em algumas etapas de sua construção.

A WASP é uma plataforma baseada na tecnologia de *Web Services* que oferece apoio ao desenvolvimento, construção e execução de aplicações sensíveis ao contexto. Como pode ser observado na Figura 2-12, extraída de [Rios, 2003], a plataforma se relaciona com o mundo exterior de três maneiras distintas, conforme o agente externo:

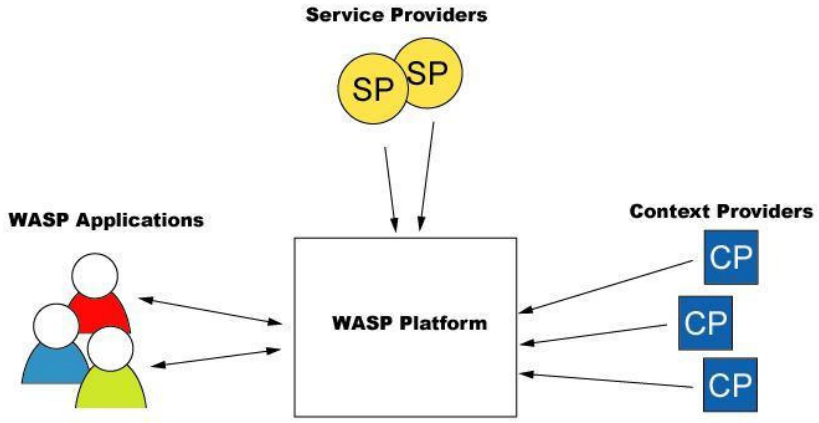

Figura 2-13 - Plataforma WASP e seus relacionamentos.

Os Provedores de Contexto (*Context Providers*) são elementos de *hardware* ou de *software* que provém à plataforma as informações contextuais coletadas de sensores ou outras fontes provedoras de contexto. Os Provedores de Serviço (*Service Providers*) são entidades que disponibilizam serviços à plataforma. Estes serviços devem ser publicados e registrados na plataforma antes de serem utilizados. As Aplicações WASP (*WASP Applications*) são os clientes da plataforma e interagem com a mesma através da assinatura de serviços previamente disponibilizados. As aplicações podem ainda, através da plataforma, acessar as informações contextuais provenientes diretamente dos Provedores de Contexto ou ainda utilizar informações pré-processadas para adaptarem-se de acordo com o ambiente em que estão inseridas.

A Figura 2-14, extraída de [Dockhorn, 2003], ilustra a arquitetura original proposta para a plataforma WASP. Nessa versão, a arquitetura da plataforma era composta por três principais componentes: o Interpretador de Contexto (*Context Interpreter*), os Repositórios (*Repositories*) e o Monitor (*Monitor*), cada um deles com uma função específica:

- *Interpretador de Contexto*: recebe as informações contextuais dos provedores de contexto, manipula essas informações e torna-as disponíveis de maneira uniforme aos demais componentes da plataforma;
- *Repositórios*: são responsáveis por dar suporte e alimentar o Monitor com informações sobre os demais elementos que estão envolvidos com a plataforma. Por exemplo, alguns dados que são armazenados nos Repositórios dizem respeito ou provém do Interpretador de Contexto e outros descrevem sobre capacidades dos Provedores de Serviço;

- *Monitor*: é o responsável por interpretar e gerenciar as assinaturas e os pedidos das aplicações para a plataforma, por isso é considerado o coração da plataforma. Para realizar os pedidos das aplicações, o Monitor se utiliza de informações fornecidas pelos Repositórios e pelo Interpretador de Contexto.

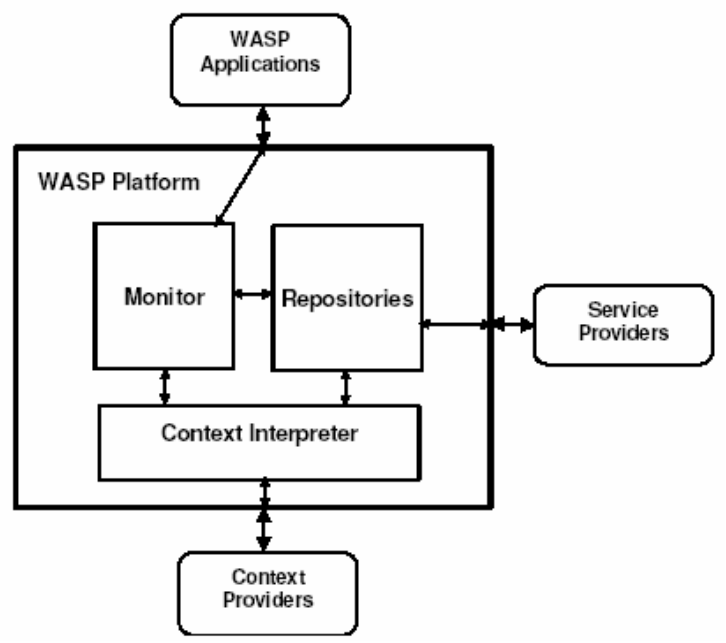

Figura 2-14 - Arquitetura da plataforma WASP.

Em [Rios, 2003], a plataforma foi estendida e estruturada para possibilitar o uso de ontologias na descrição dos dados e tarefas envolvidas com todo o sistema. As ontologias utilizadas na WASP servem para prover uma anotação semântica padronizada dos termos utilizados pelos componentes e demais atores da plataforma, além de definir uma descrição uniforme de serviços e contextos. Em [Santos, 2004], numa parceria com o LPRM/DI/UFES, a arquitetura WASP foi mais uma vez estendida, agora para dar suporte à utilização de serviços semânticos. Para tal, dois novos componentes foram incluídos na plataforma, como mostra a Figura 2-15, extraída de [Santos, 2004].

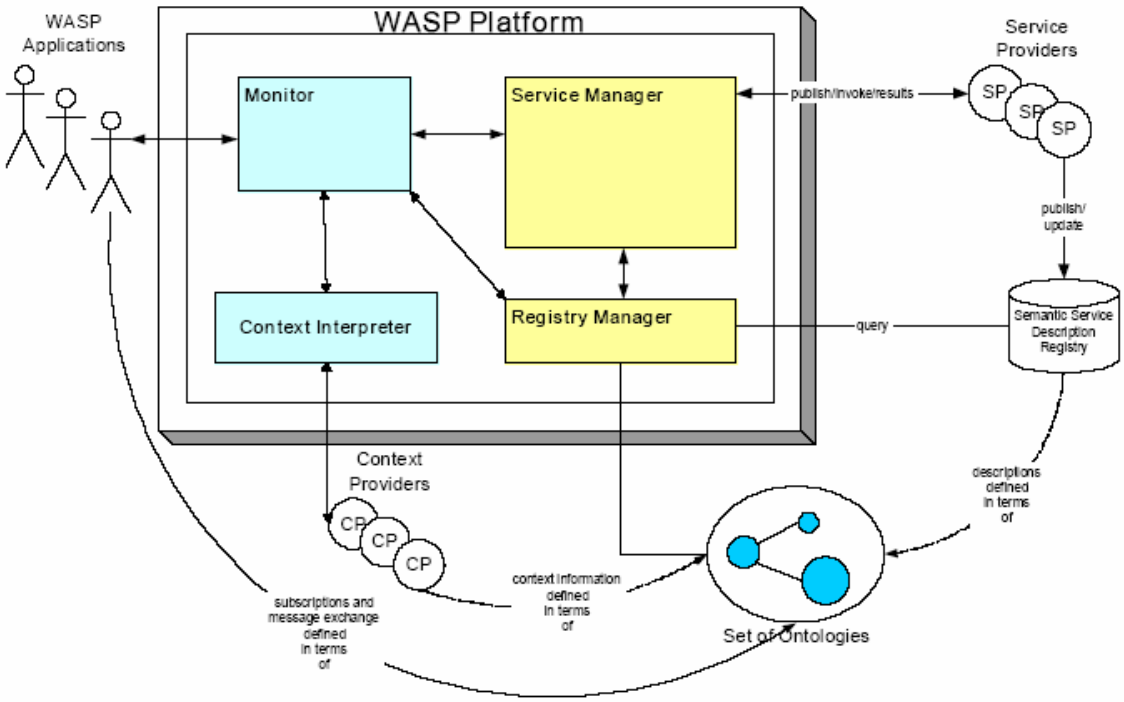

Figura 2-15 – Arquitetura WASP estendida para a utilização de serviços semânticos

O Gerenciador de Serviços (*Service Manager*) prepara conceitualmente a plataforma para a oferta de diversos tipos de funcionalidades de suporte à utilização de serviços; por exemplo, publicação de descrição semântica de serviços, descoberta e seleção de serviços, solicitação de execução de serviços, etc.

O Gerenciador de Registros (*Registry Manager*) oferece uma interface homogênea de acesso aos registros utilizados pela plataforma. Tais registros, com a inclusão do gerenciador, podem estar localizados internamente ou serem externos à plataforma.

Por fim, em [Drost, 2004], outro trabalho resultante da colaboração entre as duas instituições, uma camada de privacidade foi incluída na plataforma com o intuito de permitir um controle sobre o fluxo de informações enviado para as aplicações. Essa camada, baseia-se no uso de ontologias de tarefas [Rajpathak, 2001] para garantir maior confiabilidade das informações disponibilizadas pelos Provedores de Contexto e acessadas pelos Provedores de Serviço.

2.8. Conclusão do Capítulo

Em virtude da natureza dinâmica, heterogênea e distribuída dos ambientes em que estão inseridas as aplicações móveis sensíveis ao contexto, seus projetos e execuções requerem uma infra-estrutura computacional diferenciada, com o objetivo de apoiar o tratamento e a superação dos desafios teóricos e tecnológicos anteriormente mencionados. Tal infra-estrutura pode ser fornecida, por exemplo, de maneira integrada e transparente através de um *middleware* especializado para tal propósito.

Diversas plataformas e *frameworks* vêm sendo propostos com o intuito de atender aos requisitos da computação sensível ao contexto. Entretanto, de maneira geral, grande parte dos requisitos impostos pela ubiqüidade e sensibilidade ao contexto ainda não são atendidos por todos os projetos analisados. Em função da própria natureza e objetivos específicos de cada projeto, a maioria das plataformas apresentadas concentra-se no atendimento de um subconjunto básico desses requisitos, negligenciando às vezes questões importantes, particularmente no que se refere ao campo das aplicações ubíquas reais.

No próximo capítulo são discutidos os principais requisitos que nortearam a concepção e modelagem da arquitetura Infraware.

3. Avaliação de Requisitos

Esse capítulo faz uma avaliação das arquiteturas apresentadas no Capítulo 2 à luz de alguns requisitos e características desejáveis no âmbito das plataformas sensíveis ao contexto. É importante ressaltar que algumas dessas questões encontram-se ainda em aberto, o que impõe obstáculos efetivos à massificação da computação sensível ao contexto e abrem espaço para pesquisas adicionais.

Os seguintes pontos foram selecionados para discussão: *(i) generalidade da Informação Contextual; (ii) linguagem de subscrição; (iii) modelagem contextual e inferência de contextos; (iv) interface com sensores; (v) incerteza e confiabilidade das informações contextuais; (vi) tecnologias de distribuição e integração; (vii) segurança e privacidade; (viii) escalabilidade; (ix) descoberta de serviços; (x) personalização dos serviços para os usuários; (xi) coordenação entre aplicações, (xii) persistência das informações contextuais.*

3.1. Generalidade da Informação Contextual

Segundo [Schmidt et al., 1999] e [Chen et al., 2001], poucas informações contextuais, além de localização, vêm sendo efetivamente exploradas pelas plataformas. Esta situação pode ser explicada, em parte, pela complexidade de modelagem, reconhecimento e manipulação de elementos de contexto genéricos.

Alguns dos *middlewares* citados na literatura suportam um número muito restrito de informações contextuais. A infra-estrutura Nexus [Fritsch et al., 2000], por exemplo, tem como componente central de interesse a localização do usuário. Este tipo de restrição, apesar de reduzir o escopo do projeto, facilitando em decorrência disso a sua implantação, acaba limitando o domínio de aplicações suportadas pela plataforma. As plataformas Aura e GaiaOS também restringem o tipo de informação contextual suportada. Na plataforma *Aura*, a entidade central de interesse modelada é o usuário, cujas atividades e preferências disparam o processo de adaptação do ambiente. Na plataforma GaiaOS, o contexto de interesse está restrito a espaços ativos, isto é, espaços físicos convertidos em ambientes computacionais programáveis.

Algumas infra-estruturas se propõem à coletar e manipular informações contextuais de maneira genérica. Dentre as arquiteturas avaliadas, destacam-se o Context Toolkit e o Solar

System. Os componentes conceituais do Context Toolkit formam um *framework* para o desenvolvimento e execução de aplicações genéricas sensíveis ao contexto, sendo responsável por coletar informações contextuais de sensores, manipulá-las e disponibilizá-las para as aplicações interessadas. Durante esse processo, as informações manipuladas podem ser traduzidas ou ainda utilizadas para inferir outras informações de mais alto nível. O Solar System propõe uma abstração baseada em grafos acíclicos dirigidos para a representação e disseminação das informações contextuais. Para a manipulação das informações e inferência de novos contextos, em cada nó do grafo, são disponibilizados operadores funcionais (filtros, transformadores, unificadores ou agregadores). O Solar System também fornece um *framework* para o desenvolvimento de aplicações de uso geral. Uma outra abordagem genérica para a manipulação de informações contextuais vem sendo explorada pelas infra-estruturas CoBrA e SOCAM. Nestas arquiteturas, as informações contextuais e seus inter-relacionamentos são especificados por modelos semânticos baseados em ontologias. Devido a sua capacidade de descrever contextos complexos e a possibilidade efetiva de realização de inferência sobre as informações contextuais, esta abordagem tem se mostrado adequada para garantir a generalidade e expressividade da informação contextual em plataformas de serviços sensíveis ao contexto.

3.2. Linguagem de Subscrição

Em geral, as plataformas que se propõem a suportar informações contextuais de maneira genérica adotam o modelo *publish-subscribe* para enviar mensagens sobre a troca de contextos para as aplicações interessadas. Este modelo, apesar de acabar impondo o uso de *frameworks* para o projeto das aplicações, ao implementarem mecanismos de comunicação assíncrona, promovem um desacoplamento operacional entre cliente e servidor. Sob esta perspectiva, o cliente pode continuar o processamento assim que a plataforma tenha aceitado sua mensagem.

Entretanto, para que uma aplicação possa se subscrever para o recebimento de mensagens sobre a troca de um contexto específico, a plataforma deve definir um modelo de subscrição. O Context Fabric [Hong, 2001] utiliza a linguagem de especificação de contexto baseada em XML, denominada *Context Specification Language*, que permite aos desenvolvedores definir contextos específicos em uma linguagem de alto nível. A linguagem está intimamente ligada ao componente arquitetural chamado *Context Event Service*, que é o serviço responsável pelo processamento das subscrições (utilizando a linguagem de

especificação de contexto) e notificação das aplicações que se inscreveram no sistema sobre a ocorrência de um contexto específico. No Solar System, essa tarefa é desempenhada pelo componente Estrela (*Star*), que organiza todas as ações a serem tomadas pela plataforma, a partir do recebimento de requisições e pedidos de subscrição por parte das aplicações. Para isso, é utilizada uma *Subscription Engine*, responsável pelo processamento dos pedidos de subscrição. A plataforma WASP também define uma linguagem de subscrição baseada em XML, denominada *Wasp Subscription Language*, que é utilizada pelas aplicações para configurar ações a serem tomadas pela plataforma a partir da ocorrência de contextos de interesse.

A adoção de linguagens descritivas baseadas em padrões como XML, RDF e OWL para compor o modelo de subscrição das plataformas *context-aware* tem se mostrado uma abordagem adequada, uma vez que estas linguagens definem uma sintaxe neutra como forma de representação para especificar o relacionamento entre entidades e prover interoperabilidade estrutural com certo nível de independência.

3.3. Modelagem Contextual

Uma questão importante a ser tratada pelas plataformas é a definição de um modelo que descreva o domínio contextual em que uma determinada aplicação ou serviço está inserido. Diversas propostas de modelagem contextual são sugeridas na literatura, tais como [Schilit, 1995], [Gray et al., 2001], [Henricksen et al., 2003] e [Chen, 2004].

As tuplas de valores são a forma mais simples de estrutura de dados utilizada para modelagem da informação contextual. Schilit [Schilit, 1995] utiliza tuplas de chaves e seus respectivos valores para modelar as informações contextuais que são providas às aplicações como variáveis de ambiente. A natureza simples deste tipo de modelagem permite apenas a realização de comparações exatas, tornando este tipo de abordagem inadequada para propósitos mais sofisticados. Dentre as plataformas analisadas, destaca-se o modelo adotado pelo *Context Broker Architecture* (CoBrA). Como visto, CoBrA é uma arquitetura centralizada em um componente denominado *context broker*, que compartilha um modelo de contexto baseado em ontologia que provê um conjunto de conceitos para a comunicação entre agentes distribuídos, serviços e dispositivos. Por se apresentarem como modelos formais, ontologias podem ser úteis para realizar inferências, a fim de gerar novos fatos e/ou descobrir novos relacionamentos entre entidades. O uso de linguagens gráficas, tais como extensões de *Unified Modeling Language* (UML), conforme feito por [Henricksen et al., 2002] também

vem sendo utilizadas na modelagem das informações contextuais. Modelos baseados em UML são capazes de modelar ricas composições e inter-relacionamentos entre informações contextuais e entidades, embora não ofereçam suporte adequado para a realização de inferências acerca das informações modeladas.

A modelagem das informações contextuais tem sido um aspecto pouco explorado no desenvolvimento de plataformas de serviços sensíveis ao contexto. Por outro lado, a interoperabilidade entre diferentes plataformas e suas aplicações é um dos maiores desafios atuais da computação pervasiva. Uma das barreiras à interoperabilidade é a incompatibilidade entre os modelos de contexto subjacentes às aplicações. Modelos baseados em ontologias [Roman et al., 2002; Chen, 2004; Wang et al., 2004; Rios, 2003] apresentam uma representação formal de sua sintaxe e semântica de modo a garantir a consistência entre várias representações de contexto utilizadas por aplicações, plataformas de serviços e provedores de contexto. Neste sentido, a construção de ontologias comuns tem sido proposta como abordagem promissora para a interoperabilidade entre sistemas [Heflin et al., 2000; Mena et al., 2000]. Uma ontologia comum é uma formalização compartilhada de um determinado domínio de aplicação. Assim, uma formalização de conceitos sobre um determinado domínio poderia permitir que diversos aplicativos deste domínio compartilhassem um vocabulário comum sobre o assunto.

3.4. Interface Com Sensores

As informações contextuais podem ser adquiridas através de diferentes métodos. Em alguns casos é esperado que o usuário explicite uma informação relevante. Em outras ocasiões, estas informações podem ser adquiridas através do uso de sensores capazes de captar informações das mais distintas naturezas. Este segundo caso requer das plataformas uma infra-estrutura de suporte para a manipulação de informações contextuais provenientes de fontes de dados diversas, altamente dinâmicas, distribuídas e heterogêneas.

O Context Toolkit, por exemplo, encapsula a leitura de sensores a partir da definição de um conjunto de métodos comuns de acesso, fornecendo, assim, uma interface comum para as aplicações que utilizem contexto, independentemente dos sensores utilizados para captar a informação. A abordagem utilizada pelo GaiaOS para lidar com a heterogeneidade de seus espaços ativos é a de oferecer um componente denominado "*Unified Object Bus*"(UOB), responsável por manipular o ciclo de vida dos outros componentes que são executados no espaço ativo, sendo responsável pela criação, inspeção, remoção e nomenclatura de tais

componentes. Esta abordagem provê uma arquitetura aberta para a incorporação de novos modelos de componentes. A arquitetura SOCAM, por sua vez, apresenta as informações contextuais através de descrições OWL. Neste caso, o uso de ontologias permite que as informações contextuais sejam facilmente compartilhadas e acessadas por serviços sensíveis ao contexto. Os provedores de contexto realizam a aquisição das informações contextuais de mais baixo nível a partir de sensores e são registrados pelo serviço de localização de serviço. SOCAM define ainda provedores de contexto externos e internos. Os provedores de contexto externos obtêm informação contextual de fontes externas ao domínio ontológico, enquanto que os provedores de contexto internos obtêm informação contextual de fontes, por exemplo, sensores, localizados em um subdomínio ontológico.

A incorporação de infra-estruturas de suporte à aquisição, armazenamento e disseminação das informações contextuais às plataformas de serviços sensíveis ao contexto tem se mostrado uma abordagem interessante, uma vez que o acesso às informações contextuais é realizado de forma transparente para as aplicações. Um desafio que surge em decorrência desta abordagem é o de minimizar as restrições a um formato de dados específico para a representação destas informações.

3.5. Incerteza e Confiabilidade das Informações Contextuais

Um relevante requisito destacado por [Henricksen et al., 2002], que deve ser observado no desenvolvimento de plataformas de serviços sensíveis ao contexto, refere-se à confiabilidade e ao grau de incerteza associado às informações contextuais provenientes dos ambientes ubíquos. As informações contextuais, especialmente aquelas com características físicas, como localização, temperatura e tempo, geralmente são enviadas às plataformas através de sensores espalhados nos ambientes em que os usuários se encontram. Pelas características físicas e em virtude das fontes que as enviam (sensores físicos), esse tipo de informação contextual tipicamente possui, associada a ela, uma incerteza e uma imprecisão de valor. Além disso, essas informações também podem não refletir o ambiente que representam, podendo ter também diferentes graus de confiabilidade e persistência associados a ela, dependendo do tipo de tecnologia utilizada para a captura, obtenção e envio desses dados contextuais para as plataformas.

Os trabalhos de [Henricksen et al., 2002], [Henricksen et al., 2003] e [Henricksen et al., 2004] tratam desse tipo de problema e discutem algumas medidas que podem ser tomadas. O Context Toolkit busca trabalhar com o problema da confiabilidade através dos seus

componentes *widgets*. Outros trabalhos, tais como [Gu et al., 2004], utilizam estratégias e abordagens como o uso de redes bayesianas para lidar com as incertezas dos dados contextuais. Essa abordagem, ao tratar as informações contextuais com uma visão probabilística, tenta estimar e medir os diferentes graus de certeza e confiabilidade das informações providas por diferentes fontes de contexto.

3.6. Tecnologias de Distribuição e Integração

Diversas tecnologias vêm sendo adotadas como paradigma de distribuição pelas plataformas de serviços sensíveis ao contexto. Arquiteturas centralizadas compartilham a vantagem inegável de serem mais simples. Por outro lado, a complexidade dos sistemas atuais aliada à diversidade e dimensões da maioria das instalações computacionais, dificultam a adoção de soluções completamente centralizadas. Já as arquiteturas distribuídas, além de mais robustas, incluem a facilidade de crescimento modular e a possível distribuição de tarefas, retirando de um único ponto o custo e a responsabilidade de todo o processamento (ao custo da sua maior complexidade de implementação).

Além das abordagens canônicas acima descritas, uma boa parte das atuais plataformas tira proveito das diferentes arquiteturas existentes, adotando soluções híbridas. O *Context Broker Architecture* (CoBrA) fornece uma arquitetura baseada em agentes para o desenvolvimento de aplicações sensíveis ao contexto em espaços inteligentes. Trata-se de uma arquitetura centralizada, baseada no componente *context broker*, que compartilha um modelo de contexto através da comunicação entre agentes distribuídos, serviços e dispositivos. A utilização de agentes móveis pode introduzir algumas vantagens em relação aos paradigmas tradicionais, como robustez, tolerância à falhas e execução assíncrona.

O GaiaOS fornece uma infra-estrutura de *middleware* distribuída, baseada em CORBA, que coordena os recursos computacionais de um ambiente físico. O uso da tecnologia de distribuição CORBA oferece uma infra-estrutura para comunicação heterogênea, permitindo aos objetos de sistemas distribuídos comunicarem-se de forma transparente e multiplataforma. Já a plataforma WASP (*Web Architectures for Service Platforms*) utiliza *Web Services* como tecnologia de distribuição. Este tipo de tecnologia, difere-se principalmente da comunicação CORBA pelo fato de usar XML como formato para troca de dados e mensagens, o que também permite que sistemas de diferentes plataformas se comuniquem.

Neste sentido, a adoção de soluções de distribuição e integração híbridas, onde mecanismos centralizados interagem com módulos distribuídos, mesclam as vantagens de cada abordagem, tentando chegar a um ponto de equilíbrio entre desempenho, simplicidade, abrangência e robustez.

3.7. Segurança e Privacidade

Apesar da sua evidente importância em ambientes ubíquos, onde muitas vezes informações são colhidas sem o conhecimento e/ou consentimento explícito dos usuários, os requisitos de privacidade e segurança somente recentemente começaram a ser explorados pelas plataformas. A arquitetura CoBrA torna essa preocupação explícita ao dedicar um módulo específico para tratar e gerenciar a privacidade dos usuários das aplicações. No Context Toolkit, a segurança do sistema é feita de forma bastante simplória, através apenas do controle de acesso por origem/destino. Quando uma aplicação solicita uma informação contextual, ela deve enviar juntamente com o pedido de requisição, o seu código identificador. Apenas as informações solicitadas cujos identificadores estiverem habilitados a recebê-las são de fato enviadas para as aplicações.

Já na plataforma GaiaOS, os serviços de segurança são mais elaborados e incluem autenticação, controle de acesso, carga dinâmica segura de componentes, rastreamento seguro e privacidade de localização. Esta plataforma emprega um Serviço de Autenticação que utiliza credenciais para verificação de identidade de usuários. Estas credenciais permitem ao Serviço de Controle de Acesso prover um controle de acesso baseado em regras e perfis (*roles*). A plataforma WASP, em uma de suas extensões [Drost, 2004], também dedica uma camada para o tratamento de privacidade em torno de suas operações e para as aplicações. A abordagem utilizada se baseia no uso de ontologias de tarefas [Rajpathak, 2001] para descrever modelos de visões e de privacidade para os usuários da plataforma. Essa abordagem permite garantir a segurança e a confiabilidade das transações e execuções dos usuários que tenham determinadas restrições de privacidade.

Nas demais arquiteturas avaliadas, as preocupações com os requisitos de segurança e privacidade se restringem aos aspectos particulares do problema, como, por exemplo, a segurança das informações trocadas entre seus componentes, o tratamento de inconsistências e de confiabilidades dos dados armazenados e de perfis, e outros. Entretanto, não há, muitas vezes, a definição de uma política geral de segurança e privacidade para toda a plataforma, algo essencial em sistemas ubíquos reais.

3.8. Escalabilidade

O requisito de escalabilidade, apesar de pouco explorado em virtude da característica experimental da maioria das plataformas propostas, pode ser avaliado pelo projeto e execução de testes de estresse simulando situações reais, com muitas aplicações sendo executadas de maneira simultânea solicitando informações a todo instante. Os módulos das plataformas devem ser testados e avaliados de maneira integrada a fim de antecipar conclusões a respeito da capacidade de crescimento e desempenho das mesmas.

Quanto à escalabilidade e à capacidade de atender a um grande número de aplicações simultaneamente, a infra-estrutura proposta pelo *Solar System* traz uma abordagem interessante, que facilita o atendimento desse requisito, tendo em vista que a estrutura formada por operadores e grafos acíclicos dirigidos apresenta uma grande flexibilidade e capacidade natural de expansão. Em contrapartida, essa vantagem oferecida traz um custo considerável de gerenciamento e de manipulação de todo o sistema. De maneira bem distinta, a arquitetura CoBrA foi desenvolvida através do conceito de centralização das decisões e das operações em um *broker* controlador do sistema, distinguindo-se do Solar System, que é uma estrutura altamente distribuída. Dessa forma, a CoBrA possui um poder de gerenciamento e de controle do sistema bem maior do que o Solar System, mas deixa a desejar quando o número de aplicações solicitando serviços à plataforma cresce muito, ou seja, o *Context Broker* é, em tese, uma infra-estrutura pouco escalável.

3.9. Descoberta de Serviços

A natureza móvel e dinâmica das aplicações sensíveis ao contexto exige a descoberta de novos serviços frente às mudanças ocorridas no ambiente em que estas estão inseridas. Portanto, uma plataforma de serviços sensíveis ao contexto deve prover mecanismos e protocolos pelos quais dispositivos conectados a rede ou serviços de software tornem-se visíveis e disponíveis para as aplicações interessadas.

No processo de descoberta, a maioria das abordagens utilizadas baseia-se na comparação da descrição dos serviços, que pode ser feita com base em nomes, identificadores numéricos, árvores atributo-valor e etc. A comparação baseada em pares de atributo-valor, onde é informado o tipo de serviço e o nome, possui a desvantagem de não ser muito expressiva. Não há, portanto, um padrão confiável para identificar um mesmo serviço. As arquiteturas SOCAM e CoBRA, utilizam uma descrição mais rica para os serviços, baseada

no uso de ontologias, e especificam semanticamente conceitos e relacionamentos utilizados na descrição de um serviço, permitindo integração entre diferentes sistemas.

Além disso, normalmente é utilizada uma base de dados estática para a publicação dos serviços, onde as informações contidas não são atualizadas com freqüência e podem, por isso, não refletir a disponibilidade real dos serviços ali publicados. Por outro lado, a natureza dinâmica das aplicações sensíveis ao contexto requer informações atuais e disponíveis.

Algumas plataformas, como a GaiaOS, funcionam sem o apoio de uma base de dados, sendo a descoberta realizada através de anúncios enviados à rede. Esta abordagem pode limitar o alcance desses anúncios a um ambiente pequeno como uma rede local. O Context Toolkit dispõe de um módulo denominado *Discoverer,* responsável pela manutenção de um registro de informações de todos os componentes e suas respectivas permissões de acesso. Quando um componente é instanciado, o *Discoverer* é informado de sua presença e mantém seu registro atualizado.

A descoberta de serviços não é totalmente contemplada pelas arquiteturas avaliadas. Poucas utilizam descrição semântica para descrever serviços, ou um mecanismo dinâmico para descoberta de novos serviços. É comum na maioria das plataformas a utilização de alguma base de dados para o armazenamento das descrições dos serviços, porém, nem todas oferecem mecanismos robustos para seleção dos serviços desejados. Em um ambiente dinâmico das aplicações sensíveis ao contexto, é essencial que os mecanismos de descoberta atendam a esses aspectos, para que possam refletir com fidelidade as constantes mudanças ocorridas no ambiente.

3.10. Personalização dos Serviços Para os Usuários

Um outro requisito desejável no âmbito das plataformas sensíveis ao contexto diz respeito à adaptação dos serviços requisitados pelas aplicações à plataforma de acordo com as características e desejos dos usuários, expressos, por exemplo, através de perfis ou passados como parâmetros na requisição dos serviços. Assim, pode-se dizer que os serviços disponibilizados pela plataforma às aplicações são, de certa forma, personalizados para cada usuário.

Algumas plataformas, como o Context Toolkit, tratam dessa questão simplesmente deixando o controle das entidades de contexto para as aplicações e para os usuários (*context ownership*), ou seja, a personalização do usuário, nesse caso, é de responsabilidade das

aplicações, não existindo um controle mais aprimorado desse tema por parte da plataforma. Já a WASP disponibiliza uma série de repositórios de perfis de usuários que são analisados ao longo dos processamentos da plataforma, exigindo assim acesso e verificação dos perfis e desejos dos usuários nas ações tomadas.

Outras plataformas também verificam os perfis dos usuários antes de tomadas de decisões e disparos de ações, como a GaiaOS e a CoBrA. Há também propostas, como a apresentada em [Efstratiou et al., 2001], que utilizam os perfis dos usuários para resolver questões de impasses e conflitos entre as requisições das aplicações ou mesmo apoiar decisões de adaptação da própria plataforma em razão desses perfis. Assim, os conflitos internos das plataformas podem ser resolvidos dinamicamente sem a intervenção direta dos usuários, pois seus perfis estariam refletindo seus desejos. Existem também iniciativas de uso de padrões como o CC/PP (*Composite Capabilities/Preferences Profile*) [Indulska et al., 2003] em sistemas sensíveis ao contexto com o intuito de proporcionar uma melhor personalização dos serviços aos usuários. O CC/PP é uma descrição das capacidades de um dispositivo ou das preferências de um usuário. Essas descrições podem ser usadas para a adaptação da apresentação de um conteúdo em função de um dispositivo utilizado ou de acordo com o usuário que navega através de um documento.

Em muitas situações, os usuários podem desejar que as interações deles com o sistema sejam realizadas de forma anônima. Nestes casos, a plataforma deve estar preparada para realizar tarefas voltadas para esses usuários anônimos, sem uma identificação explícita de quem está solicitando serviços ao sistema. Essa maneira do sistema enxergar o usuário pode ser entendida como um mecanismo de preservação da privacidade do solicitante.

A personalização dos serviços para os usuários é, sem dúvida, um importante tópico dentro da área de sistemas ubíquos e sensíveis ao contexto. Os perfis previamente determinados pelos usuários das aplicações são também fontes riquíssimas de informações contextuais que podem influenciar de alguma maneira no comportamento das aplicações, caracterizando ainda mais esse tipo de sistema que se adapta a determinadas situações de acordo com o contexto envolvido.

3.11. Coordenação Entre Aplicações

Um outro desafio da computação sensível ao contexto diz respeito à resolução de conflitos e coordenação entre aplicações. Em um cenário em que diversas aplicações são executadas simultaneamente, compartilhando recursos de um mesmo ambiente e requisitando

serviços à plataforma, ações contraditórias podem ser solicitadas. Uma aplicação hipotética pode solicitar que a luminosidade de um ambiente seja aumentada e outra aplicação pode solicitar que essa luminosidade seja diminuída. Além disto, caso as aplicações tenham sido construídas sem o suporte de uma plataforma geral, nem mesmo é garantido que uma enxergará as ações da outra.

Para a resolução de conflitos, torna-se necessária a especificação de políticas de prioridades e cooperação entre aplicações. Uma das maneiras usadas para se chegar a este resultado é através de uma estrutura de agentes, cada um com seus objetivos e interface para negociação, podendo aceitar, rejeitar e modificar termos de negócio [Rahwan et al., 2005]. Pode-se, como mencionado em [Efstratiou et al., 2001], ter fatores prioritários. O trabalho desenvolvido em [Efstratiou, 2004] definiu uma série de requisitos para suportar a adaptação entre várias aplicações *context-aware*: a separação entre monitoramento e o atuador, a externalização do estado da aplicação para um modo possível de ser lido por toda a plataforma e a externalização da política de adaptação de cada aplicação a ser executada pela plataforma.

3.12. Persistência de Dados Contextuais

As aplicações sensíveis ao contexto lidam com dados heterogêneos, provenientes de vários sensores e dispositivos heterogêneos, com configurações e capacidades diferenciadas. Para se construir uma arquitetura de banco de dados para suporte a tais dados, uma das alternativas é se obter uma visão centralizada do conhecimento contextual, o que nos leva à disponibilização de um *middleware* integrador de dados. Essa alternativa é interessante à medida que preserva e se aproveita de estudos e desafios já pesquisados pela área de integração de dados e bancos de dados distribuídos; entretanto, tal tarefa é tornada mais complexa pela necessidade de se fazer a integração destes dados contextuais, de características bastante particulares, com informações tradicionais e bancos de dados legados.

Nesse novo ambiente, as fontes são dinâmicas, e tem informações extras, como precisão de medições. Deve-se averiguar sempre se as fontes estão disponíveis a fim de se evitar a inconsistência de dados. Em aplicações sensíveis ao contexto, dados espaço-temporais têm grande importância, há a necessidade de se guardar preferências de usuário, e existe ainda a necessidade de conversão das informações contextuais para diferentes formatos e níveis de abstração. Todas essas peculiaridades muitas vezes se refletem no próprio formato das consultas utilizadas pelas aplicações sensíveis a contexto. Algumas extensões para SQL foram

propostas para se lidar com intervalos de tempo e espaço. Ocasionalmente, novas linguagens são propostas [Fritsch et al., 2000].

Além dos problemas apontados anteriormente, ainda existem questões relativas à mobilidade das fontes contextuais, de onde podem ser aproveitados estudos da área de Bancos de Dados Móveis, e pró-atividade de banco de dados, estudada pela comunidade de Bancos de Dados Ativos. Todas estas questões mostram que não basta apenas utilizar Sistemas Gerenciadores de Bancos de Dados tradicionais para o armazenamento das informações contextuais, mas sim mesclar o conhecimento adquirido por diversas áreas de estudo para uma maior adequação a esses novos desafios.

3.13. Conclusão do Capítulo

O desenvolvimento de aplicações sensíveis ao contexto envolve a superação de uma série de desafios teóricos e tecnológicos [Dockhorn et al., 2003]. A análise destes desafios permitiu observar que as plataformas de serviço relatadas na literatura ainda não contemplam grande parte dos requisitos impostos pela Computação Sensível ao Contexto, o que abre espaço para propostas e investigações adicionais.

No próximo capítulo é proposta uma plataforma para o desenvolvimento e execução de aplicações móveis sensíveis ao contexto. A plataforma proposta teve sua modelagem norteada a partir dos requisitos aqui levantados e o projeto de sua arquitetura foi concebido vislumbrando-se cenários de aplicação reais..

4. A Plataforma Infraware

Este Capítulo descreve a proposta inicial de arquitetura para a plataforma Infraware. O projeto arquitetural da plataforma Infraware visa investigar e tratar, de forma integrada, a maioria das questões e desafios mencionados no Capítulo 3, formando, assim, um ambiente propício ao desenvolvimento de aplicações ubíquas reais em diferentes domínios.

A Infraware usou como base a arquitetura proposta pelo projeto WASP [Dockhorn, 2003], um projeto holandês desenvolvido conjuntamente pela *University of Twente*, *Telematica Instituut* e *Ericsson*, e que posteriormente teve continuidade no Laboratório de Pesquisas em Redes e Multimídia (LPRM) do Departamento de Informática da UFES, através de uma parceria informal com o ASNA (*Architecture and Services for Network Applications*) Group, da *Faculty of Electrical Engineering*, *Mathematics and Computer Science* da *University of Twente*. Neste sentido, algumas questões pouco exploradas pelo WASP foram tratadas em maiores detalhes pela plataforma Infraware, com destaque para: interpretação de contexto, gerência de subscrição, gerência de serviços, resolução de conflitos entre aplicações e interface com sensores heterogêneos.

A Figura 4-1 ilustra a arquitetura geral proposta para a plataforma Infraware.

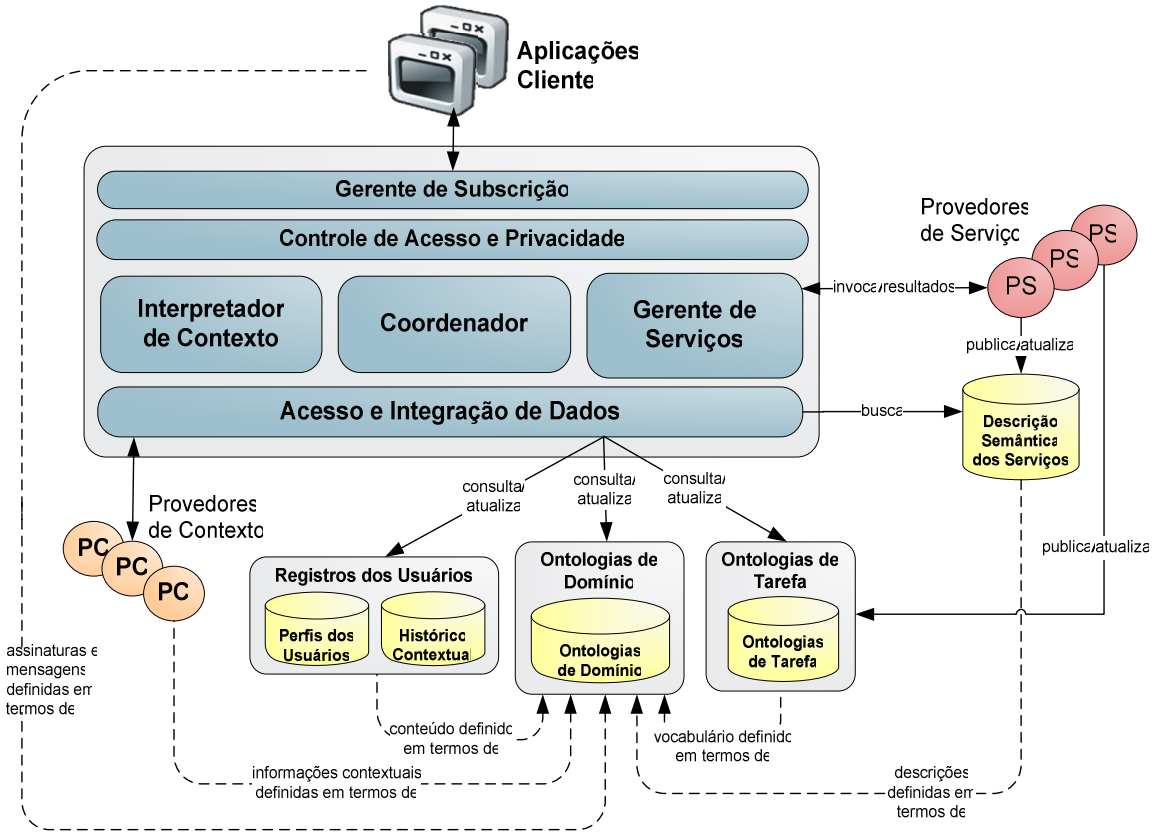

Figura 4-1 – Arquitetura Geral da Plataforma Infraware

O projeto da plataforma Infraware introduz componentes específicos para o tratamento dos requisitos funcionais relacionados ao tratamento dos pedidos de subscrição enviados pelas aplicações clientes, à interpretação de contexto, à aquisição e integração de dados contextuais heterogêneos, à gerência integrada de serviços com descrição semântica, à resolução de conflitos e coordenação entre aplicações e ao tratamento de questões relacionadas à privacidade e segurança. Além disso, para suportar à personalização dos serviços para os usuários da plataforma, são armazenados no repositório de Registros dos Usuários os perfis contendo informações referentes às preferências dos usuários, juntamente com seus históricos de informações contextuais.

A modelagem e o tratamento interno das informações contextuais são realizados através de modelos especiais que descrevem o domínio contextual em que uma determinada aplicação ou serviço está inserido. Esses modelos são largamente utilizados pelos componentes do middleware em suas funcionalidades específicas e no entendimento do tipo de dado e de informação manipulada. A infraware utiliza Ontologias [Guarino, 1998] para especificar modelos formais extensíveis que descrevem não somente o domínio das

aplicações, mas também os serviços. Essa abordagem diferenciada da arquitetura Infraware provê meios de configurar as interações entre as aplicações e a plataforma em tempo de execução (*run-time*). O uso de ontologias permite uma maior riqueza em expressividade semântica na descrição das entidades de contexto e serviços, além de possibilitar ganhos de interoperabilidade entre os componentes da Infraware.

Adicionalmente, a Infraware pode ser estendida pela adição de novos serviços e entidades bastando estender as ontologias que descrevem os modelos utilizados pela plataforma. A incorporação desse nível de flexibilidade à plataforma Infraware tem como objetivo torná-la adequada para o desenvolvimento de uma larga gama de aplicações sensíveis ao contexto, nos mais diferentes cenários de aplicação.

Os principais componentes da plataforma são descritos a seguir.

4.1. Repositórios

Na arquitetura da plataforma Infraware, destacam-se elementos responsáveis pelo armazenamento histórico de informações contextuais relacionadas a cada usuário, como o repositório Histórico de Informações Contextuais e o Repositório de Perfis de Usuários, que armazena descrições semânticas a respeito das preferências de cada usuário, oferecendo suporte extra de informações para a personalização dos diversos serviços às aplicações.

Ontologias de Domínio [Guarino, 1998] são compartilhadas entre os componentes da Infraware e descrevem conceitos gerais relacionados a um domínio de aplicação, seus relacionamentos e restrições através de uma semântica bem definida e de uma terminologia comum.

Os serviços têm suas descrições publicadas no Repositório de Descrição Semântica de Serviços. Essas descrições também são baseadas em ontologias e descrevem de maneira semântica não apenas a funcionalidade desempenhada pelo serviço, mas também os tipos de dados e informações manipuladas.

Além disso, há também o repositório de Ontologias de Tarefas [Guarino, 1998]. Essas ontologias especificam as conceituações necessárias para se definir uma tarefa ou atividade genérica, através da especialização de conceitos introduzidos nas ontologias de domínio e na descrição semântica dos serviços.

4.2. Gerente de Subscrição

As aplicações móveis sensíveis ao contexto são os clientes do *middleware* e interagem com a arquitetura através da assinatura de serviços previamente disponibilizados. Tal tarefa de subscrição é controlada pelo Gerente de Subscrição, que se vale de sentenças e expressões trocadas entre as aplicações e a plataforma, através de uma linguagem descritiva voltada a tal propósito, para resolver, interpretar e gerenciar as requisições e pedidos das aplicações. As informações contextuais provenientes principalmente dos provedores de contexto são manipuladas pela plataforma e utilizadas na tomada de decisões e no disparo de ações, em função das mudanças de contexto e das solicitações das aplicações.

O Gerente de Subscrição da plataforma Infraware fornece uma interface que permite as aplicações adicionar, remover, ou atualizar seus pedidos de subscrição. Esse componente também é responsável pela realização de tarefas como a validação sintática e semântica das mensagens trocadas entre a aplicação e a plataforma, o controle de prioridade em caso de subscrições concorrentes e a entrega das informações requisitadas para as respectivas aplicações.

Uma importante característica deste componente é o fato de ele prover flexibilidade suficiente para configurar os diversos tipos de interações entre aplicações e plataforma. Essas interações, e conseqüentemente as subscrições, devem suportar modelos de requisição do tipo *request-reponse*, *time-driven* e *event-driven*. Esses modelos determinam o modo de entrega das informações contextuais às aplicações. No modelo *request-reponse*, a plataforma se comporta de maneira análoga ao modelo cliente-servidor, em que a aplicação (o cliente) realiza uma requisição à plataforma (o servidor) e é atendida prontamente. No modelo *time-driven*, as aplicações solicitam à plataforma que as informações contextuais sejam fornecidas periodicamente. Já no modelo *event-driven*, a plataforma monitora a ocorrência de eventos de interesse das aplicações de forma assíncrona No modelo *event-driven*, uma subscrição pode determinar ao *middleware* como reagir a uma determinada correlação de eventos.

A Figura 4-2 apresenta uma visão geral da estrutura interna do módulo Gerente de Subscrição e seus sub-componentes.

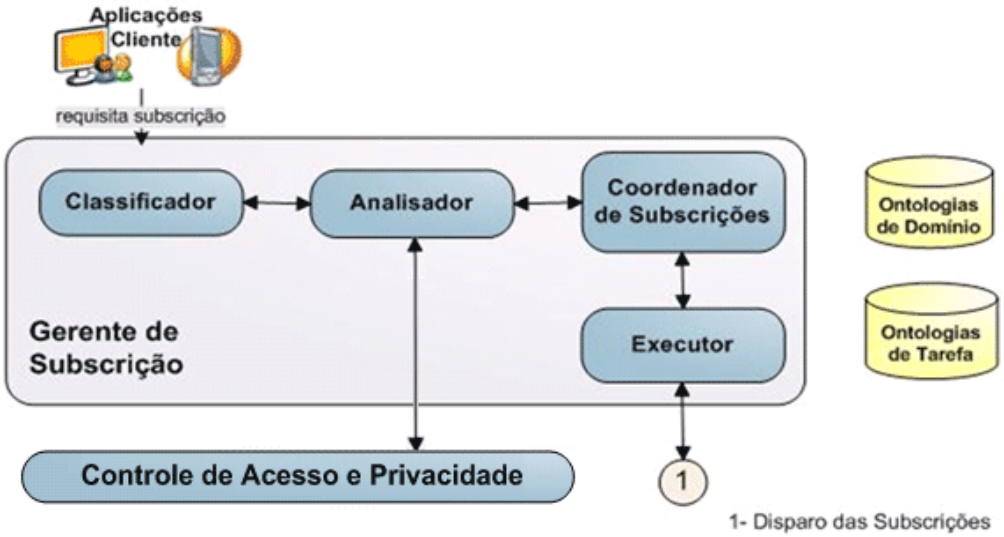

Figura 4-2 – O Gerente de Subscrição

O sub-componente Classificador é o responsável por estabelecer a ordem de atendimento entre os pedidos de subscrição recebidos pela plataforma Infraware. Para isso, é utilizada uma estrutura de fila de pedidos em que cada pedido de subscrição tem um grau de prioridade associado. Dessa maneira, requisições de maior prioridade terão preferência no atendimento de seus pedidos. Há também uma política para se evitar *starvation*, caso o número de subscrições enviadas seja elevado, já que todas as subscrições devem ser processadas.

Esses pedidos de subscrição são, posteriormente, repassados ao Analisador que, por sua vez, realiza as análises léxica e sintática das subscrições recebidas. Uma vez que os pedidos de subscrição das aplicações são descritos em termos das Ontologias de Domínio, também é realizada a validação semântica destes pedidos de acordo os termos definidos pela Ontologia de Domínio especificada para o middleware. Caso a subscrição seja uma subscrição válida, a mesma é enviada para a análise do módulo Controle de Acesso e Privacidade, que verificará se os limites de visibilidade das informações requisitadas pelas aplicações são respeitados.

O Coordenador de Subscrições é o real responsável pela conversão dos pedidos de subscrição de um formato de linguagem descritiva (ISL – *Infraware Subscription Language*) em uma estrutura de dados (SDS – *Subscription Data Structure*) que contém os campos do pedido de subscrição em formatos apropriados para serem lidos pelos demais módulos da plataforma. Finalmente, há o Executor, que é o responsável por disparar a execução das subscrições (já convertidas para o formato SDS), entre os demais componentes do

middleware. O Executor também realiza a entrega dos dados requisitados pelas aplicações após os processamentos das subscrições pelos demais módulos da plataforma.

4.3. Controle de Acesso e Privacidade

A proteção à privacidade dos usuários em ambientes de computação ubíqua torna-se um problema na medida em que sensores podem coletar informações dos usuários sem o conhecimento e/ou consentimento explícito destes. A própria fronteira entre a obtenção das informações necessárias para a execução de um serviço e a violação nos direitos a privacidade de um usuário é difícil de ser traçada. Em parte, em virtude dessa dificuldade, muitas plataformas *context-aware* simplesmente ignoram esse problema enquanto outras implementam soluções específicas em relação ao tipo de informação ou serviços utilizados.

Na arquitetura Infraware, o módulo de Controle de Acesso e Privacidade atua como um filtro sobre o fluxo de dados entre a plataforma e as aplicações, com base em um conjunto de restrições envolvendo preferências de privacidade dos usuários e as políticas de privacidade das aplicações. Resultado do trabalho iniciado por Drost [Drost, 2004] e continuado por Cruz [Cruz, 2006], esse módulo permite que os usuários controlem o acesso aa suas informações de maneira automática, transparente e não obstrutiva.

As preferências de privacidade dos usuários são descritas como um conjunto de regras que determinam quais usuários podem ter acesso a seus dados, a utilização de serviços de terceiros ou ainda os próprios serviços disponibilizados pela própria plataforma. Uma preferência é composta de um conjunto de regras, onde cada regra está associada a uma ação que é executada quando a regra é considerada valida. Essas ações permitem negar ou aceitar a requisição do dado, assim como notificar o usuário dono das informações coletadas. Além disso, toda preferência de privacidade possui um usuário responsável, não obrigatoriamente seu respectivo dono. Essa característica é necessária devido ao fato que nem sempre um usuário é responsável por suas preferências de privacidade, assim como pais podem ser responsáveis pelas regras de privacidade dos filhos.

Durante a invocação de um serviço externo à plataforma, o módulo de Controle de Acesso e Privacidade obtém as descrições das preferências do usuário e das políticas de privacidade do serviço e verifica a compatibilidade entre ambos. Se esses forem compatíveis,

o serviço é invocado a partir do módulo Gerente de Serviços. Caso contrário, uma mensagem é retornada para a aplicação indicando a incompatibilidade com o serviço requisitado. Para isso, os provedores de serviço devem publicar suas descrições no Repositório de Descrição Semânticas dos Serviços da plataforma Infraware.

A figura 4-3 apresenta uma visão geral da estrutura interna do módulo de Controle de Acesso e Privacidade.

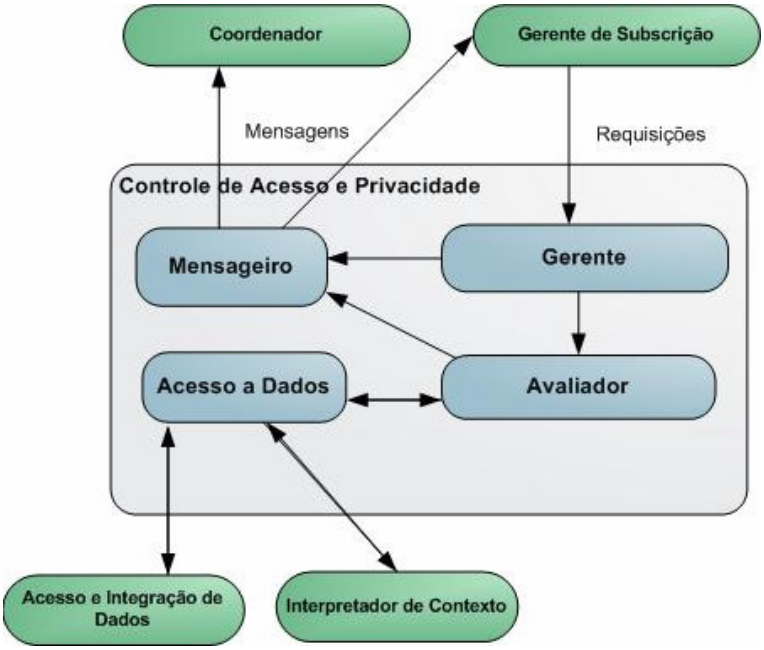

Figura 4-3 – Módulo de Controle de Acesso e Privacidade

O Gerente do módulo de Controle de Acesso e Privacidade é o sub-componente responsável pela coordenação e gerência das avaliações dos pedidos de subscrição de acordo com as preferências de privacidade dos usuários envolvidos. As diferenças entre os tipos de subscrições analisadas são relativas à invocação ou não de serviços e se os dados utilizados ou requisitados são do próprio usuário ou não. Nesse sentido, as subscrições podem ser divididas entre três tipos:

1. O usuário requisita à plataforma seus próprios dados;
2. O usuário invoca um serviço que utiliza informações suas ou de outros usuários;
3. O usuário requisita à plataforma dados de outros usuários.

O primeiro caso não oferece risco à privacidade do usuário, visto que o mesmo está requisitando a plataforma seus próprios dados. No segundo tipo é necessário verificar se a

política de privacidade do serviço invocado está de acordo com as preferências de privacidade dos usuários envolvidos e, no terceiro tipo, deve-se verificar se o usuário requisitado permite que o usuário requisitor tenha acesso a seus dados. Essas verificações são realizadas por intermédio do sub-componente Avaliador.

O componente Avaliador obtém as preferências ou contextos dos usuários, bem como as políticas dos serviços através do componente de Acesso a Dados. Este, por sua vez, disponibiliza aos demais componentes do módulo as políticas e preferências de privacidade dos usuários, as políticas dos serviços e relações entre contextos dos usuários, respectivamente. Para isso, este componente possui interfaces com outros módulos da plataforma Infraware: Acesso e Integração de Dados e Interpretador de Contexto.

Após a avaliação das políticas e preferências de privacidade dos usuários, fica a cargo do Mensageiro enviar as respostas aos componentes requisitores. Caso não haja conflito de privacidade entre os envolvidos na subscrição, o Mensageiro entregará o mesmo pedido de subscrição ao módulo indicado como receptor, e, caso contrário, enviará uma mensagem avisando sobre os conflitos detectados.

Dessa forma, o módulo de Controle de Acesso e Privacidade verifica todo o fluxo de dados entre um usuário e uma aplicação, e verifica se as preferências do usuário são compatíveis com a política de privacidade adota pela aplicação. Dessa maneira, através de mecanismos de políticas de privacidade e segurança, o módulo de Controle de Acesso e Privacidade busca garantir a segurança e a privacidade das informações contextuais providas para as aplicações.

4.4. Interpretador de Contexto

O Interpretador de Contexto é o componente responsável pela manipulação e inferência de contextos mais elaborados a partir de informações contextuais primitivas, provenientes de fontes heterogenias e distribuídas. Esse componente recebe os pedidos de interpretação das informações contextuais provenientes das aplicações, repassadas pelo componente Coordenador, e realiza inferência e manipulação semântica dos dados, gerando dessa maneira, novas informações, as quais serão utilizadas pela plataforma e aplicações. Além disso, o Interpretador de Contexto também é capaz de disparar eventos para os demais

módulos da plataforma, como o Gerente de Serviços. Os processos de inferências realizados são apoiados por informações obtidas do componente de Acesso e Integração de Dados. Maiores detalhes da operação desse componente são apresentados no Capítulo 6.

4.5. Acesso e Integração de Dados

O componente de Acesso e Integração de Dados utilizado na plataforma Infraware é resultado da incorporação e adaptação de um outro trabalho de pesquisa dentro do LPRM/DI/UFES: o CoDIMS (*Configurable Data Integration Middleware System)* [Barbosa 2000]. O CoDIMS é um ambiente configurável para integração de dados heterogênios e tem por objetivo prover transparência de localização, modelo de dados e forma de comunicação. A Figura 4-4 apresenta uma visão geral sobre este componente.

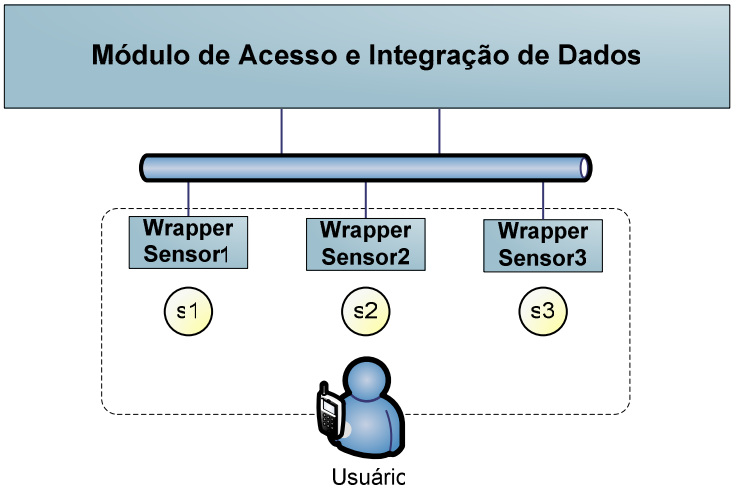

Figura 4-4 – Componente de Acesso e Integração de Dados.

Para a obtenção dos dados brutos dos sensores são utilizados agentes *wrappers* que podem implementar conversões analógico/digital, realizar algum pré-processamento sobre os dados coletados e utilizar protocolos de comunicação definidos pela plataforma. Dada a natureza heterogênea dos provedores de contexto, os dados digitalizados e armazenados nos *wrappers* podem estar em diferentes formatos, dependendo da tecnologia e padrão adotado pelo sensor utilizado.

O conjunto resultado retornado pelos *wrappers* deverá ser, então, formatado de maneira flexível e facilmente manipulável tanto pelos sub-componentes internos do

componente Acesso e Integração de Dados quanto pelos demais componentes da plataforma. Isso requer que estas informações sejam traduzidas em termos da Ontologia de Domínio especificada para a plataforma. Neste caso, para prover a integração é necessária uma transformação de modelos de dados específicos, utilizado por cada sensor, para um modelo global (único e homogêneo), definido pela plataforma.

Sendo assim, para cada domínio de aplicação e de acordo com as fontes de dados utilizadas, *wrappers* e metadados devem ser criados. Para realizar o monitoramente contínuo dos usuários com base em cada um de seus sensores, a plataforma Infraware encaminha as requisições (consultas) ao Módulo de Acesso e Integração de Dados. Este, por sua vez, é responsável pelo acesso, processamento e entrega das informações de forma tranaparente.

4.6. Gerente de Serviços

O componente Gerente de Serviços é responsável pelas atividades de publicação, descoberta e seleção de serviços na plataforma Infraware. Para isso, este componente deve descrever protocolos [Carmo, 2006] e oferecer mecanismos que permitam a descoberta automática de serviços disponíveis na rede. Uma outra atividade desempenhada por este componente diz respeito à seleção dos serviços descobertos baseada no contexto dos usuários e na descrição semântica dos serviços. Isto traz a vantagem de permitir a invocação de serviços que melhor atendam às necessidades de um usuário, ao invés de limitar a seleção com base em parâmetros pré-estabelecidos.

A Figura 4-5 apresenta uma visão geral da estrutura interna do módulo Gerente de Serviços e seus sub-componentes.

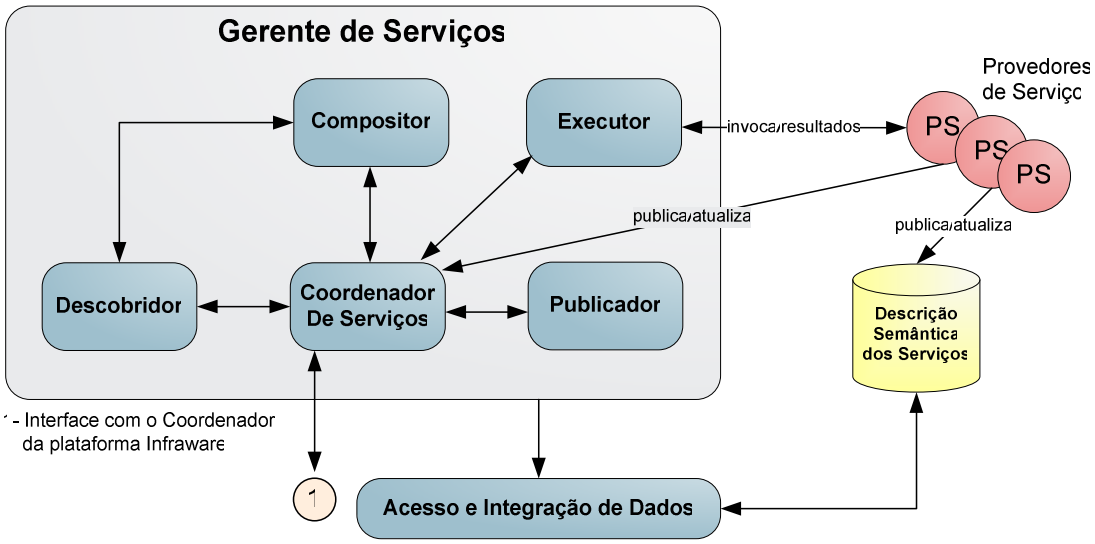

Figura 4-5 – O Gerente de Serviços.

O Coordenador de Serviços é o sub-componente responsável por controlar as interações entre os demais sub-componentes do Gerente de Serviços. Sua interface externa é acessada pelos Provedores de Serviços, pelo módulo de Acesso e Integração de Dados e pelo Coordenador da plataforma Infraware. Sendo assim, o Coordenador de Serviços atua como um roteador, encaminhando as mensagens recebidas dos Provedores de Serviço e do Coordenador da plataforma aos demais sub-componentes do Gerente de Serviços ou ao módulo de Acesso e Integração de Dados.

O Publicador é o sub-componente responsável pelo anúncio das descrições dos serviços. Ao receber uma solicitação de publicação de serviço procedente do Coordenador de Serviços, o Publicador verifica a validade da descrição a ser publicada, e, em caso afirmativo, adiciona a descrição no Repositório de Descrição Semântica de Serviços, por intermédio do módulo de Acesso e Integração de Dados. As solicitações de publicação de serviços podem ser realizadas tanto pelos Provedores de Serviços quanto pela própria plataforma. No primeiro caso, os Provedores de Serviço solicitam o anúncio de um serviço oferecido. No segundo caso, a plataforma requisita a publicação de uma descrição de composição de serviço por ela realizada. Logo, na ocorrência de uma nova requisição ao serviço previamente composto, a plataforma é capaz de encontrar sua descrição e executá-lo sem que haja a necessidade de repetir o processo de composição para o mesmo serviço.

O sub-componente Descobridor é responsável pela consulta ao módulo de Acesso e Integração de Dados em busca das descrições semânticas dos serviços disponíveis à plataforma. Esse sub-componente recebe do Coordenador de Serviços ou do Compositor as

requisições de descoberta contendo descrições semânticas das funcionalidades que o serviço a ser descoberto deve oferecer. A descoberta dos serviços é realizada através do protocolo de descoberta de serviços da plataforma Infraware [Rios, 2006]. Este protocolo utiliza uma abordagem dinâmica e segura para descoberta dos serviços e seu processo de seleção baseia-se na descrição semântica dos serviços. A descoberta dinâmica garante que o serviço descoberto esteja disponível no momento da solicitação. Para alcançar este objetivo, todos os Provedores de Serviços anunciam-se a um servidor, que armazena as descrições semânticas dos serviços e atende às requisições de descoberta feitas por clientes. Para atender este dinamismo, é utilizado o mecanismo de *leasing*. Com este mecanismo, os provedores periodicamente atualizam suas descrições junto ao servidor, garantindo que durante o tempo definido pelo *leasing* o provedor e a descrição dos serviços por ele providos estão disponíveis e atuais. O algoritmo de seleção utilizado [Costa, 2006] é baseado na descrição semântica dos serviços e permite a descoberta de serviços que atendam às características descritas pelo usuário, em vez de limitar a seleção em parâmetros pré-estabelecidos e restritos ao protocolo. Para garantir um nível aceitável de segurança das informações na descoberta dos serviços, a autenticação, a confidencialidade e o não-repúdio são aspectos considerados pelo protocolo.

O sub-componente *Compositor* é o módulo responsável pelo processo de composição semântica de serviços. Caso o *Descobridor* não encontre um serviço que, sozinho, atenda as necessidades de uma requisição, o *Coordenador de Serviços* envia ao *Compositor* um pedido de composição semântica de serviços que satisfaça à requisição original. A partir daí, baseado na descrição semântica dos serviços, o *Compositor* solicita ao *Descobridor* as sub-tarefas necessárias à composição do serviço solicitado, escolhe as sub-tarefas mais adequadas e gera o plano de execução do serviço composto.

A execução dos serviços e seu monitoramento são gerenciados pelo sub-componente *Executor.* A partir do plano de execução gerado pelo *Compositor* e enviado pelo *Coordenador*, o *Executor* invoca os serviços de forma ordenada e coreografada e gerencia as trocas de mensagens entre os mesmos. Serviços atômicos, compostos por apenas uma tarefa, são invocados diretamente pelo *Executor*, sem que seja necessária a geração do plano de execução.

4.7. Coordenador

O componente Coordenador da plataforma Infraware, como o próprio nome sugere, realiza atividades de coordenação e gerenciamento do contexto geral da plataforma, interagindo com os demais componentes do *middleware* de maneira a realizar as requisições feitas pelos usuários das aplicações. O Coordenador é responsável pelo gerenciamento, geração e disparo de planos de ações e atividades executados pelos outros componentes da plataforma. Além disso, ele também realiza, quando necessário, a resolução de conflitos entre as tarefas e atividades da Infraware e entre as diferentes requisições das aplicações.

A Figura 4-6 mostra a estrutura interna e os sub-componentes do módulo Coordenador da plataforma Infraware:

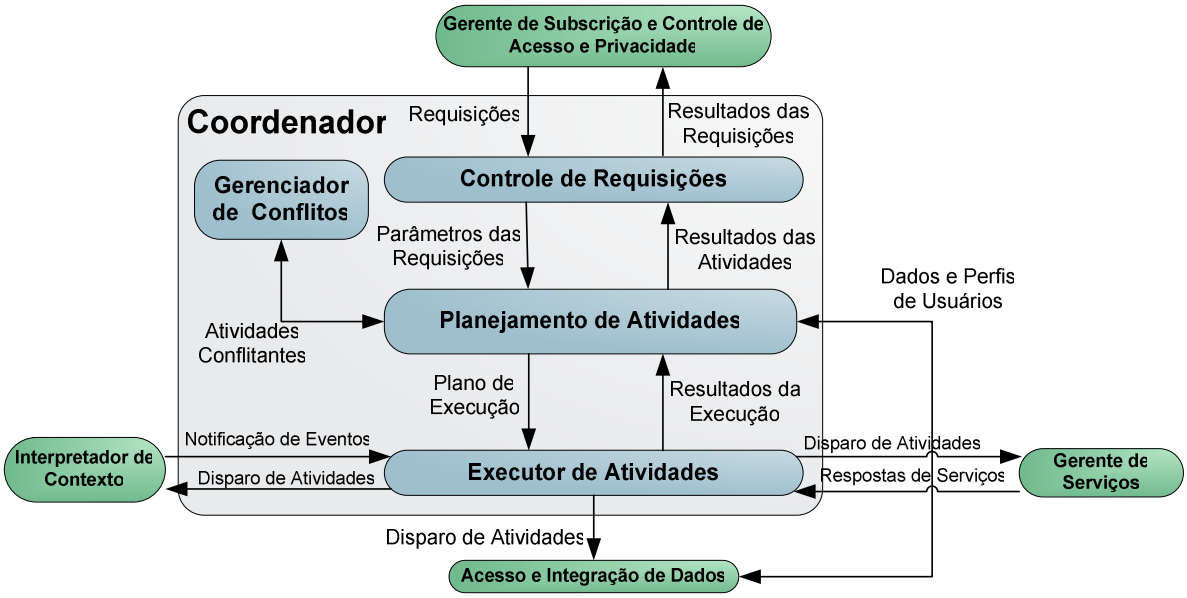

Figura 4-6 – O Componente Coordenador.

O Coordenador recebe os pedidos de subscrição previamente tratados pelo Gerente de Subscrição e devidamente filtrados e verificados pelo componente de Controle de Acesso e Privacidade. Em seguida, ele analisa e organiza as requisições, elaborando um Plano de Execução de tarefas e de atividades a ser realizado para cada requisição recebida. O sub-componente de Controle de Requisições processa e organiza as requisições entrantes, além de enviar ao sub-componente de Planejamento de Atividades dados e parâmetros necessários para a elaboração do Plano de Execução. O sub-componente de Planejamento de Atividades também utiliza informações das bases de dados e dos provedores de contexto para a formação

do Plano de Execução. Essas informações são obtidas de forma transparente e padronizadas através de solicitações ao componente de Acesso e Integração de Dados.

Em seguida, as atividades, ao serem compostas e organizadas no Plano de Execução, são executadas por intermédio do sub-componente Executor de Atividades, que dispara as respectivas ações aos componentes Interpretador de Contexto, Gerente de Serviços e Acesso e Integração de Dados. As atividades são disparadas em virtude da especificidade de cada uma e de acordo com as características das requisições e dos perfis dos usuários envolvidos. Dentre as possíveis atividades está a interpretação e a derivação de contextos específicos de um domínio ou de uma aplicação, como localização, proximidade de pessoas e ocorrência de um distúrbio na freqüência cardíaca de um paciente hospitalar. Ainda podem ser consideradas atividades solicitações de contextos primitivos ao componente de Acesso e Integração de Dados para serem enviadas ao Interpretador de Contexto, e disparo de serviços de avisos e alarmes a serem enviados a usuários de aplicações.

O sub-componente Executor de Atividades também gerencia o monitoramento de atividades que exigem um controle contínuo das informações, como o monitoramento da freqüência cardíaca de pacientes. Quando especificado no Plano de Execução, o Executor de Atividades pode disparar um pedido de monitoramento de dados cardíacos de um determinado paciente e esperar a notificação de ocorrência de algum evento observado, por exemplo, pelo Interpretador de Contexto. Além disso, respostas ou avisos de serviços realizados pelo Gerente de Serviços também são reportadas ao Executor de Atividades quando finalizadas ou na ocorrência de algum imprevisto.

Ao final do Plano da Execução, as respostas das atividades e da execução geral do Plano são retornadas aos sub-componentes de Planejamento de Atividades e de Controle de Requisições, que repassam essas respostas às aplicações solicitantes dos serviços de subscrições.

Uma importante funcionalidade atribuída ao Coordenador é a da Gerência e Resolução de Conflitos entre atividades e pedidos das aplicações. Como mencionado anteriormente, quando várias aplicações solicitam serviços simultaneamente à plataforma, ações contraditórias podem ser requeridas, tais como uma aplicação tentar aumentar e uma outra diminuir o nível de luminosidade de um ambiente. Além disso, as próprias atividades internas do *middleware* podem conflitar ao disputarem recursos, como o uso de dispositivos ou largura de banda. Para satisfazer essa necessidade de coordenação entre as atividades a serem

disparadas na plataforma, o sub-componente de Planejamento de Atividades submete o Plano de Execução elaborado para uma requisição ao sub-componente Gerenciador de Conflitos, que analisa a existência de conflitos de execução e/ou conflitos de recursos entre as atividades e propõe uma possível solução a ser realizada. Essa solução, que é baseada nos perfis e nas prioridades dos usuários, é então recebida pelo sub-componente de Planejamento de Atividades e executada no Plano de Execução corrente.

4.8. Um Cenário de Uso para a Plataforma Infraware

Esta seção apresenta um exemplo de uso da arquitetura proposta em um ambiente médico-hospitalar, ilustrando a interação entre os componentes internos da plataforma Infraware. Um ambiente médico-hospitalar é caracterizado pela presença maciça de dispositivos sensoriais responsáveis pelo monitoramento de funções vitais dos pacientes. De acordo com a enfermidade atribuída a cada paciente, sensores de diferentes naturezas são utilizados. Além disso, há um alto grau de mobilidade entre médicos e enfermeiros e a ocorrência freqüente de interrupções para a realização de atendimentos emergenciais. Em um cenário como o descrito, aplicações médicas podem explorar a natureza contextual do ambiente com o intuito de melhorar os serviços oferecidos aos pacientes e aos profissionais de saúde. Nesse caso, a plataforma pode receber e enviar dados dos pacientes aos médicos responsáveis em função dos requisitos das aplicações e das mudanças de contexto. A Figura 4-7 apresenta um cenário de uso de um sistema de Telecardiologia suportado pela plataforma Infraware.

Um médico pode solicitar o monitoramento de funções vitais de um paciente através de uma aplicação sensível ao contexto voltada a tal propósito. Inicialmente, essa aplicação realiza um pedido de subscrição à plataforma através da linguagem de subscrição definida pela Infraware e de parâmetros específicos determinados pelo usuário. O módulo Gerente de Subscrição recebe, decodifica e analisa o pedido recebido, transmitindo seus dados ao componente de Controle de Acesso e Privacidade que, através de informações de perfis do paciente, verifica se o nível de acesso requisitado pelo médico é respeitado. Em seguida, o componente Coordenador, a partir dos dados da subscrição e de modelos de privacidade e de visibilidade do usuário, gera um plano de execução de atividades que será disparado aos demais componentes do *middleware*, especialmente ao componente de Acesso e Integração de

Dados, ao Interpretador de Contexto, e ao Gerente de Serviços. Ao componente de Acesso e Integração de Dados caberá a obtenção de dados contextuais primitivos, tais como a freqüência cardíaca e a pressão sanguínea, provenientes das diversas fontes de informações existentes, que, nesse caso, são sensores ligados ao paciente.

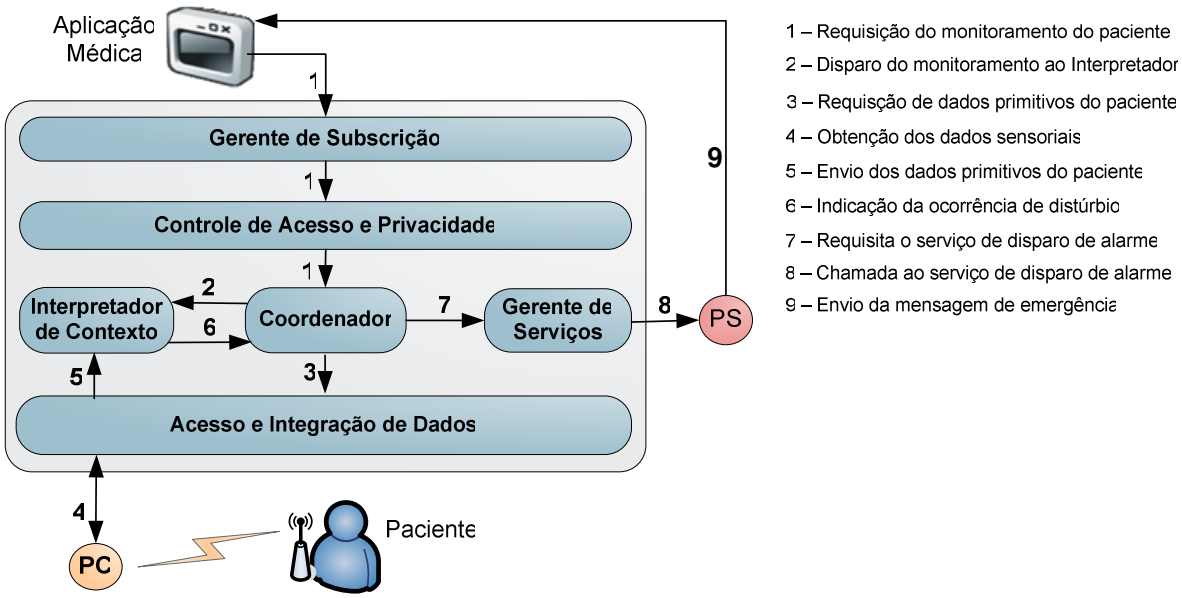

Figura 4-7 – Utilização da plataforma Infraware em um cenário médico-hospitalar

O componente de Acesso e Integração de Dados realiza consultas aos provedores de contexto e abstrai para os demais componentes do *middleware* a complexidade da tecnologia utilizada nos sensores. As informações contextuais primitivas coletadas dos sensores pelo componente de Acesso e Integração de Dados são entregues ao Interpretador de Contexto, que realiza o monitoramento desses dados e, através de regras de inferências, pode gerar novas informações contextuais mais abstratas. No caso de detecção de alguma anomalia ou distúrbios nos sinais vitais do paciente monitorado, o Interpretador de Contexto acionará o Gerente de Serviços para a execução do serviço especificado pelo pedido de subscrição original. Este serviço poderia ser, por exemplo, o envio de mensagens de emergência à aplicação solicitante do monitoramento. O componente de Descoberta de Serviços do Gerente de Serviços irá buscar e localizar os serviços de alertas disponíveis na ocorrência da emergência. Por fim, o serviço é disparado e é enviada a mensagem de emergência, alertando sobre o estado clínico do paciente monitorado.

4.9. Conclusão do Capítulo

Este capítulo apresentou uma proposta de arquitetura conceitual para a plataforma Infraware. São componentes de destaque da arquitetura proposta: o *Gerente de Serviços*, que fornece uma interface que permite aos clientes do Infraware removerem, adicionarem ou atualizarem pedidos de subscrições; o *Controle de Acesso e Privacidade*, que atua como filtro estabelecendo limites de visibilidade sobre o fluxo de dados trocado entre a plataforma e as aplicações; o módulo de *Acesso e Integração de Dados*, que provê uma interface homogênea e transparente de acesso às informações oriundas de diversas fontes de contexto; o *Gerente de Serviços*, que, baseado na descrição semântica dos serviços, é responsável pelas atividades de publicação, descoberta e seleção de serviços; o *Coordenador*, que exerce atividades de gerenciamento e monitoração do estado geral da plataforma; o *Repositório de Ontologias de Domínio*, que promove a organização do conhecimento acerca do domínio das aplicações de forma compartilhada entre os demais componentes da Infraware; e, por fim, o *Interpretador de Contexto*, responsável pela derivação e inferência de novos contextos mais elaborados a partir de informações contextuais provenientes de diferentes fontes.

O detalhamento de todos os componentes da plataforma, assim como suas respectivas implementações, estão fora do escopo deste trabalho e são, inclusive, temas de pesquisas já iniciadas no âmbito do projeto Infraware. Entretanto, com a necessidade imediata de se materializar protótipos de aplicações sensíveis ao contexto, o componente responsável pela interpretação de contexto foi escolhido para ter sua especificação interna projetada e implementada no presente trabalho. Para tanto, torna-se necessária a adoção, por parte da plataforma Infraware, de um modelo que descreva as informações contextuais e o domínio em que uma determinada aplicação ou serviço está inserido, uma vez que a escolha do modelo adotado impacta diretamente em questões relacionadas à interpretação de novos contextos.

O próximo capítulo apresenta uma avaliação de diferentes propostas de modelos de contexto referenciados na literatura da área e define o modelo a ser adotado pela plataforma Infraware.

5. Modelagem e Interpretação de Contexto

Um problema em particular, identificado no Capítulo 3, que deve ser tratado pelas plataformas de serviços sensíveis ao contexto é a definição de um modelo que descreva as informações contextuais e o domínio em que uma determinada aplicação ou serviço está inserido. Brown [Brown, 1996] e Finney [Finney et al., 1996] coincidem em afirmar que o uso do contexto está ıntimamente relacionado ao modo como o percebemos, e que se requer uma correta interpretação do mesmo para que este resulte de utilidade.

Vale ainda ressaltar que a abstração fornecida pelo modelo de contexto adotado e a expressividade da linguagem utilizada na especificação do mesmo impactam diretamente em questões relacionadas à interpretação e inferência de novos contextos mais abstratos. Nesse sentido, a inferência de contextos pode ser descrita como a capacidade de se produzir uma nova informação contextual, reinterpretanto outras informações de contexto existentes.

Diferentes propostas de modelos de contexto podem ser encontradas na literatura, contudo a maioria destas propostas apresenta flexibilidade restrita para a modelagem de aplicações em diferentes domínios de aplicação. Estas propostas também carecem, em sua maioria, de uma representação formal de sua sintaxe e semântica e resultam da diversidade de informações contextuais e de um espectro cada vez mais amplo para utilização de aplicações sensíveis a contexto [Mostéfaoui et al., 2004].

Strang e Linnhoff-Popien [Stang et al., 2004] classificam as abordagens correntes mais utilizadas para a modelagem de contexto de acordo com o esquema de estrutura de dados predominantemente utilizado para a representação das informações contextuais: modelos de pares atributo-valor, baseados em esquemas, modelos gráficos, modelos orientados a objetos, baseados em lógica e baseados em ontologias.

5.1. Modelos de Pares Atributo-valor

Os modelos baseados em pares (ou tuplas) de atributo-valor são a forma mais simples e elementar de estrutura de dados utilizada para modelagem da informação contextual. Neste

modelo, as informações contextuais são descritas como um conjunto de pares de atributo-valor, no qual um atributo descreve uma propriedade do contexto em questão. Embora seja o modelo de mais simples implementação, este modelo possui expressividade limitada devido à ausência de tipagem e interrelacionamento entre as informações contextuais representadas.

Com relação à interpretação das informações contextuais, a natureza simples deste tipo de modelagem permite apenas a realização de comparações exatas entre os valores associados a cada informação representada, tornando este tipo de abordagem inadequada para propósitos mais sofisticados. Schilit et al. utiliza pares de atributo e seus respectivos valores para modelar as informações contextuais, que são providas às aplicações como variáveis de ambiente [Schilit et al., 1993].

5.2. Modelos Baseados em Esquemas

Em modelos baseados em esquemas, as informações contextuais são representadas como estruturas de dados hierárquicas, especificadas textualmente e delimitadas por *tags* de marcação (*markup tags*). Normalmente é adotada uma alguma linguagem de marcação derivada do SGML (*Standard Generalized Markup Language*), normalmente o próprio XML (*eXtensible Markup Language*).

Diferentemente do modelo anterior, no modelo baseado em esquemas a informação contextual possui uma estrutura bem definida e de fácil extensão, permitindo a especificação de novas informações de contexto. Além disso, a representação estruturada dos dados pode ser compartilhada a partir de um grande número de aplicações, uma vez que a definição conferida pela W3C garante a uniformidade dos dados e independência de aplicações ou distribuidores. A possibilidade de criar *tags* de um modo arbitrário (respeitando sempre as regras de aninhamento) permite ainda adaptar a estrutura de um documento XML a praticamente qualquer situação específica.

Entretanto, umas das maiores críticas atribuídas a esse tipo de modelagem é que sua especificação não define semântica para as *tags* definidas pelos usuários da linguagem. Segundo [Erdmann et al., 2001], a linguagem XML permite aos usuários adicionar estruturas arbitrárias em seus documentos, mas nada diz a respeito do significado dessas estruturas. Ou seja, para fins de interpretação das informações contextuais, falta a este tipo de modelagem de

contexto expressividade semântica acerca das informações modeladas, uma vez que a estrutura estabelecida pelas *tags* não significa nada para o computador por representarem apenas uma sintaxe bem definida, omitindo sua semântica.

O *framework* Context Toolkit [Dey, 2000] encapsula as informações contextuais enviadas por sensores em arquivos no formato XML. Os demais componentes do *framework* compartilham o mesmo mecanismo de comunicação (XML sobre HTTP), o que permite a transparência no processo de aquisição de contexto. Com isso, as aplicações não necessitam saber o formato dos dados utilizados por cada sensor.

O Context Kernel [Arruda Jr, 2003] é dedicado ao gerenciamento de informações de contexto provindas das aplicações que o utilizam. Foi definida uma modelagem baseada em dimensões primitivas e dimensões derivadas. As dimensões primitivas podem ser armazenadas e recuperadas independentemente de outras dimensões. As dimensões derivadas são determinadas por meio de regras que contêm premissas e inferências. O armazenamento e recuperação das informações contextuais primitivas são realizados com base em seis dimensões pré-definidas: *who* (identificação), *where* (localização espacial), *when* (localização temporal), *how* (modo de captura ou de acesso), *what* (atividade) e *why* (motivação). A estrutura básica é definida utilizando-se a linguagem XML para classificação e estruturação das mensagens. No fragmento XML da Figura 5-1, uma aplicação está armazenando o seguinte contexto primitivo: “o professor José Gonçalves está no LPRM, em um sábado de manhã”.

```
<context>
  <primitive>
    <whoS><who type="teacher" value="José Gonçalves"/></whoS>
    <whereS><where type="room" value="LPRM"/></whereS>
    <whenS><when type="date" value="Sábado AM" qualifier="date"/></whenS>
  </primitive>
</context>
```

Figura 5-1 – Exemplo de modelagem de contexto adotada pelo Context kernel

5.3. Modelos Gráficos

Essa categoria utiliza modelagens conceituais para representar as informações contextuais e seus inter-relacionamentos, normalmente modelados por meio de linguagens

gráficas, tais como extensões de UML (*Unified Modeling Language*). Modelos gráficos são, em geral, capazes de modelar composições e inter-relacionamentos entre informações contextuais, embora sejam limitados quanto à modelagem do comportamento dinâmico do contexto.

A utilização de formalismos como a OCL (*Object Constraint Language*), por exemplo, pode ser usada na eliminação de ambigüidades, na adição de semânticas de restrições e inclusão de pré e pós-condições aos modelos, tornando os diagramas e especificações mais consistentes. Embora a estrutura estática da UML possa ser representada em OCL por meio das regras de boa-formação, a sua semântica dinâmica ainda permanece descrita pelo uso de linguagem natural e ambígua.

A abordagem visual de representação adotada por estes modelos exige ainda que os elementos gráficos da simbologia adotada sejam posteriormente transformados em ações, atributos, condições, eventos, métodos (operações) e código, a partir da especificação de um conjunto de diagramas de modelagem. Esse desacoplamento explícito entre artefatos conceituais e executáveis reais permite que o resultado da implementação não siga à risca o modelo especificado.

O modelo de contexto adotado por Henricksen [Henricksen et al., 2002] e seus atributos adicionais (incluindo características de tempo e classificação das informações contextuais) utilizava tanto modelos de Entidades e Relacionamentos quanto diagramas UML. Posteriormente, esse modelo foi reformulado para o *Object-Role Modeling* (ORM) para a modelagem de contexto [Henricksen et al., 2003; Henricksen et al., 2004]. Trata-se de uma extensão da técnica ORM, com o detalhamento de algumas características das informações contextuais e descrição de suas propriedades. Em ORM, o conceito básico de modelagem é um fato, e a modelagem de um domínio envolve a identificação apropriada de tipos de fatos e posteriormente o papel que cada entidade desempenha sobre os fatos. Em sua extensão à ORM, Henricksen classifica os fatos de acordo com sua natureza estática (fatos que se mantêm inalterados) ou dinâmica, sendo que estes últimos são ainda classificados de acordo com a forma pela qual foram obtidos (*profiled*, *sensed* ou *derived*). Além disso, são consideradas questões a respeito da natureza temporal de alguns fatos e a relação de dependência entre fatos.

A Figura 5-2 ilustra um exemplo onde é utilizada a modelagem proposta por Henricksen [Henricksen et al., 2003]. No modelo sugerido, deseja-se modelar o cenário em

que uma pessoa (*Person*) envolvida na realização de alguma atividade (*Activity*) possua permissão para a utilização de algum dispositivo (*Device*). Este dispositivo, por sua vez, captura periodicamente dados referentes à sua localização e estes dados são também utilizados para a determinação da localização da pessoa em questão. Esta mesma pessoa está ainda associada a um canal de comunicação. Este, por sua vez, possui um modo de comunicação e requer a utilização de algum dispositivo. Com base nesses fatos, o modelo destaca, por exemplo, a relação de dependência entre a atividade desenvolvida pela pessoa e sua respectiva localização.

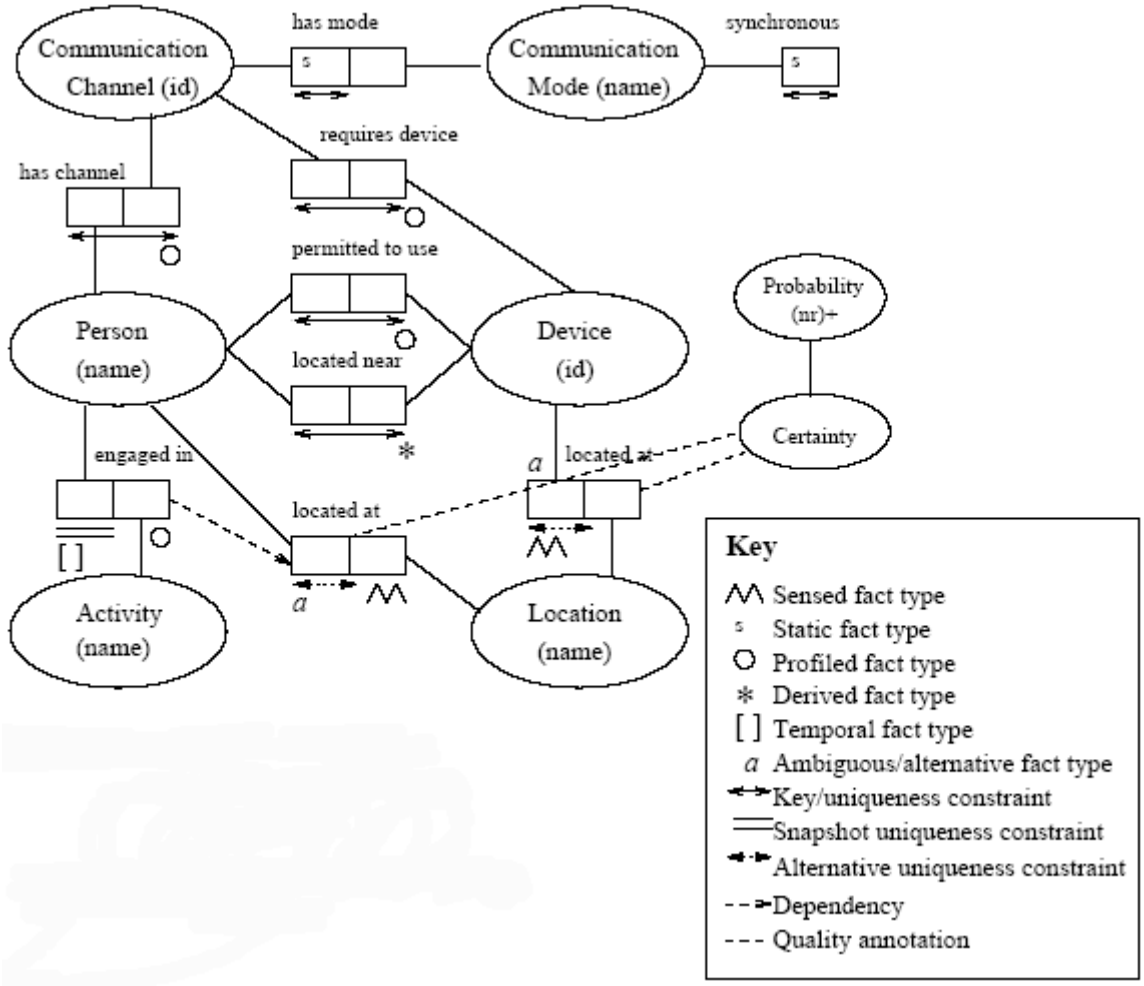

Figura 5-2 – Exemplo de modelo gráfico construído a partir de extensões à ORM

Considerando-se as particularidades das aplicações ubíquas, a utilização de uma linguagem de especificação como UML na representação de contexto não tem se mostrado apropriada. Há poucos trabalhos que se apóiam na UML para a definição do modelo de contexto e alguns manifestam inadequação desse modelo para o propósito específico de modelagem de contexto [Henricksen et al., 2002; Henricksen et al., 2003]. As dificuldades

relatadas são devidas às formulações genéricas providas pela UML, para tratar uma ampla gama de possibilidades conceituais de modelagem dos mais diferentes domínios, que lhe conferem o status de linguagem de modelagem de propósito geral. Esses problemas poderiam ser minimizados caso dispuséssemos de um metamodelo de contexto formalmente definido, a partir do quail representações válidas de contexto pudessem ser geradas e manipuladas, facilitando o mapeamento das informações contextuais descritas em um modelo específico de domínio para outro.

5.4. Modelos Baseados em Objetos

Nos modelos orientados a objetos as informações contextuais são estruturadas em hierarquias de classes, enfatizando aspectos como abstração, encapsulamento, polimorfismo e herança. Os detalhes de processamento das informações contextuais são encapsulados em objetos e o acesso a essas informações é fornecido através de interfaces.

A utilização de modelos baseados em objetos para a especificação de informações contextuais não explora a descrição semântica das mesmas e são, portanto, incapazes de implementar algoritmos que realizem inferência e interpretação de novos contextos a partir apenas da descrição destas informações. Por conseguinte, as próprias estruturas utilizadas para representação das informações contextuais implementam a lógica das aplicações, encapsulando-a em métodos, objetos e classes que implementam algumas estratégias de adaptação – tais como o disparo de ações quando um determinado contexto tornar-se disponível. Observe que os elementos responsáveis pela inferência de contextos seguem um sistema fortemente tipado e são limitados a representar as informações contextuais previamente implementadas. Essa estratégia impossibilita, portanto, a implantação de novos tipos de contexto em tempo de execução, tornando esta abordagem inadequada para o tratamento de informações contextuais de propósito geral.

O sistema JCAF (*Java Context-Awareness Framework*) [Jakob, 2005] oferece uma infra-estrutura de execução e um *framework* básico para o desenvolvimento de aplicações sensíveis ao contexto. A infra-estrutura de execução do JCAF é composta por serviços contextuais (*context services*), aos quais cabe hospedar entidades. Estas, por sua vez, são objetos Java que modelam o comportamento e armazenam o contexto de objetos do mundo real. Observadores de entidades (*entity listeners*) são aplicações Java interessadas em receber

notificações na ocorrência de determinadas alterações no estado de uma entidade. O sistema JCAF incorpora ainda os monitores de contexto (objetos responsáveis pela captura das informações contextuais), atuadores (objetos que alteram o contexto) e transformadores (os quais têm o propósito de realizar interpretações limitadas a respeito das informações contextuais manipuladas). Por fim, JCAF inclui um componente que controla e autentica o acesso aos demais componentes da arquitetura. O sistema é fortemente acoplado à biblioteca de classes de Java e utiliza extensivamente o conceito de entidades para representar objetos do mundo real. Uma vez que toda entidade do mundo real deve ter uma representação interna no sistema, o modelo proposto por JCAF incentiva a instanciação de diversos objetos, os quais frequentemente possuem funcionalidades bastante simples. A Figura 5-3 exibe um trecho de código exemplificando a implementação de uma entidade (*Place*) a partir das classes implementadas pelo *framework* do JCAF.

```
public class Place extends LocatableEntity {
  protected ContextTransformer transformer = null;
  ...
  public void contextHasChanged(ContextEvent event) {
    if (transformer == null) {
      transformer = (RFIDToEntityIDTransformer).
                     getEntityEnvironment().getTransformerRepository().
                     getContextTransformer(RFIDToEntityIDTransformer.class);
    }
    if (event.getType().equals(RFIDItem.class)) {
      ContextItem[] in = new ContextItem[1];
      in[0] = event.getItem();
      try {
        EntityIDItem id = (EntityIDItem) transformer.translate(in);
        Entity entity = getContextService().getEntity(id.getEntityId());
        getContextService().setContextItem(this.getId(), entity);
      } catch (ContextTransformerException e) {...}
    }
  }
}
```

Figura 5-3 – Trecho de código exemplificando a modelagem baseada em objetos do JCAF

A classe *Place* é declarada como subclasse de *LocatableEntity*, classe disponibilizada pelo *framework* JCAF que modela entidades que possam conter informações relacionadas à localização. O método *contextChanged()* implementa as ações a serem executadas na ocorrência de alguma alteração do contexto relacionadas a localização da entidade *Place*. Um componente transformador (*RFIDToEntityIDTransformer*) é utilizado para traduzir a

informação obtida de um sensor *RFID* para o formato *EntityID*, utilizado para identificar objetos da classe *Place*.

5.5. Modelos Baseados em Lógica

Modelos de contexto baseados em lógica possuem um alto grau de formalismo e utilizam expressões lógicas para definir as condições sob as quais conclusões ou fatos serão derivados, a partir de um conjunto de outras expressões ou fatos. Para descrever estas condições como um conjunto de regras, um sistema formal é utilizado. As informações contextuais são, portanto, definidas como fatos, expressões ou regras. Este conjunto de asserções define um modelo lógico e permite uma forma de raciocínio não dependente da interpretação subjetiva dos símbolos utilizados nas asserções.

Normalmente, a especificação dos modelos apresenta algumas hipóteses sobre o contexto no qual o sistema irá operar (leis físicas, entidades e propriedades que o sistema deve satisfazer), permitindo, assim, a escrita de asserções que representem o conhecimento de uma determinada entidade (física ou abstrata).

Exemplos importantes incluem lógica proposicional, lógica de primeira ordem, lógica temporal e lógicas de alta ordem. Gray e Salber propõem um modelo de contexto que utiliza lógica de primeira ordem como uma representação formal para as derivações e relacionamentos envolvendo informações contextuais [Gray et al., 2001].

Apesar de seu alto grau de formalismo, definido por um conjunto de regras estritamente formais, modelos baseados em lógica não vêm sendo largamente utilizados na modelagem de contexto. Dentre as principais razões para a não utilização destes modelos destacam-se a dificuldade de implementação, a complexidade na elaboração, avaliação e explicação dos modelos e a falta de padronização das linguagens lógicas.

5.6. Modelos Baseados em Ontologias

Ontologias também são utilizadas para modelagem de contexto. Uma ontologia é uma descrição formal dos conceitos e dos relacionamentos existentes em um determinado domínio [Guarino, 1998]. Basicamente, uma ontologia consiste de conceitos, relações, definições, propriedades e restrições descritas na forma de axiomas [Falbo et al., 1998].

Essa abordagem para a modelagem de contexto foca na construção de uma ontologia para um domínio específico, no qual a aplicação está inserida, com o objetivo de permitir o compartilhamento dos conceitos a respeito do domínio entre vários sistemas, uma vez que uso desses modelos ajuda a eliminar ambigüidades com relação a informações contextuais, que poderiam ter diferentes significados em sistemas distintos. A interoperabilidade entre sistemas pode ser conseguida mesmo sem o compartilhamento de um modelo comum, uma vez que equivalências sintáticas entre as entidades e conceitos representados por diferentes modelos podem ser expressas de forma a estabelecer uma terminologia comum. Por apresentarem este grau de formalismo, ontologias podem ser úteis para realizar inferências sobre um domínio, a fim de gerar novos fatos ou descobrir novos relacionamentos, aspecto importante para aplicações móveis sensíveis a contexto.

O projeto CoBrA – *Context Broker Architecture* [Chen, 2004] fornece uma arquitetura baseada em agentes para o desenvolvimento de aplicações sensíveis ao contexto em espaços inteligentes. Como visto no Capítulo 2, trata-se de uma arquitetura centralizada num componente, o *context broker*, que compartilha um modelo de contexto baseado em ontologia, provendo um conjunto de conceitos para a comunicação entre agentes distribuídos, serviços e dispositivos. Essa ontologia provê uma representação explícita do tipo de informação de contexto que o *context broker* é capaz de compartilhar e processar. A capacidade de inferência de novos contextos foi desenvolvida com base em tecnologias da Web Semântica e a linguagem OWL (*Ontology Web Language*) foi escolhida para a especificação dos modelos por ser mais expressiva que RDF (*Resource Description Framework*). Apesar de as duas linguagens consistirem no uso de triplas (classe, propriedade, valor) para representar as relações entre os diversos conceitos, RDF possui limitações claras quando à possibilidade de realização de inferências. Em contrapartida, a linguagem OWL se apresenta muito mais poderosa, utilizando a Lógica Descritiva [Baader et al., 2003] para explicitação do conhecimento. A Figura 5-4 ilustra a representação em OWL de alguns conceitos

relacionados à localização, definidos em uma das ontologias utilizadas no projeto CoBrA (*COBRA-ONT Location Ontology*).

```
<owl:Class rdf:ID="ThingHasLocationContext">
  <rdfs:label>ThingHasLocationContext</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://daml.umbc.edu/cobra/space-basic#SpatialThing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#locationContext"/>
      <owl:minCardinality rdf:datatype="#nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="locationContext">
  <rdfs:label>isLocated</rdfs:label>
  <rdfs:domain rdf:resource="#ThingHasLocationContext"/>
  <rdfs:range rdf:resource="#LocationContextDescription"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="LocationContextDescription">
  <rdfs:label>LocationContextDescription</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://daml.umbc.edu/cobra/time-basic#TemporalEvent"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="locatedIn">
  <rdfs:label>locatedIn</rdfs:label>
  <rdfs:domain rdf:resource="#LocationContextDescription"/>
  <rdfs:range rdf:resource="http://daml.umbc.edu/cobra/space-basic#SpatialThing"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="locatedNearBy">
  <rdfs:label>locatedNearBy</rdfs:label>
  <rdfs:domain rdf:resource="#LocationContextDescription"/>
  <rdfs:range rdf:resource="http://daml.umbc.edu/cobra/space-basic#SpatialThing"/>
</owl:ObjectProperty>
```

Figura 5-4 – Trecho de código da COBRA-ONT Location Ontology

Modelos baseados em ontologias são particularmente apropriados para a modelagem das informações contextuais. Estes modelos oferecem boas potencialidades na realização de inferências, na modelagem de contextos complexos e no tratamento de situações relacionadas com a qualidade, ambiguidade e inexatidão das informações. Além disso, a utilização de primitivas de modelagem mais simples do que aquelas utilizadas pelos modelos lógicos torna a elaboração, avaliação e explicação dos modelos baseados em ontologias mais intuitivas. Por último, as iniciativas de padronização de linguagens para a construção de ontologias por parte da W3C (*World Wide Web Consortium*) tendem a potencializar a integração entre diferentes plataformas e aplicações sensíveis ao contexto.

5.7. Avaliação dos Modelos Analisados

A adoção de modelos adequados à representação de domínios de aplicações sensíveis ao contexto deve ser uma preocupação no projeto de plataformas de suporte a aplicações sensíveis ao contexto, uma vez que alguns modelos não asseguram precisão à sintaxe e à semântica abstrata das informações contextuais. Além disso, a maioria dos modelos analisados possui expressividade limitada para a representação das informações contextuais de propósito gerais, sendo boas opções no suporte a aplicações de domínios específicos.

Strang e Linnho-Popien avaliam em [Stang et al., 2004] cada um desses seis modelos de contexto, tendo como base os seguintes requisitos:

- **Composição distribuída:** dada a natureza inerentemente distribuída das aplicações sensíveis ao contexto, as técnicas de modelagem devem suportar, de alguma maneira, a distribuição dos modelos de contexto;
- **Validação parcial da informação:** como conseqüência da distribuição, as técnicas de modelagem devem ainda permitir que componentes distribuídos de uma aplicação avaliem, mesmo que parcialmente, a conformidade das informações contextuais instanciadas de acordo com as especificações do modelo;
- **Riqueza e qualidade da informação:** as técnicas de modelagem devem ser capazes de modelar ricas composições e inter-relacionamentos entre informações contextuais além de permitir representar características referentes à qualidade das informações contextuais manipuladas;
- **Suporte à incompletude e ambigüidade:** as técnicas de modelagem devem ser capazes de lidar com a natureza incompleta e ambígua das informações contextuais;
- **Nível de formalismo:** as técnicas de modelagem devem permitir a representação da semântica da informação contextual em um nível mais rico do que o nível da sintaxe.
- **Aplicabilidade:** este requisito tem por objetivo analisar cada uma das técnicas de modelagem quanto à sua aplicabilidade e custo para a implementação de sistemas, em especial plataformas de serviços sensíveis ao contexto.

A Figura 5-5 apresenta os resultados da avaliação de cada modelo de contexto a partir de cada requisito analisado. Os autores concluem que apenas o modelo orientado a objetos e o modelo baseado em ontologias são suficientemente ricos para atender a todos esses requisitos.

Requisito / Abordagem	Composição distribuída	Validação parcial	Riqueza e qualidade da informação	Suporte à incompletude e ambigüidade	Nível de formalismo	Aplicabilidade
Modelos de pares Atributo-valor	-	-	--	--	--	+
Modelos gráficos	+	++	-	-	+	++
Modelos baseados em esquemas	--	-	+	-	+	+
Modelos baseados em objetos	++	+	+	+	+	+
Modelos baseados em lógica	++	-	-	-	++	-
Modelos baseados em ontologias	++	++	+	+	++	+

Figura 5-5 – Avaliação dos Modelos de Contexto Analisados

5.8. Conclusão do Capítulo

Este capítulo apresentou diferentes propostas para modelagem de contexto referenciadas na literatura com vistas à definição de um modelo adequado para a plataforma Infraware. No particular caso da plataforma proposta, deseja-se um modelo de contexto flexível e suficientemente expressivo de maneira a suportar a natureza genérica das informações contextuais. Além disso, são características desejáveis o suporte a inferência de novos contextos e o compartilhamento dos conceitos a respeito do domínio entre vários sistemas, dada a natureza distribuída da plataforma. O modelo deve ainda permitir a implantação de novas informações contextuais em tempo de execução, sem nenhum impacto sobre as aplicações clientes do *middleware*. A análise realizada sobre as abordagens para modelagem de contexto mostrou que o modelo baseado em ontologias apresenta-se como um modelo adequado e suficiente expressivo para atender a esses requisitos.

O próximo capítulo apresenta os principais aspectos relacionados ao projeto e a implementação do Interpretador de Contexto da plataforma Infraware, este componente utiliza modelos baseados em ontologias para a representação das informações contextuais e regras de produção para a inferência de novos contextos.

6. O Interpretador de Contexto

Este capítulo tem como objetivo apresentar o projeto e a implementação do componente Interpretador de Contexto da plataforma Infraware. Esse componente é o responsável pela realização de interpretação e inferências de novas informações contextuais mais elaborados a partir de informações contextuais primitivas, provenientes de diferentes fontes, com a finalidade de abstrair para as aplicações a complexidade do processo de inferência e tratamento das informações contextuais em ambientes ubíquos.

O Interpretador de Contexto da Infraware utiliza modelos baseados em ontologias para descrever os domínios das aplicações, bem como as informações contextuais. Esses modelos permitem a definição de conceitos e relações através de uma semântica bem definida e são compartilhadas por todos os módulos da plataforma, aplicações e serviços através de uma terminologia comum.

6.1. Interpretação de Contexto

O processo de interpretação de contexto pode ser descrito como o conjunto de mecanismos e procedimentos que realizam a derivação e a inferência de novas informações contextuais relevantes para o uso das aplicações, a partir de informações básicas provenientes das diferentes fontes de contexto. Esse processo tem como propósito facilitar o entendimento de uma determinada situação pelas aplicações e auxiliá-las em suas tomadas de decisões.

Na plataforma Infraware, o processo de interpretação de contexto dá-se em diferentes níveis. Inicialmente, as informações contextuais são coletadas a partir das diferentes fontes de contexto, tipicamente sensores reais com diferentes graus de complexidade e tecnologias. Essa coleta é realizada por estruturas de software hierárquicas que representam sensores virtuais, denominadas *wrapers,* instanciadas para cada tipo de sensor. Os *warapers*, em função da sua percepção, podem combinar diferentes pré-processamentos sobre os dados coletados dos sensores, aplicando diferentes filtros de compensação e redução de ruído, até à extração das informações contextuais, que são posteriormente normalizadas de acordo com o modelo de contexto especificado pela plataforma. A inferência de novas informações contextuais e a percepção de contexto, conforme especificado pelas aplicações, por sua vez,

são consideradas funções de nível mais alto desempenhadas pelo componente Interpretador de Contexto.

Em sua primeira versão [Calvi et al., 2005], o Interpretador de Contexto da plataforma Infraware utilizava grafos acíclicos dirigidos como estrutura de *software* para manipulação, interpretação e inferência de contextos e adotava o modelo gráfico de contexto proposto por [Henricksen et al., 2002] para a representação das informações contextuais. Essa abordagem mostrou-se incompatível com a estratégica de desenvolvimento atual do projeto *Infraware*, uma vez que o modelo informal de contexto adotado carecia de expressividade semântica para interoperabilidade com os demais módulos da plataforma.

Em contrapartida, como discutido no Capítulo 5, modelos baseados em ontologias mostraram-se uma alternativa adequada no projeto e desenvolvimento da plataforma *Infraware*, uma vez que sua notação formal permite uma especificação do domínio de forma não ambígua, possibilitando a realização de inferências consistentes, além de garantir uma representação compartilhada e reutilizável das informações entre os demais módulos da plataforma.

6.2. Inferências e Ontologias

Em pesquisas no campo da Representação do Conhecimento e do Raciocínio Baseado em Regras, normalmente são definidas linguagens formais para a representação do conhecimento e métodos de inferência associados a essas linguagens. Isso permite a construção de sistemas capazes de derivar informações implícitas através da análise do conhecimento representado explicitamente.

Modelos baseados em ontologias fornecem caracterização lógica para a interpretação de objetos, classes e relações, possibilitando, assim, processos de inferência semanticamente efetivos. A Figura 6-1, extraída de [Timm 2005], apresenta um exemplo simples de ontologia descrevendo alguns relacionamentos entre os conceitos: forma, círculo e quadrado.

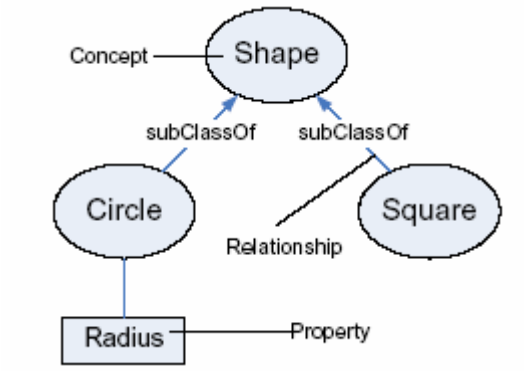

Figura 6-1 – Um exemplo simples de ontologia

Um tipo básico de relação suportado pelas ontologias é a relação hierárquica é-um (*IS-A*). Com essa relação, é possível realizar a análise das relações de especialização e generalização de entidades em diferentes níveis. No exemplo apresentado pela Figura 6-1, esta relação é descrita pela relação *subClassOf* .

Ontologias também podem definir relações não hierárquicas. Por exemplo, a relação "tem-interesse-em" pode ser definida entre os conceitos "pessoa" e "interesse", sem que se trate de um relacionamento hierárquico.

Além de definir relações, as ontologias também permitem a definição de restrições Por exemplo, em uma ontologia sobre pessoas, pode-se construir uma restrição sobre o conceito "pessoa" baseada na relação "possui-nome", ou seja: "uma pessoa possui exatamente um nome". Desta maneira, foi definida uma restrição sobre o conceito pessoa.

Modelos baseados em ontologias também permitem a inferência de novas informações por meio de regras de inferência. Por exemplo, considere uma ontologia em que o termo "parente" defina um relacionamento mais geral do que "irmão". Se Pedro é irmão de José, o sistema é capaz de inferir que Pedro e José são parentes sem que esse fato tenha sido explicitamente declarado.

6.3. Linguagem para Representação de Ontologias

À primeira vista, qualquer linguagem de representação formal do conhecimento, ou mesmo informal, poderia ser usada para representar ontologias [Falbo, 1998]. Entretanto, para a realização de inferências torna-se necessária a utilização de uma linguagem formal, ou seja, uma linguagem dotada de símbolos não ambíguos e formulações exatas.

Recentemente, a W3C (*World-Wide Web Consortium*) tem se esforçado na evolução e padronização de novas linguagens para definição de páginas e "recursos" da *Web* baseadas em conhecimento estruturado na forma de ontologias. Dentre as linguagens de representação criadas, destacam-se linguagens como RDF (*Resource Description Framework*) [RDF, 2004], RDF-S (*Resource Description Framework-Schema*) [RDFS, 2004], DAML+OIL (*DARPA Agent Markup Language + Ontology Inference Layer*) [DAML+OIL, 2001] e OWL (*Web Ontology Language*) [OWL, 2004]. Todas estas linguagens têm por base a idéia de declarações na forma de triplas. Por exemplo, uma declaração sobre uma pessoa poderia ser na forma: (João, ama, Maria), onde "João" e "Maria" são instâncias do conceito "pessoa" e "ama" é uma relação declarada entre estas instâncias.

A linguagem OWL foi escolhida para a especificação do modelo de contexto da plataforma Infraware por permitir a troca e o reuso da definição de conceitos e relações através de uma semântica bem definida e suficientemente expressiva, além de ser, a linguagem recomendada atualmente pela W3C para a representação de ontologias e conteúdos da *web* semântica.

A OWL é subdividida em três linguagens que provêem níveis incrementais de expressividade:

- **OWL Lite** - oferece suporte para que classificações hierárquicas com restrições simples. Por exemplo, nas restrições de cardinalidade são permitidas para apenas valores iguais a zero ou um. A OWL Lite é a mais simples a ser implementada e pode ser uma boa alternativa para a construção de taxionomias e *tesauros*.
- **OWL DL** – provê maior grau de expressividade do que OWL Lite e garante que todas as conclusões são computáveis (completude) e todas as computações terminam em tempo finito (decidíveis). Em OWL DL, classes podem ser definidas por operações de união, interseção, complemento e disjunção. Entretanto, a separação de tipos é exigida. Por exemplo, uma classe não pode ser um indivíduo ou uma propriedade ao mesmo tempo. A sigla DL possui correspondência com a Lógica Descritiva (D*escription Logics*) [Baader et al., 2003].
- **OWL Full** – provê máxima expressividade e liberdade sintática. Com OWL Full, os usuários podem aumentar o vocabulário pré-definido, porém não há garantia de computabilidade e decidibilidade. Assim, máquinas de inferência poderão não derivar conclusões ou poderão não ser computáveis em tempo finito. Aqui, não cabem as

restrições de separação de tipos da OWL DL e é possível manipular e modificar metaclasses.

Para a especificação dos modelos de contexto da plataforma Infraware, escolheu-se OWL DL pela necessidade de restrições de cardinalidade diferentes das suportadas por OWL Lite e pela garantia de computabilidade e decibilidade – não oferecidas pela OWL Full.

6.4. Modelagem de Contexto Com Ontologias

A utilização de ontologias para a descrição do conhecimento de um domínio envolve alguns princípios básicos como a especificação das classes presentes no domínio, o agrupamento destas classes em hierarquias, a identificação das propriedades e relacionamentos de forma a refletir a realidade do domínio descrito. As instâncias definem o conhecimento factual e uma configuração de instâncias representa os indivíduos e deve respeitar os conceitos e axiomas expressos na ontologia. Uma vez descrito, o conhecimento de um domínio pode ser manipulado através de regras lógicas para a inferência de novos conhecimentos não explicitamente especificados, mas que derivam do conhecimento inicial.

A Figura 6-2 ilustra uma representação simplificada de um domínio envolvendo o relacionamento entre os funcionários de um hospital e pacientes, descrito em OWL.

```
<owl:Class rdf:ID="Place">
    <rdfs:label>Place</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Department">
    <rdfs:label>Department</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Place"/>
</owl:Class>

<owl:Class rdf:ID="Hospital">
    <rdfs:label>Company</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Place"/>
</owl:Class>

<owl:TransitiveProperty rdf:ID="isPartOf">
    <owl:inverseOf rdf:resource="#hasPart" />
    <rdfs:domain rdf:resource="#Department"/>
    <rdfs:range  rdf:resource="#Hospital"/>
</owl:TransitiveProperty>

<owl:Class rdf:ID="Person">
    <rdfs:label>Person</rdfs:label>
    <rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#hasChild"/>
        <owl:allValuesFrom rdf:resource="#Person"/>
    </owl:Restriction>
```

```
</owl:Class>

<owl:FunctionalProperty rdf:ID="isInside">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range  rdf:resource="#Department"/>
</owl:FunctionalProperty>

<owl:ObjectProperty rdf:about="isLocatedIn">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Place"/>
</owl:ObjectProperty>

<owl:SymmetricProperty rdf:ID="isFriendOf">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>

<owl:TransitiveProperty rdf:ID="employerOf">
    <owl:inverseOf rdf:resource="#employedBy"/>
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range  rdf:resource="#Hospital"/>
</owl:TransitiveProperty>

<owl:TransitiveProperty rdf:ID="employedBy">
    <owl:inverseOf rdf:resource="#employerOf"/>
    <rdfs:domain rdf:resource="#Hospital"/>
    <rdfs:range  rdf:resource="#Person"/>
</owl:TransitiveProperty>

<owl:DatatypeProperty rdf:ID="email">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:Class rdf:about="Emergency">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasEmergency"/>
        <owl:allValuesFrom>
          <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
              <owl:Thing rdf:about="#tachycardia"/>
              <owl:Thing rdf:about="#bradycardia"/>
              <owl:Thing rdf:about="#arrhythmia"/>
            </owl:oneOf>
          </owl:Class>
        </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="Patient">
  <rdfs:label>Paciente</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasEmergency">
  <rdfs:domain rdf:resource="#Patient" />
  <rdfs:range rdf:resource="#Emergency" />
</owl:ObjectProperty>
```

Figura 6-2 - Ontologia de Domínio descrevendo as relações funcionários de um hospital e pacientes

No exemplo apresentado pela Figura 6-2, a sintaxe *rdf:ID="Place"* é usada para nomear a classe *Place*, que representa um lugar. Segundo [Smith et al., 2004], esta classe,

dentro do documento em que foi definida, pode ser referenciada usando-se a expressão *#Place*. Outras ontologias podem referenciar esta classe usando a forma completa *"http://www.lprm.inf.ufes.br/infraware#Place"*.

Hierarquias de classes podem ser criadas usando-se a construção *rdfs:subClasseOf*. No exemplo da Figura 6-2, as classes *Department* e *Hospital* são definidas como subclasses de *Place*. Portanto, o conjunto de indivíduos das classes *Department* e *Hospital* forma, respectivamente, um subconjunto do conjunto de indivíduos da classe *Place*.

As propriedades são relações binárias usadas para estabelecer relacionamentos entre indivíduos ou entre indivíduos e valores de dados. Estes relacionamentos permitem afirmar fatos gerais sobre os membros das classes e podem também especificar fatos sobre indivíduos [Smith et al., 2004]. A OWL define duas categorias principais de propriedades:

- Propriedades de dados tipados (*datatype properties*): relação entre indivíduos e tipos de dados.
- Propriedades de objetos (*object properties*): relação entre indivíduos.

Uma propriedade de objeto é definida como instância da classe *owl:ObjectProperty*. Uma propriedade de dado tipado é definida como uma instância da classe *owl:DatatypeProperty*. Tanto *owl:ObjectProperty* quanto *owl:DatatypeProperty* são subclasses da classe RDF *rdf:Property* e herdam as propriedades: *rdfs:subPropertyOf*, *rdfs:domain* e *rdfs:range*.

No modelo apresentado pela Figura 6-2, a propriedade *isPartOf* tem como domínio (*domain*) a classe *Department*, ou seja, esta propriedade deve ser usada somente por indivíduos que são instâncias da classe *Department*. Sua imagem (*range*) é a classe *Hospital*, ou seja, os valores que esta propriedade pode assumir devem ser instâncias da classe *Hospital*. A propriedade *email* relaciona *Person* a uma *string*, ou seja, uma pessoa tem um endereço de e-mail. Os possíveis valores assumidos por esta propriedade são definidas pelo recurso *rdf:resource="&xsd;string"*, um dado tipado definido na XML *Schema*. Outras referência para este e outros dados tipados podem ser encontrados em [Smith et al., 2004].

As propriedades podem ser definidas como simétricas. Por exemplo, a propriedade *isFriendOf* é declarada como simétrica. Portanto de uma pessoa A é dita amiga de uma pessoa B, então se pode deduzir que a pessoa B também é amiga da pessoa A.

Uma propriedade funcional é aquela que possui um único valor de *y* para cada instância de *x*. Portanto, não tem mais do que um único valor para cada indivíduo. Uma propriedade com esta característica é dita ter cardinalidade mínima 0 e cardinalidade máxima 1. No exemplo da Figura 6-2, a propriedade *isInside* é declarada como funcional, esta definição não permite que instâncias da classe *Person* estejam lotadas em mais de um departamento de um hospital ao mesmo tempo. Uma propriedade pode ser também o inverso de uma outra propriedade. Por exemplo, a propriedade *employerOf* é o inverso da propriedade *employedBy* e representa a relação entre os empregados e o hospital.

A OWL permite ainda que restrições sejam impostas sobre propriedades. Uma restrição é um tipo especial de descrição de classe, ou seja, descreve uma classe anônima de indivíduos que satisfazem as restrições. As restrições podem ser de valores (*allValuesFrom*, *someValuesFrom* e *hasValue*) ou de cardinalidade (*maxCardinality*, *minCardinality* e *Cardinality*). De acordo com o modelo da Figura 6-2, todas as instâncias da classe *Person* que usam a propriedade *hasChild* devem ter como valores associados a esta propriedade instâncias da própria classe *Person*. Esta restrição é análoga ao quantificador universal da lógica de predicados. Isto é, não é requerido que uma pessoa tenha filhos, mas se tiver, estes devem ser da classe *Person*.

6.5. Arquitetura Conceitual do Interpretador de Contexto

O Interpretador de Contexto da plataforma Infraware é composto por cinco módulos que, juntos, são capazes de realizar o processo de interpretação de contexto e de informações contextuais. A Figura 6-3 ilustra os principais módulos do Interpretador de Contexto da plataforma Infraware.

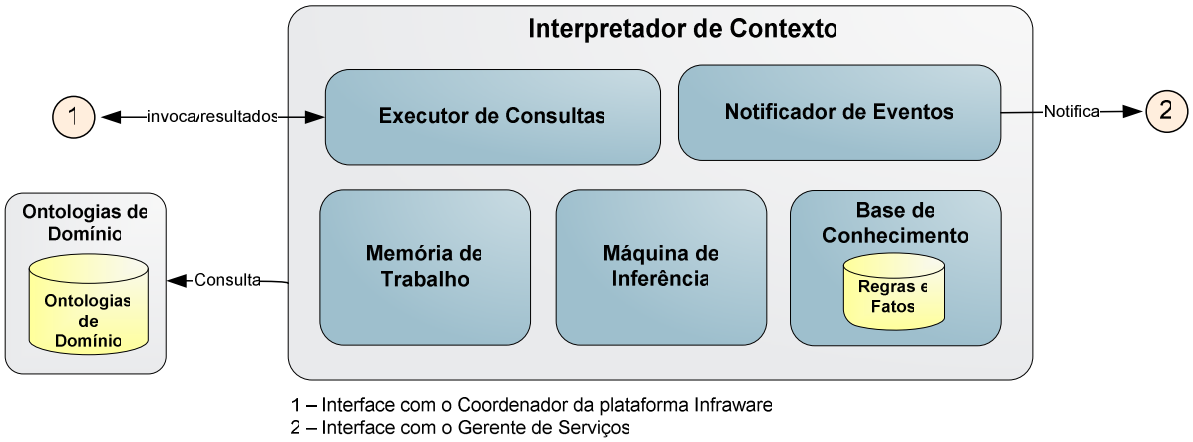

Figura 6-3 – Projeto Conceitual do Interpretador de Contexto

As *Ontologias de Domínio* são compartilhadas entre os componentes da Infraware e descrevem conceitos gerais relacionados a um domínio de aplicação, seus relacionamentos e axiomas. O armazenamento das ontologias de domínio da plataforma Infraware em um repositório acessível a todos os módulos da plataforma promove a organização do conhecimento de forma compartilhada além de facilitar o seu reuso. Além disso, o isolamento destas especificações acerca do domínio das aplicações permite que estas ontologias sejam extendias ou redefinidas para diferentes domínios, uma vez que o modelo deve ter associação com a realidade, mais do que com aspectos técnicos relacionados à implementação.

A *Base de Conhecimento* armazena o conhecimento instanciado a partir das Ontologias de Domínio, ou seja, ela é a representação de um conhecimento específico. Esse conhecimento pode ser dividido em duas partes:

- **Conhecimento Intensional** (*Intensional Knowledge*): conhecimento geral sobre o domínio do problema. Representa o conhecimento sobre as classes de indivíduos com as mesmas características gerais.
- **Conhecimento Extensional** (*Extensional Knowledge*): conhecimento sobre cada indivíduo que faz parte de uma classe.

Esse conhecimento é especificado através de um conjunto de regras *if-then* e fatos conhecidos. Uma regra estabelece um relacionamento entre clausulas (asserções ou fatos) e, dependendo da situação, pode ser usado para gerar uma nova informação ou para executar o disparo de alguma ação. Eis um exemplo de uma sentença lógica transformada em regra de produção:

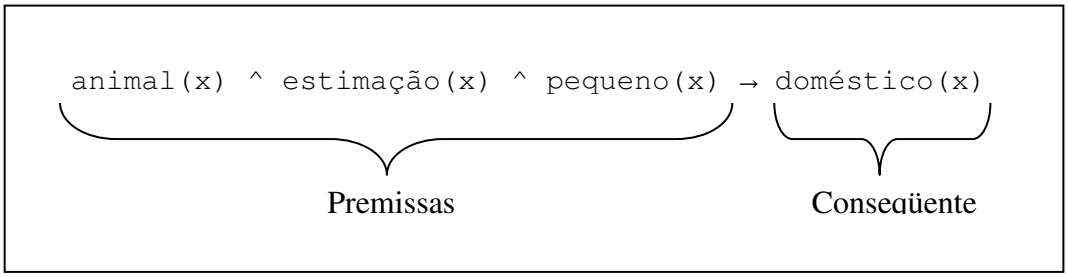

Figura 6-4 – Exemplo de sentença lógica transformada em regra

A regra ilustrada pela Figura 6-4 pode ser entendida como: "Para todo x (quantificador universal implícito), se existe um fato afirmando que x é um animal e outro fato afirmando que x é de estimação e outro fato afirmando que x é pequeno, então, isto implica a validade de um novo fato, o de que x é doméstico". Estas regras são denominadas regras de produção justamente por produzirem novos fatos.

No conseqüente da regra, tanto pode haver a criação de novos fatos na base de conhecimento (regras de produção) quanto à execução de comandos e procedimentos. Nesse caso, a regra definida é do tipo *event-condition-action* (ECA) e o evento caracterizado pelas premissas determina a execução de um método ou a invocação de um serviço.

A *Memória de Trabalho* armazena os fatos que são concluídos em uma sessão de execução. A *Máquina de Inferência* realiza o processo que combina os fatos da Memória de Trabalho com a Base de Regras e Fatos para concluir novos fatos ou identificar contextos específicos. O módulo *Executor de Consultas* permite que o estado atual dos fatos interpretados seja consultado pelos demais módulos da plataforma. O módulo *Notificador de Eventos* é o componente responsável por notificar o ***Gerente de Serviços*** da plataforma Infraware sobre a ocorrência de um evento ou contexto monitorado.

6.6. Implementação

Em sua implementação atual, o ***Interpretador de Contexto*** da plataforma Infraware utiliza a API *Jena Semantic Web Framework* [McBride, 2001] para manipulação e inferência de informações contextuais. Jena é um *framework* escrito em Java, desenvolvido pelo *HP Labs Semantic Web Programme* com o objetivo de permitir a construção de aplicações com suporte à Web Semântica. Além do suporte as linguagens RDF, DAML+OIL e OWL, a API

Jena também oferece componentes para a construção de máquinas de inferência baseadas em regras.

As Ontologias de Domínio, expressas em OWL, definem uma descrição de conceitos do domínio das aplicações. Os pedidos de subscrição das aplicações são traduzidos pela definição de regras de inferência, escritas na linguagem GRL (*Generic Rule Language*), especificada pela API Jena. Essas regras expressam contextos possíveis e são avaliadas pelo ***Interpretador de Contexto***, que utiliza as informações enviadas pelos provedores de contexto e as informações semânticas capturadas da Ontologia de Domínio para disparar ações específicas na ocorrência de um determinado contexto. Através da linguagem declarativa RDQL (*Resource Description Query Language*) [Seaborne, 2004], as aplicações podem ainda realizar consultas semânticas sobre as informações contextuais com suporte à inferência e validação sintática.

A Figura 6-5 exibe o diagrama de componentes, em notação UML, do interpretador de contexto implementado.

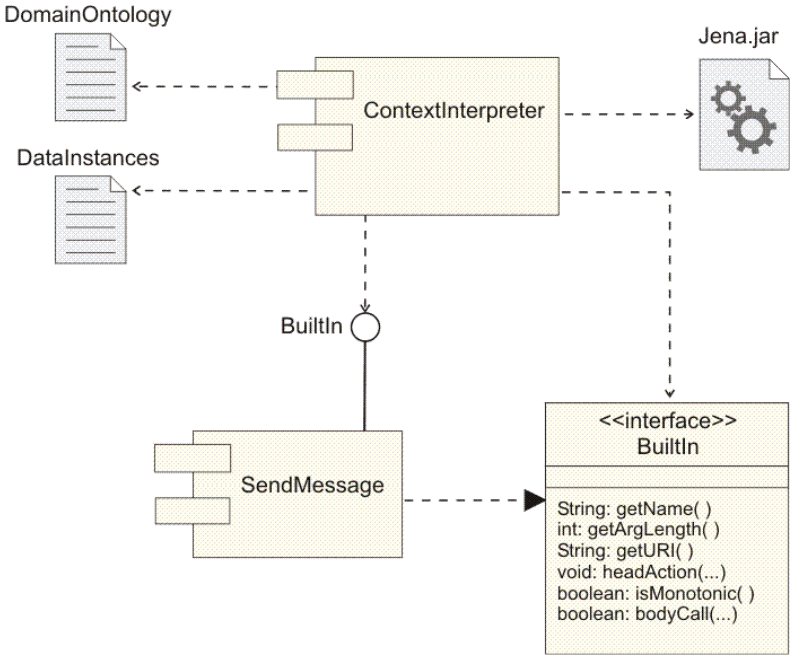

Figura 6-5 – Diagrama de componentes do Interpretador de Contexto

Como pode ser observado na Figura 6-5, o Interpretador de Contexto depende de dois arquivos de dados. Um destes arquivos contém as ontologias de domínio (*DomainOntology*), expressas em OWL, que são compartilhadas entre os demais componentes da Infraware e definem os termos usados para descrever e representar um domínio de aplicação, definindo, assim, um vocabulário, uma estrutura e possíveis restrições dos metadados que descrevem as

informações contextuais e demais entidades relacionadas. O segundo arquivo (*DataInstances*) armazena o conhecimento instanciado a partir das Ontologias de Domínio, ou seja, é a representação das entidades, indivíduos e informações do mundo real. É importante ressaltar que, em tempo de execução, as ontologias de domínio podem ser alteradas com a inclusão de novas classes ou propriedades. O processo de interpretação de contexto se dá a partir da alteração dos valores referentes às propriedades definidas para cada indivíduo (instância) de modo a refletir o estado real (capturado pelos provedores de contexto).

O componente *SendMessage*, que dispara a execução de um serviço de notificação, implementa (realiza) a interface *BuiltIn*. Esta interface define um conjunto de métodos que devem ser implementados pelas classes que definem operadores que realizam chamadas a procedimentos primitivos (*builtins*). Portanto, além dos operadores já disponibilizados pela API Jena (Tabela 5-1), as regras de inferência podem conter também operadores personalizados, como o implementado pelo componente *SendMessage*. Pode-se dizer então que o componente *SendMessage* exporta a interface *BuiltIn*. Por outro lado, o componente *ContextInterpreter* depende da mesma interface para importar novos operadores primitivos (*builtIn operators*).

6.6.1. Máquina de Inferência

O suporte à inferência oferecida pela ferramenta Jena permite que diferentes máquinas de inferência sejam integrados às bases de dados e ontologias para a criação dos modelos de inferência, implementado pela classe *InfModel*. Quaisquer consultas que sejam feitas ao modelo, após este ter sido integrado a uma máquina de inferência, trarão como resultados, além dos dados originais, todas as assertivas derivadas pela máquina de inferência através das regras e das propriedades definidas na ontologia.

Atualmente, a API Jena oferece as seguintes máquinas de inferência:

- *Transitive Reasoner* – oferece suporte apenas as propriedades transitivas e simétricas de *rdfs:subPropertyOf* e de *rdfs:subClassOf;*
- RDFS *Rule Reasoner* – implementa um subconjunto configurável das funcionalidades descritas pelo RDFS;

- OWL, OWL Mini, OWL Micro *Reasoners* – implementam de maneira ainda incompleta as funcionalidades descritas pela OWL Lite;
- DAML *Micro Reasoner* – implementa capacidades mínimas de inferência para DAML e é utilizada internamente para a habilitação da API legada DAML;
- *Generic Rule Reasoner* – máquina de inferência baseada em regras com suporte a encadeamento progressivo (*forward chaining*), encadeamento regressivo (*backward chaining*) e estratégias de execução híbridas.

O *Generic Rule Reasoner* (GRR) foi escolhido para a implementação da máquina de inferência do Interpretador de Contexto por ser o único com suporte a criação de regras que seguem uma sintaxe própria. Além disso, é possível a importação de todas as funções das outras máquinas de inferência, o que torna o GRR a mais poderosa máquina de inferência disponibilizada pela ferramenta Jena..

Como mencionado anteriormente, o GRR suporta três tipos de mecanismos de ativação de regras: *(i)* encadeamento progressivo (*Forward Chaining*), *(ii)* encadeamento regressivo (*Backward Chaining*) e *(iii)* um modo híbrido.

No encadeamento progressivo, a parte esquerda da regra é comparada com a descrição da situação atual, contida na memória de trabalho. As regras que satisfazem a esta descrição têm sua parte direita executada, o que, em geral, significa a introdução de novos fatos na memória de trabalho ou execução de alguma ação através da invocação de serviços (*builtin procedures*). Cada vez que uma regra causa a introdução de novos fatos, outras regras podem ser recursivamente ativadas. Estas regras também podem ser ativadas cada vez que são criadas, alteradas ou removidas triplas do modelo de instâncias. A inferência acaba quando nenhum encadeamento de regras for mais ativado. O algoritmo usado no mecanismo encadeamento progressivo é o RETE [Forgy, 1982]. Este algoritmo trabalha incrementalmente e é considerado um algoritmo rápido, apesar de seu grande consumo de espaço em memória.

No encadeamento regressivo é utilizado um mecanismo dirigido por metas. O sistema faz o caminho inverso, partindo de uma suposta solução do problema (meta) e tenta verificar a veracidade da suposição através de suas condições, que passam a ser então submetas a serem também provadas. Este processo ocorre de forma sucessiva até que um conjunto de condições verificáveis seja encontrado.

A máquina de inferência do Interpretador de Contexto foi configurada para utilizar o modo híbrido de ativação de regras suportado pela API Jena. Nesse modo, sempre que o mecanismo de encadeamento progressivo é executado, é armazenado um conjunto de premissas na área de deduções. Todas as regras que porventura criem novas regras para trás, vão instanciar-las de acordo com as variáveis armazenadas na área de deduções e são, posteriormente, repassadas as regras instanciadas para o mecanismo regressivo. Todas as consultas são resolvidas posteriormente pelo mecanismo regressivo, que utiliza a mistura dos dados brutos e das deduções obtidas a partir do mecanismo progressivo. Esta abordagem híbrida permite ao Interpretador de Contexto derivar novos fatos e disparar serviços a partir do encadeamento progressivo além da realização de consultas a partir do encadeamento regressivo.

6.6.2. Generic Rule Language

Para a definição das regras de inferência, o *Generic Rule Reasoner* define a linguagem *Generic Rule Language* (GRL). Como mencionado anteriormente, essas regras podem derivar novos fatos (contextos possíveis) ou disparar ações na ocorrência de eventos ou condições e são avaliadas pelo Interpretador de Contexto.

Uma regra é definida como uma instância da classe *Rule*, que por sua vez, contém uma lista de premissas e conclusões sobre essas premissas. Opcionalmente, uma regra possui um nome e um sentido (progressivo ou regressivo). Uma premissa ou uma conclusão pode ser apenas uma tripla, uma tripla estendida ou uma chamada a procedimento primitivo (*builtin operator*). A Figura 6-6, extraída de [Reynolds, 2005], apresenta uma notação informal e simplificada para a linguagem GRL.

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

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

hterm     :=   term
           or   [ bare-rule ]

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

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

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

Figura 6-6 – Notação informal e simplificada da Generic Rule Language

A Figura 6-7 descreve uma regra de produção capaz de introduzir novos fatos na memória de trabalho do Interpretador. Neste exemplo, sempre que uma pessoa estiver localizada em algum dos departamentos do hospital em que trabalha, então um novo fato é produzido indicando que a atividade corrente daquela pessoa é *'working'*.

```
[Regra01: (?p rdf:type Person)
          (?p isInside ?d)
          (?d rdf:type Department)
          (?d isPartOf ?o)
          (?o rdf:type Hospital)
          (?p employerOf ?o)
  -> (?p currentActivity 'working')  ]
```

Figura 6-7 – Exemplo de regra de produção para a introdução de novos fatos.

A Figura 6-8 ilustra uma regra descrevendo o pedido de subscrição realizado por uma aplicação. Neste pedido, sempre que uma pessoa estiver localizada no mesmo lugar que algum de seus amigos, uma mensagem é enviada informando sobre o ocorrido. Neste caso, o módulo Gerente de Serviços irá disparar serviços de envio de mensagens de texto as pessoas envolvidas.

```
[Regra01: (?p1 rdf:type Person)
          (?p1 isFriendOf ?p2)
          (?p2 rdf:type Person)
          (?p1 isInside ?l)
          (?p2 isInside ?l)
  -> SendMessage(?p1,'your friend' + ?p2 + 'is also in this place.') ]
```

Figura 6-8 – Exemplo de regra de produção para a invocação de um serviço.

6.6.3. RDF Data Query Language

Além da inferência de novas informações contextuais e disparo de ações na ocorrência de eventos, o Interpretador de Contexto permite que aplicações consultem o estado corrente das informações interpretadas. Essas consultas são expressas em RDQL, uma linguagem de consulta declarativa para RDF suportada pelos modelos de inferência do Jena e capaz de extrair informações destes modelos de acordo com uma determinada instrução de consulta.

Apesar de executar consultas sobre modelos RDF, a RDQL pode ser usada em modelos descritos em OWL, uma vez que estes modelos são descritos sobre RDF. A sintaxe RDQL é bem parecida com o SQL (*Structured Query Language*), aplicando-se praticamente o mesmo padrão com algumas características particulares. As condições são escritas como triplas *(sujeito, propriedade, valor)* e as variáveis são representadas com o ponto de interrogação "?" seguido pela sua designação. Os recursos (*resources*), por sua vez, são delimitados por "<>". As seguintes cláusulas são definidas em RDQL:

- **SELECT** – cláusula que mostra o resultado da consulta. Nela são escolhidas variáveis que retornarão os dados da consulta
- **FROM** – cláusula que especifica o modelo como um URI (*Uniform Resource Identifier*)
- **WHERE** – especifica o grafo padrão com a lista de triplas. Por exemplo, pode-se perguntar quais valores satisfazem a tripla para um dado sujeito e propriedade.
- **AND** – especifica uma ou várias expressões booleanas.
- **USING** – cláusula que possibilita a utilização de um prefixo para representar um *namespace* através de uma *string*.

A Figura 6-9 apresenta um exemplo de consulta simples, cujo resultado retorna todos os recursos representados pela URI que contenham a propriedade “idade” maior que 18 anos.

```
SELECT ?recurso
WHERE (?recurso, <xs:idade>, ?idade)
AND ?idade >= 18
USING xs FOR <http://www.lprm.inf.ufes.br/Pessoa#>
```

Figura 6-9 – Exemplo de consulta em RDQL

6.6.4. Interface de Programação

O Interpretador de Contexto provê uma API (*Application Program Interface*) que representa as interfaces fornecidas e requeridas para sua utilização. Sob a ótica do desenvolvimento de software baseado em componentes, uma interface corresponde a “uma coleção de pontos de acesso a serviços, cada um com uma semântica estabelecida” [Szyperski, 1998].

A API provida pelo Interpretador de Contexto disponibiliza serviços relativos à utilização do componente como a inclusão ou alteração de fatos na memória de trabalho, registro de procedimentos primitivos (*builtins*), inclusão e remoção de regras e execução de consultas sobre os fatos interpretados. O componente foi implementado sobre a plataforma Java J2SE versão 1.5 e utiliza internamente a API Jena em sua versão 2.4.

A Figura 6-10 apresenta o diagrama de classes utilizado pelo componente Interpretador de Contexto. A classe abstrata *Model* oferece suporte à criação de modelos que constituem documentos carregados em memória. Estes modelos podem ser tanto modelos que descrevam o domínio (*DomainModel*) quanto modelos de dados instanciados (*DataModel*). A classe *Rule* representa as regras de inferências armazenadas internamente ao Interpretador. Estas regras são compostas por um nome (atributo *name*), por um cabeçalho (atributo *head*) e por um corpo (atributo *body*). A classe *ContextInterpreter* representa o Interpretador de Contexto em si, e possui como atributos o *GenericRuleReasoner* (importado da API Jena), responsável pelas inferências; um modelo de inferência (definido pela classe *InfModel*, também importado da API Jena) que implementa a memória de trabalho e a base de conhecimento; e uma string contendo o URI para a ontologia de domínio.

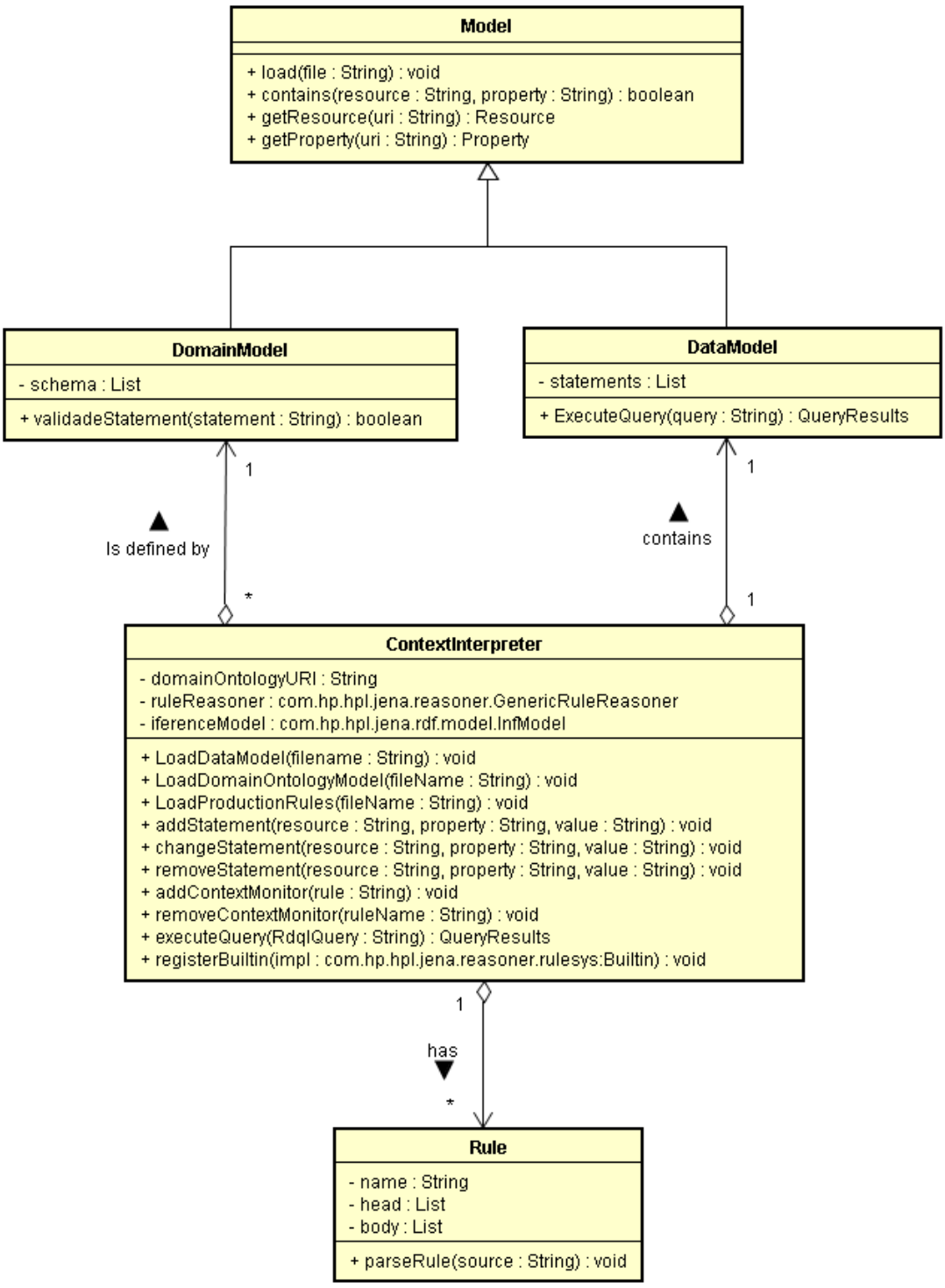

Figura 6-10 – Diagrama de classes utilizadas pelo Interpretador de Contexto

A classe *ContextInterpreter* implementa ainda os principais métodos relativos à utilização do Interpretador de Contexto:

- **LoadDomainOntologyModel** – carrega o modelo do domínio a partir de um arquivo owl contendo as definições da ontologia;

- **LoadDataModel** – carrega o modelo de dados a partir de um arquivo owl contendo as instâncias;
- **LoadProductionRules** – carrega as regras de produção a partir de um arquivo texto contendo as regras expressas em GRL. Estas regras permitem a derivação de novas informações;
- **addStatement, changeStatement e removeStatement** – permitem a inclusão, alteração e remoção de triplas no formato: (recurso, propriedade, valor);
- **addContextMonitor e removeContextMonitor** – permitem a inclusão e remoção de regras expressas em GRL do tipo *event-condition-action.* Estas regras permitem a execução de ações na ocorrência de contextos específicos;
- **registerBuiltin** – permite o registro de novos procedimentos primitivos (*builtins*, utilizado no disparo das ações;
- **executeQuery** – permite a realização de consultas expressas em RDQL sobre o estado atual das informações interpretadas.

O código-fonte da interface de programação do Interpretador de Contexto pode ser encontrado no Apêndice A.

6.6.5. Interface de Depuração

Com a evolução dos ambientes de programação, os programadores têm se tornado cada vez mais exigentes em relação aos produtos utilizados. Atualmente, uma das ferramentas essenciais no desenvolvimento de sistemas é o depurador, que permite aos desenvolvedores acompanhar passo a passo o que está ocorrendo dentro do sistema.

A interface de depuração do Interpretador de Contexto implementa uma interface gráfica para acompanhamento da execução das inferências realizadas pelo interpretador, permitindo a identificação de erros referentes à definição tento dos modelos de contexto quanto das regras de inferência.

A Figura 6-11 apresenta a interface de depuração do Interpretador de Contexto da plataforma Infraware sendo utilizada para a carga do arquivo contendo as Ontologias de Domínio.

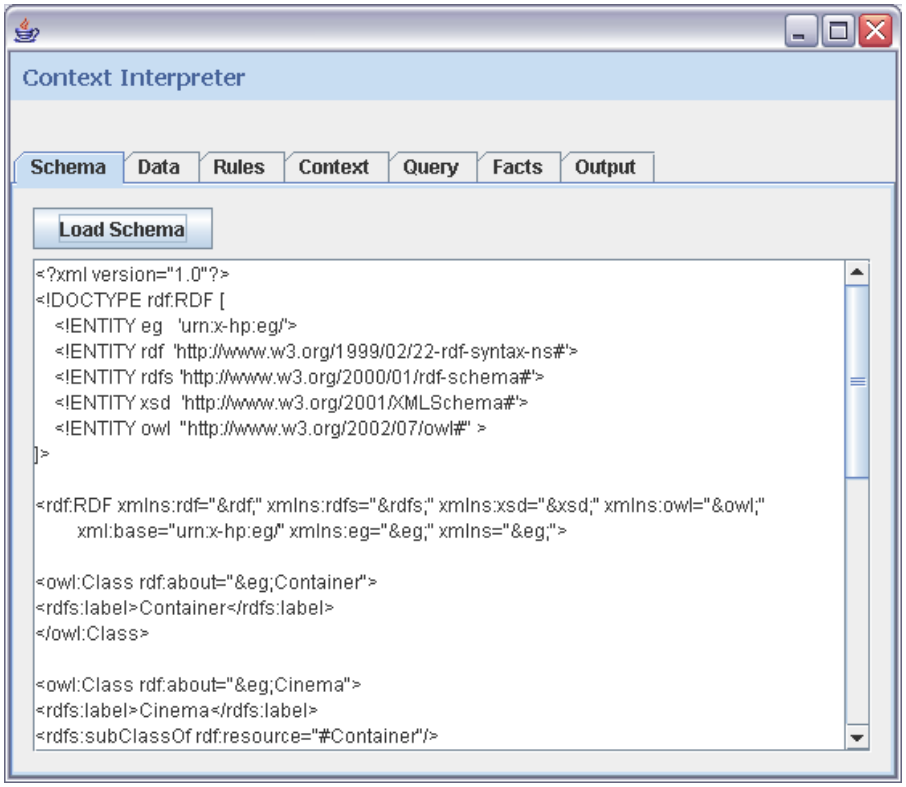

Figura 6-11 – Carga das Ontologias de Domínio

A Figura 6-12 apresenta a interface de depuração sendo usada para a carga do arquivo contendo os indivíduos instanciados:

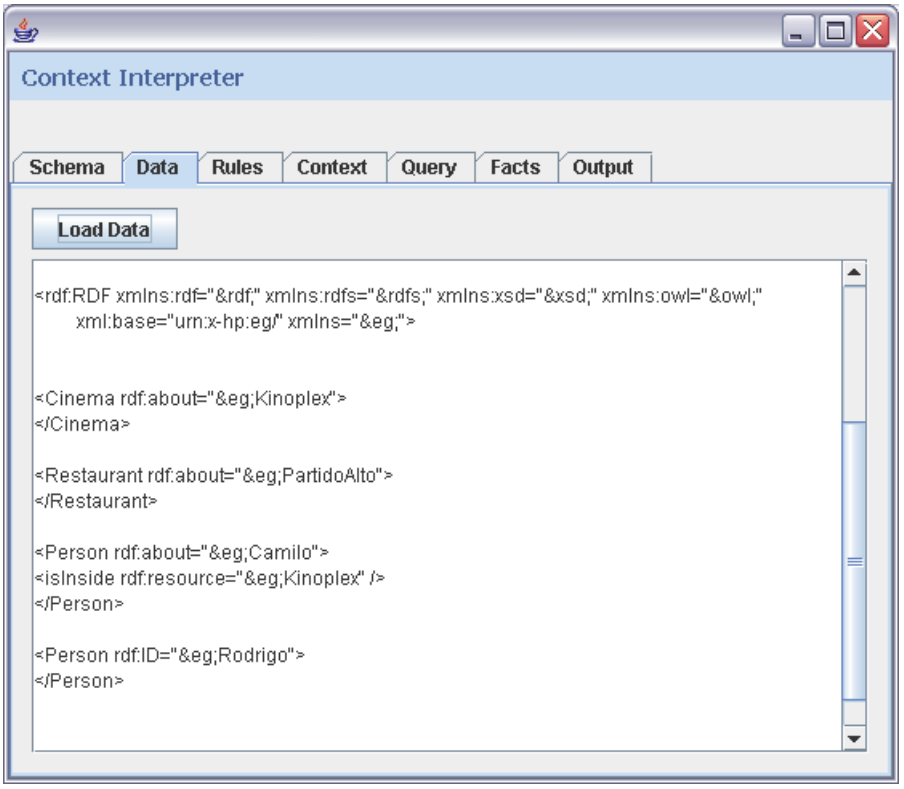

Figura 6-12 – Carga dos indivíduos instanciados

A Figura 6-13 apresenta a interface de depuração sendo utilizada para consultar todos os fatos armazenados na memória de trabalho do Interpretador de Contexto:

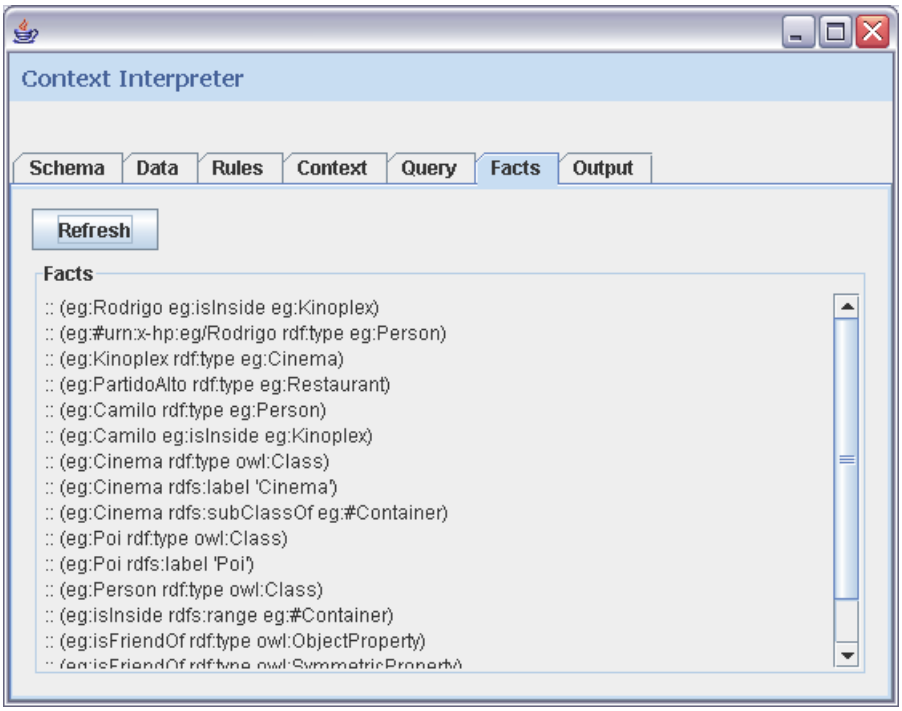

Figura 6-13 – Consulta aos fatos conhecidos e inferidos pelo Interpretador

Na Figura 6-14, a interface de depuração é utilizada para a execução de uma consulta em RDQL. O resultado da consulta retorna todos os indivíduos instanciados e suas respectivas localizações.

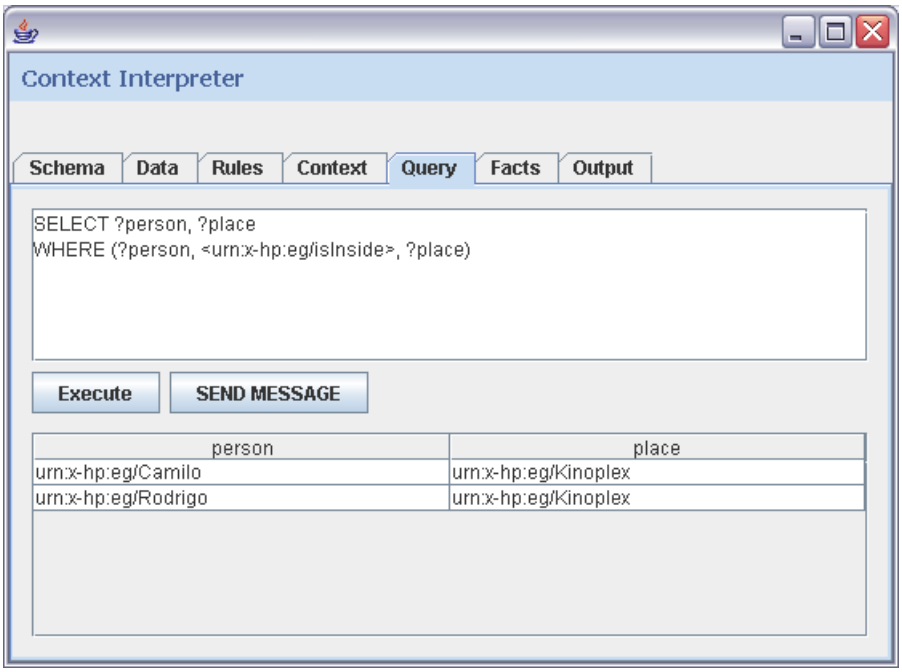

Figura 6-14 – Execução de uma consulta em RDQL

A Figura 6-15 apresenta a interface de depuração sendo utilizada para o envio de uma informação contextual através da tripla (*Rodrigo, isInside, Kinoplex*). Esta funcionalidade visa simular a aquisição das informações contextuais de diferentes provedores de contexto e permite tanto a alteração como a inclusão de novos fatos na memória de trabalho do Interpretador de Contexto.

Também na Figura 6-15, a interface de depuração é utilizada para o envio de uma subscrição através de uma regra do tipo *event-condition-action*. Esta regra dispara o serviço de envio de mensagens no caso de dois amigos estarem localizados em um mesmo lugar.

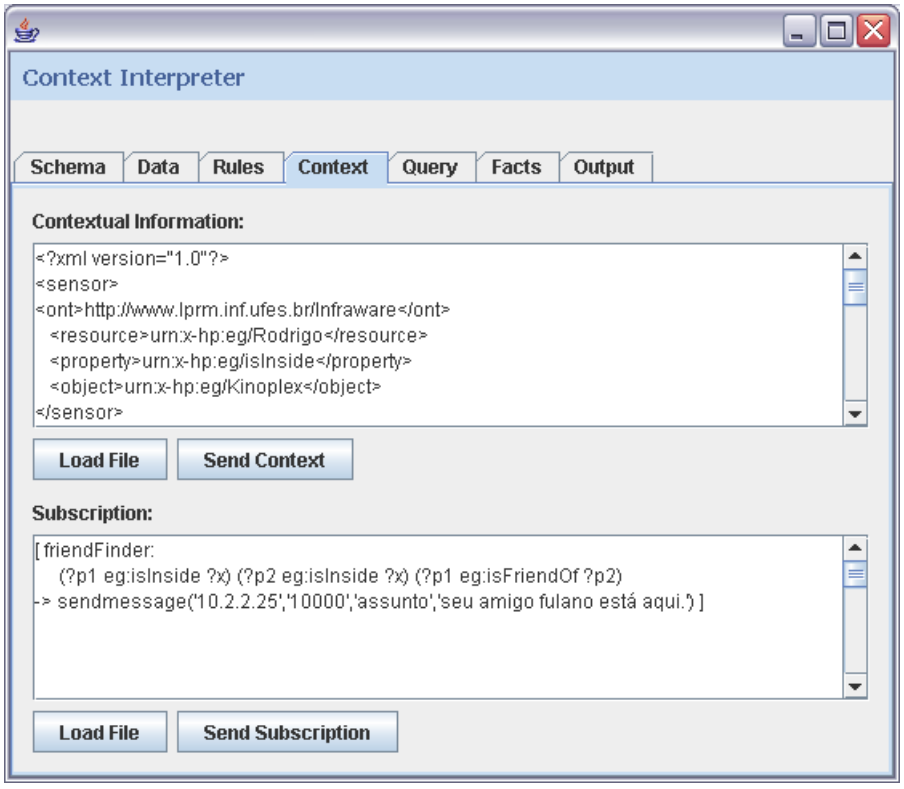

Figura 6-15 – Envio de informações contextuais e subscrições

O código-fonte referente à interface de depuração do Interpretador de Contexto pode ser encontrado no Apêndice B.

6.7. Cenário de Aplicação

Como prova de conceito, esta seção ilustra a utilização do Interpretador de Contexto a partir do desenvolvimento de algumas aplicações-piloto. A implementação dessas aplicações permite avaliar as decisões de projeto relacionadas à modelagem e inferência de informações

contextuais bem como a tecnologia empregada na implementação do Interpretador de Contexto.

Os protótipos desenvolvidos simulam o processo de aquisição das informações contextuais a partir dos sensores, uma vez que a implementação do componente de Acesso e Integração de Dados encontra-se fora do escopo deste trabalho, sendo objeto de implementação em outros trabalhos em andamento no projeto Infraware.

6.7.1. Cenário Turístico

O domínio de aplicação relacionado ao turismo é particularmente rico para a Computação Sensível ao Contexto uma vez que a mobilidade dos usuários e seus respectivos interesses pessoais podem ser explorados pelas aplicações com a intenção de produzir serviços mais flexíveis, adaptáveis e personalizados.

Tendo-se como referência um turista realizando um passeio em uma cidade desconhecida, pode-se imaginar um conjunto de aplicações capazes de auxiliá-lo: mapas interativos, recebimento de informações sobre pontos turísticos personalizadas (de acordo com o perfil e preferências dos usuários) e acesso a serviços de infra-estrutura, gastronomia e entretenimento (hotéis, restaurantes, bares, cinemas, teatros e etc.).

Os usuários das aplicações são identificados como principais atores do sistema e interagem com a plataforma Infraware através de aplicações rodando em dispositivos portáteis como h*andhelds*, telefones celulares ou outros dispositivos móveis. A partir do cenário descrito, são identificados os elementos que compõem o domínio de aplicação analisado. A Figura 6-16 ilustra uma versão simplificada da ontologia de domínio para as aplicações relacionadas ao turismo. Esta ontologia foi escrita em OWL e pode ser encontrada no Apêndice C.

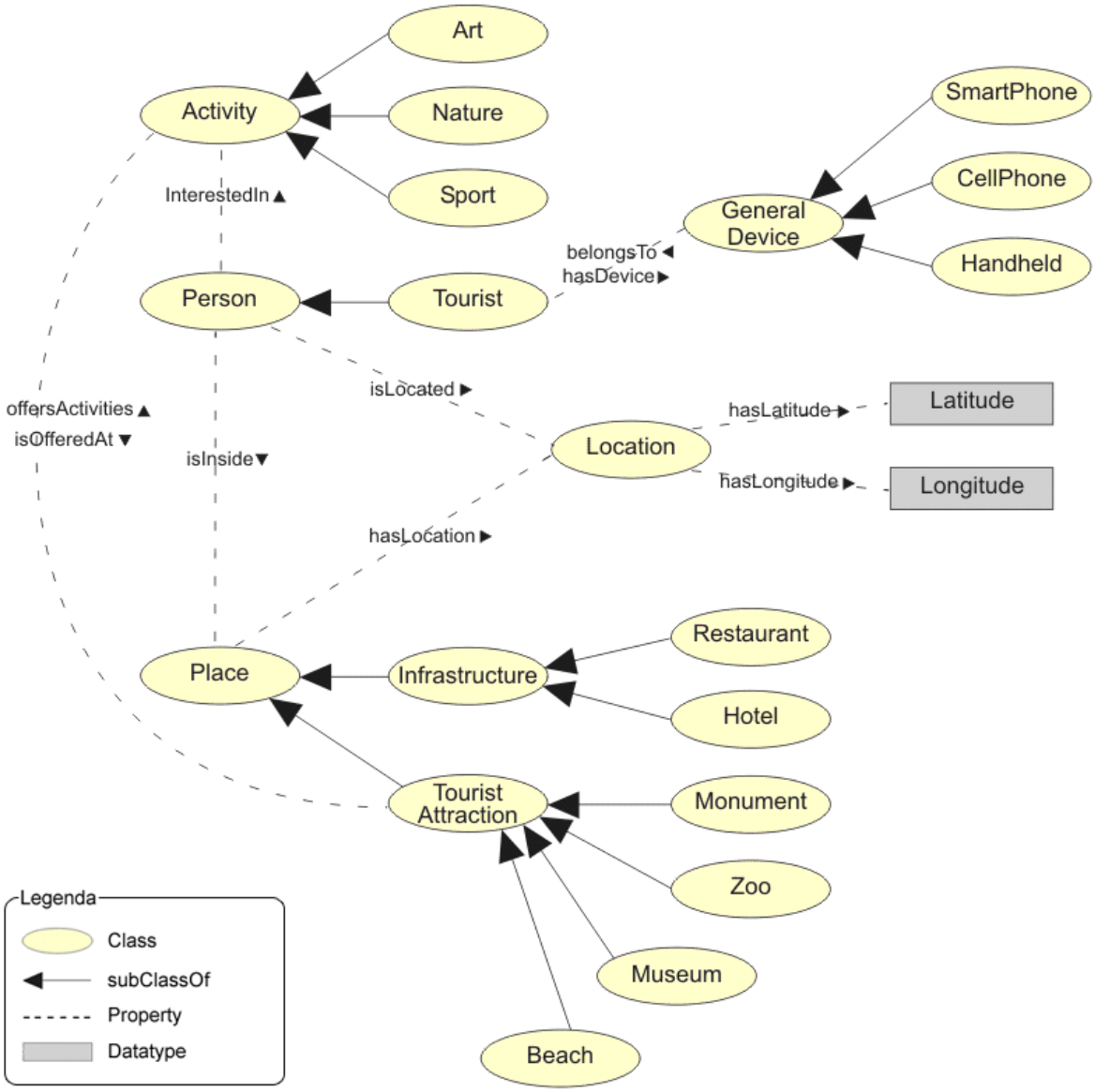

Figura 6-16 – Ontologia simplificada para o domínio do turismo

Uma possível aplicação poderia interessar-se em enviar informações a respeito dos pontos turísticos visitados pelos usuários no momento em que a visita é realizada, seguindo o modelo de subscrição *event-driven* suportado pela plataforma Infraware. Apesar dos dispositivos móveis permitirem o acesso as informações a qualquer hora e lugar, pode ser interessante personalizar o conteúdo das informações enviadas de acordo o tipo de dispositivo empregado. Por exemplo, usuários de aparelhos celulares comuns podem se sentir desconfortáveis ao lerem mensagens muito longas em *displays* pequenos. Um outro fator que justifica a personalização do conteúdo enviado aos usuários está relacionado à grande variação de recursos para a apresentação de conteúdo multimídia entre dispositivos

heterogêneos. As Figuras 6-17 e 6-18 descrevem dois pedidos de subscrição para o envio de informações turísticas para diferentes classes de dispositivos.

```
[subscription01: (?t rdf:type Tourist)
                 (?t hasDevice ?d)
                 (?d rdf:type CellPhone)
                 (?t isInside ?p)
                 (?p rdf:type TouristAttraction)
                 -> SendSMSTouristInformation(?d,?p) ]
```

Figura 6-17 – Regra para o envio de informações turísticas através do serviço SMS

```
[subscription02: (?t rdf:type Tourist)
                 (?t hasDevice ?d)
                 (?d rdf:type SmartPhone)
                 (?t isInside ?p)
                 (?p rdf:type TouristAttraction)
                 -> SendMMSTouristInformation(?d,?p) ]
```

Figura 6-18 – Regra para o envio de informações turísticas através do serviço MMS

O primeiro pedido de subscrição (Figura 7-2) invoca o serviço de envio de informações turísticas, através do envio de mensagens de texto SMS (*Short Message Service*), sempre que um turista portador de um dispositivo da classe *CellPhone* visitar um ponto turístico da cidade. Já o segundo pedido de subscrição (Figura 7-3) envia as informações aos turistas portadores de dispositivos da classe *SmartPhone* através do envio de mensagens MMS (*Multimedia Messaging Service*). Em ambos os serviços, são passados como parâmetros o dispositivo portado pelo usuário e o ponto turístico visitado, através das variáveis "?d" e "?p".

A Figura 6-19 ilustra o recebimento das mensagens nos diferentes dispositivos:

Figura 6-19 – Recebimento de informações turísticas em diferentes dispositivos

Ainda no cenário do turismo, uma outra aplicação poderia interesar-se por enviar aos usuários sugestões de lugares a serem visitados, de acordo com os interesses pessoais dos próprios usuários. A partir de uma solicitação realizada por um usuário, a aplicação requisita tais informações a plataforma Infraware através de um pedido de subscrição do tipo *request-response*, uma vez que a resposta da requisição é aguardada de maneira síncrona pela aplicação.

A Figura 6-20 ilustra este pedido de subscrição realizado pela aplicação através de uma consulta RDQL em que são selecionados os pontos turísticos e suas respectivas atividades, de acordo o interesse de um usuário específico. No exemplo ilustrativo, assumiremos que as atividades de interesse do usuário são: *Nature* e *Sport*.

```
SELECT ?touristAtraction, ?activity
WHERE (?person, <rdf:type>, <v:Peron>),
      (?person, <v:name>, ?name),
      (?person, <v:InterestedIn>, ?activity),
      (?activity, <v:isOfferedAt>, ?touristAtraction)
AND ?name eq "Rodrigo Mantovaneli Pessoa"
USING vcard FOR <http://www.w3.org/2001/vcard-rdf/3.0#>
      v FOR <http://lprm.inf.ufes.br/TourismOntology#>';
```

Figura 6-20 – Consulta aos pontos turísticos de interesse de um usuário

Esta consulta é processada pelo componente Interpretador de Contexto após ser validada pelo Gerente de Subscrição e o resultado do processamento é enviado para a aplicação requisitante. A Figura 6-21 ilustra o resultado da consulta sendo exibido para o usuário da aplicação.

Figura 6-21 – Sugestões para visitas a pontos turísticos baseadas nas preferências dos usuários

6.8. Trabalhos Relacionados

A interpretação de contexto vem sendo explorada em algumas pesquisas e projetos de infra-estruturas de suporte a aplicações sensíveis ao contexto, seja através de módulos ou componentes específicos para tal propósito, como um Interpretador de Contexto, seja através de mecanismos e algoritmos desenvolvidos especialmente para este objetivo.

Um exemplo de estratégia utilizada para a interpretação de contexto é a adotapa pelo *Solar System* [Chen et al., 2002]. Nela, utiliza-se uma estrutura baseada em Grafos Acíclicos Dirigidos para a manipulação e o refinamento de informações contextuais, proporcionando a interpretação de contexto através de combinações de operadores em seus grafos.

O *Context Toolkit* [Dey, 2000], um dos trabalhos pioneiros na área, oferece um framework conceitual de componentes para apoiar o desenvolvimento e a execução de aplicações sensíveis ao contexto. A interpretação de contexto ocorre através dos componentes Interpreters, sendo estes os componentes responsáveis pela combinação de uma ou mais informações para a produção de uma nova informação contextual. Esta abordagem torna a implementação de cada componente Interpreter dependente do domínio das aplicações.

Recentemente, com o desenvolvimento das tecnologias e pesquisas de apoio a computação sensível ao contexto, alguns trabalhos começaram a explorar o uso de ontologias para modelagem semântica e realização de inferências em ambientes ubíquos. O *Context Broker Architecture* (CoBrA) [Chen, 2004] fornece uma arquitetura baseada em agentes para o desenvolvimento de aplicações sensíveis ao contexto em espaços inteligentes, por exemplo, salas de reuniões. Trata-se de uma arquitetura centralizada no componente *context broker*, que compartilha um modelo de contexto entre agentes distribuídos, serviços e dispositivos. CoBrA define a ontologia SOUPA, descrita em OWL, para a manipulação e inferência de informações contextuais, utilizando a API Jena em sua máquina de inferência para a manipulação de ontologias.

A arquitetura *Service-Oriented Context-Aware Middleware* (SOCAM) [Gu et al., 2005] define um modelo formal de contexto baseado em ontologias com o objetivo de prover prototipação rápida de serviços e aplicações context-aware. As informações contextuais são descritas em OWL, permitindo o compartilhamento de conhecimento contextual entre diferentes entidades e a realização de inferências sobre essas informações. O Interpretador de Contexto da SOCAM é constituído de uma máquina de inferência (*Context Reasoner*) e de

uma base de conhecimentos contextuais (*Context KB*), sendo responsável pelo processamento e a interpretação de contexto. O processo de inferência também é baseado em regras de inferência e pode utilizar o encadeamento progressivo e regressivo.

A Tabela 6-1 ilustra a comparação entre o Interpretador de Contexto desenvolvido para a plataforma Infraware e outros trabalhos relacionados. São analisados aspectos relacionados à: especificação e modelagem das informações contextuais, interpretação de informações contextuais, comunicação distribuída e ferramentas de suporte ao desenvolvimento de aplicações.

	Especificação/ Modelagem	Interpretação	Comunicação distribuída	Ferramentas de desenvolvimento
Solar System	Grafos	Combinação de operadores do grafo	TCP/IP Sockets	-
Context Toolkit	XML	Interpretador	XML sobre HTTP	-
CoBrA	OWL	Máquina de Inferência	RDF/XML sobre HTTP	-
SOCAM	OWL	Máquina de Inferência	RDF/XML sobre HTTP	-
Infraware	OWL	Máquina de Inferência	RDF/XML sobre HTTP	Interface de Depuração

Tabela 6-1 Comparação entre o Interpretador de Contexto e trabalhos relacionados.

Com relação à especificação e modelagem das informações contextuais, tanto a estrutura baseada em grafos, proposta pelo *Solar System*, quanto à estrutura hierárquica baseada em tags utilizada pelo *Context Toolkit* não exploram a descrição semântica das informações contextuais e são, portanto, incapazes de implementar algoritmos que realizem inferência e interpretação de novos contextos a partir apenas da descrição destas informações. Em contrapartida, a abordagem para a modelagem de contexto focada na construção de ontologias para a especificação de domínios explora a descrição semântica das informações além de permitir o compartilhamento dos conceitos a respeito dos domínios entre vários sistemas. O compartilhamento destes conceitos ajuda a eliminar ambigüidades com relação a informações contextuais, que poderiam ter diferentes significados em sistemas distintos.

Com relação à interpretação das informações contextuais, as máquinas de inferência utilizadas nas infra-estruturas CoBra, SOCAM e Infraware utilizam a semântica da descrição das informações contextuais especificadas em OWL para viabilizar mecanismos de inferência a partir dos quais novas informações são produzidas. Já as abordagens utilizadas pelo *Solar System* e *Context Toolkit* exige que operadores de grafos ou componentes Interpreters sejam implementados pelos desenvolvedores das aplicações.

A comunicação distribuída utilizada pelo *Solar System*, baseada em TCP/IP Sockets não é transparente para as aplicações clientes, enquanto que as abordagens baseadas em XML ou XML/RDF sobre HTTP utilizam as vantagens oferecidas pelos *Web Services* para suportar

a concepção de sistemas fracamente acoplados, para interoperar em ambientes de computação ubíquos e distribuídos.

Com relação ao suporte a ferramentas de desenvolvimento, o Interpretador de Contexto da plataforma Infraware confere vantagens sobre os outros trabalhos analisados, sendo a única abordagem a oferecer aos desenvolvedores das aplicações uma interface gráfica de depuração, que permite aos desenvolvedores acompanhar passo a passo a execução das inferências realizadas pelo interpretador, permitindo a identificação de erros referentes à definição tento dos modelos de contexto quanto das regras de inferência.

6.9. Conclusão do Capítulo

Neste Capítulo foi apresentado o Interpretador de Contexto da plataforma Infraware, responsável pela inferência de novas informações contextuais e pela percepção de contexto, conforme especificado pelas aplicações. Para validar a abordagem proposta, foram desenvolvidas aplicações-piloto no cenário do turismo. Os testes de validação, embora de natureza simples, permitiram observar a adequação da estratégia adotada no projeto e implementação do Interpretador de Contexto com os objetivos traçados pela plataforma Infraware.

A utilização das informações contextuais providas pelo Interpretador de Contexto às aplicações-piloto possibilitou sintonizar o comportamento do dispositivo móvel com o seu usuário, permitindo a estas aplicações detectar e reagir a mudanças do ambiente e se adaptarem de acordo com o contexto dos seus usuários. Como conseqüência, espera-se uma melhora na experiência dos usuários com seus dispositivos móveis.

Atualmente, o Interpretador de Contexto vem sendo integrado aos outros componentes já implementados da plataforma Infraware através da disponibilização de seus métodos por *web services* remotos. Espera-se que, ao final da integração, seja possível o desenvolvimento de aplicações reais.

7. Conclusões e Perspectivas Futuras

As aplicações móveis sensíveis ao contexto trouxeram, sem dúvida, uma série de novos desafios teóricos e tecnológicos, inerentes da dinamicidade e da heterogeneidade características, que devem ser explorados e utilizados de maneira a potencializar o uso dessas aplicações nos mais diferentes domínios. A utilização de uma infra-estrutura de *middleware* como a Infraware é potencialmente promissora em diversos cenários, uma vez que a plataforma oferece facilidades para a concepção de uma ampla variedade de aplicações sensíveis ao contexto. O projeto TeleCardio [Telecardio, 2005], por exemplo, vem investigando a utilização da plataforma Infraware na concepção de um sistema de baixo custo e flexível para o telemonitoramento da atividade cardíaca de pacientes através do eletrocardiograma.

Este trabalho propôs a versão inicial da arquitetura conceitual da plataforma Infraware, que atualmente encontra-se em desenvolvimento no Laboratório de Pesquisas em Redes e Multimídia da UFES. Exemplos de facilidades oferecidas por este middleware incluem um modelo de contexto, um mecanismo de interpretação e percepção de contexto, um gerente de serviços semânticos, uma camada para o controle de acesso e privacidade, uma interface uniforme para a aquisição dos dados dos sensores e um módulo gerente de subscrições. Foram apresentados os principais componentes do *middleware*, com destaque para o Interpretador de Contexto, que desempenha um papel fundamental na arquitetura proposta.

O Interpretador de Contexto proposto utiliza tecnologias de representação e estruturação semântica de informações contextuais. A utilização dessas tecnologias, em especial de ontologias, permite a criação de modelos formais, semânticos e extensíveis que descrevam o domínio das aplicações a partir da especificação de conceitos, relações e axiomas. O compartilhamento desses modelos entre os demais componentes da plataforma promove a integração entre as aplicações e a Infraware. A extensibilidade proporcionada pelo uso de ontologias permite ainda que mudanças no modelo acarretem alterações mínimas nas aplicações desenvolvidas. A especificação semântica das informações contextuais e do domínio das aplicações possibilita ainda a interpretação e inferência de novas informações contextuais, baseados nas informações semânticas descritas nos modelos.

Um tema a ser futuramente investigado refere-se à definição e à aplicação de um metamodelo de contexto definido a partir da própria linguagem OWL, contemplando

parâmetros relacionados a qualidade e confiabilidade das informações contextuais, como proposto por [Fuchs et al., 2005]. O uso do metamodelo aumentaria potencialmente o reuso das soluções utilizadas na especificação dos modelos que definem as informações contextuais e o domínio das aplicações. A existência de um metamodelo de contexto formalmente definido também facilitaria o mapeamento de informações contextuais descritas em um modelo específico de domínio para outro, aumentando a portabilidade e a interoperabilidade entre as aplicações desenvolvidas. Uma primeira iniciativa no sentido de propor um metamodelo de contexto para a plataforma Infraware, especificado usando a linguagem *Meta Object Facility* (MOF), foi proposta por [Leite, 2006].

Uma outra técnica que aumentaria o poder de expressão do Interpretador de Contexto seria a incorporação de lógica fuzzy à sua máquina de inferência. Através de regras fuzzy, situações do tipo “se o nível de qualidade de um canal de comunicação for alta, então realize a atividade 1". Neste caso, o nível de qualidade depende de uma avaliação que não é precisa, mas que é importante para o desenvolvimento da atividade.

Há ainda planos de um projeto futuro para que o Interpretador de Contexto suporte a validade temporal dos fatos armazenados em sua base de conhecimento. Estes fatos são normalmente relacionamentos que associam duas entidades no âmbito da dimensão temporal e suas respectivas validades seriam especificadas através de seus rótulos temporais. As regras e eventos especificados seriam acrescidos de operadores de lógica temporal, onde o tempo é caracterizado por uma linha seqüencial simples de eventos. Isto permitiria expressar quantitativamente o tamanho dos intervalos temporais, a distância temporal entre os eventos e a viabilidade das eventos modelados quanto às proposições lógicas estabelecidas.

Algumas questões importantes não foram deliberadamente exploradas no escopo deste trabalho, constituindo, assim, temas de estudos futuros. São exemplos: a manipulação de informações contextuais imprecisas e ambíguas, e análises estatísticas quanto ao desempenho, disponibilidade e escalabilidade da plataforma e do Interpretador de Contexto, de forma a se estabelecer uma base comparativa para investigações adicionais. Espera-se que, com a finalização da implementação e da integração dos módulos Gerente de Serviços, Gerente de Subscrição e Coordenador, uma análise mais apurada e eficiente do desempenho da infra-estrutura geral da Infraware seja realizada.

8. Referências

Akman, V.; Surav, M. (1997) The use of situationtheory in context modeling. ComputationalIntelligence 13, 3, 427–438.

Andreão, R. V. (2004) Segmentation de battements ECG par approche markovienne: application à la détection d'ischémies, Tese de Doutorado, Dep. EPH, Institut National des Télécommunications, Evry, França.

Arruda Jr, C.R.E. (2003) "Context Kernel: um Web service baseado nas dimensões de informação de contexto"; 1; 85p.; Dissertação (Mestrado em Ciências da Computação e Matemática Computacional) - Universidade de São Paulo. São Carlos.

Baader, F. et al (2003) "The Description Logic Handbook – Theory, Implementation and Applications". Cambridge University Press.

Barbosa, A. C. P. (2000) Middleware para Integração de Dados HeterogêneosBaseado em Composição de Frameworks, Tese de Doutorado, PUC-RJ, 2000.

Barwise, J.; Perry, J. (1983) Situations and Attitudes. MIT Press, , Cambridge, MA.

Bechhofer, S., van Harmelen, F., et al. (2004) OWL Web Ontology Language Reference: www.w3.org/TR/2004/REC-owl-ref-20040210/, W3C Recommendation. Acessado em julho/2006.

Brown., P. J. (1996) The stick-e document: A framework for creating context-aware applications. Proceedings of the ElectronicPublishing, pages 259–272.

Calvi, C. Z.; Pessoa, R. M.; Pereira Filho, J. G. (2005) Um interpretador de contexto para plataformas de serviços context-aware. Anais SEMISH, São Leopoldo (RS).

Carmo, R. R. M. (2006) "Um Protocolo de Descoberta de Serviço para Sistemas Sensíveis ao Contexto". Dissertação de Mestrado, Programa de Pós-Graduação em Informática, Universidade Federal do Espírito Santo.

Chen, G.; Kotz, D. (2002): SOLAR: A Pervasive-Computing Infrastructure for Context-Aware Mobile Applications. Technical Report TR2002-421, Dartmouth College.

Chen, H. (2004): An Intelligent Broker Architecture for Pervasive Context-Aware Systems. Ph.D. Thesis, University of Maryland, USA.

COSTA, A. C. M. (2006) "Um Algoritmo de Comparação Semântica de Serviços, baseado em Ontologia e Sensível ao Contexto para a Plataforma Infraware". Projeto Final dog curso de Ciência da Computação. Universidade federal do Espírito Santo.

Cruz, A. O. (2006) "Um Estudo Sobre Controle de Privacidade em Plataformas de Serviços Sensíveis ao Contexto". Projeto Final de curso. Universidade federal do Espírito Santo. Orientador: José Gonçalves Pereira Filho.

DAML+OIL (2001) "DAML+OIL Reference Description". Disponível em: <http://www.w3.org/TR/daml+oil-reference>. Último acesso: junho de 2006.

Dey, A. K. (2000): Providing Architectural Support for Building Context-Aware Applications. Ph.D. Thesis, Georgia Institute of Technology.

Dockhorn Costa, P. D. (2003): Towards a Services Platform for Context-Aware Applications. Master Thesis, University of Twente, The Netherlands.

Dockhorn Costa, P.; Pereira Filho, J. G.; Sinderen M. v. (2003): Architectural Requirements for Building Context-Aware Services Platforms. In Proceedings of the Ninth Open European Summer School and IFIP Workshop on Next Generation Networks (EUNICE 2003), pp. 62-69.

Drost, C. (2004) Privacy in Context-Aware Systems. BSc. thesis, University of Twente, Enschede, Netherlands, December.

Efstratiou, C., (2004) "Coordinated Adaptation for Adaptive Context-aware Applications". PhD Thesis, Lancaster University, England.

Efstratiou, C., Cheverst, E., Davies, N., Friday, A., (2001) "An architecture for the effective support of adaptive context-aware applications"

Erdmann, M. and Studer, R. (2001) "How to Structure and Access XML Documents With Ontologies". Data and Knowledge Engineering, v. 36, p.317-335.

Falbo, R. A., Menezes, C. S. and Rocha, A.R.C. (1998) Using Ontologies to Improve Knowledge Integration in Software Engineering Environments, Proc. of SCI'98/ISAS'98, USA, July.

Finney, J. and Davies, N. (1996) The flexibleubiquitous monitor project. Proceedingsof the Third Computer Networks Sympo-sium.

Flinn, J. D. (2001) Narayanan, and M. Satyanarayanan. Self-Tuned Remote Execution for Pervasive Computing. In Proc. 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), pages 61--66, Schloss Elmau, Germany, May.

Forgy, C. L. (1982) RETE: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19, 17-37.

Fritsch, D., Klinec, D., Volz, S. (2000) Nexus - Positioning and Data Management Concepts for Location Aware Applications. In: Proceedings of the 2nd International Symposium on Telegeoprocessing. Disponível <http://citeseer.ist.psu.edu/511280.html>. Acesso em junho de 2006

Fuchs F., Hochstatter I., Krause M., Berger M. (2005) "A Metamodel Approach to Context Information". Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference, p. 8-14.

Gray, P. D. and Salber, D. (2001): Modelling and Using Sensed Context Information in the Design of Interactive Applications. In: Little, M. R. and Nigay, L. (eds). Engineering for

Human-Computer Interaction. Lecture Notes in Computer Science (2254), Springer-Verlag, pp. 317-336.

Gruber, T. R. (1995) Toward Principles for the Design of Ontologies used for Knowledge Sharing, In Special Issue of the International Journal of Human-Computer Studies. n. 43. pp. 907-928.

Gu, T., Pung, H. K., and Zhang, D. Q. (2005). A service-oriented middleware for building context-aware services. In Journal of Network and Computer Applications. 28, 1 (Jan. 2005).

Gu, T.; Pung, H. K.; Zhang, D. Q,. (2004) A Bayesian Approach for Dealing with Uncertain Contexts. In Proceedings of the Second International Conference on Pervasive Computing (Pervasive 2004), in the book "Advances in Pervasive Computing" published by the Austrian Computer Society, vol. 176, ISBN 3-85403-176-9. Vienna, Austria, April.

Guarino, N. (1998) Formal Ontology and InformationSystems. in: N. Guarino, (Ed.) Formal Ontology inInformation Systems. pp. 3-15, IOS Press, Amsterdam, Netherlands.

Heflin, J. and Hendler, J. (2000) Semantic Interoperability on the Web. In Proceedings of Extreme Markup Languages 2000. Graphic Communications Association, 2000. pp. 111-120.

Henricksen, K. and Indulska, J. (2004): Modelling and Using Imperfect Context Information. In Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications, Workshop on Context Modelling and Reasoning (CoMoRea'04). IEEE Computer Society, pp. 33-37.

Henricksen, K., Indulska, J. and Rakotonirainy, A. (2002): Modeling Context Information in Pervasive Computing Systems. In Proceedings of the First International Conference on Pervasive Computing (Pervasive´02). Lecture Notes in Computer Science (2414), pp. 167-180.

Henricksen, K., Indulska, J. and Rakotonirainy, A. (2003): Generating Context Management Infrastructure from High-Level Context Models. In Proceedings of the 4th International Conference on Móbile Data Management, Industrial Track Proceedings. Melbourne (Australia), pp. 1-6.

Hong, Jason I. (2001) Context Fabric: Infrastructure Support for Context-Aware Systems. Phd. Thesis, The University of California at Berkeley, 2001.

Indulska J., Robinson R., Rakotonirainy A., Henricksen K. (2003) Experiences in Using {CC/PP} in Context-Aware Systems, in 4th International Conference on Mobile Data Management (MDM). Lecture Notes in Computer Science, January, pp. 247-261.

INFRAWARE (2005): Projeto INFRAWARE - Uma Plataforma para Desenvolvimento de Aplicações Móveis e Context-Aware. Departamento de Informática/UFES, Financiamento: FAPES – Fundação de Apoio à Ciência e Tecnologia do Espírito Santo, processo nº 30897041/2005.

Jakob E. Bardram. (2005) "The Java Context Awareness Framework (JCAF)". In Third International Conference on Pervasive Computing, volume 3468 of Lecture Notes in Computer Science, pages 98–115. Springer.

Langheinrich, M. (2002): A privacy awareness system for ubiquitous computing environments,4th International Conference on Ubiquitous Computing (Ubicomp 2002), LNCS No. 2498, Springer-Verlag, pp. 237-245, September.

LEITE, M. M.; de FARIAS, C. R. G.; PESSOA, R. M.; CALVI, C. Z.; PEREIRA FILHO, J. G.. (2006) "Um Metamodelo MOF para o Desenvolvimento de Aplicações Móveis Sensíveis ao Contexto". Proceedings of the: First Workshop on Ontologies and Metamodeling in Software and Data Engineering (WOMSDE), 2006, Florianópolis - SC, Brazil.

McBride, B. (2001) Jena: Implementing the RDF model and syntax specification. In: Proceedings of the 2nd International Workshop on the Semantic Web.

Mena, E.; Illarramendi, A.; Kashyap, V.; and Sheth, A. (2000) OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies, In: International journal on Distributed And Parallel Databases(DAPD), ISSN 0926-8782, 8(2):223-272.

Mostéfaoui, G. K., Pasquier-Rocha, J., Brezillon, P. (2004): Context-Aware Computing: A Guide for the Pervasive Computing Community. In Proceedings of the IEEE/ACS International Conference on Pervasive Services (ICPS´04). IEEE Computer Society, pp. 39-48.

Noble, B. D. et al. (1997) Agile application-aware adaptation for mobility. In: Sixteen ACM Symposium on Operating Systems Principles. Saint Malo, França:[s.n.]. p. 276–287.

Oliveira, K. (1999) Modelo para construção de ambientes de desenvolvimento de software orientados a domínio. 222 f. Tese (Doutorado em Engenharia de Sistemas e Computação). COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro.

OWL (2004) Web Ontology Language Overview. McGuinness, Deborah L. and van Harmelen, Frank. W3C Recommendation. 2004. Disponível em: <http://www.w3.org/TR/owl-features/>. Último acesso: junho de 2006.

Rahwan, I., Graham, C., Sonenberg, L., (2005) Supporting impromptu coordination using automated negotiation. In M. Barley, N. Kasabov (eds.), Intelligent Agents and Multi-Agent Systems VII, Proceedings of the 7th Pacific Rim International Workshop on Multi-Agents (PRIMA), volume 3371 of Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, Germany.

Rajpathak, D.G (2001)., The Task Ontology Component of the Scheduling Library (Pilot Study), Knowledge Media Institute, October.

RDF (2004) RDF Primer. Disponível em <http://www.w3.org/TR/rdf-primer/>. Último acesso: junho de 2006.

RDFS (2004) RDF Vocabulary Description Language 1.0: RDF Schema. Disponível em: <http://www.w3.org/TR/2000/CR-rdf-schema-20000327>. Último acesso: junho de 2006.

Reynolds, D. (2005) Jena Inference Engine User Manual. Disponível em: <http://jena.sourceforge.net/inference/index.html>. Último acesso: junho de 2006.

Rios, D.M., (2003) "Modelling context information using ontology-based techniques", Master Thesis, University of Twente, Enschede, Netherlands, August.

Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R., & Nahrsedt, K. (2002). Gaia: A Middleware platform for active spaces. ACM SIGMOBILE Mobile Computing and Communication Review, 6(4):65-67.

Rubel P., Gouaux F. et al. (2001), Towards Intelligent and Móbile Systems for Early Detection and Interpretation of Cardiological Syndromes. Computers in Cardiology, vol. 28, n° 1.

Santos, L. O. B. da S. (2004): Semantic Services Support for Context-Aware Platforms. Dissertação de Mestrado, Programa de Pós-Graduação em Informática, Universidade Federal do Espírito Santo (UFES).

Satyanarayanan, M. (2002) The evolution of coda. ACM Transac-tions on Computer Systems, v. 20, n. 2, p. 85–124, may

Schilit, B. N. (1995) "A Context-Aware Systems Architecture for Mobile Distributed Computing", Ph.D. Thesis, Columbia University.

Schilit, B. N.; Theimer, M., Welch, B. (1993) Customizing mobile applications. InProceedings of USENIX Mobile & Location-Independent Computing Symposium, pages 129-138,Cambridge, Massachusetts, August.

Schmidt, A.; Beigl, M.; and Gellersen, H. (1999) "There is more to Context than Location." In: Computers & Graphics Journal, Elsevier, Volume 23, No.6, December, pp 893-902.

Seaborne, A. (2004) RDQL - A Query Language for RDF. www.w3.org/Submission/2004/SUBM-RDQL-20040109/. Acessado em julho/2006.

Smith, M. K; Welty, C; Mcguinness, D. L. (2004) "OWL Web Ontology Language Guide." http://www.w3.org/TR/2004/REC-owl-guide-20040210/, acessado em junho de 2006.

Souza, J. P., Garlan, D. (2002) "Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments", Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture, Kluwer Academic Publishers, August 25-31, 2002. pp. 29-43.

Szyperski, C.. (1998) Component Software: Beyond Object Oriented Programming. AddisonWesley Publishing Company, ISBN: 0-201-17888-5.

Strang, T., Linnhoff-Popien, C. (2004). A Context Modelling Survey. In First InternationalWorkshop on Advanced Context Modelling, Reasoning And Management, Nottingham, England, September.

TeleCardio (2005): Projeto Telecardio - Telecardiologia a Serviço do Paciente em Ambientes Hospitalares e Residenciais. DI/DEE/UFES, Financiamento: FAPES - Fundação de Apoio à Ciência e Tecnologia do Espírito Santo, processo nº 31024866/2005.

Timm, J. T .E., Gannod G. C. (2005) A Model-Driven Approach for Specifying Semantic Web Services. Proceedings of the IEEE International Conference on Web Services (ICWS'05) - Volume 00, Pages: 313 – 320, ISBN:0-7695-2409-5.

Viterbo Filho J., Sacramento V., da Rocha R.C.A., Endler M. (2006) MoCA: Uma Arquitetura para o Desenvolvimento de Aplicações Sensíveis ao Contexto para Dispositivos Móveis, Proc. of the XXIV Brazilian Symposium on Computer Networks (SBRC 2006), Tool Session, Curitiba, May 2006.

Wang X., et al., (2004) "Ontology-Based Context Modeling and Reasoning using OWL", Context Modeling and Reasoning Workshop at PerCom 2004. http://citeseer.ist.psu.edu/wang04ontology.html

Weiser M. (1991) "The Computer for the Twenty-First Century" Scientific American, pp. 94-10, September.

Yamim, A. C. (2004) Arquitetura para um Ambiente de Grade Computacional Direcionado às Aplicações Móveis, Distribuídas e Conscientes do Contexto da Computação Pervasiva. Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Porto Alegre.

Apêndice A: Código Fonte – Interface de Programação

```
package ContextInterpreter;
/*
 * Interpreter.java
 * Created on 21 de Fevereiro de 2006, 13:44
 */

import java.util.*;
import java.io.*;
import com.hp.hpl.jena.*;
import com.hp.hpl.jena.vocabulary.*;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.rdf.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.rdf.listeners.*;
import com.hp.hpl.jena.rdf.arp.*;
import com.hp.hpl.jena.util.*;
import com.hp.hpl.jena.reasoner.*;
import com.hp.hpl.jena.reasoner.rulesys.*;
import com.hp.hpl.jena.reasoner.transitiveReasoner.*;
import com.hp.hpl.jena.rdql.*;

/**
 *
 * @author Rodrigo Mantovaneli Pessoa
 */
public class Interpreter {
    private Model schema;
    private Model data;
    private List rules;
    private InfModel infModel;
    private GenericRuleReasoner ruleReasoner;

    /** Creates a new instance of Interpreter */
    public Interpreter(String schemaFile, String dataFile, String rulesFile) {
        com.hp.hpl.jena.reasoner.rulesys.BuiltinRegistry.theRegistry.register(new
SendMessage());
        schema = ModelLoader.loadModel("file:" + schemaFile);
        data = ModelLoader.loadModel("file:" + dataFile);

        StringBuffer ruleSrc = new StringBuffer();
        try {
          BufferedReader in = new BufferedReader(new FileReader(rulesFile));
          String str;
          while ((str = in.readLine()) != null) {
            ruleSrc.append(str + "\n");
          }
          in.close();
        } catch (IOException e) {
            System.out.println(e);
        }
        rules = new LinkedList();
        rules.addAll(Rule.parseRules(ruleSrc.toString()));
        ruleReasoner = new GenericRuleReasoner(rules);
        ruleReasoner = (GenericRuleReasoner)ruleReasoner.bindSchema(schema);
        infModel = ModelFactory.createInfModel(ruleReasoner, data);
    }

    public void reset() {
        infModel.reset();
```

```
    }

    public void addStatement(Resource res, Property prop, Object obj) {
        infModel.reset();
        this.data.add(res,prop,obj);
    }

    public void printStatements(Resource s, Property p, Resource o) {
        Model m = infModel;
        for (StmtIterator i = m.listStatements(s,p,o); i.hasNext(); ) {
            Statement stmt = i.nextStatement();
            System.out.println(" :: " + PrintUtil.print(stmt));
        }
    }

    public void addContextMonitor(String ruleSrc) {
        rules.addAll(Rule.parseRules(ruleSrc));
        ruleReasoner = new GenericRuleReasoner(rules);
        ruleReasoner = (GenericRuleReasoner)ruleReasoner.bindSchema(schema);
        infModel = ModelFactory.createInfModel(ruleReasoner, data);
    }

    public void changeObject(String resourceStr, String propertyStr, Object obj) {
      infModel.reset();
      Resource r = data.getResource(resourceStr);
      Property p = data.getProperty(propertyStr);
      if (r.getProperty(p) != null) {
          r.getProperty(p).changeObject(obj);
      }
      else {
        infModel.reset();
        this.data.add(r,p,obj);
      }
      infModel.prepare();
    }

    public void changeObject(String resourceStr, String propertyStr, String objStr)
{
      infModel.reset();
      Resource r = data.getResource(resourceStr);
      Property p = data.getProperty(propertyStr);
      Resource o = data.getResource(objStr);
      if (r.getProperty(p) != null) {
          r.getProperty(p).changeObject(o);
      }
      else {
        infModel.reset();
        this.data.add(r,p,o);
      }
      infModel.prepare();
    }

    public QueryResults executeQuery(String strQuery) {
        Query query = new Query(strQuery);
        query.setSource(infModel);
        QueryEngine qe = new QueryEngine(query);
        QueryResults results = qe.exec();
        return results;
    }

    public String getStatementsAsText() {
        StringBuffer strBuff = new StringBuffer();
        Model m = infModel;
        for (StmtIterator i = m.listStatements(); i.hasNext(); ) {
            Statement stmt = i.nextStatement();
            strBuff.append(" :: " + PrintUtil.print(stmt) + "\n");
        }
        return strBuff.toString();
```

```
    }


}

package ContextInterpreter;
/*
 * SendMessage.java
 * Created on 4 de Janeiro de 2006, 15:53
 */

import com.hp.hpl.jena.reasoner.rulesys.*;
import com.hp.hpl.jena.reasoner.rulesys.builtins.*;
import com.hp.hpl.jena.util.PrintUtil;
import com.hp.hpl.jena.graph.*;
import java.net.*;
import java.io.*;

/**
 *
 * @author Rodrigo Mantovaneli Pessoa
 */
public class SendMessage extends BaseBuiltin {

    /** Creates a new instance of SendMessage */
    public SendMessage() {
    }

    /**
     * Return a name for this builtin, normally this will be the name of the
     * functor that will be used to invoke it.
     */
    public String getName() {
        return "sendmessage";
    }

    /**
     * This method is invoked when the builtin is called in a rule body.
     * @param args the array of argument values for the builtin, this is an array
     * of Nodes, some of which may be Node_RuleVariables.
     * @param length the length of the argument list, may be less than the length
of the args array
     * for some rule engines
     * @param context an execution context giving access to other relevant data
     * @return return true if the buildin predicate is deemed to have succeeded in
     * the current environment
     */
    public boolean bodyCall(Node[] args, int length, RuleContext context) {
        String AreaCode = getArg(0, args, context).toString();
        AreaCode = AreaCode.substring(1,AreaCode.length()-1);
        String PhoneNumber = getArg(1, args, context).toString();
        PhoneNumber = PhoneNumber.substring(1,PhoneNumber.length()-1);
        System.out.println("number: " +PhoneNumber);
        String Subject = getArg(2, args, context).toString();
        Subject = Subject.substring(1,Subject.length()-1);
        String MessageBody = getArg(3, args, context).toString();
        MessageBody = MessageBody.substring(1,MessageBody.length()-1);

        String servidorSocket = AreaCode;
        int port = Integer.parseInt(PhoneNumber);
        Socket socket = null;
        String lineToBeSent;
        BufferedReader input;
        PrintWriter output;
        int ERROR = 1;
        try {
            socket = new Socket(servidorSocket, port);
            System.out.println("Connected with server " +
```

```
            socket.getInetAddress() +
            ":" + socket.getPort());
    }
    catch (UnknownHostException e) {
        System.out.println(e);
        return false;
    }
    catch (IOException e) {
        System.out.println(e);
        return false;
    }
    try {
        input = new BufferedReader(new InputStreamReader(System.in));
        output = new PrintWriter(socket.getOutputStream(),true);
        lineToBeSent =
                "<message>" +
                "  <subject>" + Subject + "</subject>" +
                "  <sender>INFRAWARE</sender>" +
                "  <body>"+ MessageBody +"</body>" +
                "</message>";
        output.println(lineToBeSent);
        System.out.println(lineToBeSent);
    }
    catch (IOException e) {
        System.out.println(e);
         return false;
    }
    try {
        socket.close();
    }
    catch (IOException e) {
        System.out.println(e);
        return false;
    }
    return true;
}


/**
 * This method is invoked when the builtin is called in a rule head.
 * Such a use is only valid in a forward rule.
 * @param args the array of argument values for the builtin, this is an array
 * of Nodes.
 * @param context an execution context giving access to other relevant data
 */
public void headAction(Node[] args, int length, RuleContext context) {
    String AreaCode = getArg(0, args, context).toString();
    AreaCode = AreaCode.substring(1,AreaCode.length()-1);
    String PhoneNumber = getArg(1, args, context).toString();
    PhoneNumber = PhoneNumber.substring(1,PhoneNumber.length()-1);
    System.out.println("number: " +PhoneNumber);
    String Subject = getArg(2, args, context).toString();
    Subject = Subject.substring(1,Subject.length()-1);
    String MessageBody = getArg(3, args, context).toString();
    MessageBody = MessageBody.substring(1,MessageBody.length()-1);

    String servidorSocket = AreaCode;
    int port = Integer.parseInt(PhoneNumber);
    Socket socket = null;
    String lineToBeSent;
    BufferedReader input;
    PrintWriter output;
    int ERROR = 1;
    // onnect to server
    try {
        socket = new Socket(servidorSocket, port);
        System.out.println("Connected with server " +
            socket.getInetAddress() +
```

```
                ":" + socket.getPort());
        }
        catch (UnknownHostException e) {
            System.out.println(e);
        }
        catch (IOException e) {
            System.out.println(e);
        }
        try {
            input = new BufferedReader(new InputStreamReader(System.in));
            output = new PrintWriter(socket.getOutputStream(),true);
            lineToBeSent =
                    "<message>" +
                    "  <subject>" + Subject + "</subject>" +
                    "  <sender>INFRAWARE</sender>" +
                    "  <body>"+ MessageBody +"</body>" +
                    "</message>";
            output.println(lineToBeSent);
            System.out.println(lineToBeSent);
        }
        catch (IOException e) {
            System.out.println(e);
        }
        try {
            socket.close();
        }
        catch (IOException e) {
            System.out.println(e);
        }
    }

}
```

Apêndice B: Código Fonte – Interface de Depuração

```
package ContextInterpreter;
/*
 * InterpreterShell.java
 * Created on 17 de Abril de 2006, 15:35
 */

import javax.swing.JFileChooser;
import java.io.*;
import org.xml.sax.InputSource;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import com.hp.hpl.jena.rdql.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.net.*;


/**
 *
 * @author  Rodrigo Mantovaneli Pessoa
 */

class ConsoleStream extends PrintStream {
    private javax.swing.JTextArea console;

    public ConsoleStream(javax.swing.JTextArea textArea) {
        super(System.out);
        console = textArea;
    }

    public void println(String str) {
        console.append(str + "\n");
    }

    public void println(Object obj) {
        console.append(obj.toString() + "\n");
    }

    public void print(String str) {
        console.append(str);
    }

    public void print(Object obj) {
        console.append(obj.toString());
    }

}

public class InterpreterShell extends javax.swing.JFrame {

    /** Creates new form InterpreterShell */
    public InterpreterShell() {
        initComponents();

        //Altera a stream padrão de saída:
      PrintStream stdout = null;
      try {
            stdout = new ConsoleStream(jTextAreaOutput);
```

```
            //stdout = new PrintStream(new FileOutputStream("./output.out"));
       } catch (Exception e) {
            System.out.println("Redirect:  Unable to open output file!");
            System.exit(1);
       }
       System.setOut(stdout);
        System.out.println("Bem Vindo ao Interpretador de Contexto!");
    }


    public void ParseSensorXML(String xml) {
        try {
            DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
            InputSource input = new InputSource(new StringReader(xml));
            Document doc = builder.parse(input);
            NodeList nodes = doc.getElementsByTagName("sensor");
            for (int i = 0; i < nodes.getLength(); i++) {
                Element element = (Element) nodes.item(i);
                String resource = getChildTagValue(element, "resource" );
                String property = getChildTagValue(element, "property" );
                String object   = getChildTagValue(element, "object" );
                interpreter.changeObject(resource,property,object);
            }
        }
        catch (Exception e) {
           e.printStackTrace();
        }
    }

    private static String getChildTagValue( Element elem, String tagName ) throws
Exception {
        NodeList children = elem.getElementsByTagName( tagName );
        if( children == null ) return null;
        Element child = (Element) children.item(0);
        if( child == null ) return null;
        return child.getFirstChild().getNodeValue();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     */
    private void initComponents() {
        jFileChooser1 = new javax.swing.JFileChooser();
        jTabbedPane1 = new javax.swing.JTabbedPane();
        jpSchema = new javax.swing.JPanel();
        jbLoadSchema = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTextAreaSchema = new javax.swing.JTextArea();
        jpData = new javax.swing.JPanel();
        jbLoadData = new javax.swing.JButton();
        jScrollPane2 = new javax.swing.JScrollPane();
        jTextAreaData = new javax.swing.JTextArea();
        jpRules = new javax.swing.JPanel();
        jbLoadRules = new javax.swing.JButton();
        jScrollPane3 = new javax.swing.JScrollPane();
        jTextAreaRules = new javax.swing.JTextArea();
        jpContext = new javax.swing.JPanel();
        jPanel5 = new javax.swing.JPanel();
        jScrollPane6 = new javax.swing.JScrollPane();
        jTextAreaContext = new javax.swing.JTextArea();
        jbSendContext = new javax.swing.JButton();
        jbLoadFileContext = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
        jPanel6 = new javax.swing.JPanel();
        jScrollPane9 = new javax.swing.JScrollPane();
        jTextAreaSubscription = new javax.swing.JTextArea();
        jLabel3 = new javax.swing.JLabel();
```

```
        jbLoadFileSubscription = new javax.swing.JButton();
        jbSendSubscription = new javax.swing.JButton();
        jPanel2 = new javax.swing.JPanel();
        jPanel3 = new javax.swing.JPanel();
        jScrollPane7 = new javax.swing.JScrollPane();
        jTextAreaQuery = new javax.swing.JTextArea();
        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jPanel4 = new javax.swing.JPanel();
        jScrollPane8 = new javax.swing.JScrollPane();
        jpFacts = new javax.swing.JPanel();
        jScrollPane5 = new javax.swing.JScrollPane();
        jTextAreaFacts = new javax.swing.JTextArea();
        jbRefreshFacts = new javax.swing.JButton();
        jpOutput = new javax.swing.JPanel();
        jScrollPane4 = new javax.swing.JScrollPane();
        jTextAreaOutput = new javax.swing.JTextArea();
        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();

        jFileChooser1.setDialogTitle("Open File");

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        jTabbedPane1.setPreferredSize(new java.awt.Dimension(460, 320));
        jbLoadSchema.setText("Load Schema");
        jbLoadSchema.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jbLoadSchemaActionPerformed(evt);
            }
        });


jScrollPane1.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SC
ROLLBAR_ALWAYS);
        jScrollPane1.setPreferredSize(new java.awt.Dimension(400, 400));
        jTextAreaSchema.setEditable(false);
        jTextAreaSchema.setLineWrap(true);
        jTextAreaSchema.setRows(1);
        jTextAreaSchema.setWrapStyleWord(true);
        jTextAreaSchema.setMaximumSize(null);
        jTextAreaSchema.setPreferredSize(null);
        jScrollPane1.setViewportView(jTextAreaSchema);

        org.jdesktop.layout.GroupLayout jpSchemaLayout = new
org.jdesktop.layout.GroupLayout(jpSchema);
        jpSchema.setLayout(jpSchemaLayout);
        jpSchemaLayout.setHorizontalGroup(

jpSchemaLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jpSchemaLayout.createSequentialGroup()
                .addContainerGap()

.add(jpSchemaLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(jpSchemaLayout.createSequentialGroup()
                        .add(jScrollPane1,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 482, Short.MAX_VALUE)
                        .addContainerGap())
                    .add(jpSchemaLayout.createSequentialGroup()
                        .add(jbLoadSchema)
                        .add(288, 288, 288))))
        );
        jpSchemaLayout.setVerticalGroup(

jpSchemaLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jpSchemaLayout.createSequentialGroup()
                .addContainerGap()
                .add(jbLoadSchema)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
```

```
                .add(jScrollPane1, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
289, Short.MAX_VALUE)
                .addContainerGap())
        );
        jTabbedPane1.addTab("Schema", jpSchema);

        jbLoadData.setText("Load Data");
        jbLoadData.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jbLoadDataActionPerformed(evt);
            }
        });


jScrollPane2.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SC
ROLLBAR_ALWAYS);
        jScrollPane2.setPreferredSize(new java.awt.Dimension(400, 400));
        jTextAreaData.setEditable(false);
        jTextAreaData.setLineWrap(true);
        jTextAreaData.setRows(1);
        jTextAreaData.setWrapStyleWord(true);
        jTextAreaData.setMaximumSize(null);
        jTextAreaData.setPreferredSize(null);
        jScrollPane2.setViewportView(jTextAreaData);

        org.jdesktop.layout.GroupLayout jpDataLayout = new
org.jdesktop.layout.GroupLayout(jpData);
        jpData.setLayout(jpDataLayout);
        jpDataLayout.setHorizontalGroup(

jpDataLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jpDataLayout.createSequentialGroup()
                .addContainerGap()

.add(jpDataLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(jScrollPane2,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 482, Short.MAX_VALUE)
                    .add(jbLoadData))
                .addContainerGap())
        );
        jpDataLayout.setVerticalGroup(

jpDataLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jpDataLayout.createSequentialGroup()
                .addContainerGap()
                .add(jbLoadData)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jScrollPane2, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
289, Short.MAX_VALUE)
                .addContainerGap())
        );
        jTabbedPane1.addTab("Data", jpData);

        jbLoadRules.setLabel("Load Rules");
        jbLoadRules.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jbLoadRulesActionPerformed(evt);
            }
        });


jScrollPane3.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SC
ROLLBAR_ALWAYS);
        jScrollPane3.setPreferredSize(new java.awt.Dimension(400, 400));
        jTextAreaRules.setEditable(false);
        jTextAreaRules.setLineWrap(true);
        jTextAreaRules.setRows(1);
        jTextAreaRules.setWrapStyleWord(true);
```

```
        jTextAreaRules.setMaximumSize(null);
        jTextAreaRules.setPreferredSize(null);
        jScrollPane3.setViewportView(jTextAreaRules);

        org.jdesktop.layout.GroupLayout jpRulesLayout = new
org.jdesktop.layout.GroupLayout(jpRules);
        jpRules.setLayout(jpRulesLayout);
        jpRulesLayout.setHorizontalGroup(

jpRulesLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jpRulesLayout.createSequentialGroup()
                .addContainerGap()

.add(jpRulesLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(jpRulesLayout.createSequentialGroup()
                        .add(jScrollPane3,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 482, Short.MAX_VALUE)
                        .addContainerGap())
                    .add(jpRulesLayout.createSequentialGroup()
                        .add(jbLoadRules)
                        .add(298, 298, 298))))
        );
        jpRulesLayout.setVerticalGroup(

jpRulesLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jpRulesLayout.createSequentialGroup()
                .addContainerGap()
                .add(jbLoadRules)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jScrollPane3, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
289, Short.MAX_VALUE)
                .addContainerGap())
        );
        jTabbedPane1.addTab("Rules", jpRules);

        jpContext.setLayout(new java.awt.GridLayout(2, 1));


jScrollPane6.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SC
ROLLBAR_ALWAYS);
        jTextAreaContext.setLineWrap(true);
        jTextAreaContext.setRows(1);
        jTextAreaContext.setWrapStyleWord(true);
        jTextAreaContext.setMaximumSize(null);
        jTextAreaContext.setPreferredSize(new java.awt.Dimension(400, 400));
        jScrollPane6.setViewportView(jTextAreaContext);

        jbSendContext.setText("Send Context");
        jbSendContext.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jbSendContextActionPerformed(evt);
            }
        });

        jbLoadFileContext.setText("Load File");
        jbLoadFileContext.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jbLoadFileContextActionPerformed(evt);
            }
        });

        jLabel2.setText("Contextual Information:");

        org.jdesktop.layout.GroupLayout jPanel5Layout = new
org.jdesktop.layout.GroupLayout(jPanel5);
        jPanel5.setLayout(jPanel5Layout);
        jPanel5Layout.setHorizontalGroup(
```

```
jPanel5Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel5Layout.createSequentialGroup()
                .addContainerGap()

.add(jPanel5Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(jScrollPane6,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 482, Short.MAX_VALUE)
                    .add(jLabel2)
                    .add(jPanel5Layout.createSequentialGroup()
                        .add(jbLoadFileContext)
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                        .add(jbSendContext)))
                .addContainerGap())
        );
        jPanel5Layout.setVerticalGroup(

jPanel5Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel5Layout.createSequentialGroup()
                .addContainerGap()
                .add(jLabel2)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jScrollPane6, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
110, Short.MAX_VALUE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(jPanel5Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jbLoadFileContext)
                    .add(jbSendContext)))
        );
        jpContext.add(jPanel5);


jScrollPane9.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SC
ROLLBAR_ALWAYS);
        jTextAreaSubscription.setLineWrap(true);
        jTextAreaSubscription.setRows(1);
        jTextAreaSubscription.setWrapStyleWord(true);
        jTextAreaSubscription.setMaximumSize(null);
        jTextAreaSubscription.setPreferredSize(new java.awt.Dimension(400, 400));
        jScrollPane9.setViewportView(jTextAreaSubscription);

        jLabel3.setText("Subscription:");

        jbLoadFileSubscription.setText("Load File");
        jbLoadFileSubscription.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jbLoadFileSubscriptionActionPerformed(evt);
            }
        });

        jbSendSubscription.setText("Send Subscription");
        jbSendSubscription.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jbSendSubscriptionActionPerformed(evt);
            }
        });

        org.jdesktop.layout.GroupLayout jPanel6Layout = new
org.jdesktop.layout.GroupLayout(jPanel6);
        jPanel6.setLayout(jPanel6Layout);
        jPanel6Layout.setHorizontalGroup(

jPanel6Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel6Layout.createSequentialGroup()
                .addContainerGap()
```

```
.add(jPanel6Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(jScrollPane9,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 482, Short.MAX_VALUE)
                    .add(jLabel3)
                    .add(jPanel6Layout.createSequentialGroup()
                        .add(jbLoadFileSubscription)
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                        .add(jbSendSubscription)))
                .addContainerGap())
        );
        jPanel6Layout.setVerticalGroup(

jPanel6Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel6Layout.createSequentialGroup()
                .addContainerGap()
                .add(jLabel3)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jScrollPane9, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
99, Short.MAX_VALUE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(jPanel6Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jbLoadFileSubscription)
                    .add(jbSendSubscription))
                .addContainerGap())
        );
        jpContext.add(jPanel6);

        jTabbedPane1.addTab("Context", jpContext);

        jPanel2.setLayout(new java.awt.BorderLayout());

        jTextAreaQuery.setColumns(20);
        jTextAreaQuery.setRows(5);
        jScrollPane7.setViewportView(jTextAreaQuery);

        jButton1.setText("Execute");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jButton2.setText("SEND MESSAGE");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        org.jdesktop.layout.GroupLayout jPanel3Layout = new
org.jdesktop.layout.GroupLayout(jPanel3);
        jPanel3.setLayout(jPanel3Layout);
        jPanel3Layout.setHorizontalGroup(

jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel3Layout.createSequentialGroup()
                .addContainerGap()

.add(jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(jScrollPane7,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 482, Short.MAX_VALUE)
                    .add(jPanel3Layout.createSequentialGroup()
                        .add(jButton1)
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                        .add(jButton2)))
                .addContainerGap())
```

```
        );
        jPanel3Layout.setVerticalGroup(

jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel3Layout.createSequentialGroup()
                .addContainerGap()
                .add(jScrollPane7, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
96, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.add(jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jButton1)
                    .add(jButton2)))
        );
        jPanel2.add(jPanel3, java.awt.BorderLayout.NORTH);

        org.jdesktop.layout.GroupLayout jPanel4Layout = new
org.jdesktop.layout.GroupLayout(jPanel4);
        jPanel4.setLayout(jPanel4Layout);
        jPanel4Layout.setHorizontalGroup(

jPanel4Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel4Layout.createSequentialGroup()
                .addContainerGap()
                .add(jScrollPane8, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
482, Short.MAX_VALUE)
                .addContainerGap())
        );
        jPanel4Layout.setVerticalGroup(

jPanel4Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel4Layout.createSequentialGroup()
                .addContainerGap()
                .add(jScrollPane8, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
182, Short.MAX_VALUE)
                .addContainerGap())
        );
        jPanel2.add(jPanel4, java.awt.BorderLayout.CENTER);

        jTabbedPane1.addTab("Query", jPanel2);


jScrollPane5.setBorder(javax.swing.BorderFactory.createTitledBorder("Facts"));

jScrollPane5.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SC
ROLLBAR_ALWAYS);

jTextAreaFacts.setBackground(javax.swing.UIManager.getDefaults().getColor("Panel.ba
ckground"));
        jTextAreaFacts.setLineWrap(true);
        jTextAreaFacts.setRows(1);
        jTextAreaFacts.setWrapStyleWord(true);
        jTextAreaFacts.setFocusable(false);
        jTextAreaFacts.setMaximumSize(null);
        jTextAreaFacts.setPreferredSize(new java.awt.Dimension(400, 400));
        jScrollPane5.setViewportView(jTextAreaFacts);

        jbRefreshFacts.setText("Refresh");
        jbRefreshFacts.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jbRefreshFactsActionPerformed(evt);
            }
        });

        org.jdesktop.layout.GroupLayout jpFactsLayout = new
org.jdesktop.layout.GroupLayout(jpFacts);
        jpFacts.setLayout(jpFactsLayout);
```

```
        jpFactsLayout.setHorizontalGroup(

jpFactsLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jpFactsLayout.createSequentialGroup()
                .addContainerGap()

.add(jpFactsLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(jScrollPane5,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 482, Short.MAX_VALUE)
                    .add(jbRefreshFacts))
                .addContainerGap())
        );
        jpFactsLayout.setVerticalGroup(

jpFactsLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jpFactsLayout.createSequentialGroup()
                .addContainerGap()
                .add(jbRefreshFacts)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jScrollPane5, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
289, Short.MAX_VALUE)
                .addContainerGap())
        );
        jTabbedPane1.addTab("Facts", jpFacts);


jScrollPane4.setBorder(javax.swing.BorderFactory.createTitledBorder("Output"));

jScrollPane4.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SC
ROLLBAR_ALWAYS);
        jScrollPane4.setPreferredSize(new java.awt.Dimension(400, 400));

jTextAreaOutput.setBackground(javax.swing.UIManager.getDefaults().getColor("Panel.b
ackground"));
        jTextAreaOutput.setEditable(false);
        jTextAreaOutput.setLineWrap(true);
        jTextAreaOutput.setRows(1);
        jTextAreaOutput.setWrapStyleWord(true);
        jTextAreaOutput.setMaximumSize(null);
        jTextAreaOutput.setPreferredSize(null);
        jScrollPane4.setViewportView(jTextAreaOutput);

        org.jdesktop.layout.GroupLayout jpOutputLayout = new
org.jdesktop.layout.GroupLayout(jpOutput);
        jpOutput.setLayout(jpOutputLayout);
        jpOutputLayout.setHorizontalGroup(

jpOutputLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jpOutputLayout.createSequentialGroup()
                .addContainerGap()
                .add(jScrollPane4, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
482, Short.MAX_VALUE)
                .addContainerGap())
        );
        jpOutputLayout.setVerticalGroup(

jpOutputLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(org.jdesktop.layout.GroupLayout.TRAILING,
jpOutputLayout.createSequentialGroup()
                .addContainerGap()
                .add(jScrollPane4, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
318, Short.MAX_VALUE)
                .addContainerGap())
        );
        jTabbedPane1.addTab("Output", jpOutput);

        getContentPane().add(jTabbedPane1, java.awt.BorderLayout.CENTER);
```

```
        jPanel1.setLayout(new java.awt.BorderLayout());

        jPanel1.setPreferredSize(new java.awt.Dimension(10, 60));
        jLabel1.setBackground(new java.awt.Color(200, 221, 242));
        jLabel1.setFont(new java.awt.Font("Tahoma", 1, 14));
        jLabel1.setForeground(new java.awt.Color(70, 98, 153));
        jLabel1.setText("  Context Interpreter");
        jLabel1.setOpaque(true);
        jLabel1.setPreferredSize(new java.awt.Dimension(34, 30));
        jPanel1.add(jLabel1, java.awt.BorderLayout.NORTH);

        getContentPane().add(jPanel1, java.awt.BorderLayout.NORTH);

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jbSendSubscriptionActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jbSendSubscriptionActionPerformed
        interpreter.addContextMonitor(jTextAreaSubscription.getText());
    }//GEN-LAST:event_jbSendSubscriptionActionPerformed

    private void jbLoadFileSubscriptionActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_jbLoadFileSubscriptionActionPerformed
        if (jFileChooser1.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
            File file = jFileChooser1.getSelectedFile();
            StringBuffer strBuff = new StringBuffer();
            try {
                BufferedReader in = new BufferedReader(new FileReader(file));
                String str;
                while ((str = in.readLine()) != null) {
                    strBuff.append(str + "\n");
                }
                in.close();
            }
            catch (IOException e) {
                System.out.println(e);
            }
            jTextAreaSubscription.setText(strBuff.toString());
         }
    }//GEN-LAST:event_jbLoadFileSubscriptionActionPerformed

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton2ActionPerformed
        String servidorSocket = "Guarapari";
        String mensagem = "";
        int port = 10000;
        Socket socket = null;
        String lineToBeSent;
        BufferedReader input;
        PrintWriter output;
        int ERROR = 1;
        // onnect to server
        try {
            socket = new Socket(servidorSocket, port);
            System.out.println("Connected with server " +
                socket.getInetAddress() +
                ":" + socket.getPort());
        }
        catch (UnknownHostException e) {
            System.out.println(e);
            //System.exit(ERROR);
        }
        catch (IOException e) {
            System.out.println(e);
            //System.exit(ERROR);
        }
        try {
            input = new BufferedReader(new InputStreamReader(System.in));
```

```
            output = new PrintWriter(socket.getOutputStream(),true);
            lineToBeSent =
                    "<mensagem>" +
                    "  <paciente id=100>" +
                    "    <nome>rodrigo</nome>" +
                    "    <alerta>olá mundo</alerta>" +
                    "  </paciente>" +
                    "</mensagem>";
            output.println(lineToBeSent);
            System.out.println(lineToBeSent);
        }
        catch (IOException e) {
             System.out.println(e);
        }
        try {
            socket.close();
        }
        catch (IOException e) {
            System.out.println(e);
        }
    }//GEN-LAST:event_jButton2ActionPerformed

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton1ActionPerformed
        QueryResults results = interpreter.executeQuery(jTextAreaQuery.getText());
        Object[] columnNames = results.getResultVars().toArray();
        DefaultTableModel model = new DefaultTableModel(columnNames,0);
        javax.swing.JTable table = new javax.swing.JTable(model);
        jScrollPane8.setViewportView(table);

        while (results.hasNext()) {
            ResultBinding binding = (ResultBinding)results.next();
            Object[] line = new Object[columnNames.length];
            for (int i = 0; i < line.length; i++) {
                String str = columnNames[i].toString();
                Object objResult = binding.get(columnNames[i].toString());
                line[i] = objResult;
            }
            model.addRow(line);
        }
    }//GEN-LAST:event_jButton1ActionPerformed

    private void jbLoadFileContextActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jbLoadFileContextActionPerformed
        if (jFileChooser1.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
            File file = jFileChooser1.getSelectedFile();
            StringBuffer strBuff = new StringBuffer();
            try {
                BufferedReader in = new BufferedReader(new FileReader(file));
                String str;
                while ((str = in.readLine()) != null) {
                    strBuff.append(str + "\n");
                }
                in.close();
            }
            catch (IOException e) {
                System.out.println(e);
            }
            jTextAreaContext.setText(strBuff.toString());
         }
    }//GEN-LAST:event_jbLoadFileContextActionPerformed

    private void jbSendContextActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jbSendContextActionPerformed
        ParseSensorXML(jTextAreaContext.getText());
    }//GEN-LAST:event_jbSendContextActionPerformed
```

```
    private void jbRefreshFactsActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jbRefreshFactsActionPerformed
    String stmText = "";
    if (interpreter != null) {
        stmText = interpreter.getStatementsAsText();
    }
    jTextAreaFacts.setText(stmText);
    }//GEN-LAST:event_jbRefreshFactsActionPerformed

    private void jbLoadRulesActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jbLoadRulesActionPerformed
            if (jFileChooser1.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
{
                rulesFile = jFileChooser1.getSelectedFile().toString();
                File file = jFileChooser1.getSelectedFile();
                StringBuffer strBuff = new StringBuffer();
                try {
                    BufferedReader in = new BufferedReader(new FileReader(file));
                    String str;
                    while ((str = in.readLine()) != null) {
                        strBuff.append(str + "\n");
                    }
                    in.close();
                } catch (IOException e) {
                    System.out.println(e);
                }
                jTextAreaRules.setText(strBuff.toString());
            }
            if ((rulesFile != null)
                && (dataFile != null)
                && (schemaFile != null)) {
                interpreter = new Interpreter(schemaFile, dataFile, rulesFile);
            }

    }//GEN-LAST:event_jbLoadRulesActionPerformed

    private void jbLoadDataActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jbLoadDataActionPerformed
            if (jFileChooser1.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
{
                dataFile = jFileChooser1.getSelectedFile().toString();
                File file = jFileChooser1.getSelectedFile();
                StringBuffer strBuff = new StringBuffer();
                try {
                    BufferedReader in = new BufferedReader(new FileReader(file));
                    String str;
                    while ((str = in.readLine()) != null) {
                        strBuff.append(str + "\n");
                    }
                    in.close();
                } catch (IOException e) {
                    System.out.println(e);
                }
                jTextAreaData.setText(strBuff.toString());
            }
            if ((rulesFile != null)
                && (dataFile != null)
                && (schemaFile != null)) {
                interpreter = new Interpreter(schemaFile, dataFile, rulesFile);
            }
    }//GEN-LAST:event_jbLoadDataActionPerformed

    private void jbLoadSchemaActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jbLoadSchemaActionPerformed
            if (jFileChooser1.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
{
                schemaFile = jFileChooser1.getSelectedFile().toString();
                File file = jFileChooser1.getSelectedFile();
```

```
            StringBuffer strBuff = new StringBuffer();
            try {
                BufferedReader in = new BufferedReader(new FileReader(file));
                String str;
                while ((str = in.readLine()) != null) {
                    strBuff.append(str + "\n");
                }
                in.close();
            } catch (IOException e) {
                System.out.println(e);
            }
            jTextAreaSchema.setText(strBuff.toString());
        }
        if ((rulesFile != null)
            && (dataFile != null)
            && (schemaFile != null)) {
            interpreter = new Interpreter(schemaFile, dataFile, rulesFile);
        }
}//GEN-LAST:event_jbLoadSchemaActionPerformed

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new InterpreterShell().setVisible(true);
        }
    });
}

// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton2;
private javax.swing.JFileChooser jFileChooser1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JPanel jPanel3;
private javax.swing.JPanel jPanel4;
private javax.swing.JPanel jPanel5;
private javax.swing.JPanel jPanel6;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JScrollPane jScrollPane4;
private javax.swing.JScrollPane jScrollPane5;
private javax.swing.JScrollPane jScrollPane6;
private javax.swing.JScrollPane jScrollPane7;
private javax.swing.JScrollPane jScrollPane8;
private javax.swing.JScrollPane jScrollPane9;
private javax.swing.JTabbedPane jTabbedPane1;
private javax.swing.JTextArea jTextAreaContext;
private javax.swing.JTextArea jTextAreaData;
private javax.swing.JTextArea jTextAreaFacts;
private javax.swing.JTextArea jTextAreaOutput;
private javax.swing.JTextArea jTextAreaQuery;
private javax.swing.JTextArea jTextAreaRules;
private javax.swing.JTextArea jTextAreaSchema;
private javax.swing.JTextArea jTextAreaSubscription;
private javax.swing.JButton jbLoadData;
private javax.swing.JButton jbLoadFileContext;
private javax.swing.JButton jbLoadFileSubscription;
private javax.swing.JButton jbLoadRules;
private javax.swing.JButton jbLoadSchema;
private javax.swing.JButton jbRefreshFacts;
```

```
    private javax.swing.JButton jbSendContext;
    private javax.swing.JButton jbSendContext1;
    private javax.swing.JButton jbSendSubscription;
    private javax.swing.JPanel jpContext;
    private javax.swing.JPanel jpData;
    private javax.swing.JPanel jpFacts;
    private javax.swing.JPanel jpOutput;
    private javax.swing.JPanel jpRules;
    private javax.swing.JPanel jpSchema;
    private ContextInterpreter.Interpreter interpreter;
    private String schemaFile;
    private String dataFile;
    private String rulesFile;
}
```

Apêndice C: Ontologia de Turismo – OWL

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns="http://www.lprm.inf.ufes.br/TourismOntology.owl#"
  xml:base="http://www.lprm.inf.ufes.br/TourismOntology.owl">
  <owl:Ontology rdf:about="">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      A simple example ontology for tourism domain.</rdfs:comment>
    <owl:versionInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
     1.0 by Rodrigo Mantovaneli (rpessoa@inf.ufes.br)</owl:versionInfo>
  </owl:Ontology>

<owl:Class rdf:ID="Person">
 <rdfs:label>Person</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Tourist">
 <rdfs:label>Tourist</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>

<owl:Class rdf:ID="GeneralDevice">
 <rdfs:label>GeneralDevice</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="CellPhone">
 <rdfs:label>CellPhone</rdfs:label>
 <rdfs:subClassOf rdf:resource="#GeneralDevice"/>
</owl:Class>

<owl:Class rdf:ID="SmartPhone">
 <rdfs:label>CellPhone</rdfs:label>
 <rdfs:subClassOf rdf:resource="#GeneralDevice"/>
</owl:Class>

<owl:Class rdf:ID="Handheld">
 <rdfs:label>HandheldDevice</rdfs:label>
 <rdfs:subClassOf rdf:resource="#GeneralDevice"/>
</owl:Class>

<owl:Class rdf:ID="Place">
 <rdfs:label>Place</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="TouristAttraction">
 <rdfs:label>A place of interest where tourists normally visit.</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Place"/>
</owl:Class>

<owl:Class rdf:ID="Infrastructure">
  <rdfs:label>Infrastructure</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Place"/>
</owl:Class>
```

```
<owl:Class rdf:ID="Museum">
 <rdfs:label>Museum</rdfs:label>
 <rdfs:subClassOf rdf:resource="#TouristAttraction"/>
</owl:Class>

<owl:Class rdf:ID="Beach">
 <rdfs:label>Beach</rdfs:label>
 <rdfs:subClassOf rdf:resource="#TouristAttraction"/>
</owl:Class>

<owl:Class rdf:ID="Monument">
 <rdfs:label>Monument</rdfs:label>
 <rdfs:subClassOf rdf:resource="#TouristAttraction"/>
</owl:Class>

<owl:Class rdf:ID="Zoo">
 <rdfs:label>Zoo</rdfs:label>
 <rdfs:subClassOf rdf:resource="#TouristAttraction"/>
</owl:Class>

<owl:Class rdf:ID="Hotel">
 <rdfs:label>Hotel</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Infrastructure"/>
</owl:Class>

<owl:Class rdf:ID="Restaurant">
 <rdfs:label>Restaurant</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Infrastructure"/>
</owl:Class>

<owl:Class rdf:ID="Activity">
 <rdfs:label>Activity</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Nature">
 <rdfs:label>Nature</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Activity"/>
</owl:Class>

<owl:Class rdf:ID="Art">
 <rdfs:label>Art</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Activity"/>
</owl:Class>

<owl:Class rdf:ID="Sports">
 <rdfs:label>Surfing</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Activity"/>
</owl:Class>

<owl:Class rdf:ID="Location">
 <rdfs:label>Location</rdfs:label>
</owl:Class>

<owl:FunctionalProperty rdf:ID="isLocated">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 <rdfs:domain rdf:resource="#Person"/>
 <rdfs:range rdf:resource="#Location"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="hasLocation">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 <rdfs:domain rdf:resource="#Place"/>
 <rdfs:range rdf:resource="#Location"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="isInside">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 <rdfs:domain rdf:resource="#Person"/>
```

```
 <rdfs:range rdf:resource="#Place"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="hasDevice">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 <rdfs:domain rdf:resource="#Tourist" />
 <rdfs:range rdf:resource="#GeneralDevice" />
</owl:ObjectProperty>

<owl:InverseFunctionalProperty rdf:about="belongsTo">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#GeneralDevice"/>
  <rdfs:range rdf:resource="#Tourist"/>
  <owl:inverseOf rdf:resource="#hasDevice"/>
</owl:InverseFunctionalProperty>

<owl:FunctionalProperty rdf:ID="offersActivities">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 <rdfs:domain rdf:resource="#TouristAttraction"/>
 <rdfs:range rdf:resource="#Activity"/>
</owl:FunctionalProperty>

<owl:InverseFunctionalProperty rdf:about="isOfferedAt">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#Activity"/>
  <rdfs:range rdf:resource="#TouristAttraction"/>
  <owl:inverseOf rdf:resource="#offersActivities"/>
</owl:InverseFunctionalProperty>

<owl:FunctionalProperty rdf:ID="InterestedIn">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 <rdfs:domain rdf:resource="#Person"/>
 <rdfs:range rdf:resource="#Activity"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="hasLatitude">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty" />
  <rdfs:domain rdf:resource="#Location"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
</owl:FunctionalProperty>

<owl:DatatypeProperty rdf:ID="hasLongitude">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty" />
  <rdfs:domain rdf:resource="#Location"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="isInside">
 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 <rdfs:domain rdf:resource="#Person"/>
 <rdfs:range rdf:resource="#Place"/>
</owl:FunctionalProperty>

</rdf:RDF>
```