UNIVERSIDADE CATÓLICA DE PELOTAS
CENTRO POLITÉCNICO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

EXEHDA-DA: Controle da Adaptação Dinâmica ao Contexto para o middleware EXEHDA

por
Nelsi Warken

Relatório de Conclusão da Disciplina
Dissertação de Mestrado I

Orientador: Prof. Dr. Adenauer Corrêa Yamin

DMI-2009/1-009

Pelotas, julho de 2009

Dedico... este trabalho ao meu filho Júlio Warken Zabaleta, meu constante incentivador e a grande obra da minha vida.

AGRADECIMENTOS

Agradeço ao meu orientador e amigo, Adenauer Corrêa Yamin, sempre presente e com quem muito aprendo, tanto em sua linha de pesquisa, onde é um grande e reconhecido especialista, quanto na área das relações humanas, onde é uma pessoa extremamente gentil e amiga, harmonizando e qualificando, diariamente, todos seus ambientes de contexto.

Agradeço a Embrapa Clima Temperado, chefia e demais funcionários, pela grande oportunidade de passar por esta experiência tão gratificante, especial e renovadora, a realização do meu Curso de Mestrado em Ciência da Computação.

Agradeço também aos colegas, professores e funcionários do PPGINF e Laboratório de Informática pelo apoio, companheirismo, carinho, aprendizado e amizade. É um imenso prazer o convívio com grandes e bons amigos.

As tecnologias mais profundas e duradouras são aquelas que desaparecem. Elas dissipam-se nas coisas do dia a dia até tornarem-se indistinguíveis.

MARK WEISER

Você precisa ser a mudança que você quer ver no mundo.

GANDHI

SUMÁRIO

LISTA DE FIGURAS 7

LISTA DE TABELAS 8

LISTA DE ABREVIATURAS E SIGLAS 9

RESUMO 11

ABSTRACT 12

1 INTRODUÇÃO 13
1.1 Tema 14
1.2 Motivação 14
1.3 Objetivos 15
1.4 Estrutura do Texto 16

2 COMPUTAÇÃO UBÍQUA 18
2.1 Projetos de Computação Ubíqua 20

3 ONTOLOGIAS: PRINCIPAIS CONCEITOS 22
3.1 Categorias de Ontologias: 23
3.2 Tecnologias para Tratamento das Ontologias 24
3.2.1 XML 26
3.2.2 RDF e RDF Schema 26
3.2.3 OWL 27
3.2.4 API Jena 28
3.2.5 SPARQL 31
3.2.6 Protégé 31

4 COMPUTAÇÃO AUTÔNOMA 33
4.1 Elementos Autônomos 34
4.2 Políticas 35
4.3 Computação Ubíqua Autônoma 37

5 CONTROLE DA ADAPTAÇÃO NA UBICOMP 38
5.1 Principais Conceitos Associados 38
5.2 Projetos Relacionados ao Controle da Adaptação na UbiComp 42
5.2.1 Comparação entre os modelos 51

5.2.2 Considerações sobre os modelos 53

6 EXEHDA-DA: FUNDAMENTOS E VISÃO GERAL 55
6.1 Questões consideradas para elaboração da proposta 55
6.2 Características Envolvidas na Definição do EXEHDA-DA 58
6.3 Trabalhos Relacionados ao Modelo Proposto 60
6.4 Modelagem do EXEHDA-DA 62
6.4.1 Estrutura do Serviço de Adaptação Dinâmica Proposto 63
6.4.2 Estrutura do Modelo Semântico do EXEHDA-DA 66

7 PLANO DE TRABALHO 72
7.1 Etapas de Trabalho 72
7.2 Cronograma 74

8 CONSIDERAÇÕES FINAIS 75
8.1 Publicações Realizadas 77

REFERÊNCIAS 78

ANEXO A GLOSSÁRIO 84

LISTA DE FIGURAS

Figura 3.1 Estrutura Web Semântica proposta pela W3C 25
Figura 3.2 Características e funções nas camadas 26
Figura 3.3 Recuperação de Informações WS 29
Figura 3.4 Camadas da API Jena (DICKINSON, 2008) 29
Figura 3.5 Motor Inferência da API Jena (REYNOLDS, 2008) 30

Figura 4.1 Gerente e Elementos Autônomos (KEPHART; CHESS, 2003) 35

Figura 5.1 Premissas da Computação Ubíqua (YAMIN, 2004) 41

Figura 6.1 Responsabilidade pela Adaptação 56
Figura 6.2 Serviço de Adaptação no *Middleware* EXEHDA (YAMIN, 2004) . . 57
Figura 6.3 Adaptação da Figura Controle da Adaptação Multi-nível (YAMIN, 2004) 57
Figura 6.4 Relação entre os serviços do EXEHDA para o Controle da Adaptação 64
Figura 6.5 Visão Geral do EXEHDA-DA 64
Figura 6.6 Processo do Controle da Adaptação no EXEHDA-DA 66
Figura 6.7 Ontologia Básica do Contexto Ubíquo - OntUbi 68
Figura 6.8 Ontologia da Adaptação - ONTADAPT - Política da Aplicação para Adaptação (Classes) 69
Figura 6.9 Ontologia da Adaptação - ONTADAPT - Política da Aplicação para Adaptação (Relacionamentos) 69

LISTA DE TABELAS

Tabela 5.1 Comparativo: Características (1 a 12) e Modelos Context-aware . . . 52

LISTA DE ABREVIATURAS E SIGLAS

AMS	Arrythmia Monitoring System
API	Application Program Interface
ASCII	American Standard Code for Information Interchange
CONON	Context Ontology
DAML	DARPA Agent Markup Language
ECG	Eletrocardiograma
EXEHDA	Execution Environment for Highly Distributed Applications
FIPA	Foundation for Intelligent Physical Agents
GMLC	Gateway Mobile Location Center
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUMO	General User Model Ontology
G3PD	Grupo de Pesquisa em Processamento Paralelo e Distribuído
HP	Hewlett-Packard
HTTP	HyperText Transfer Protocol
ISAM	Infra-estrutura de Suporte às Aplicações Móveis Distribuídas
JAX-RPC	Java API for XML-based Remote Procedure Call
JCAF	Java Context Awareness Framework
JPEG	Joint Photographic Experts Group
LBS	Location Based Service
LDT	Location-Determining Technologies
OIL	Ontology Integration Language
OWL	Web Ontology Language
PC	Personal Computer
PDA	Personal Digital Assistants

PHP	Hypertext Preprocessor
PHS	Personal Handphone System
PoI	Point of Interest
QoS	Quality of Service
RDF	Resource Description Framework
RDFS	RDF Schema
RGB	Red, Green and Blue
RMI	Remote Method Invocation
SECAS	Simple Environment for Context Aware Systems
SGML	Standard Generalized Markup Language
SIG	Sistema de Informação Geográfica
SOCAM	Service-Oriented Context-Aware Middleware
SOAP	Simple Object Access Protocol
SOUPA	Standard Ontology for Ubiquitous and Pervasive Applications
SPARQL	Protocol and RDF Query Language
UbiComp	Ubiquitous Computing
UCPEL	Universidade Católica de Pelotas
UFPEL	Universidade Federal de Pelotas
UFRGS	Universidade Federal do Rio Grande do Sul
UFSM	Universidade Federal de Santa Maria
UDDI	Universal Description, Discovery and Integration
URI	Universal Resource Identifier
URL	Universal Resource Locator
WSDL	Web Services Description Language
W3C	World Wide Web Consortium
XHTML	eXtensible Hypertext Markup Language
XML	eXtensible Markup Language

RESUMO

O objetivo central deste trabalho é avaliar o emprego dos conceitos e tecnologias referentes a Web Semântica e Sistemas Autônomos na concepção de mecanismos de controle da adaptação ao contexto na Computação Ubíqua, considerando suas principais características e desafios de pesquisa. Na computação ubíqua, os diversos sistemas computacionais interagem com o ser humano a todo o momento, não importando onde ele esteja, constituindo um ambiente altamente distribuído, heterogêneo, dinâmico, móvel e de composição mutável. As aplicações deste ambiente devem ser adaptáveis, considerando o contexto em que estão inseridas. Ontologias são especificações formais dos conceitos de um determinado domínio, constituindo o núcleo da Web Semântica. Aplicações semânticas buscam interpretar o significado dos dados, permitindo estabelecer relacionamentos e inferências entre os mesmos. A Computação Autônoma indica que os sistemas computacionais deveriam desempenhar funções automáticas de configuração, tratamento, otimização e proteção, substituindo tarefas complexas do usuário por políticas descritas em alto nível por administradores e/ou programadores. Considerando todos estes aspectos, a pesquisa desenvolvida nesta dissertação de mestrado tem como eixo conceber uma proposta para controlar as adaptações quando da tomada de decisões, considerando o contexto, produzido por informações monitoradas, informações semânticas e inferências a partir destas mesmas informações. A premissa é culminar em um mecanismo de adaptação genérico, que poderá ser utilizado tanto pelo *middleware*, quanto por diferentes aplicações, em tempo de execução. Este mecanismo que está sendo concebido chama-se EXEHDA-DA, EXEHDA-*Dynamic Adaptation*, serviço de controle da adaptação dinâmica ao contexto para o *middleware* EXEHDA.

Palavras-chave: Computação ubíqua, computação pervasiva, adaptação ao contexto, controle da adaptação, gerência da adaptação, adaptação dinâmica, aplicações adaptáveis ao contexto.

TITLE: “EXEHDA-DA: DYNAMIC ADAPTATION CONTROL TO THE CONTEXT FOR THE MIDDLEWARE EXEHDA”

ABSTRACT

The main purpose of this study is to assess the use of concepts and technologies for the Semantic Web and Autonomics Systems in the design of mechanisms to control the adaptation to the context in Ubiquitous Computing, considering its main characteristics and research challenges. In ubiquitous computing, the various computational systems interact with human beings all the time, no matter where you are, providing a highly distributed environment, heterogeneous, dynamic, mobile and changing composition. The applications of this environment must be adaptable, considering the context in which they are embedded. Ontologies are formal specifications of concepts of a given field, forming the core of the Semantic Web. Applications seek to interpret the semantic meaning of the data, allowing establish relationships between data and inferences. The Autonomic Computing indicates that computing systems should perform automatic configuration, treatment, protection and optimization, replacing the user's complex tasks for high-level policies described by administrators and/or programmers. Considering all these aspects, the proposed research has as focus to concept a propose to control the adjustments when making decisions, considering the context, generated by tracked information, semantic information and inferences from these same information. The premisse is a generic mechanism for adaptation that can be used in run-time, both by middleware or by various applications. This mechanism being designed is called EXEHDA-DA, EXEHDA-Dynamic Adaptation, service of dynamic adaptation control to the context for the middleware EXEHDA.

Keywords: ubiquitous computing, pervasive computing, contex-aware adaptations, management adaptations, adaptations control, dynamic adaptation, context-aware applications.

1 INTRODUÇÃO

Apresentaremos, nesta breve introdução, o tema escolhido para o desenvolvimento do trabalho, assim como, as motivações para escolha do foco do trabalho, os objetivos e metas considerados como premissas no andamento deste trabalho.

As tecnologias mais profundas são aquelas que desaparecem, elas se integram na vida cotidiana até se tornarem indistinguíveis da mesma (WEISER, 1991). Esta frase do clássico artigo sobre a Computação para o século 21, sintetiza um pouco do que é esperado com a Computação Ubíqua. O termo trata do acesso ao ambiente computacional do usuário, isto é, ao espaço ubíquo do usuário, em qualquer lugar, todo o tempo com qualquer dispositivo. A computação e seus diversos sistemas interagem com o ser humano a todo o momento, não importando onde ele esteja, em casa, no trabalho e na rua, constituindo um ambiente altamente distribuído, heterogêneo, dinâmico, móvel, mutável e com forte interação entre homem e máquina (COSTA; YAMIN; GEYER, 2008).

As aplicações precisam entender e se adaptar ao ambiente, compreendendo o contexto em que estão inseridas (MACIEL; ASSIS, 2004). Essa nova classe de sistemas computacionais, adaptativos ao contexto, abre perspectivas para o desenvolvimento de aplicações mais ricas, elaboradas e complexas, que exploram a natureza dinâmica e a mobilidade do usuário. Entretanto, o desenvolvimento de aplicações que se adaptem continuamente ao ambiente e permaneçam funcionando mesmo quando o indivíduo se movimentar ou trocar de dispositivo (GRIMM; BERSHAD, 2003), continua um desafio de pesquisa em aberto. Ainda existem limitações para o desenvolvimento de tais softwares, pois poucas linguagens e ferramentas estão disponíveis para a programação de aplicações adaptáveis às mudanças de contexto. (WANT; PERING, 2005).

Contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é qualquer pessoa, lugar ou objeto que é considerado relevante para a interação entre usuário e a aplicação, incluindo o próprio usuário e a própria aplicação. Esta visão foi introduzida na publicação (DEY; ABOWD; SALBER, 2001).

Através da adaptação é feita a avaliação de elementos do contexto, como localização, tempo e atividades do usuário, permitindo o desenvolvimento de aplicações sensíveis ao mesmo (HENRICKSEN; INDULSKA; RAKOTONIRAINY, 2006).

O G3PD, grupo de pesquisa no qual esta dissertação se encontra em desenvolvimento, participa de um consórcio de pesquisa formado por diversas universidades do Rio Grande do Sul. O G3PD participa do desenvolvimento do *middleware* EXEHDA, *Execution Environment for Highly Distributed Applications*. O EXEHDA é um *middleware* adaptativo ao contexto, baseado em serviços, que tem por objetivo criar e gerenciar um

ambiente ubíquo. As aplicações deste ambiente são distribuídas, móveis e adaptativas ao contexto.

1.1 Tema

O tema deste trabalho é o controle da adaptação dinâmica ao contexto, considerando os aspectos introduzidos pela Computação Ubíqua e as demandas do *middleware* EXEHDA.

1.2 Motivação

Computação Ubíqua é um novo paradigma e uma tendência mundial, em função do rápido avanço tecnológico dos dispositivos móveis, redes sem fio, grades computacionais, *Web Services*, *Web* Semântica, Sistemas Autônomos e outras tecnologias relacionadas a integração de sistemas. São assuntos modernos e que vão se interligando naturalmente. A adaptação ao Contexto torna-se um componente importante e fundamental na Computação Ubíqua.

Aplicações que se adaptam ao espaço ubíquo tornam a vida do usuário mais confortável e, por consequência, melhoram a sua qualidade de vida, podendo fazer com que suas necessidades, sejam atendidas com mais facilidade e, muitas vezes até, sem que ele mesmo tenha que fazer a solicitação. A computação ubíqua é centrada no usuário e é reconhecida como um novo paradigma, decorrente do rápido avanço tecnológico dos dispositivos móveis, redes sem fio, redes de sensores, dispositivos e sistemas inteligentes, sistemas distribuídos, grades computacionais, *Web Services* e *Web* Semântica. Nesta visão, o Controle da Adaptação ao contexto é essencial para os Sistemas Ubíquos e tem ganho espaço na literatura especializada (COSTA; YAMIN; GEYER, 2008).

Ainda não temos conhecimento de linguagens de programação para aplicações que se adaptem ao espaço ubíquo ou de modelos para desenvolvimento, de maneira genérica, de aplicações adaptáveis ao contexto. Prover soluções adaptáveis a esses contextos tão diversos, que facilite o desenvolvimento de aplicações baseado no reuso, constitui um grande desafio para a Engenharia de Software.

Assim, motivada por estas idéias, este trabalho propõe-se a explorar o uso de Ontologias, Serviços Web e Sistemas Autônomos para Controle da Adaptação ao Contexto na Computação Ubíqua, facilitando e otimizando o desenvolvimento de aplicações *context-aware*.

Ontologia, além de ser um elemento importante da Web Semântica, evolução da Web atual que é centrada em aspectos sintáticos, é um bom instrumento para especificar os complexos conceitos da computação ubíqua, seu contexto (entidades ou elementos de contexto como usuários, dispositivos, serviços, localização, códigos por dispositivos e serviços, entre outros), atuando como um padrão para descrições, características, configurações e perfis, que poderão ser compartilhados pelas aplicações.

A Computação Ubíqua idealiza a criação de ambientes carregados de dispositivos computacionais que se integram a vida das pessoas de forma transparente, adaptando-se automaticamente de acordo com as situações observadas no contexto em que estão inseridos. A Computação Autônoma, também denominada Computação Autonômica, indica que os sistemas computacionais deveriam desempenhar funções automáticas de

configuração, tratamento, otimização e proteção, substituindo tarefas complexas por políticas descritas em alto nível por usuários e administradores.

Apesar dos dois paradigmas apresentarem focos distintos, é possível perceber que eles possuem características em comum. Por exemplo, a capacidade de adaptação ao contexto de forma transparente para os usuários está presente nos Sistemas de Computação Ubíqua e de Computação Autônoma. Com isso, alguns autores já definem o termo de Computação Ubíqua Autônoma, nos casos em que as tecnologias desenvolvidas apresentam características dos dois paradigmas (O'DONNELL; LEWIS; WADE, 2005; RANGANATHAN; CAMPBELL, 2004).

Pode-se tornar uma tendência natural, a utilização combinada das áreas de ontologias e computação ubíqua e sistemas autônomos, pelo avanço rápido tanto da Web Semântica como dos ambientes ubíquos, além de facilitar a análise e a reutilização destas novas aplicações e a integração entre estas. A padronização para o desenvolvimento de aplicações ubíquas poder ser interessante e desejável, em função das demandas crescentes dos usuários (HILERA; RUIZ, 2006).

Outra motivação é a possibilidade de sugerir alternativas para o mecanismo de controle de adaptação, funcional e não funcional, para o *middleware* EXEHDA, foco atual de desenvolvimento do G3PD.

1.3 Objetivos

O objetivo central do trabalho é avaliar a exploração das tecnologias de Web Semântica e Sistemas Autônomos como mecanismos de controle da adaptação ao contexto na computação ubíqua, considerando as principais características e desafios de pesquisa da mesma. Será criado um modelo ontológico para o ambiente computacional provido pelo *middleware* EXEHDA, e a proposta é tomar decisões automáticas de adaptação para este ambiente, com base em informações monitoradas, informações semânticas e inferências a partir das mesmas. Assim, a proposta do trabalho é criar um modelo de controle da adaptação dinâmica de aplicações em ambiente ubíquo.

Como objetivos específicos a serem buscados neste modelo de controle, destacaríamos que o mesmo:

- tome decisões automáticas, sem intervenção do usuário, de acordo com seu perfil e preferências;
- tenha adaptação dinâmica à mudança de contexto;
- possa encontrar o melhor estado adaptativo, quando uma mudança ocorrer em seu contexto;
- possua um mecanismo para a adaptação que possa ser reusável e customizada para diferentes demandas de aplicações.

A idéia é controlar estas adaptações quando da tomada de decisões considerando o contexto, com base em informações monitoradas, informações semânticas e inferências a partir destas mesmas informações. A sugestão consta de um mecanismo de adaptação genérico que será utilizado por todas aplicações, em tempo de execução. Neste modelo

podemos ter, no repositório de adaptação, todas as regras, políticas e ações globais e ações específicas de cada aplicação, utilizadas pelo servidor de adaptação, para a partir destes dados e das mudanças do contexto, inferir o que e onde a ação de adaptação deverá ser executada. Para que a aplicação seja de alto nível, genérica e com bom desempenho, pode-se pensar em permitir uma inserção no modelo de políticas de adaptação (regras, ações e restrições), por parte do desenvolvedor da aplicação, utilizando a meta-programação. O modelo de controle de adaptação ao contexto proposto pretende possibilitar uma evolução incremental das especificações de políticas, regras e ações de adaptação. Permitindo a reutilização e a customização destas para o desenvolvimento de novas aplicações *context-aware*. Além disso, deverá possuir mecanismos para administrar situações de conflito quando da tomada de decisões do servidor de adaptação, perseguindo a premissa de adaptar os serviços da computação ubíqua ao ambiente sem, ou com mínima, intervenção explícita do usuário.

1.4 Estrutura do Texto

O texto é composto por sete capítulos.

No capítulo 1 encontra-se a introdução, onde é mostrado o tema do trabalho, a motivação para escolha do tema e os objetivos a serem perseguidos no andamento deste trabalho.

No capítulo 2 é descrito a Computação Ubíqua com as características que podem ser encontradas em seu cenário, projetos em Computação Ubíqua e os conceitos relacionados ao tema.

No capítulo 3 são apresentados os fundamentos teóricos sobre ontologias com seus principais conceitos, categorias e as tecnologias para tratamento das ontologias, mostrando as principais linguagens envolvidas.

No capítulo 4 são apresentados os conceitos da Computação Autônoma, dos elementos autônomos, das políticas envolvidas e o novo paradigma, intitulado Computação Ubíqua Autônoma.

No capítulo 5 é apresentado o estudo realizado sobre o foco principal do trabalho, o controle da adaptação na Computação Ubíqua. Inicialmente são apresentados os conceitos associados a este assunto. Uma breve descrição de 20 trabalhos relacionados ao controle da adaptação, que foram estudados. Foi elaborada uma comparação entre os modelos, considerando 12 aspectos importantes na Computação Ubíqua. Após são apresentadas as considerações a respeito dos modelos e das características consideradas.

No capítulo 6, EXEHDA-DA: Fundamentos e Visão Geral, são identificadas as questões a serem consideradas e resolvidas para as aplicações adaptáveis ao contexto e para o EXEHDA, *middleware* para o qual será sugerida a inserção do mecanismo proposto. Neste capítulo é definido o modelo de controle de adaptação dinâmica com todas as suas características, comparando-o com outros trabalhos de adaptação ao contexto. Também é mostrada a modelagem do serviço com a estrutura do serviço de adaptação dinâmica para o EXEHDA, bem como a estrutura inicial do modelo semântico proposto.

No capítulo 7, é descrito a metodologia utilizada para elaboração da dissertação de mestrado com todas as etapas do trabalho e o respectivo cronograma de atividades.

No capítulo 8, são feitas considerações finais a respeito deste texto e são relacionadas as publicações feitas no decorrer do desenvolvimento deste trabalho.

Como anexo A, temos o glossário com as definições e conceitos dos termos con-

siderados mais importantes no estudo da Computação Ubíqua, e que foram empregados no estudo realizado até agora.

2 COMPUTAÇÃO UBÍQUA

Este capítulo apresenta um resumo da fundamentação teórica, aspectos históricos, funcionalidades e exemplos relativos a Computação Ubíqua. Também apresenta algumas características encontradas em um cenário ubíquo, bem como relaciona alguns projetos de Computação Ubíqua.

O termo Computação Ubíqua, foi definido pela primeira vez pelo cientista Mark Weiser, que no início dos anos 90, já previa um aumento nas funcionalidades e na disponibilidade de serviços de computação para os usuários finais, entretanto com a visibilidade destes serviços sendo a menor possível. A computação não seria exclusividade de um computador, mas de diversos dispositivos conectados entre si.

A Computação Ubíqua neste contexto não significa um computador que possa ser transportado para a praia, o campo ou o aeroporto. Mesmo o mais poderoso *notebook*, com acesso a Internet ainda foca a atenção do usuário num simples equipamento. Comparando à escrita, carregar um super *notebook* é como carregar um livro muito importante. Personalizar este livro, mesmo escrevendo milhões de outros livros, não significa capturar o real poder da Literatura (WEISER, 1991).

Há alguns milhares de anos atrás, as antigas civilizações da Suméria, Babilônia, Assíria e Pérsia criaram um mecanismo para representar pensamentos na forma de símbolos abstratos; nascia a escrita moderna. Conhecimento que durante muito tempo foi acessível exclusivamente aos mais conceituados especialistas das letras, hoje está integralmente imerso em nosso cotidiano e imperceptivelmente, sendo uma tecnologia consumida em larga escala.

A concretização do pensamento de Weiser, e consequentemente a ubiquidade da informática, terá atingido sua plenitude quando a computação for aplicada com a mesma naturalidade que hoje é utilizada a língua escrita e os motores, agora elétricos e embarcados, para realização de atividades do cotidiano, ou seja, a comunidade terá alcançado a era da computação ubíqua quando o computador deixar de ser peça única e de uso genérico e multiplicar-se para usos especializados.

O computador é uma interface de comunicação com um mundo cibernético e, como todas as interfaces de sucesso, ele tende a desaparecer. Por exemplo, os óculos são uma interface entre o ser humano e o mundo mas, o ser humano não concentra suas atenções nos óculos, mas no mundo. Uma pessoa cega, ao utilizar sua bengala, percebe o mundo ao seu redor, e não a bengala. Nestes casos, as interfaces passam a tornar-se inconscientes (WEISER, 1993).

A Computação Ubíqua, também conhecida como Computação Invisível (NORMAN, 1998), é definida como sendo o terceiro grande paradigma computacional, pre-

cedido pelo império dos mainframes e pela onda da computação pessoal (WEISER; BROWN, 1997). Com foco em demandas administrativas no passado e hoje presente em cenários específicos, os computadores de grande porte foram definidos sobre uma arquitetura onde, poucas máquinas de imenso poder de processamento, são compartilhadas simultaneamente por inúmeros usuários. Com o domínio dos microcomputadores, observamos a máquina como peça íntima de uso pessoal e exclusivo. Na Computação Ubíqua, veremos a tecnologia discretamente nos envolvendo e agindo nos bastidores de nossas vidas, formando uma malha de dispositivos inteligentes em torno de cada indivíduo, sem contudo impactar profundamente nosso modo de viver.

A estratégia de empregar mobilidade para prover computação a qualquer momento e em qualquer lugar é considerada uma abordagem reativa ao acesso da informação, e prepara para a pró-atividade da computação ubíqua: todo tempo, em todo lugar (SAHA; MUKHERJEE, 2003). Surge o termo *everywhere* (em todo lugar) para denominar a nova classe de software necessária para a computação ubíqua (CAHILL et al., 2003). Todavia, ainda existem muitas limitações para o desenvolvimento de tais softwares, pois poucas linguagens e ferramentas estão disponíveis para a programação (WANT; PERING, 2005).

O panorama de aplicações sensíveis ao contexto deve prever a mobilidade de equipamentos e usuários, denominada mobilidade física, e também dos componentes da aplicação e serviços, chamada de mobilidade lógica. Para isso, as aplicações devem ter o estilo siga-me, facultando que o usuário possa acessar seu ambiente computacional independente da localização, do tempo e do dispositivo utilizado (YAMIN, 2004; YAMIN et al., 2005a).

Os sistemas ubíquos nascem com o intuito de estarem presentes a todo o momento nos ambientes físicos e computacionais em que estiverem envolvidos. Para tal, as aplicações precisam ”entender” e se adaptar ao ambiente, compreender o contexto em que estão inseridas (MACIEL; ASSIS, 2004). Essa nova classe de sistemas computacionais, sensíveis ao contexto, abre perspectivas para o desenvolvimento de aplicações muito mais ricas, elaboradas e complexas, que exploram a natureza dinâmica e a mobilidade do usuário. A dificuldade se encontra no desenvolvimento de aplicações que se adaptem continuamente ao ambiente e permaneçam funcionando mesmo quando o indivíduo se movimentar ou trocar de dispositivo (GRIMM; BERSHAD, 2003).

A Computação Ubíqua trata do acesso ao ambiente computacional do usuário, em qualquer lugar, todo o tempo com qualquer dispositivo, constituindo um ambiente altamente distribuído, heterogêneo, dinâmico, móvel, mutável e com forte interação entre homem e máquina (AUGUSTIN, 2003).

Para ilustrar um cenário de aplicação da computação ubíqua, podemos utilizar o exemplo: (SATYANARAYANAN, 2001)

Luciana está no portão 5 do Aeroporto Internacional de Brasília a espera de seu vôo para Patos de Minas, MG. Neste período, pelo *notebook*, ela revisa sua dissertação de mestrado pois necessita enviá-la por *e-mail*, através da rede *wireless* do aeroporto, ao seu orientador para correções. Entretanto, devido à limitação da largura de banda disponível naquele instante, causada pela alta taxa de utilização da rede no perímetro do portão 5, o envio imediato do documento de Luciana, que contém vários *megabytes* de texto, imagens e planilhas, é inviabilizado, devido à iminência de seu embarque. Em um ambiente ubíquo, como o do projeto Aura da Carnegie Mellon University (GARLAN, 2001), seria possível verificar a impossibilidade de transferência completa do documento de Luciana antes da partida de seu vôo e, através de observações na programação de decolagens do

aeroporto e avaliações climáticas e de tráfego aéreo para previsão de atrasos, orientar a usuária a dirigir-se ao portão mais próximo cujo fluxo na rede não impeça a total conclusão de suas atividades. Esta pró-atividade do sistema agrega conforto e segurança à usuária, na medida em que garante o cumprimento de seus compromissos e proporciona-lhe maior comodidade.

Ainda existe muita discussão e controvérsia nos termos Computação Pervasiva (*Pervasive Computing*) e Computação Ubíqua (*Ubiquitous Computing*). O termo *pervasivo* não existe na língua Portuguesa e em inglês significa espalhado, integrado, universal. O termo *ubíquo* significa onipresente. Os dois termos também são tratados sem distinção por alguns pesquisadores. Neste trabalho, para fins de padronização, será adotado Computação Ubíqua ou simplesmente *UbiComp*. Neste contexto, o usuário desloca-se e comunica-se com os recursos computacionais do ambiente em que se encontra para configuração dinâmica de novos serviços, de tal forma que as necessidades do usuário sejam satisfeitas, da maneira mais natural e transparente possíveis.

2.1 Projetos de Computação Ubíqua

A seguir são relacionados alguns projetos de grande porte para Computação Ubíqua, que proporcionam uma infra-estrutura base para a concepção de aplicações para ambientes ubíquos, integrando várias funcionalidades.

Projeto Aura (GARLAN, 2001): gira em torno da manutenção da lista de tarefas a serem executadas diariamente pelos usuários da *UbiComp*. O projeto Aura é uma proposta que visa projetar, implementar, empregar e avaliar sistemas de larga escala para demonstrarem o conceito de **aura de informação pessoal** que se espalha pelas diversas infra-estruturas computacionais. É um projeto com diversas frentes que investiga novas arquiteturas para o ambiente ubíquo, com premissas básicas em pró-atividade e auto-ajuste do sistema às necessidades do usuário. O projeto tem seu foco no usuário, suas tarefas e preferências, prevê a ênfase na dimensão pessoal.

Projeto Gaia (ROMAN; AL., 2002): desenvolvimento de uma infra-estrutura base para a construção de aplicativos, *tratando dos Espaços Ativos*, os quais traduzem uma visão de futuro onde o espaço habitado pelas pessoas é interativo e programável. Os usuários interagem com seus escritórios, casa, carros, para requisitar informações, beneficiar-se dos recursos disponíveis e configurar o comportamento de seu habitat. Dados e tarefas estão acessíveis e são mapeados dinamicamente para os recursos convenientes e presentes na localização corrente. Este ambiente interativo, centrado no usuário, requer uma infra-estrutura de software para operar com os recursos, observar propriedades do contexto, assistir o desenvolvimento e execução das aplicações. *Middleware* usado para prototipar gerenciamento de recursos e fornecer uma interface orientada ao usuário.

Projeto *EasyLiving* (BRUMITT et al., 2000): este projeto define um conjunto de tecnologias destinadas à integração dinâmica de dispositivos para o enriquecimento da experiência do usuário na utilização do espaço. Seu objetivo é permitir, através de uma arquitetura pré-estabelecida, que mais dispositivos possam se integrar ao sistema de elementos que relaciona o usuário com o ambiente inteligente que o abriga. Utiliza

integração dinâmica de dispositivos para o enriquecimento da experiência do usuário. A estrutura do projeto é composta por *middleware*, modelagem geométrica, sensoriamento e descrição de serviços.

Projeto *Gator Tech* (HELAL et al., 2005): *Gator Tech Smart House* é um projeto de uma casa inteligente conduzido pelo Laboratório de Computação Móvel e Pervasiva em colaboração com o *College of Public Health and Health Professions* da Universidade da Flórida. Seu objetivo central é a criação de ambientes assistidos computacionalmente, capazes de reação com base no monitoramento de seu estado e de seus habitantes. Propõe ambientes que reajam à alteração do comportamento de seus integrantes. Utiliza um *middleware* programável, onde a arquitetura é composta por módulos executáveis e por bibliotecas de programação, facilitando a evolução do sistema.

Projeto ISAM (YAMIN et al., 2005b): Infra-estrutura de Suporte às Aplicações Móveis Distribuídas, desenvolvido pela UFRGS. Como uma de suas atividades, o projeto desenvolve o **EXEHDA**: *Execution Environment for Highly Distributed Applications* que é um *middleware* direcionado às aplicações distribuídas, móveis e conscientes do contexto da computação ubíqua (YAMIN, 2004). Este *middleware* é baseado em serviços, que tem por objetivo criar e gerenciar um ambiente ubíquo, onde as aplicações são distribuídas, móveis e adaptativas ao contexto. Seus serviços são organizados em quatro grandes subsistemas: execução distribuída, adaptação, comunicação e acesso ubíquo.

A ausência de foco de um ou outro projeto em determinado atributo não configura sua deficiência naquele item. Isto é, o fato de o projeto Aura não focar na viabilização de uma biblioteca de desenvolvimento, não necessariamente implica que ele não possua esta infra-estrutura de programação, mas sim que este não é o foco de sua pesquisa.

Através da evolução dos Sistemas de Informação Distribuídos (SID), percebido inicialmente com o desenvolvimento da Internet, e a ampliação das opções de conexões, verifica-se que a Computação Ubíqua já é realidade, comprovado pelos benefícios que a Computação Móvel trouxe aos usuários. Celulares com acesso à *Web, Laptops*, Redes *WI-FI*, Lousas Digitais, *iPods* e o *iPhone*, permitem ao mais leigo, sem perceber, a utilização a qualquer momento e em qualquer lugar de um sistema de computação, através de um *software* e/ou uma interface.

Neste capítulo foram abordados aspectos sobre a Computação Ubíqua, sua história, conceitos e características considerados mais importantes, assim como, alguns de seus projetos.

Na continuidade serão explorados aspectos referentes a Web Semântica: seus principais conceitos, sua classificação em categorias e as tecnologias e linguagens mais importantes.

3 ONTOLOGIAS: PRINCIPAIS CONCEITOS

Este capítulo apresenta as tecnologias de Web Semântica, com destaque em Ontologias. São apresentados seus principais conceitos, sua classificação em categorias, alguns projetos de ontologias, e as ferramentas e linguagens consideradas mais importantes para o tratamento das ontologias.

A definição mais aceita e citada pelos autores da área de Computação é a que define ontologia como uma especificação formal e explícita de uma conceituação compartilhada (FENSEL, 2000), onde:

- conceituação - se refere ao modelo abstrato do mundo real;
- explícita - significa que os conceitos e seus requisitos são definidos explicitamente, definições de conceitos, instâncias, relações, restrições e axiomas;
- formal - indica que a ontologia é processável por máquina, permite raciocínio automático e possui semântica lógica formal;
- compartilhada - significa que uma ontologia captura o conhecimento apresentado não apenas por um único indivíduo, mas por um grupo, sendo uma terminologia comum da área modelada ou acordada entre os desenvolvedores.

Uma ontologia não se resume somente a um vocabulário, também possui relacionamentos e restrições (axiomas) entre os conceitos definidos pelo vocabulário e, através de regras de inferência, é possível derivar novos fatos baseando-se em fatos existentes.

Ontologias constituem o núcleo da Web Semântica. Aplicações Semânticas buscam interpretar o significado de textos e outros formatos de dados, permitindo estabelecer relacionamentos e inferências entre os dados. A reutilização e a conectividade são características fundamentais para tais aplicações.

Web Semântica (Web 3.0) se caracteriza pela análise de páginas da Web indexados por motores de busca, que permitem anotar documentos e recursos na Web semântica com informações baseadas em ontologias (BERNERS-LEE; HENDLER; LASSILA, 2001). A Web 3.0 também conhecida como Web Ubíqua, possibilita o acesso às páginas em qualquer lugar ou a qualquer momento, em dispositivos diversos: roupas, eletrodomésticos, móveis. O conceito está relacionado com computação ubíqua, que é o modelo onde a ação computacional é invisível e onipresente, introduzida em nosso ambiente e no cotidiano.

Ontologia pode ser considerado um bom instrumento, embora não o único, para especificar os conceitos da computação ubíqua, além de ser o elemento central na emergente Web Semântica, novo paradigma como evolução da atual Web (Web Sintática). Web Semântica se caracteriza pela análise de páginas da Web indexados por motores de busca, que permitindo anotar documentos e recursos na Web semântica com informações baseadas em ontologias (BERNERS-LEE; HENDLER; LASSILA, 2001).

3.1 Categorias de Ontologias:

Apresentamos algumas classificações de ontologias, as quais foram as mais utilizadas nos trabalhos estudados referentes à Computação Ubíqua.

Ontologias de domínio: genéricas ou específicas, descrevem conhecimento sobre um domínio ou sub-domínio.

Ontologias como Artefatos de Software: usadas como artefatos de diversos tipos no processo de desenvolvimento de aplicações ou durante a execução de uma aplicação. (HILERA; RUIZ, 2006)

A categoria genérica refere-se a ontologias cujo principal objetivo é o de representar o conhecimento de um determinado domínio ou de um contexto de interesse, sub-domínio. A existência de uma ontologia para conceituar plenamente um domínio ou numa área de conhecimento, poderia auxiliar na anotação dos recursos e localização, por exemplo, permitindo evitar as ambiguidades e contradições, que podem ser produzidas quando utilizamos diversos termos e conceitos.

Podemos observar alguns projetos abaixo, específicos de ontologias para a Computação Ubíqua:

- **SOUPA** (*Standard Ontology for Ubiquitous and Pervasive Applications*) (CHEN et al., 2004): combina vocabulários de uso comum de diferentes ontologias. SOUPA inclui conceitos como *Agent*, que representa os usuários com as propriedades como crenças, desejos, intenções ou planos; *Action*, *Time*, *Device*, ou *Local*. Esta ontologia foi criada utilizando a linguagem OWL que é usada para representação de conhecimento. SOUPA é composta de um conjunto de ontologias, porém estas não são completamente importadas, apenas parte do vocabulário destas é que é utilizado. Esta combinação de ontologias é um dos pontos fortes existentes, pois desta maneira SOUPA pode fornecer uma ontologia consensual para os desenvolvedores de sistemas ubíquos. As ontologias existentes no SOUPA estão divididas em dois diferentes conjuntos: (i) *Core*, que define conceitos comuns a diversas aplicações ubíquas, estes conceitos são representados em nove ontologias diferentes; (ii) *Extension*, que estendem as ontologias presentes no Core, definindo conceitos mais específicos, os quais podem não ser comuns a diversas aplicações ubíquas.

- **CONON** (*Context Ontology*) (WANG; GU; ZHANG, 2004): prevê um contexto geral, incluindo conceitos comuns às aplicações conscientes de contexto. Esta ontologia contem um conjunto de conceitos de nível superior, tais como, *ContextEntity*, *Location*, *Person*, *Activity*, *IndoorSpace*, *Device*. Fornece flexível extensibilidade para acrescentar conceitos específicos em diferentes domínios, por

exemplo, *Cellphone* pode ser uma sub-classe de *Device*.

- **FIPA** (*Foundation for Intelligent Physical Agents*) - *Device Ontology Specification* (FIPA, 2001): é um exemplo de ontologia, que integra-se com outras, com o objetivo de reutilizar o conhecimento, com alguns conceitos como *Device*, *HardwareDescription*, *SoftwareDescription* e *ConnectionDescription*. Pode ser usada como referência para expressar as capacidades de diferentes dispositivos em um sistema de computação ubíqua.

- **GUMO** (*General User Model Ontology*) (HECKMANN et al., 2007): uma ontologia de alto nível para modelagem de usuários ubíquos. GUMO procura abranger todas as classes, predicados e instâncias referentes às condições e modelos do usuário, aos dispositivos dos sistemas e aos ambientes. Esta ontologia dividiu as descrições das dimensões do modelo do usuário em três partes: auxiliar, predicado e alcance de valores.

O uso de ontologias como artefato de software podem ser utilizados em tempo de desenvolvimento ou manutenção e em tempo de execução. No desenvolvimento, auxiliam na especificações de requisitos, modelagem conceitual de sistemas, atividades de suporte, gerenciamento de projeto, reuso de conhecimento.
Em tempo de execução podemos ter dois tipos de ontologias, segundo (HILERA; RUIZ, 2006):

- **Ontologias como artefatos arquiteturais**: ontologias são parte da arquitetura de software, como um componente adicional, cooperando com o resto do sistema para atender o objetivo da aplicação. Aplicações orientadas a ontologia (*Ontology-driven software*): onde a arquitetura de *software* é caracterizada por usar uma ou mais ontologias como elementos centrais do sistema. Utilizados em Sistemas baseados em repositório de conhecimento, que é formado por ontologia e motor de inferência.

- **Ontologias como informações de recursos**: são utilizadas pela aplicação para propósitos específicos, como uma fonte de informações, normalmente remota, onde podem ser feitas consultas para a operação da aplicação, Aplicações Cientes da Ontologia (*Ontology-aware Software*).

3.2 Tecnologias para Tratamento das Ontologias

As linguagens para ontologias devem permitir a seus usuários escrever conceitualizações formais explícitas sobre modelos de domínios. Seus principais requisitos são: sintaxe bem definida, suporte eficiente ao raciocínio, semântica formal, suficiente potencial de expressividade e conveniência de expressão.

Expressividade X Raciocínio
Quanto mais rica a linguagem, mais ineficiente se torna o suporte ao raciocínio. Às vezes

a expressividade pode ultrapassar as fronteiras da computabilidade, portanto é necessário um compromisso entre expressividade e o suporte ao raciocínio.

Raciocínio sobre o conhecimento
Pertinência a Classes: se X é uma instância de uma classe C e C é subclasse de outra classe D, então é possível inferir que X é uma instância de D.
Equivalência de Classes: se a classe A é equivalente à classe B e B é equivalente à classe C, então A também é equivalente a C.
Consistência: se X é instância das classes A e B, mas A e B são disjuntas, isto indica certamente um erro na ontologia.
Classificação: certos pares propriedade-valor são condição suficiente para pertinência em determinadas classes.
Se um indivíduo X satisfaz tais condições para uma classe A, pode-se concluir que X deve ser uma instância de A.

W3C, *World Wide Web Consortium* (http://www.w3.org/)
São os detentores dos padrões XML, RDF, e OWL. Em meados da década de 1990 a W3C começou a trabalhar em uma linguagem de marcação que combinasse a flexibilidade da SGML com a simplicidade da HTML. O principio do projeto era criar uma linguagem que pudesse ser lida por software, e integrar-se com as demais linguagens.

A figura 3.1 mostra as camadas da Web Semântica, estrutura proposta pela W3C.

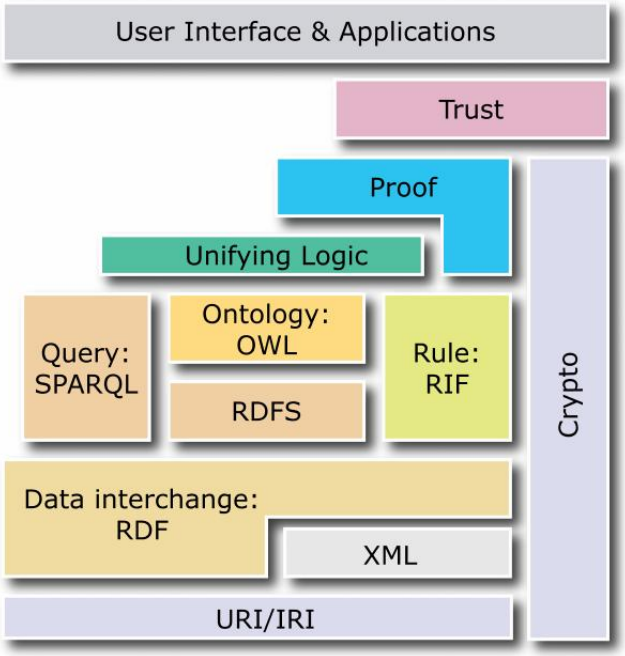

Figura 3.1: Estrutura Web Semântica proposta pela W3C

A figura 3.2 mostra as características e funções esperadas nas camadas da Web Semântica.

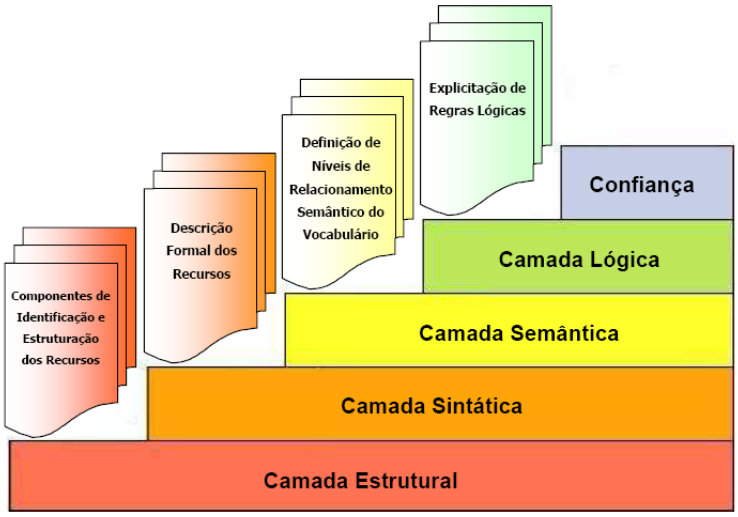

Figura 3.2: Características e funções nas camadas

3.2.1 XML

XML, (*eXtensible Markup Language*), é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais. É um subtipo de SGML, *Standard Generalized Markup Language*, ou Linguagem Padronizada de Marcação Genérica, capaz de descrever diversos tipos de dados. Seu propósito principal é a facilidade de compartilhamento de informações através da Internet. Entre linguagens baseadas em XML incluem-se XHTML (formato para páginas Web), RDF, SDMX ,SMIL, MathML (formato para expressões matemáticas), NCL, XBRL, XSIL e SVG (formato gráfico vetorial). O XML é considerado um bom formato para a criação de documentos com dados organizados de forma hierárquica, como se vê frequentemente em documentos de texto formatados, imagens vetoriais ou bancos de dados. Pela sua portabilidade, um banco de dados pode através de uma aplicação escrever em um arquivo XML, e um outro banco distinto pode ler então estes mesmos dados.

3.2.2 RDF e RDF Schema

O modelo RDF (*Resource Description Framework*) não é uma linguagem, mas um modelo, sendo descrito com a utilização do XML. Sua característica está na forma com que o modelo de dados utiliza metadados para descrever recursos, apresentando um significado ao recurso.

O recurso pode ser caracterizado como um objeto e é sempre identificado por um URI, que através de *statements* possibilita a construção básica de um modelo em RDF. Para tal, utiliza-se a indicação de uma tupla, (predicado, sujeito, objeto), permitindo a ligação entre os valores e os recursos, cuja função é a de prover uma maior representação de modelos complexos. Além de ser apresentado como tupla, o modelo RDF permite uma visualização através de grafos, convencionados pela W3C, que possibilitam representar recursos através de nós, propriedades através de arcos e literais por retângulos.

A viabilidade para representar a interligação das propriedades e os outros recursos mostra-se limitada. Com isso, surgiu o RDF Schema (BRICKLEY; GUHA, 2004), uma extensão de RDF, que veio fornecer descrição de grupos de recursos e os relacionamentos

existentes entre eles. Existe a flexibilidade de criar vocabulários, representados por classes e propriedades com características restritas, podendo serem reaproveitados em outros modelos.

Uma característica bem clara da forma de reuso dos recursos e das propriedades já existentes está na utilização de sintaxes de outras linguagens. A especificação de uma linguagem não tenta enumerar um vocabulário específico para descrição das classes e propriedades de um documento RDF. Mas, a característica do RDF Schema de utilizar apenas as características (recursos, propriedades) que o convém, facilita a descrição de documentos RDF com a utilização de várias sintaxes.

3.2.3 OWL

Segundo a W3C, a maneira mais completa de representar ontologias é sobre OWL (*Web Ontology Language*), esta linguagem é uma extensão do RDFS e permite obter um maior nível de expressividade. A linguagem OWL fornece suporte a metadados RDF, abstrações de classes, generalização, agregação, relações de transitividade, simetria e detecção de inconsistências. A W3C lançou a OWL como um padrão no uso de ontologias em 2008. A linguagem OWL é uma revisão baseada em pesquisa da linguagem DAML+OIL, (*Machine Actionable Semantics*).

A OWL é uma linguagem para definir e instanciar ontologias na Web. Uma ontologia OWL pode incluir descrições de classes e suas respectivas propriedades e seus relacionamentos. OWL foi projetada para o uso por aplicações que precisam processar o conteúdo da informação ao invés de apenas apresentá-la aos humanos. Ela facilita mais a possibilidade de interpretação por máquinas do conteúdo da Web do que XML, RDF e RDFS (RDF Schema), por fornecer vocabulário adicional com uma semântica formal.

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

OWL atualmente tem três sub-linguagens, também chamadas espécies:

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

- *OWL DL*: suporta aqueles usuários que querem a máxima expressividade, enquanto

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

- *OWL Full*: é direcionada àqueles usuários que querem a máxima expressividade e a liberdade sintática do RDF sem nenhuma garantia computacional. Por exemplo, em OWL Full uma classe pode ser tratada simultaneamente como uma coleção de indivíduos e como um indivíduo por si mesma. OWL Full permite que uma ontologia aumente o vocabulário pré-definido de RDF ou OWL. É improvável que algum software de inferência venha a ser capaz de suportar completamente cada recurso da OWL Full.

Cada uma destas sub-linguagens é uma extensão da espécie anterior, tanto em relação ao que pode ser expressado, como em relação ao que pode ser concluído.

3.2.4 API Jena

Jena é uma API (*Application Program Interface*) Java desenvolvida pela HP, Hewlett-Packard Company, (MCBRIDE, 2008), utilizada para desenvolver aplicações baseadas em Semântica Web. A Jena permite manipulação dinâmica de modelos ontológicos e o desenvolvimento de aplicações baseadas em Semântica Web.

Esta API está dividida essencialmente 5 áreas relativas ao processamento de ontologias:

- processamento e manipulação de modelos RDF;
- implementação da linguagem de consulta SPARQL;
- processamento e manipulação de ontologias descritas em OWL ou DAML;
- inferência sobre OWL e RDFS;
- persistência de modelos de ontologias sobre bases de dados.

A Figura 3.3 mostra a estrutura da recuperação de informações da Web Semântica.

A Jena caracteriza-se por possibilitar a criação e manipulação de grafos RDF, representadas pelos recursos (*resource*), propriedades (*properties*) e *literals*, formando as tuplas que irá dar origem aos objetos criados pelo java. Assim, esse conjunto de objetos é usualmente chamado de *model*. Model pode se considerar como o conjunto de *statements* que forma o grafo por completo (VERZULLI, 2001).

A arquitetura base da API de ontologias do Jena é composta por 3 camadas, mostradas na Figura 3.4:

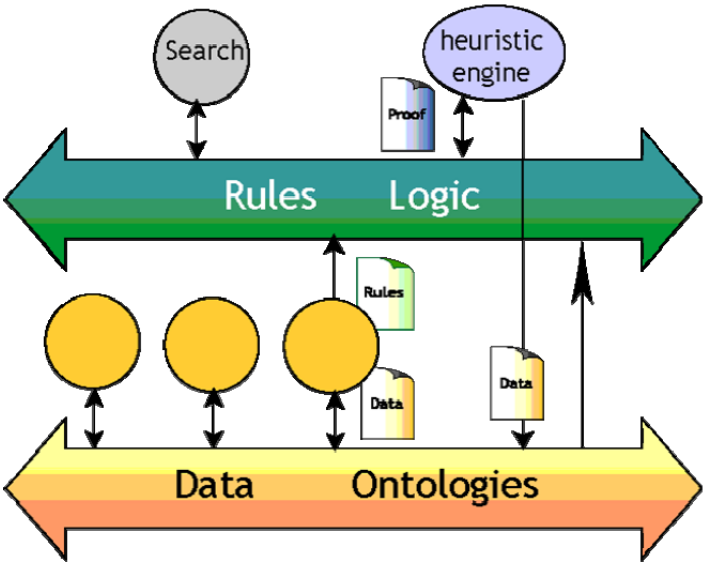

Figura 3.3: Recuperação de Informações WS

- *Ontology Model*: contém todas as classes necessárias para trabalhar com ontologias descritas em OWL, DAML, OIL ou RDFS. Neste módulo a classe mais relevante é a OntModel que representa um modelo ontológico. Esta classe no entanto é uma interface.
- *Graph Interface Reasoner*: permite fazer inferências sobre modelos OWL. O uso das inferências sobre modelos semânticos é permitir obter informação adicional (inferida) sobre as ontologias.
- *Graph Interface Base RDF*: OWL está definido sobre RDFs. O Jena usa a API de RDF para manipular as ontologias.

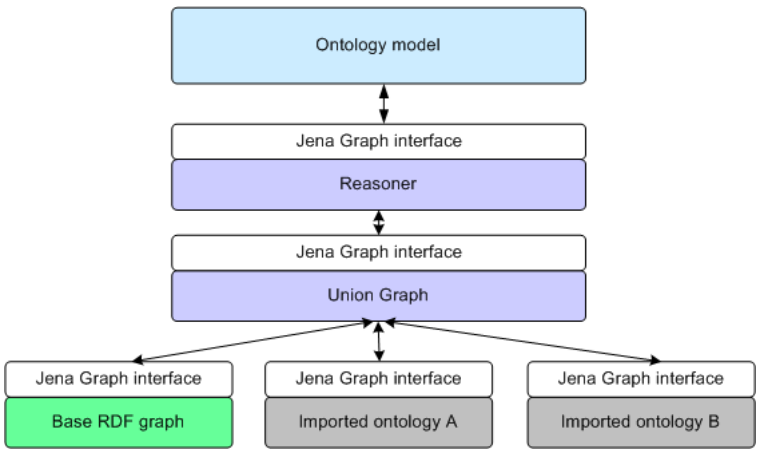

Figura 3.4: Camadas da API Jena (DICKINSON, 2008)

Em todas as camadas existem padrões que são usadas para permitir que a API seja mais genérica. Nomeadamente é usada a *Factory Pattern* para a construção das clas-

ses principais para cada submódulo. O primeiro módulo é *ModelFactory*: encontra a implementação apropriada para aquela interface. De uma forma genérica existe sempre uma classe que representa um conceito (ex. *OntModel*, *Reasoner*), uma *Factory* com métodos estáticos para construir instancias dessa classe e um *Registry* que a fábrica usa para localizar as implementação da superclasse que representa o conceito.
A Figura 3.5 mostra os módulos.

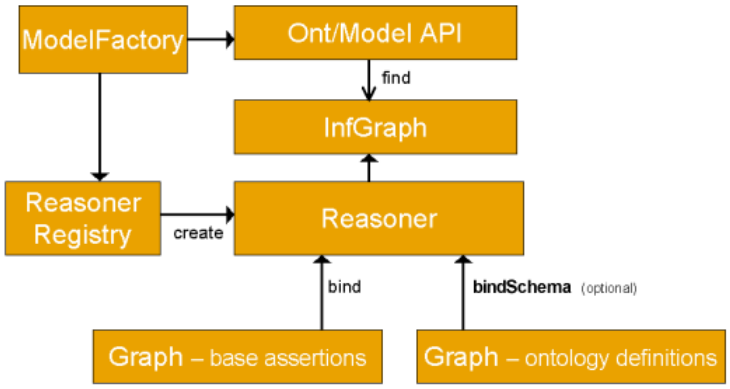

Figura 3.5: Motor Inferência da API Jena (REYNOLDS, 2008)

O módulo *Reasoner* além de fazer inferências, também pode ser usado para fazer processamento e transformações. (REYNOLDS, 2008)A arquitetura do *Reasoner* é feita de tal forma para que novos motores de inferência possam ser adicionados facilmente e permite que esses motores possam ser tanto locais como remotos. Os motores de inferência, que vêm incluídos com o Jena em si, são básicos e são apenas usados apenas para realizar tarefas comuns. No entanto estão disponíveis outros motores de terceiros. A classe principal desta API é o *Reasoner*. Este executa o processamento sobre ontologias. Os *reasoners* podem ser criados a partir a *ReasonerFactory* que por sua vez usa o *ReasonerRegistry* para encontrar o *reasoner* apropriado. É no ReasonerRegistry que novos motores de inferência podem ser adicionados. A operação mais básica que é usada no *Reasoner* é o *validate*. Este método permite verificar se o modelo ontológico está de acordo com as regras de inferência especificadas no *Reasoner*.

Os *reasoners* de OWL do Jena funcionam aplicando regras tipo *if-then-else* sobre instancias OWL. O outro *reasoner* disponível é o de uso geral. Este suporta inferência baseada em regras sobre grafos RDF (*Rule description Framework*). As regras estão definidas sobre BNF e assemelham-se às linguagens lógicas de 1ª ordem como o PROLOG. A classe principal do motor genérico é o *GenericRuleReasoner*. A comunicação entre os módulos é feita pela *Jena Graph Interface* que representa um modelo genérico de dados para comunicação entre módulos. O OWL está definido sobre RDFS, logo o Jena usa as suas *APIs* de *RDF* apara poder manipular as ontologias. Dai a existência do 3º módulo.

O Jena vem com um *reasoner* genérico que depois pode ser implementado por terceiros. Esse *reasoner* tem 2 motores de inferência:

- *Forward Chaining Engine*: verifica cada regra para avaliar se cada uma é válida, se sim executa a regra obtendo novos dados e continua aplicando as regras tendo em conta os novos dados.

- *Backward Chaining Engine*: verifica o objetivo e verifica posteriormente que regras satisfazem o objetivo (similar ao Prolog).

Atualmente existem três *reasoners* que podem ser usados externamente com o Jena através da interface DIG: o *Pellet*, o *Racer* e o *FaCT*.

Queries: Jena possui classes para executar *queries* sobre modelos ontológicos. Essas *queries* são feitas em RDQL. Existe também outra linguagem mais avançada para executar *queries*, SPARQL. O RDQL executa *queries* sobre modelos RDF mas pode ser usado em modelos descritos em OWL, uma vez que estes são descritos sobre RDF. O RDQL trata o RDF como um conjunto de triplas (Sujeito, Propriedade, Valor). Por exemplo, podemos perguntar qual os Valores que satisfazem a tripla para um dado sujeito e propriedade. O RDQL segue a mesmo sintaxe básica que o SQL:

SELECT variáveis WHERE condições

As condições são escritas como triplas: (Sujeito Propriedade Valor). As variáveis por sua vez são representadas com o ponto de interrogação seguido pela designação da variável: ?a, ?b. Os *resources* são enclausurados entre <>.

Persistência: além de poder usar pastas para armazenar as ontologias, o Jena permite armazenar as ontologias em bases de dados relacionais. Atualmente o Jena suporta as bases de dados: MySQL, Oracle e PostgreSQL.

3.2.5 SPARQL

A W3C também lançou como recomendação o SPARQL em 2007. SPARQL (*Protocol and RDF Query Language*) é a nova tecnologia de buscas para Web Semântica. Definida como uma linguagem de consulta e protocolo de acesso a dados em RDF. Esta inovação supera as limitações de buscas locais e o acesso a formatos únicos.

A equipe do W3C RDF *Data Access Working Group* divulgou três recomendações para a tecnologia: a SPARQL *Query Language* para RDF; o protocolo SPARQL para RDF; e a SPARQL *Query Results* para o formato XML. A Oracle, HP e IBM participam deste grupo de trabalho.

SPARQL possui três partes: linguagem de consulta, formato dos resultados e protocolo de acesso. A linguagem é baseada num modelo de triplas, filtros para adicionar restrições, possui partes opcionais e partes alternativas.

SPARQL foi desenvolvida para permitir buscas em diferentes fontes de informação, independente do formato dos resultados.

3.2.6 Protégé

Protégé é um ambiente extensível e independente de plataforma escrito em Java. É uma das ferramentas mais utilizadas para criação e edição de ontologias e bases de conhecimento, suportando a criação, visualização e manipulação de ontologias em vários formatos. O Protégé possui uma vasta quantidade de *plugins*, importa e exporta ontologias em diversos formatos, principalmente OWL, facilitando a reutilização e o intercâmbio de ontologias, além de incorporar diversas outras funcionalidades, como visualizadores e integradores, além de possibilitar o uso de *plugins* desenvolvidos por

usuários. O Protégé foi desenvolvido na Universidade de Stanford e está disponível para utilização gratuitamente.

Entre as vantagens apresentadas por essas linguagens, destacam-se (KNUBLAUCH et al., 2004):

- são otimizadas para a representação de conhecimento estruturado em um nível elevado de abstração;
- os modelos de domínio podem ser disponibilizados na Internet e compartilhados entre múltiplas aplicações;
- é baseada em dialetos não ambíguos da lógica formal, permitindo a utilização de serviços inteligentes baseados no raciocínio (reasoning), possibilitando em tempo de execução, a definição dinâmica de classes, a reclassificação de instâncias e a execução de consultas lógicas complexas;
- essas linguagens operam sobre estruturas similares às linguagens orientadas a objeto, e podem ser eficazmente integradas aos componentes de software tradicionais.

O presente trabalho fará uso de ontologias através das tecnologias: RDF e RDF Schema, OWL, Protégé (OWL-DL), API Jena e SPARQL.

Neste capítulo foram tratados os fundamentos teóricos sobre ontologias com seus principais conceitos, classificação em categorias, alguns projetos específicos de ontologias e as tecnologias para tratamento das ontologias, mostrando as principais linguagens envolvidas.

Como continuidade do trabalho será introduzido um novo paradigma, chamado Computação Autônoma. O próximo capítulo informará as principais fundamentações deste assunto, bem como descreverá o conceito de elemento e gerente autônomo, políticas e mostrará um novo conceito chamado Computação Ubíqua Autônoma.

4 COMPUTAÇÃO AUTÔNOMA

Este capítulo registra os principais aspectos identificados, quando da revisão de literatura, referente a Computação Autônoma e sistemas auto-configuráveis. Seus conceitos são introduzidos bem como sua afinidade com a Computação Ubíqua.

A necessidade de integração de sistemas computacionais heterogêneos (KEPHART; CHESS, 2003) tem dificultado as operações de gerência e manutenção nos sistemas corporativos. As milhares de linhas de código desses sistemas tem apresentado níveis de complexidade que desafiam a capacidade da compreensão humana (AHMED et al., 2007).

Em 2001, Paul Horn, vice-presidente da área de pesquisas da IBM, definiu o termo *Computação Autônoma* para descrever uma solução para a crescente complexidade envolvida nos sistemas de software (HORN, 2001). Inspirado no sistema nervoso autônomo dos humanos, a computação autônoma apresenta uma visão onde os sistemas de software possuem a capacidade de auto-gerência, baseados em informações contextuais e guiados por políticas descritas em alto nível por seus administradores (MENASCE; KEPHART, 2007). Assim, as complexidades de baixo-nível seriam abstraídas dos humanos, permitindo que os administradores de software deixem de lado as rotineiras operações de manutenção, concentrando-se em políticas descritas em alto nível. Nos últimos anos a computação autônoma tem chamado a atenção da comunidade científica que, por sua vez, tem incentivado as pesquisas nessa área em busca de uma padronização de conceitos, modelos e aplicações que estejam inseridos na visão de (HORN, 2001).

Apesar de existirem algumas interpretações e adaptações da visão inicial, a definição mais aceita na comunidade, classifica a computação autônoma em quatro aspectos principais detalhados a seguir (KEPHART; CHESS, 2003; LIN; MACARTHUR; LEANEY, 2005):

1. Auto-configuração: o aspecto de auto-configuração define que os sistemas de software devem se adaptar automaticamente de acordo com as mudanças ocorridas no ambiente em que estão inseridos. Entre outras coisas, isto permite que o sistema continue funcionando normalmente quando uma nova entidade de software se integra ao sistema.

2. Auto-tratamento: em computação autônoma, a capacidade de auto-tratamento é responsável por detectar, diagnosticar e reparar problemas resultantes de *bugs* ou falhas de *hardware* e *software*. Aplicações com esta capacidade se baseiam em dados de *log* e monitores que indicam falhas no sistema. Uma vez diagnosticado o

problema, o sistema deve aplicar as mudanças necessárias para reparar a falha ou indicar o tratamento apropriado para sanar o problema.

3. Auto-otimização: sistemas de banco de dados como Oracle e DB2 apresentam centenas de parâmetros de configuração que devem ser ajustados para otimizar o funcionamento. Para isso, os administradores devem ter o conhecimento das responsabilidades de cada parâmetro e verificar qual a melhor combinação para utilizar no sistema em que está administrando. Em computação autônoma, tais ajustes deveriam ser desempenhados de forma automática, tanto em tempo de instalação, quanto em tempo de execução, com a finalidade de melhorar o desempenho do sistema.

4. Auto-proteção: apesar da existência de *firewalls* e ferramentas de detecção de intrusão, em muitas situações os humanos devem estar presentes para decidir como proteger o sistema a partir de ataques maliciosos. Por exemplo, ao identificar um grande número de requisições com um determinado IP em um curto espaço de tempo, servidores de nomes poderiam caracterizar uma situação de ataque e bloquear as requisições originadas desta máquina automaticamente. Neste sentido, aplicações com a capacidade de auto-proteção preservam o funcionamento do sistema se defendendo de ataques maliciosos e falhas em cascata, além de antecipar problemas com base em relatórios de *log*, requisições e histórico.

A automatização das operações relacionadas a estes aspectos se baseia em informações identificadas no próprio sistema e no ambiente em que o mesmo está inserido, através de mecanismos que tem ciência do contexto. Sistemas que desempenham estes aspectos são conhecidos como sistemas autônomos que possuem a característica de auto-gerência. O que não quer dizer que os aspectos de configuração, tratamento, otimização e proteção estejam inseridos em dimensões ortogonais de uma característica maior de auto-gerência. Em algumas situações estes aspectos se combinam e se sobrepõem na busca da auto-gerência.

4.1 Elementos Autônomos

Os sistemas autônomos podem ser vistos como coleções interativas de elementos autônomos. Cada elemento autônomo é um constituinte individual do sistema, que contém recursos e entrega serviços a humanos ou outros elementos autônomos. Esses elementos gerenciam seu comportamento interno e suas relações com outros elementos autônomos verificando se estão de acordo com políticas que humanos ou outros elementos estabeleceram. Para dar suporte aos elementos autônomos e suas interações, sugere-se uma infra-estrutura distribuída e orientada a serviços (KEPHART; CHESS, 2003).

Um elemento autônomo irá tipicamente consistir de um ou mais elementos gerenciados e um gerente de autonomia que os controla e representa, como ilustrado na Figura 4.1. O elemento gerenciado vai ser essencialmente equivalente ao que encontramos em sistemas não autônomos. Ele pode corresponder a um recurso de *hardware*, como armazenamento ou CPU, ou pode ser um recurso de software como um banco de dados, um serviço de diretórios ou um grande sistema legado. Se observado sob um nível mais alto, um elemento gerenciado pode ser visto como um serviço, ou um negócio individual. De forma geral, elementos autônomos serão dotados de ciclos de vida complexos,

tratando normalmente com várias *threads* de atividades e continuamente monitorando e respondendo ao ambiente em que estão situados.

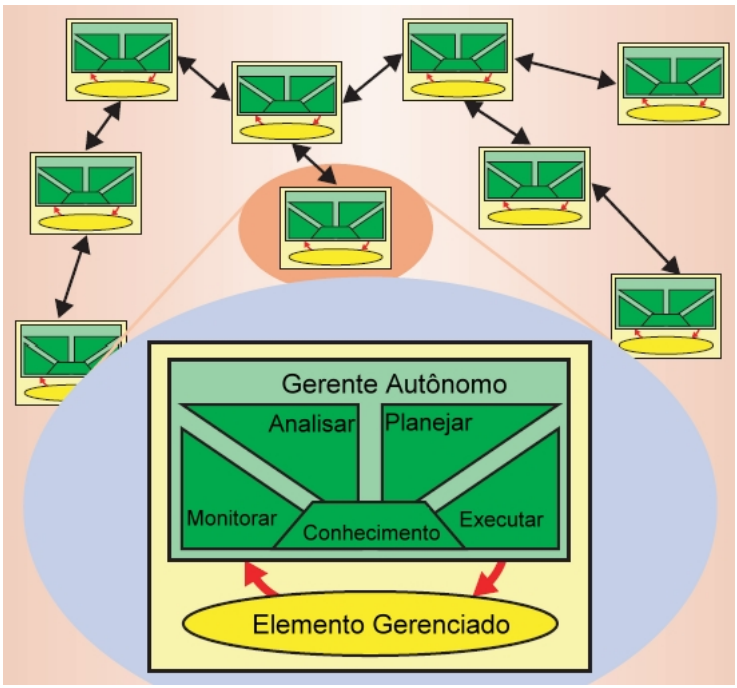

Figura 4.1: Gerente e Elementos Autônomos (KEPHART; CHESS, 2003)

4.2 Políticas

A transformação de diretivas em ações a serem realizadas pelos elementos é de grande importância para o comportamento dos sistemas autônomos. Os comportamentos são determinados através de políticas descritas em alto nível por administradores e usuários do sistema autônomo. Uma política é uma representação, em uma forma padrão, do comportamento de Computação Ubíqua Autônoma desejado e, também, das restrições sobre o comportamento. Neste caso, os sistemas autônomos devem considerar duas questões importantes:

Como os administradores do sistema deveriam expressar estas políticas?

Como o Sistema Autônomo usaria tais políticas para gerenciar o seu próprio comportamento?

Neste sentido, os Sistemas Autônomos utilizam abordagens amplamente discutidas nas disciplinas de Inteligência Artificial. Em (WHITE et al., 2004) são apresentados três tipos de políticas que podem ser utilizadas na concepção de sistemas autônomos:

- Políticas de ação: é a forma de especificação mais básica, a qual é tipicamente expressa na forma SE < condição > ENTÃO < ação >. Um elemento autônomo que emprega políticas de ação deve medir ou sintetizar as quantidades presentes nas condições e deve executar as ações sempre que as condições são satisfeitas;

- Políticas de objetivos: descrevem condições que podem ser atingidas sem especificação de *como*. Neste caso, em vez de informar o que o sistema deve fazer, as políticas de objetivos descrevem o estado final que o sistema deve atingir em determinadas situações. Neste sentido, é possível afirmar que as políticas de objetivos são mais robustas do que políticas de ação, pois um humano ou outro elemento autônomo pode dar direções para outro elemento sem que haja requisição do conhecimento detalhado das atividades internas do elemento.

- Políticas de função de utilidade: especificam o quão desejáveis são estados alternativos, um com relação ao outro. Essa relação entre os estados pode ser obtida através de atribuição numérica ou ordenação parcial ou total dos estados. Funções de utilidade são definidas como extensões das políticas de objetivo, pois determinam automaticamente o objetivo mais importante em uma dada situação.

Abordagens situação-ação que especificam exatamente o que fazer em determinadas situações (GARLAN et al., 2004; LUTFIYYA et al., 2001). Especificações baseadas em ação para situação são, sem dúvida, a mais popular e são utilizadas em diferentes domínios relacionados com as redes e sistemas distribuídos, tais como redes de computadores, bases de dados ativas, sistemas inteligentes.

Abordagens orientadas a objetivos: representam uma forma de alto nível de especificação comportamental, estabelecendo a realização de objetivos, deixando o sistema ou o *middleware* determinar as ações necessárias para alcançar aqueles objetivos (KEPHART; CHESS, 2003). Especificações de objetivos capturam as relações entre contexto e mecanismos de adaptação em uma maneira concisa.

Abordagens baseadas em utilidade estendem as abordagens orientadas a objetivos (WALSH et al., 2004; KEPHART; DAS, 2007). Funções de utilidade atribuem um valor escalar real para os estados de sistema. No nosso caso a utilidade será utilizada nos diversos parâmetros de cada atributo. O gerenciador do serviço de adaptação calcula as variantes de utilidade, selecionando a variante de mais alta utilidade, para cada parâmetro dentro de cada atributo do tipo de adaptação.

Abordagens situação-ação exigem a descrição explícita de cada situação. Quanto mais elementos de contexto, mais condições e as suas regras devem ser especificadas. Além disso, estas abordagens utilizam regras simples que falham para utilizar algumas dependências entre adaptação e contexto. Elas não são, portanto, apropriadas para o objetivo do nosso trabalho. Usando abordagens orientadas a objetivos, é possível expressar políticas em alto nível. No entanto, abordagens orientadas a metas também têm grandes desvantagens. Elas não conseguem capturar dependências entre adaptações e metas, e os conflitos entre os objetivos. Por exemplo, uma meta para a otimização do uso da CPU pode ser alcançada através da suspensão de aplicações de menor prioridade, mas tal suspensão teria impacto sobre a meta para um serviço de alta disponibilidade. Além disso, elas não fornecem qualquer mecanismo para comparar ações de adaptação quando várias ações podem ser aplicadas para atingir uma meta. Funções de utilidade utilizam a lógica para decisões de adaptação de uma forma precisa, e são, portanto, mais adequadas do que políticas de metas quando a adaptação desencadeia efeitos e interferência, ou quando são metas em conflito.

Na modelagem do mecanismo de adaptação proposto, EXEHDA-DA, discutido na seção 6.4, as funções de utilidade são utilizadas para expressar os pesos dos parâmetros e as dependências entre os atributos das variantes de adaptação, o contexto e as necessidades dos usuários e, assim, nós escolhemos uma abordagem baseada em utilidade para fazer face à complexidade do raciocínio da adaptação. As políticas de adaptação serão expressas utilizando funções de utilidade.

4.3 Computação Ubíqua Autônoma

A Computação Ubíqua idealiza a criação de ambientes carregados de dispositivos computacionais que se integram a vida dos humanos de forma transparente, adaptando-se automaticamente de acordo com as situações observadas no contexto em que estão inseridos. A Computação Autônoma indica que os sistemas computacionais deveriam desempenhar funções automáticas de configuração, tratamento, otimização e proteção, substituindo tarefas complexas por políticas descritas em alto nível por usuários e administradores ou desenvolvedores.

Apesar dos dois paradigmas apresentarem focos distintos, é possível perceber que eles possuem características em comum. Por exemplo, a capacidade de adaptação ao contexto de forma transparente para os usuários, assim como as políticas de adaptação, estão presentes nos Sistemas de Computação Ubíqua e de Computação Autônoma. Com isso, alguns autores já definem o termo de Computação Ubíqua Autônoma, nos casos em que as tecnologias desenvolvidas apresentam características dos dois paradigmas (O'DONNELL; LEWIS; WADE, 2005; RANGANATHAN; CAMPBELL, 2004).

Este capítulo tratou da Computação Autônoma, seus principais conceitos, características, descreveu as políticas e os seus tipos, bem como mostrou a afinidade entre a UbiComp e a Computação Autônoma.

O próximo capítulo trata as questões do controle da adaptação na UbiComp, principal foco deste trabalho. Apresentará os desafios, dúvidas e problemas relativos a este tema. Também fará uma descrição, comparação e uma breve avaliação entre os 20 projetos estudados relativos ao controle da adaptação.

5 CONTROLE DA ADAPTAÇÃO NA UBICOMP

Neste capítulo serão sintetizados os problemas, desafios e características do controle da adaptação na computação ubíqua. também serão descritos, comparados e avaliados 20 trabalhos relacionados ao controle da adaptação.

O estudo realizado caracterizou que o desenvolvimento de aplicações *context-aware*, tem muitos aspectos a serem considerados e desafios a serem vencidos, constituindo uma área de pesquisa interessante e com muitos focos de pesquisa a serem explorados.

5.1 Principais Conceitos Associados

Para que um sistema ubíquo tenha o mínimo de intrusão no mundo real, sua infraestrutura de software deve ser sensível ao contexto (DOURISH, 2004), ou seja, possuir uma base extensa de informações a respeito das pessoas e do ambiente que as abriga e, através destes dados, adaptar seu comportamento, da melhor forma possível, para completa satisfação das expectativas de seus usuários.

Uma vez que é possível perceber o contexto, é necessário usar essa informação e agir de forma pró-ativa. Gerência de contexto é a ação de responder a uma mudança detectada. No nosso modelo, a idéia é tomar decisões baseadas em informações sentidas, monitoradas ou inferidas.

Podemos diferenciar as adaptações em estáticas e dinâmicas. Adaptação estática diz respeito à redefinição e reconfiguração de arquiteturas e componentes das aplicações em tempo de concepção, quando alteram requisitos funcionais e não funcionais. Adaptação dinâmica acontece em tempo de execução da aplicação, devido à alterações dos recursos e condições de contexto. Por exemplo, aplicações podem querer reagir dinamicamente a flutuações na conectividade da rede, capacidade da bateria, aparecimento de novos dispositivos e serviços, ou a uma mudança nas preferências do usuário. No nosso trabalho focamos a adaptação dinâmica, onde as mudanças dinâmicas acontecem no contexto de execução e a necessidade de adaptação na aplicação torna-se evidente, especialmente nos cenários onde os usuários se deslocam de um local para outro, ou então trocam de dispositivo.

Adaptação dinâmica é um tema em destaque na Computação Ubíqua. Ela é necessária toda vez que existe uma diferença muito grande entre o fornecimento de recursos e a sua demanda (SATYANARAYANAN, 2001). Através da adaptação é feita a avaliação

de elementos do contexto, como localização, tempo e atividades do usuário, permitindo o desenvolvimento de aplicações sensíveis ao mesmo *context-aware* (HENRICKSEN; INDULSKA; RAKOTONIRAINY, 2006). O tratamento dinâmico da adaptação à medida que as aplicações são executadas ainda não estão resolvidas para a Computação Ubíqua. As condições de contexto são pró-ativamente monitoradas e o modelo de execução deve permitir que tanto a aplicação como ele próprio reajam às alterações no ambiente através de mecanismos de adaptação. Este processo requer a existência de múltiplos caminhos de execução, ou de configurações alternativas, as quais exibem diferentes perfis de utilização dos recursos computacionais (YAMIN et al., 2005a). Na computação Ubíqua, as aplicações necessitam adaptar-se ao contexto, chamadas **aplicações *context-aware***, são as aplicações de natureza dinâmica e adaptativa.

Normalmente, dispositivos da computação móvel estão limitados em termos de poder de processamento, capacidade de armazenamento e capacidade de comunicação. Além disso, a carga da bateria é limitada. Assim, consumo de recursos e escalabilidade da infra-estrutura de adaptação são as principais preocupações. Geralmente, a adaptação é um ponto crucial na concepção de tais sistemas. Para atingir escalabilidade, o número de variáveis que são avaliadas em tempo de execução, em reação a uma mudança de contexto deve ser mantida tão baixo quanto possível, tentando eliminar as configurações iniciais que não são viáveis em determinada situação.

Os trabalhos referentes à adaptação ao contexto apresentam aspectos comuns (YAMIN, 2004):

1. atuam gerenciando as larguras de banda utilizadas nas comunicações;

2. contemplam um domínio específico para as aplicações, tais como: multimídia e informações dependentes do contexto e,

3. usualmente implementam somente uma estratégia de adaptação: alteração no formato dos dados, replicação de dados ou migração de tarefas.

A localização é um aspecto-chave para os sistemas com mobilidade, pois a localidade influi significativamente no contexto disponibilizado para os mecanismos de adaptação. O contexto representa uma abstração peculiar da Computação ubíqua, e inclui informações sobre recursos, serviços e outros componentes do meio físico de execução. Nesta ótica, o ambiente de execução deve fornecer informações de contexto que extrapolam a localização onde o componente móvel da aplicação se encontra. O contexto de execução é definido por elementos computacionais que podem ser mensurados e obtidos, tais como poder computacional e memória disponíveis no processador, banda-passante e latência do canal de comunicações, consumo de energia, localização e preferências do usuário. Através da adaptação, em decorrência das informações de contexto, o comportamento interno do próprio *middleware* pode ser alterado dinamicamente.

O contexto caracteriza informações sobre o estado dos dispositivos envolvidos no processamento, sobre os recursos de rede, sobre a localização do usuário, dentre outros. Deste modo, o monitoramento do contexto, a modelagem das informações provenientes do contexto e a notificação das mesmas são aspectos centrais para as aplicações. Faz-se necessária a existência de configurações alternativas de acordo com o contexto, e deverá existir uma efetiva colaboração entre o sistema e a aplicação na gerência dos procedimentos adaptativos (DEY; ABOWD; SALBER, 2001).

O modelo de contexto é associado a um conjunto de atributos que descrevem o estado das entidades. Entidades são elementos abstratos do modelo e representam a fonte da informação. Atributos são associados a objetos específicos (entidades) ou chamados de elementos de contexto. O conjunto dos elementos de contexto de interesse da aplicação definirá seu contexto. O relacionamento entre os elementos de contexto é estabelecido explicitamente por relações que definem como é a coleta, qual a origem e o tipo da informação. O estado do elemento de contexto é uma das possibilidades previstas à qual a aplicação pode se ajustar. O estado é monitorado e o dado coletado é interpretado para traduzir uma possibilidade válida, gerando um dado contextualizado.

A adaptação se refere à alteração no comportamento, na estrutura e/ou na interface da aplicação, em resposta a trocas arbitrárias no estado do elemento de contexto. Os tipos de adaptações que podem ser empregadas pela aplicação dependem da natureza desta e dos recursos que ela requer.

Alguns exemplos de comportamento adaptativo incluem:

1. variedade de filtros (compressão, omissão, conversão de formato) inseridos entre o servidor e o cliente;
2. alteração da fidelidade de saída;
3. alterações nas dimensões da interface;
4. migração de componentes para máquinas mais adequadas (com maior poder computacional, com maior memória, com rede de comunicação mais veloz, etc.);
5. funcionalidade conectada em face à desconexão utilizando *buffering* e *prefetching*.

Considera-se que as abstrações dos sistemas operacionais/linguagens de programação existentes não são suficientes, nem apropriadas para tratar as necessidades de adaptação das aplicações ubíquas.

A adaptação não-funcional consiste na capacidade do sistema atuar sobre a localização física dos componentes das aplicações, seja no momento de uma instanciação do componente, seja, posteriormente, via migração do mesmo. Ambas operações demandam a existência de mecanismo para instalação sob demanda do código, assim como mecanismos para descoberta e alocação dinâmicas de recursos e acompanhamento de seu estado.

Por sua vez, a adaptação funcional consiste na capacidade do sistema atuar sobre a seleção da implementação do componente a ser utilizado em um determinado contexto de execução, isto é, atua na seleção do código de execução do serviço de uma aplicação.

O EXEHDA (*Execution Environment for Highly Distributed Applications*) é um *middleware* direcionado às aplicações distribuídas, móveis e conscientes do contexto da Computação Ubíqua (YAMIN, 2004; YAMIN et al., 2005a). Este *middleware* é baseado em serviços, que tem por objetivo criar e gerenciar um ambiente ubíquo. As aplicações deste ambiente são distribuídas, móveis e adaptativas ao contexto.A figura 5.1 mostra os serviços do EXEHDA, organizados em quatro grandes subsistemas: execução distribuída, adaptação, comunicação e acesso ubíquo. O suporte à adaptação no EXEHDA

está associado à operação do subsistema de reconhecimento de contexto e adaptação. Este subsistema inclui serviços que tratam desde a extração da informação bruta sobre as características dinâmicas e estáticas dos recursos que compõem o ambiente ubíquo, passando pela identificação em alto nível dos elementos de contexto, até o disparo das ações de adaptação em reação a modificações no estado de tais elementos de contexto.

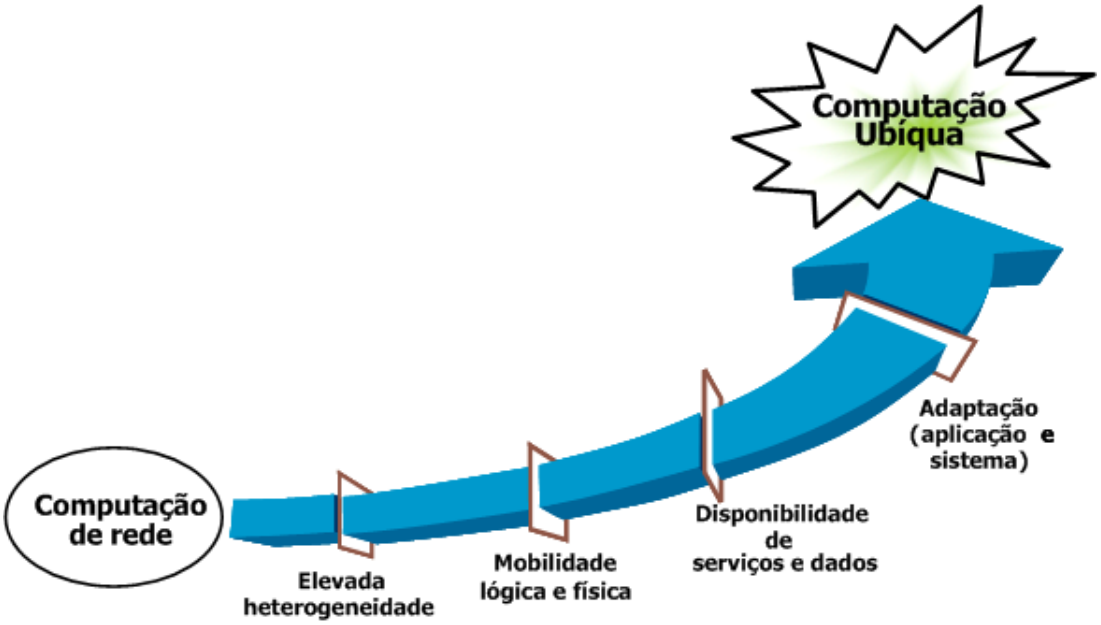

Figura 5.1: Premissas da Computação Ubíqua (YAMIN, 2004)

O processo de publicação, descoberta, seleção e ligação das arquiteturas orientadas a serviços revela uma potencial solução para o compartilhamento de recursos nos ambientes de computação ubíqua. Uma vez que os serviços são descobertos sob demanda, é possível lidar com a heterogeneidade e dinamicidade encontrada nestes ambientes, utilizando a descoberta de serviços sobre os dispositivos que estão dispersos no ambiente.

As políticas de adaptação são centrais na estratégia de colaboração entre o *middleware* e a aplicação, quando da execução dos comandos de adaptação por parte do EXEHDA. Estas políticas referem-se às orientações da aplicação para a tomada de decisão do ambiente de execução, o qual controla o comportamento geral da aplicação, buscando deste modo, o equilíbrio entre transparência (uso de políticas gerais) e a participação da aplicação na gerência do processo de adaptação. A adaptação automática contempla aspectos de propósito-geral não considerando as necessidades peculiares de cada aplicação. O usuário quando conhecedor do comportamento da aplicação pode otimizar o processo de adaptação, orientando a estratégia a ser empregada em relação a componentes específicos da aplicação. Logo, as políticas podem ser de natureza global, válidas para toda aplicação; ou de natureza específica, válidas para alguns componentes em particular. Políticas específicas são semelhantes às globais, diferenciando-se pela identificação do componente (ente) ao qual se referem.

5.2 Projetos Relacionados ao Controle da Adaptação na UbiComp

Nesta seção serão relacionados alguns projetos adaptativos ao contexto, sob a ótica da Computação Ubíqua. Para a escolha dos projetos foi utilizado o foco no controle da adaptação na UbiComp. São descritas, de maneira sucinta, algumas características de cada modelo e após é demonstrado, através de uma tabela comparativa, a situação encontrada em relação a alguns requisitos, considerados importantes na Computação Ubíqua.

Dos diversos projetos estudados, foram relacionados 20, dos quais o resumo descritivo se encontra a seguir.

1. **Omnipresent** (ALMEIDA, 2006): um Sistema Ciente de Contexto baseado em Arquitetura Orientada a Serviço. Arquitetura baseada em *Web services* que proporciona várias funcionalidades: (i) um serviço de mapas, com os pontos de interesse de acordo com o perfil do usuário; (ii) um serviço de rota, com o qual o usuário pode encontrar o menor caminho entre dois pontos no mapa; (iii) um *Web service* de anúncio de produtos e serviços que permitem que usuários possam vender ou comprar de forma que o anúncio ou a busca pelo produto é realizada de forma dinâmica, enquanto ele se movimenta pela cidade; (iv) um serviço que monitora o contexto do usuário e do ambiente ao seu redor, permitindo que lembranças e alertas sejam ativados de acordo com o contexto e não apenas com o tempo, como em agendas eletrônicas; v) um modelo de representação do contexto usando ontologias, baseado nas principais características do contexto do usuário e do ambiente. Como se trata de uma arquitetura orientada a serviço, estes serviços podem ser acessados independentemente. Usuários acessam apenas os serviços necessários. Outros serviços podem ser adicionados à arquitetura desde que sigam o padrão *Web service* da W3C. Possui a facilidade de acréscimo de outras informações de contexto. Novas classes na ontologia podem ser acrescentadas em tempo de execução, permitindo uma rápida expansão de informações, como novas categorias de produtos, novos tipos de estado fisiológico e novas categorias de interesses, além de possibilitar regras de inferência. Em relação à monitoração do contexto, o Omnipresent possibilita a construção de regras complexas que podem ser adicionadas ou removidas a qualquer momento.

2. **SOCAM** (GU; PUNG; ZHANG, 2005): *Service-Oriented Context-Aware Middleware*, é uma arquitetura para a construção de um serviço móvel ciente de contexto. O modelo do contexto é baseado em ontologia que provê um vocabulário para representar e compartilhar conhecimento de contexto em um domínio da computação onipresente. O projeto da ontologia do contexto é composto de dois níveis hierárquicos. Um nível contém ontologias individuais sobre vários subdomínios, por exemplo, o domínio de uma casa, um escritório, um veículo, etc. Um nível mais alto contém conceitos gerais sobre as outras ontologias, esse nível é chamado de ontologia generalizada ou ontologia de alto nível. A arquitetura SOCAM é composta pelos seguintes elementos: (i) Provedores de Contexto: provêem uma abstração do sensoriamento de baixo nível. Cada provedor de contexto precisa ser registrado em um serviço de registro, para que outros usuários possam descobrir esses provedores. Os provedores de contexto podem ser externos ou inter-

nos; (ii) Serviço de Localização de Serviço, SLS: permite usuários, agentes e aplicações descobrirem e localizarem diferentes provedores de contexto; (iii) Interpretador de Contexto: provê contexto de alto nível através da interpretação de contexto de baixo nível; (iv) Serviços Cientes de Contexto: são aplicações que fazem uso dos diferentes níveis da arquitetura SOCAM. Os desenvolvedores dessas aplicações podem predefinir regras e especificar que métodos devem ser invocados quando uma condição for verdadeira. Todas as regras são salvas em um arquivo e pré-carregadas no *reasoner*.

SOCAM foi projetado como um serviço de componentes independentes que podem ser distribuídos numa rede heterogênea e podem se comunicar entre si. Todos os componentes são implementados em Java. Para a comunicação entre os componentes distribuídos é usado Java RMI, que permite objetos distribuídos invocarem métodos de outros objetos remotos. A interação entre os componentes do SOCAM ocorre, resumidamente, da seguinte forma: um provedor de contexto pode adquirir dados sobre o contexto de vários sensores heterogêneos. Diferentes provedores de contexto registram seus serviços no SLS. As aplicações móveis ciente de contexto são capazes de localizar um provedor e obter dados sobre um determinado contexto. O interpretador de contexto e o serviço móvel ciente de contexto também podem ser registrados no SLS. A arquitetura SOCAM segue uma arquitetura semelhante ao padrão *Web Service*, na qual os serviços são registrados em um diretório público e podem ser encontrados e utilizados por outros serviços. Porém, a arquitetura não é independente de linguagem de programação, pois os componentes trocam mensagens usando Java RMI, tornando difícil a integração entre servidores heterogêneos. Além disso, as regras de contexto devem ser carregadas previamente no sistema para que passem a funcionar.

3. **CybreMinder** (DEY; ABOWD, 2000): *Context-Aware System for Supporting Reminders* é uma ferramenta ciente de contexto que ajuda a criar e gerenciar lembranças. Uma lembrança é um tipo especial de mensagem que enviamos para nós mesmos ou para outros, para informar sobre atividades futuras que devemos tratar. Uma importante informação de contexto usado pela ferramenta é a localização, que pode ser em relação a algum lugar ou em relação a uma outra pessoa. O sistema foi implementado em Java e é dividido em duas partes principais: (i) criar lembranças: o CybreMinder apresenta uma interface semelhante ao de um *e-mail*, com assunto, corpo da mensagem, uma lista de outras pessoas a quem a lembrança interessa, nível de prioridade, data e hora de expiração e a situação (contexto) que a mensagem deve ser entregue; (ii) entregar lembranças: a lembrança é entregue quando a situação (contexto) associada for satisfeita ou porque o tempo expirou. A lembrança pode ser entregue via *e-mail*, celular, *handheld*, entre outros.

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

4. **Context Toolkit** (SALBER; DEY; ABOWD, 1999): é um software que ajuda a construir aplicações cientes de contexto. Ele promove três principais conceitos para construir aplicações cientes de contexto: separar a aquisição de contexto do uso de contexto; agregação de contexto e interpretação de contexto. Agregação e interpretação facilitam a aplicação a obter o contexto requerido. *Framework* em Java para programação

de aplicações. Fornece classes para programar sensores, agregadores, tradutores e notificadores. O Context Toolkit consiste de três blocos básicos: (i) *widgets*: agregação e interpretador de contexto. *Widgets* são responsáveis principalmente por coletar e encapsular informação sobre um dado contexto, como localização. Os widgets também dão suporte a serviços que permitem afetar o ambiente, como por exemplo, controlar a intensidade de luz de um local de acordo com a luminosidade detectada pelos sensores; (ii) agregação: responsável por unir toda informação de contexto de uma entidade particular (pessoa, lugar ou objeto). O interpretador de contexto é usado para abstrair ou interpretar o contexto, que é constituído por elementos de um ambiente que detectam a presença de pessoas ou objetos; (iii) aplicação: são as aplicações que usufruem dos componentes do Context Toolkit.

5. **AROUND** (JOSÉ et al., 2003): provê uma infra-estrutura de localização de serviços que permite as aplicações selecionarem serviços que são associados a uma determinada área espacial. A principal diferença entre a arquitetura AROUND e as outras aplicações baseadas em localização é que ela utiliza dois modelos distintos de seleção de serviços: (i) modelo baseado em distância: onde o cliente seleciona os serviços localizados dentro de sua região de proximidade, delimitada pela criação de um raio. A desvantagem desse modelo é que quanto maior o raio, mais coisas que não são de interesse do usuário estarão dentro de sua área de proximidade; (ii) modelo baseado em escopo: cada serviço tem seu escopo associado a um espaço físico. O cliente seleciona aqueles serviços que o escopo inclui em sua própria localização, o cliente é capaz de descobrir um serviço se o mesmo estiver localizado dentro do escopo desse serviço.

No modelo baseado em distância, o foco está na localização do provedor de serviço, enquanto no modelo baseado em escopo, o foco está na área geográfica definida pelo uso do serviço. Estas diferenças fazem com que cada modelo seja o melhor para um determinado tipo de serviço. O modelo baseado em distância é o mais adequado para serviços que tenham uma forte associação com um específico ponto no espaço. O modelo baseado em escopo é o mecanismo utilizado pela arquitetura AROUND para associar serviços com localização. O escopo de um serviço é registrado em um conjunto de contexto de localização. Neste caso, contexto de localização é uma representação simbólica de uma área num espaço físico, como por exemplo, um departamento de um campus universitário, um laboratório dentro de um departamento, uma praça pública, um bairro. Na arquitetura, existem dois tipos de relacionamentos: (i) contido, que se refere a inclusão espacial de áreas dentro da área de um outro contexto; (ii) adjacente, que expressa a proximidade espacial entre dois contextos de localização, permite que usuários consultem serviços próximos mesmo estando fora do escopo. A principal desvantagem da arquitetura AROUND é que ela não leva o contexto do usuário em consideração ao apresentar os serviços. Apenas a localização e o deslocamento do usuário são utilizados pela ferramenta para descobrir em que área de serviço ele está. A comunicação entre os componentes é baseada em CORBA.

6. **CoBrA** (CHEN; FININ; JOSHI, 2003): é uma arquitetura baseada em agentes para apoiar computação com conhecimento de contexto em espaços inteligentes. A principal característica da arquitetura é a presença de um *context broker* inteligente, que é responsável pela manutenção e gerenciamento de um modelo compartilhado de contextos para o benefício da comunidade de agentes. O *Broker* centralizado

foi projetado para possibilitar duas importantes características que são chave para a realização da computação ubíqua, que são o suporte ao recurso limitado dos dispositivos móveis e preocupações com privacidade e segurança do usuário. Através do uso de uma representação explícita de ontologia, CoBrA permite que o conhecimento de contexto possa ser capaz de derivar informações adicionais. Este conhecimento pode também ser facilmente compartilhado por agentes distribuídos usando linguagens e protocolos padrões de comunicação (FIPA-ACL, KQML e SOAP/XML-RPC). A ontologia CoBrA define um conjunto de vocabulários para descrever pessoas, agentes, lugares e uma apresentação de eventos para dar suporte a um sistema de uma sala de reunião sendo agrupada em quatro temas distintos, mas relacionados: (i) Conceitos que definem lugares físicos e os relacionamentos especiais associados; (ii) Conceitos que definem agentes, humanos ou de software, e seus atributos; (iii) Conceitos que descrevem o contexto da localização de um agente em um campus universitário; (iv) Conceitos que descrevem as atividade de contexto de um agente, incluindo as funções dos palestrantes e espectadores e seus desejos e intenções em um evento de apresentação.

7. **Online Aalborg Guide** (ANDERSEN; CHENG; KLITGAARD-NIELSEN, 2003): é um protótipo construído usando o *framework* baseado em LBS, *Location Based Services*, desenvolvido no departamento de ciências da computação da Universidade de Aalborg. Utilizado para implementar um guia *on-line* para turistas que visitam a cidade de Aalborg. As características básicas da ferramenta são: (i) visualizar a localização dos PoI (*Point of Interest*) mais próximos; (ii) permite ao usuário salvar os PoI para uso posterior, como o *bookmark* de um Web *browser*; (iii) permite ao usuário adicionar novos PoI, submetendo o nome e a descrição do mesmo; (iv) o usuário pode adicionar novos comentários e fotos a um PoI já existente; (v) os provedores podem enviar anúncios de acordo com as preferências e localização do usuário; (vi) permite ao usuário encontrar o menor caminho para um PoI; (vii) obter informações sobre outros usuários; (viii) o usuário pode editar o seu perfil antes e durante uma viagem; (ix) serviço de mapas. Um mapa do ambiente é exibido a toda hora com uma indicação da posição do usuário e os PoI mais próximos.

O Online Aalborg Guide usa uma mistura de tecnologia *push* e *pull*. Os PoI mais próximos são continuamente atualizados e mostrados na tela do celular. Dessa forma, o usuário não precisa interagir com o dispositivo para ver que PoI estão próximos. Porém, se o usuário deseja obter mais informações sobre um PoI, ele deve fazer uma requisição ao servidor e a informação é apresentada na tela do dispositivo. No protótipo nem todas as características foram implementadas.

8. **Flame2008** (WEISSENBERG; VOISARD; GARTMANN, 2004): é um protótipo de uma plataforma de integração para *Web services* inteligentes personalizados para as olimpíadas de 2008 em Beijing. A principal idéia desse projeto é, através de um dispositivo móvel, realizar *push* de ofertas significantes baseadas na situação atual e no perfil do usuário. Todas as dimensões do contexto (localização, calendário, perfil) são representadas através de ontologias. Com agregação dessas dimensões, o Flame2008 define situações que são registradas no sistema e pode ser composto de várias ontologias, loc, act e cal são *namespaces* para diferentes ontologias. Definida a situação em que o usuário se encontra e o seu perfil, o sistema se encarrega de buscar serviços que se encaixam nesses parâmetros. O perfil é composto por interesses, preferências e dados

pessoais do usuário. Interesses são informações estáticas que não se alteram com a mudança de contexto, por exemplo, se o usuário possui interesse por música clássica e obras de arte. As preferências podem depender da situação, por exemplo, um usuário em sua cidade pode preferir comida italiana quando vai a um restaurante, mas quando ele está viajando pode preferir experimentar a comida local. Cada usuário pode manter seu perfil através de um conjunto de propriedades que o caracterizam. Além disso, há sensores, que obtêm informações do ambiente atual do usuário (localização e tempo). O resultado são instâncias de uma ontologia de alto nível que são usadas para implicitamente construir um perfil de situação que é semanticamente comparada com os perfis de todas as situações conhecidas pelo sistema, e implicitamente comparada com todos os perfis de serviços registrados. A desvantagem é que ele não trata o relacionamento com outros usuários, ou seja, o sistema não monitora o contexto de contatos do cliente. Além disso, a busca por produtos e serviços é baseada apenas na categoria do produto e no perfil do usuário, não se importando com outras características do produto ou propriedades do serviço.

9. **Nexus** (DÜRR et al., 2004): é uma plataforma com o propósito de dar suporte a todos os tipos de aplicações cientes de contexto através de um modelo de contexto global. Servidores de contexto podem ser integrados à plataforma para compartilhar e usufruir das informações providas pelos outros servidores de contexto. Esses servidores de contexto são chamados de modelos locais ou modelos de contexto. Os modelos locais contém diferentes tipos de informações do contexto. Por exemplo, um modelo que representa as estradas, um modelo que representa as casas em uma cidade, um modelo que representa as pessoas, entre outros. No caso do modelo de pessoas, são usados sensores para manter os dados atualizados no modelo. Os servidores de contexto presentes na camada de serviço armazenam os modelos locais. Para serem integrado a plataforma, os serviços devem seguir uma certa interface descrita em XML e registrados em um serviço chamado *Area Service Register* para que possam ser descobertos dinamicamente. A camada de federação funciona como um mediador entre as aplicações e os servidores de contexto. Ele possui a mesma interface dos servidores de contexto, mas não armazenam modelos. Ele analisa a consulta da aplicação, determina os servidores de contexto que podem responder a consulta e distribui a consulta para esses serviços. O nodo Nexus também possui a capacidade de adicionar serviços que possuem suas próprias interfaces e que usam os modelos de contexto para processar informação e fornecê-la para as aplicações. No topo da arquitetura estão as aplicações cientes de contexto que podem usar a plataforma de três formas diferentes: (i)as aplicações podem enviar consultas e obter informações sobre o ambiente ao redor, como por exemplo, PoI, amigos próximos; (ii) as aplicações podem registrar um evento para receber uma notificação quando um certo estado do mundo ocorrer; (iii) as aplicações podem usar os serviços do nodo Nexus, como os serviços de navegação, para enriquecer as funcionalidades da aplicação. A plataforma Nexus possui uma arquitetura semelhante ao padrão *Web service* [ACKM04]. Os servidores de contexto funcionariam como provedores de serviços que são registrados em um serviço de diretório; o ASR funcionaria como um serviço de diretórios semelhante ao UDDI no padrão *Web service*. Porém, na plataforma Nexus, a descoberta de serviços é realizada pelo nodo Nexus na camada de federação e não pelas aplicações como no padrão *Web service*. A plataforma Nexus monitora o espaço físico, ou seja, o contexto do ambiente. O Nexus é capaz de transmitir anúncios para vários usuários em uma determinada área, mas não faz busca automática de produtos ou serviços de interesse do

usuário. Além disso, a plataforma não é capaz de deduzir informação a partir dos dados do contexto.

10. **ICAMS** (NAKANISHI et al., 2002): é um sistema cliente-servidor que usa celulares para tornar a comunicação mais eficiente através de informações de localização e agenda dos usuários. O sistema usa o serviço de localização PHS, *Personal Handphone System* oferecida pela companhia de telecomunicações japonesa NTT DoCoMo. O celular atualiza a sua localização a cada 15 minutos e tem uma precisão de 100 metros. Basicamente, o ICAMS auxilia no redirecionamento de mensagens telefônicas ou *e-mail*. Os usuários do sistema ICAMS são amigos que desejam compartilhar localização e outras informações. Quando um usuário acessa o servidor, um script PHP, *Hypertext Preprocessor* consulta o banco de dados e retorna um arquivo CHTML chamado *TopPage*. Este arquivo apresenta os outros usuários do sistema em ordem de proximidade. Ícones e setas são usados para indicar se um amigo está se movendo e em que direção em relação ao usuário. Quando o usuário clica no nome de um amigo no *TopPage*, um segundo PHP consulta o banco de dados e retorna um CHTML chamado *MemberPage*. Essa página contém mais detalhes sobre aquele amigo: nome, o último local em que esteve, se o amigo está parado ou se movendo, a distância em relação ao usuário e as opções de contato (telefones, e-*mails* e outros). O usuário pode clicar no número de telefone ou no e-mail para estabelecer uma comunicação. Se o amigo, por exemplo, estiver em sua casa, o telefone residencial é listado primeiro no *MemberPage*. Mas se o amigo estiver numa reunião no trabalho, o seu e-*mail* de escritório será o primeiro na lista. Através do Web *browser* do celular, o usuário pode entrar com sua programação , agenda, selecionando o dia, hora e conteúdo da programação. Pode ordenar os contatos para a programação criada em ordem de preferência. Dessa forma, seus amigos saberão qual a melhor forma de entrar em contato para cada situação do dia. Esse sistema possui algumas desvantagens: baixa precisão em relação à localização; as únicas informações do contexto que são analisadas pelo sistema são a localização do usuário e sua agenda; o redirecionamento das mensagens não é automático.

11. **AMS** (LISZKA et al., 2004): *Arrhythmia Monitoring System* é um trabalho na área de telemedicina com o objetivo de prever ataques cardíacos, arritmias. O AMS coleta sinas de um ECG (eletrocardiograma) combinado com os dados de um GPS e os transmitem a uma estação remota para visualização e monitoração. A arquitetura do sistema é composta dos seguintes componentes: (i) *wearable server*: é um pequeno coletor de dados que o paciente fica conectado. Ele é composto por um ECG (eletrocardiograma) que coleta as atividades elétricas do músculo cardíaco através de biosensores; (ii) *central server*: é um pequeno servidor localizado próximo ao cliente. Ele executa várias funções: compressão, tratar sinais do GPS e detectar sinais de arritmia rudimentares. O *central server* é geralmente um dispositivo móvel como um *Palm*, que conectado ao *wearable server* e a um GPS, transmite esses sinais para o *call center*; (iii) *call center*: é através do qual um profissional médico monitora o sinal ECG e transmite um alerta caso detecte alguma situação de risco. Os dados dos biosensores são armazenados em um *buffer* e a cada 20 segundos, esse *buffer* é transmitido para o *call center*. O sistema também possui um botão de pânico pelo qual o cliente pode enviar um alerta para o sistema central que se encarrega de chamar um serviço de emergência (190, por exemplo) passando a localização do usuário.

12. **SOAM** (VÁZQUEZ; IPIÑA; SEDANO, 2006): *An Environment Adaptation Model for the ubique Semantic Web*. Em SOAM a mais importante entidade da arquitetura é o *smobject*, uma abreviação de "*smart object*". Um *smobject* é um agente, na forma de uma peça de software, representando um dispositivo inteligente, vários dispositivos ou parte do evento de um dispositivo. *Smobject* captura um subconjunto de condições ambientais, que fornecem informação de contexto percebida em requisição solicitada, e age sobre o mesmo ou outro subconjunto de condições ambientais, a fim de modificá-lo e adaptá-lo quando necessário. *Smobjects* necessitam de acesso interno de sensores, *effectors*, portas de comunicação, capacidade de armazenamento, se disponível no dispositivo, e assim por diante. A verdadeira força da arquitetura de agentes baseados em *smobject* no SOAM é padronizar a forma como a informação dos sensores e *effectors* é representada e acessada. *Smobjects* gerenciam três diferentes tipos de estruturas de informação que demostram as funções de *smobject*: informação de contexto, capacidades e limitações. No SOAM, os dados são trocados através de interfaces de comunicação padrões dos *smobjects* e interagem com o dispositivo anfitrião, utilizando os seus internos sensores e effectors. Capacidades e limitações estão representados e trocadas em XML utilizando estruturas declaradas em uma gramática chamada SOAM *Datatypes XML Schema*, ao mesmo tempo a Informação de Contexto é representada usando RDF e Ontologias de domínio de acordo com a especificação OWL. Padrão HTTP é utilizado para propósito de transporte e negociação entre outras entidades e *smobjects*, a fim de recuperar e armazenar esta estrutura de informação no SOAM. Os *smartobjects* recebem ou percebem informações de dispositivos, usuários e procuram ou inferem perfis e configurações destes dispositivos e usuários, modelados numa Ontologia. Todas estas informações necessárias para efetuar as adaptações, são enviadas pelos smartobjects para o *orquestrador* que deverá efetivá-las. Não foi explicado como o *orquestrador* funciona e nem como continua o processo de adaptação.

13. **CAMidO** (AYED et al., 2005): é um middleware baseado em meta-modelo ontológico para descrição de contexto. Este meta-modelo é escrito em OWL e permite que os desenvolvedores reusem vocabulários definidos. Dessa forma, é possível que seja feita uma descrição do contexto e dos sensores pelos quais os dados são coletados. A arquitetura do middleware é baseada em componentes e para adaptar uma aplicação para as alterações do contexto, faz uso dos seguintes componentes: (i) Collection Manager é responsável por receber as informações obtidas dos sensores; (ii) *Context Analyzer* é responsável por filtrar a informação contextual e determinar mudanças relevantes. A filtragem de contexto consiste em encontrar mudanças de contexto através da comparação do valor da nova informação coletada com a antiga; (iii) Context Interpreter é responsável por interpretar a informação enviada pelo *Collection Manager*. Uma vez que essa informação esteja disponível no *Context Analyzer* é possível que seja criado um serviço para disponibilizar essa informação às aplicações que necessitem dela. Dessa forma, quando uma aplicação precisa fazer uso de determinada informação contextual, o desenvolvedor implementa um serviço que é responsável por receber informações sobre a informação contextual em que o mesmo possui interesse.

14. **JCAF** (BARDRAM, 2005): *Java Context Awareness Framework* [Bar05], é um *middleware* para computação ubíqua baseado em Java, em eventos e serviços. Basi-

camente, ele é constituído de duas partes: *Context-Awareness Programming Framework* e *Context-Awareness Runtime Infrastructure*. O *Context-Awareness Programming Framework* provê um conjunto de classes e interfaces Java para o desenvolvimento de aplicações ubíquas e serviços de contexto, *Context Services*, que constituem a *Context-Awareness Runtime Infrastructure*. Cada serviço de contexto deve lidar com as informações de um ambiente em particular, por exemplo, uma sala ou um quarto, estando os mesmos dispostos em uma topologia ponto a ponto, para que possam assim se comunicar e trocar informações de contexto entre si. As informações providas por cada serviço de contexto podem ser recuperadas através de requisições e respostas, *request-response*, ou registrando-se como ouvintes de mudanças no contexto. Um aspecto interessante a ser observado sobre a modelagem de contexto no JCAF é que novas entidades e ítens de contexto podem ser adicionados a um serviço de contexto sempre que necessário, mesmo em tempo execução. Porém, toda a modelagem da informação contextual fica a cargo do desenvolvedor da aplicação.

15. **CORTEX** (SORENSEN et al., 2004): este projeto propõe o uso de um modelo baseado em objetos sensíveis, *sentient object model*. Este modelo é uma abstração para o modelo de componentes que usa objetos sensíveis para permitir o desenvolvimento de aplicações distribuídas. Ele permite a construção do sistema baseado nestes objetos sensíveis. Um objeto sensível é componente de software capaz de sentir o ambiente através do consumo de eventos via canais de eventos ou outros objetos sensíveis. O projeto CORTEX desenvolveu um *middleware* para prover o suporte necessário ao desenvolvimento de aplicações baseadas em objetos sensíveis. O *middleware* consiste de vários componentes *frameworks*, CF onde cada CF corresponde a uma área de pesquisa. Existem dois CFs que são definidos pelo projeto, o Context CF e o Publish-Subscribe CF. O primeiro registra-se para eventos dos sensores e possivelmente de outros objetos sensíveis. O último implementa um modelo de comunicação entre os objetos.

16. **LOTUS** (MOREIRA BUBLITZ, 2007): é uma infra-estrutura projetada com o objetivo de abordar o problema da disponibilização da informação contextual às aplicações em uma abordagem integrada. Foi adotada uma solução que reúne as soluções baseadas em ontologias com as soluções baseadas em *middlewares*. Das soluções baseadas em *middlewares* foram aproveitados mecanismos que permitem que seja feita a descoberta de nós e serviços mesmo em redes heterogêneas. Com relação às ontologias, foi desenvolvido um módulo que permite a representação da informação contextual. Além disso, foram desenvolvidos mecanismos para permitir que a informação presente nas ontologias seja atualizada dinamicamente e um mecanismo para que a informação seja disponibilizada às aplicações. Foi demonstrado ainda, por meio do estudo de caso, que é possível o desenvolvimento de aplicações cientes de contexto usando o Lotus. Neste estudo de caso, foi possível constatar a viabilidade da infra-estrutura, uma vez que a informação contextual é disponibilizada para o desenvolvedor, podendo este focar na lógica de negócio da aplicação. Além disso, a informação contextual é realmente compartilhada entre as aplicações, permitindo que a mesma informação esteja disponível para várias aplicações, independentemente.

17. **ADAPT!** (SOUZA, 2008): foi desenvolvida através da linguagem Java definindo modelo de entidades, conjunto de predicados, tradutores, engenhos de políticas

e provedores de informação na busca da auto-configuração no ambiente físico do *Embedded*. Apresentada uma infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis. Foram tratados aspectos de dinamicidade, heterogeneidade, ciência de contexto e permitindo a utilização de diversos engenhos de política, que podem raciocinar paralelamente sobre as informações contextuais do ambiente. Utiliza uma infra-estrutura de rede *UPnP*, tecnologia *Universal Plug and Play* define uma arquitetura que permite a conectividade em redes ubíquas ponto-a-ponto sem a necessidade de configurações. Utilizando protocolos conhecidos, como TCP, IP, UDP e HTTP, os dispositivos que suportam UPnP podem, dinamicamente, entrar na topologia da rede, obter um endereço IP, disponibilizar seus serviços e ainda verificar a presença e capacidade dos dispositivos que estão na mesma rede. O Adapt! provê uma aplicação web para a administração do sistema, onde é possível verificar o estado do banco de dados que representa o ambiente, adicionar políticas, adicionar informações e verificar quais dispositivos UPnP estão presentes no ambiente. Utilizando mensagens baseadas em XML, estas operações são realizadas automaticamente de forma transparente para o usuário, deixando a infra-estrutura de rede invisível. Foi implementado, a principio, para as atividades: Iniciando e Parando a Reprodução de Áudio, Ligando e Desligando a Lâmpada de uma Sala. Não trabalhava com Ontologias.

18. **SECAS** (CHAARI; LAFOREST, 2007): *Simple Environment for Context Aware Systems* é um projeto que trata com a adaptação das aplicações ao contexto (preferências do usuário e do ambiente, dispositivos). Uma plataforma que tem como propósito fazer os serviços, dados e interfaces de usuário de aplicações adaptáveis às situações de contexto. Trabalha com facetas (dimensões): perfil da rede, descrição/preferências do usuário, características dos dispositivos do usuários, localização, ambiente. A troca de um valor de parâmetro em uma faceta, assinala uma nova situação contextual. Adaptação da aplicação ao contexto: troca de dados com o usuário, adaptando o conteúdo e a interface para visualização. Mostra uma nova definição do contexto, separando dados da aplicação dos parâmetros do contexto. Foi utilizada a representação das redes Petri para descrever os serviços da aplicação e suas dependências acoplados a serviços Java XML-RPC para estabelecer a adaptação. Foi implementado no ambiente médico, focando o impacto do contexto estático na aplicação, objetivando validar a sua estratégia de adaptação.

19. **PROTEUS** (TONINELLI et al., 2007): um modelo semântico de política adaptativa ao *context-aware*. Trata o contexto como a classe principal para definição e adoção de políticas. Abordagem baseada em descrições lógicas ontológicas e regras em linguagem de programação. O modelo considera três tipos de adaptação: (i)adaptação de políticas: consiste em instruir o sistema para um contexto que mudou e que deve continuar ativo se determinadas condições de contexto forem mantidas. Esta política de adaptação automaticamente prolonga a validade de uma política ativa; (ii) adaptação de ações: representa a habilidade de encontrar ações alternativas, permitidas ou obrigatórias, caso a ação original determinada pelo estado corrente não possa ser executada; (iii) adaptação de contexto: consiste em identificar um contexto alternativo onde ações permitas/obrigatórias poderão ser realizadas. Permite recursos serem controlados baseados na visibilidade do contexto, obtida pelo paradigma da semântica, com descrição de alto nível do contexto e uso de raciocínio em contexto e políticas.

20. **CAPPUCINO** (PARRA; DUCHIEN, 2008): é um projeto que objetiva empreender a concepção, implantação e execução de software em ambiente aberto, móvel e ubíquo, o qual requer adaptação por todo ciclo de vida do software, considerando a heterogeneidade de dispositivos em que o software será executado. Será considerada a estrutura cliente/servidor em vez do esquema peer-to-peer. O escopo do projeto prevê um ambiente aberto, onde usuários entram e saem do ambiente e onde são providos uma série de serviços desconhecidos para eles. Para lidar com a mobilidade utiliza, como solução, a adaptabilidade de software. É carregada uma aplicação *bootstrap* extensível no dispositivo, por ftp ou http, que cria um processo de adaptação no qual todo dispositivo móvel aprende como gerenciar a informação de contexto para comunicar-se com diferentes provedores e obter acesso para serviços e informação customizada.

5.2.1 Comparação entre os modelos

Os modelos foram comparados, conforme as características funcionais de um sistema ubíquo que se adapta ao contexto.

As principais características consideradas foram:

1. Monitoração em tempo real: para perceber alterações no contexto, as informações obtidas dos sensores e dispositivos devem ser as mais atuais possíveis;
2. Monitoração de vários tipos de contexto: contexto do usuário, contexto do ambiente, contexto das aplicações, contexto computacional;
3. Localização: essa característica permite que o usuário visualize a sua localização, a localização de seus serviços e seus dispositivos;
4. Análise do perfil do usuário: informações sobre as suas preferências para determinado serviço;
5. Anúncio de produtos ou serviços: esta característica faz parte do contexto do ambiente e permite com que o usuário receba anúncios de produtos e/ou serviços que estejam próximos de sua localização;
6. Orientado a serviço: os serviços são executados sob demanda e permite que novos serviços heterogêneos sejam adicionados a arquitetura de forma padronizada, acrescentando mais funcionalidades ao sistema.
7. Descoberta de recursos ou serviços: mostra se a solução provê os mecanismos necessário para a descoberta de recursos/serviços;
8. Modelagem do domínio: indica se a solução permite que a informação referente ao domínio da aplicação seja modelada;
9. Suporte a raciocínio: indica se a solução provê algum mecanismo que permita que seja realizado algum raciocínio sobre a informação adquirida;
10. Compartilhamento: indica se a solução permite que a informação contextual seja compartilhada pelas aplicações;

11. Histórico: se existe alguma forma de registro das situações de contexto e/ou adaptações realizadas pelo modelo;

12. Ontologias: se o modelo usou semântica para modelagem de contexto, ou especificação de regras ou condições para adaptação, descrição ou seleção de ações de adaptação.

A Tabela 5.1 mostra quais modelos possuem as características enumeradas acima. Os campos marcados com "+" indicam que o modelo possui a característica. Os campos sem marcação indicam que a ferramenta não possui a funcionalidade especificada.

Tabela 5.1: Comparativo: Características (1 a 12) e Modelos Context-aware

MODELOS	01	02	03	04	05	06	07	08	09	10	11	12
1. OMNIPRESENT	+	+[1]	+	+	+	+		+	+		+	+
2. SOCAM	+	+	+		+	+			+	+		
3. CYBREMINDER	+	+	+							+		
4. Context Toolkit	+	+	+	+			+	+				
5. AROUND	+		+							+		
6. CoBrA	+							+	+	+		+
7. Online Aalborg Guide	+	+[2]	+	+								
8. FLAME2008	+	+[2]	+	+	+	+		+			+	+
9. NEXUS	+	+[3]	+			+	+	+		+		
10. ICAMS	+	+[4]	+	+						+		
11. AMS	+	+[5]	+	+								
12. SOAM	+	+[6]	+					+	+	+		+
13. CAMidO	+	+[6]						+	+			+
14. JCAF		+				+		+		+		
15. CORTEX	+							+				
16. LOTUS	+	+					+	+	+	+		+
17. ADAPT!		+[7]					+	+	+			
18. SECAS		+[7]	+	+				+	+			+
19. PROTEUS		+[8]		+				+	+			+
20. CAPPUCINO		+[7]	+				+					

Observações na tabela comparativa:
[1]:Menos o contexto computacional.
[2]:Localização e perfil do usuário.
[3]:Servidor de contexto para cada contexto específico.
[4]:Localização e agenda do usuário.
[5]:Localização e batimento cardíaco.
[6]:Apenas informações de sensores.
[7]:Dispositivos e seus recursos.
[8]:Apenas contexto computacional.

5.2.2 Considerações sobre os modelos

Em relação à monitoração do contexto, o Omnipresent possibilita a construção de regras que podem ser adicionadas ou removidas a qualquer momento. Em outras aplicações, como o SOCAM, as regras devem ser pré-carregadas no sistema para que passem a monitorar o contexto. No Flame2008, o usuário escolhe situações pré-definidas no sistema, no qual, os anúncios de produtos devem ser apresentados ao cliente através do seu dispositivo móvel. No Omnipresent, o usuário pode criar regras para encontrar pontos de interesse quando certas situações acontecem, por exemplo, quando estiver com fome, para listar as lanchonetes num raio de 1 Km. Por sua vez o Flame2008 analisa o histórico do usuário para deduzir novas informações sobre o seu perfil.

Muitos trabalhos dependentes do contexto, são aplicações sensíveis apenas à localização, *location-aware applications*. Mudando a localização do usuário, a aplicação propõe diferentes informações, serviços ou produtos, por exemplo em shoppings ou aplicações voltadas ao turismo. Outras aplicações são adaptativas quanto ao conteúdo, em função de perfil do usuário. Muitas aplicações Web utilizam este tipo de adaptação. Outras ainda, são adaptativas ao dispositivo utilizado, onde a interface do usuário muda de acordo com o dispositivo. Também temos alguns exemplos de adaptação na área médica, onde alguns softwares adaptam o conteúdo a ser visualizado em função do usuário (administrador, médico, enfermeiro, atendente, residente, entre outros) e do dispositivo utilizado, adaptando o formato, interface do usuário (resolução da imagem, cores, sintetização de voz, tamanho de tela, imagens, texto). Nestas aplicações podem ser visualizados exames médicos, como eletrocardiogramas, ultrassons, análises laboratoriais. Outros tipos de adaptações de conteúdo também podem ser encontrados, como adaptação de idiomas, compressão, decomposição, agregação de dados. Temos um número bem expressivo de trabalhos em adaptação de conteúdo Web para dispositivos móveis (telefone celular, PDA).

Em relação aos trabalhos relacionados na tabela comparativa de Modelos versus características de ambiente ubíquo, as abordagens consideradas atendem alguns ou grande parte dos requisitos, porém nenhuma delas contempla todas as características na sua totalidade. Os modelos focam alguns aspectos e deixam lacunas em outros. Por exemplo, a maioria dos *middlewares* focam a etapa de aquisição da informação contextual, levando em conta questões como descoberta de nós e serviços em redes heterogêneas, porém não abordam a forma como a informação é representada e como é feita a inferência sobre a mesma. Já as abordagens que usam ontologias, lidam bem com a parte de representação da informação, porém tem menos preocupação com a forma de obtenção da informação, muitas vezes não provendo nenhum suporte para isso. Além disso, as soluções, de um modo geral, referem-se a um determinado domínio de aplicação, principalmente as soluções baseadas em ontologias, ou à determinada tecnologia de rede, mais comum nos *middlewares*.

O controle da adaptação ainda é muito abstrato e específico a determinado contexto ou a determinada aplicação. Os trabalhos existentes ainda não possuem uma solução genérica e padronizada para as aplicações ubíquas.

Neste capítulo foi abordado o controle da adaptação ubíqua, seus problemas, dúvidas, impasses, desafios e classificações. Também foi resumido o estudo dos trabalhos relacionados, sendo discutidos 20 projetos e apresentadas sua descrição, comparação e avaliação destes, em relação aos 12 aspectos considerados importantes para o controle

da adaptação na UbiComp.

No próximo capítulo será detalha a proposta para o controle da adaptação dinâmica ubíqua. Será detalhado o modelo ontológico proposto. Também serão apresentadas as potencialidades deste modelo.

6 EXEHDA-DA: FUNDAMENTOS E VISÃO GERAL

Os sistemas com arquitetura estática, monolítica e centralizada tornam-se inviáveis para a área da Computação Ubíqua. No espaço ubíquo ocorrem mudanças continuamente, fazendo com que o software tenha que se adaptar e reagir às mudanças de forma dinâmica. Desta maneira, temos que ter uma visão diferenciada no desenvolvimento de softwares adaptáveis ao contexto, seja na forma do seu desenvolvimento como da sua estruturação.

6.1 Questões consideradas para elaboração da proposta

As tecnologias orientadas a serviços, que podem ser selecionados de acordo com as suas capacidades funcionais, podem ser integrados e requisitados remotamente, afetando a profundidade do dinamismo e flexibilidade das aplicações. *Middlewares* para publicação e utilização de serviços, assim como, Computação em *Grid*, que facilita a virtualização e independência de recursos, seja do ponto de vista de fornecedor atual, quanto de sua localização física. Ambientes de computação autônoma que oferecem recursos para que sistemas se auto-configurem, sejam mantidos, otimizados e protegidos automaticamente. Estas são tecnologias que lidam com a complexidade, heterogeneidade e incertezas de sistemas e ambientes de software modernos, que devem ser considerados no âmbito da UbiComp.

Premissas de Estudo e Pesquisa:

- as decisões para adaptação da aplicação serão o mais possível automáticas, considerando preferências e perfil do usuário, contexto, dispositivos e recursos;
- o ambiente de execução deve ter a definição do que é relevante para a aplicação (elementos de contextos + estados + localização);
- o ambiente de execução deve ser capaz de obter as informações de contexto, processá-las e promover a adaptação da aplicação em função destas informações.

Desafios de Pesquisa:

- qual o nível de colaboração entre a aplicação e o *middleware* e como acontece esta colaboração no controle da adaptação?

- como realocar os recursos de hardware e software no controle da adaptação, mantendo a aplicação operacional?
- como resolver situações de conflito na tomada de decisões da adaptação?
- como diminuir a complexidade e o tempo de desenvolvimento das aplicações adaptáveis ao contexto?
- como viabilizar todos estes desafios, perseguindo a eficiência tanto da aplicação quanto do *middleware*?

Quem fará o Controle da Adaptação, Aplicações ou *Middleware*? A figura 6.1 mostra uma visão geral do controle da adaptação, identificando os extremos, *middleware* ou aplicação, como responsáveis pela adaptação.

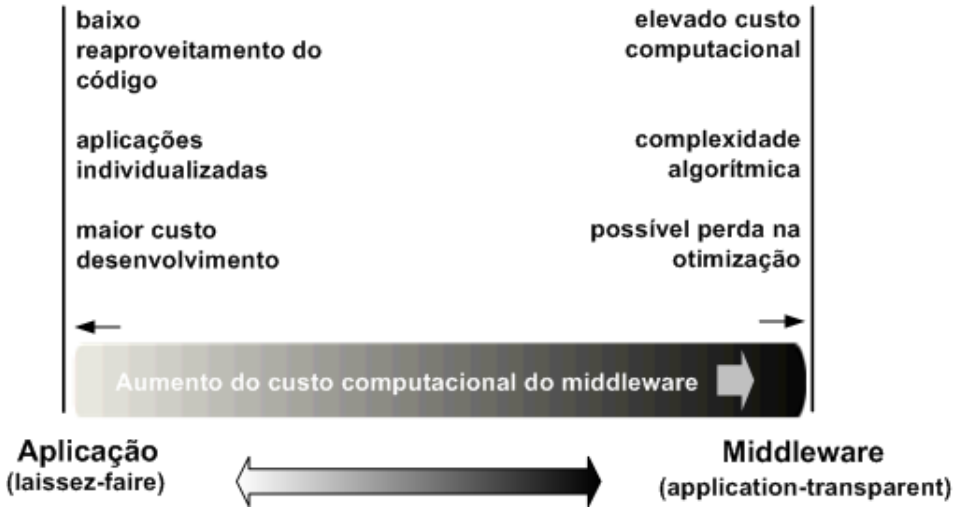

Figura 6.1: Responsabilidade pela Adaptação

O desafio do controle da adaptação ao contexto está em determinar onde este controle deverá ser efetuado, nas aplicações ou no *middleware*. A proposição consta de um serviço de controle de adaptação que seja utilizado tanto pelas aplicações, quanto pelo *middleware*. Uma adaptação Multi-nível Colaborativa, onde todo nível adaptativo fica apenas nos nodos base do *middleware*. Os nodos são heterogêneos, onde uns podem ter mais serviços que outros. Cada nodo pode ter um perfil de execução e os serviços podem ser carregados por *boot* ou por demanda. A essência da colaboração da adaptação se encontra em juntar desempenho com serviços sob demanda. Na nossa visão tanto o *middleware* quanto as aplicações devem perseguir este modelo.

Este mecanismo será uma sugestão de um módulo, Controle de Adaptação funcional e não funcional, para o *middleware* EXEHDA. Na Figura 6.2 é mostrado o módulo no EXEHDA que trata da Adaptação ao Contexto. Como áreas de aplicações pretende-se explorar soluções para Medicina e Agropecuária.

A Figura 6.3 mostra a da adaptação e seus níveis de controle.

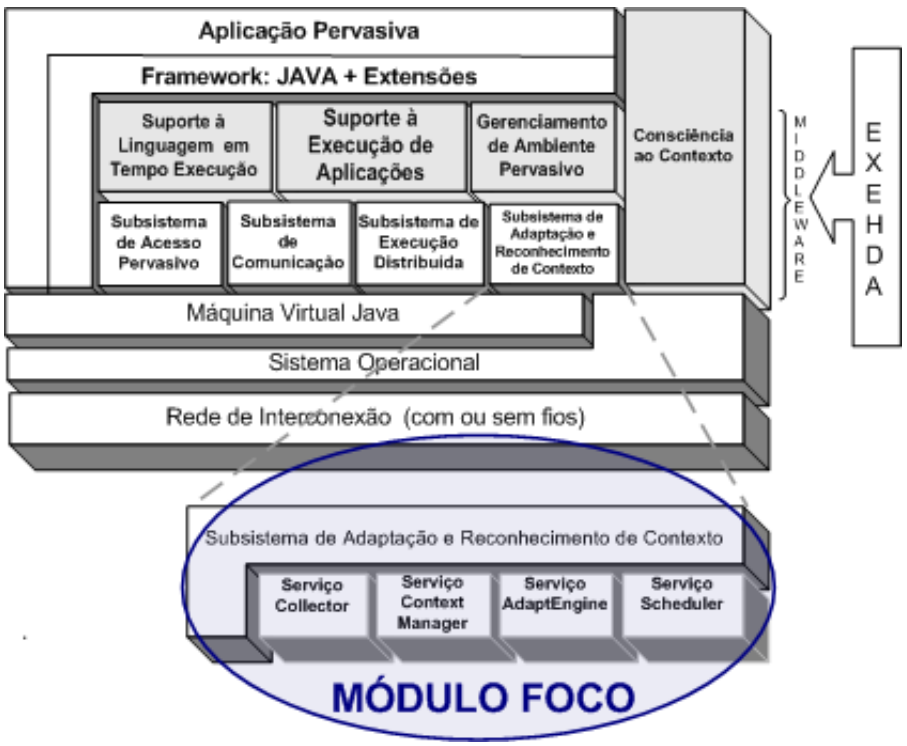

Figura 6.2: Serviço de Adaptação no *Middleware* EXEHDA (YAMIN, 2004)

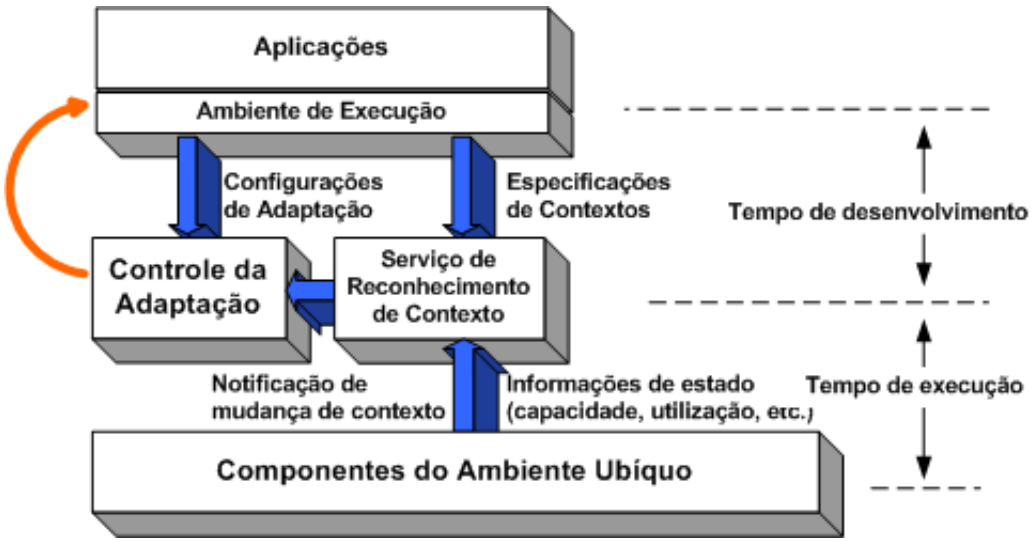

Figura 6.3: Adaptação da Figura Controle da Adaptação Multi-nível (YAMIN, 2004)

Tipos de adaptação dinâmica que são considerados:

- Adaptação Não-Funcional: atua sobre a gerência da execução distribuída, operações de mapeamento, reescalonamento, instanciação remota, migração.
- Adaptação Funcional: adaptação do código da aplicação, atua sobre a seleção da implementação em determinado contexto.

Atualmente, os principais elementos de contexto considerados pelo EXEHDA são o tipo de equipamento, o estado de ocupação dos seus recursos e a situação da conectividade no momento.

6.2 Características Envolvidas na Definição do EXEHDA-DA

Na monitoração de contexto, observa-se que os múltiplos elementos de contexto devem ser considerados. Além disso, estes elementos básicos, tais como o custo da rede, podem ser agregados com elementos complexos ou derivados, tais como previsão de localização. Espera-se que o conjunto de elementos relevantes de contexto e suas fontes produtoras evoluirão ao longo do tempo, da mesma forma que as aplicações. Na definição de cenários móveis, muitas vezes, novos elementos de contexto são introduzidos gradualmente como o problema a ser resolvido passa a ser melhor compreendido ou novas funcionalidades são adicionadas na aplicação. Estas possibilidades levam as seguintes considerações arquiteturais: (i)monitoração do contexto deverá ser mantida separada da aplicação e realizada através de elementos reutilizáveis pelo *middleware*. O contexto de *middleware* deve ser extensível e suportar a inclusão de novos elementos de contexto e novas formas de raciocínio. Estas duas possibilidades já estão sendo consideradas no trabalho sobre Sensibilidade ao Contexto na Computação Ubíqua no *middleware* EXEHDA, EXEHDA-SS: Suporte Semântico (VENECIAN, 2008).

Além da reusabilidade e extensibilidade dos elementos do contexto, a separação da execução da aplicação, do controle das suas adaptações, deve simplificar o desenvolvimento de aplicações adaptáveis. Gerenciamento do contexto torna-se responsabilidade do Servidor de Tratamento de Contexto do *middleware* EXEHDA, serviço EXEHDA-SS.

Relativo ao controle da adaptação, observamos múltiplas relações entre contexto e mecanismos de adaptação, bem como os impactos dos efeitos da adaptação. Assim, os elementos de contexto não podem ser considerados separadamente quando do raciocínio sobre a adaptação. Por exemplo, as capacidades de áudio devem ser consideradas em conjunto com a atividade do usuário, antes de escolher a modalidade Interface do Usuário (IU), isto é, voz ou IU com base em texto. Também a implementação de uma adaptação pode ter efeito sobre o contexto. Por exemplo, a seleção da modalidade IU tem um impacto sobre o consumo de recursos de dispositivos portáteis. Mais elementos de contexto foram introduzidos e mais as relações entre eles que temos que considerar. Com a generalização de cenários, sabemos que é muito difícil captar todas essas

relações (FLOCH; STAV; HALLSTEINSEN, 2006). Novas relações poderão ser introduzidas, envolvendo as aplicações e os elementos de contexto, a medida que elementos evoluam.

Em (OREIZY et al., 1999) foram examinados dois tipos de abordagens para a auto-adaptação em aplicações context-aware: (i)abordagens internas, em que a gestão de contexto e adaptação são realizados como parte da aplicação utilizando características de linguagem de programação; (ii)abordagens externas, onde estes mecanismos são realizados por um *middleware* independente. O principal inconveniente da abordagem interna é a complexidade introduzida pelo entrelaçamento da adaptação com os comportamentos da aplicação. Além disso, pouco suporte para a evolução deste tipo de software. Estas desvantagens tornam abordagens internas inadequadas no contexto dos serviços ubíquos. Portanto, escolhemos a abordagem externa para o mecanismo de adaptação.

Assim, na nossa proposta os mecanismos de adaptação serão realizados externamente a aplicação. As abordagens externas requerem políticas de adaptação a serem descritas separadamente das aplicações. Estas políticas são utilizadas pelo Servidor de Adaptação para raciocinar e decidir sobre a adaptação.

Quanto às políticas de adaptação, como visto em 4.2, funções de utilidade podem ser especificadas para expressar as dependências entre os parâmetros de atributos das variantes de adaptação, o contexto e as necessidades dos usuários. Foi escolhida uma abordagem baseada em utilidade para fazer face à complexidade do raciocínio da adaptação. As políticas de adaptação, deste modo, serão expressas utilizando funções de utilidade.

Embora abordagens baseadas em utilidade permitam a tomada de decisões, elas, também, exigem a especificação em detalhes dos atributos e parâmetros das adaptações de cada aplicação, pelos desenvolvedores. A função utilidade calcula a utilidade de uma adaptação da aplicação como uma função dos atributos previstos para as diferentes adaptações e do contexto atual. O objetivo é selecionar a adaptação da aplicação que maximiza a função de utilidade, satisfazendo as limitações dos recursos fornecidos pelo ambiente. Definir a função utilidade para aplicações complexas pode ser uma tarefa difícil. Ela poderia ser adequadamente expressa, por exemplo, como uma soma ponderada das dimensões das funções de utilidade de suas propriedades, onde os pesos manifestam preferências do usuário (isto é, importância relativa de uma dimensão para o usuário). Um função de utilidade dimensional mede a satisfação do usuário em um atributo da dimensão. Políticas de adaptação para múltiplas aplicações, podem igualmente ser expressas como uma soma ponderada das utilidades de cada aplicação, onde os pesos expressam as prioridades de aplicação do usuário. Quando os recursos são escassos isto pode levar à adaptações onde aplicações de baixa prioridade serão candidatas a serem suspensas, a fim de maximizar a satisfação global do usuário.

Para o tipo de mecanismo de adaptação das aplicações existem duas técnicas populares de adaptações: adaptação por parâmetros e adaptação composicional (POLADIAN et al., 2004). Adaptação por parâmetros apóia a sintonia de aplicações através da modificação de variáveis do programa e parâmetros de implantação. Adaptação composicional permite a modificação da estrutura dos componentes da aplicação e substituição de componentes. Parametrização é um modo intuitivo e eficaz para implementar variabilidade, mas é menos poderoso do que a adaptação composicional e também pode levantar preocupações em relação a escalabilidade, se os valores dos parâmetros possuírem uma variação muito grande. Portanto, focaremos a adaptação composicional de componentes.

O usuário interage com a aplicação em um contexto específico, tal como definido

pelos valores dos atributos indicados como preferências/necessidades do usuário. Utilidade representa uma medida de quão bem uma configuração da aplicação se adapta num determinado contexto. A utilidade de uma adaptação da aplicação é computada como uma função dos atributos representando a interação entre a aplicação e seu contexto.

A função de utilidade considera os atributos da aplicação e de execução do ambiente atual, bem como as preferências do usuário. A função utilidade é definida pelo desenvolvedor da aplicação. A função de utilidade calcula a soma ponderada das diferenças entre os atributos ofertados pelo ambiente e os necessários pela aplicação, onde os pesos refletem as prioridades das necessidades do usuário. Dentro deste modelo, o objetivo do Servidor EXEHDA-DA pode ser formulado como a garantia de que, a qualquer momento, será executada a adaptação com a mais alta utilidade e que não viole as limitações de recursos impostos pelo ambiente.

Na prática, o contexto muda quando o usuário se move para locais diferentes, ou quando o ambiente muda (ou seja, diferentes horas do dia, diferentes condições de luz, sinais mais fracos de rede sem fio, etc.) Como consequência, as configurações possíveis são avaliadas usando a função de utilidade, e a aplicação é apropriadamente adaptada. Por exemplo, a regra da IU é resolvida dinamicamente a ser executada por uma IU visual (normal) ou o áudio (mãos-livres), dependendo do contexto do usuário (dirigido à mãos ocupadas, instalando uma antena, ou não).

Em resumo, adotamos uma abordagem externa de adaptação onde as capacidades de adaptação de aplicações são realizadas por um serviço do *middleware* independente. As políticas de adaptação serão expressas utilizando funções de utilidade. A adaptação é baseada em adaptação composicional de componentes. Seguimos uma abordagem de arquitetura centrada no *middleware*, onde componentes genéricos, em tempo de execução, raciocinam sobre o controle da adaptação, chamado de EXEHDA-DA (*Dynamic Adaptation*): Serviço de Controle de Adaptação Dinâmica ao Contexto do EXEHDA. A adaptação precisa conhecer a estrutura de componentes e limitações da aplicação, bem como os vários contextos e dependências de recursos. Isto implica em que o *middleware* trabalhe, em tempo de execução, em um modelo arquitetônico, onde as aplicações especificam previamente todas as variantes e limitações de adaptação. O EXEHDA-DA realizará, de maneira dinâmica, adaptações funcionais e não funcionais.

6.3 Trabalhos Relacionados ao Modelo Proposto

Nesta seção, os trabalhos relacionados foram selecionados para servirem de ferramenta de comparação com o modelo proposto para o EXEHDA-DA. Esta escolha se deve às características e premissas do modelo proposto.

Quanto aos trabalhos relacionados, ainda são muito orientados à QoS, com foco na adaptação de *middleware* a nível de serviços. Nos últimos anos, tem havido um interesse crescente na sensibilidade ao contexto e computação móvel, focando na adaptação à nível da aplicação. Este trabalho se enquadra nessa categoria. Um levantamento global de projetos de middlewares para sistemas de software adaptativos é fornecido em (MCKINLEY et al., 2004). No nosso modelo o controle da adaptação é um serviço do *middleware* EXEHDA, incidindo sobre adaptação de serviços do próprio *middleware* como sobre as aplicações.

Carisma (CAPRA; EMMERICH; MASCOLO, 2003)] é um *middleware* para

computação móvel *peer-to-peer* que explora o princípio de reflexão para suportar a construção de conscientes-contexto, aplicações adaptáveis. Carisma incide sobre a adaptação do *middleware* a nível de serviços. Planejamento no Carisma consiste em escolher entre um punhado de políticas de adaptação baseadas em regras pré-definidas (a que se refere o perfil da aplicação) usando funções de utilidade e resolvendo conflitos entre as políticas utilizando um procedimento do tipo leilão. Embora as políticas baseadas em regras adotadas são naturais e simples de utilizar, existem algumas desvantagens em comparação com as funções de utilidade do EXEHDA-DA. Em primeiro lugar, as regras prevêem menor transparência para o desenvolvedor, exigindo raciocínio em termos de ações de reconfiguração de baixo nível, em vez do projeto arquitetural semântico. No EXEHDA-DA, ações de reconfiguração de nível mais baixo são automaticamente determinadas pelo *middleware* e a adaptação é conduzida por atributos para adaptações funcionais e não-funcionais, enquanto as políticas baseadas em regras do Carisma não consideram previsão de adaptações não-funcionais. Ainda, CARISMA manuseia apenas um número fixo de políticas de adaptação enquanto a nossa abordagem da função de utilidade permite selecionar a melhor adaptação entre um leque de alternativas, que são automaticamente inferidas pelo servidor de adaptação. Além disso, o EXEHDA-DA permite, a qualquer momento, o desenvolvedor da aplicação incluir novas adaptações ou novos atributos, devido à política da aplicação ser mantida externamente para configuração dos perfis das políticas, regras e ações.

Chisel (KEENEY; CAHILL, 2003), é um framework aberto para adaptações dinâmicas de serviços utilizando reflexão dirigida à políticas, de maneira consciente ao contexto. É baseado na decomposição de aspectos extra-funcionais (não-funcionais) de um serviço em comportamentos alternativos. Então, um algoritmo orientado a política é usado para dinamicamente selecionar a alternativa mais adequada, com base nas mudanças de contexto. Mudanças de contexto desencadeiam um processo que inteligentemente determina a adaptação a ser realizada. Chisel nivela políticas de adaptação declarativas legíveis para humanos para instruir o comportamento adaptativo das aplicações. Estas políticas são definidas com uma linguagem de forma personalizada e essencialmente baseadas em regras. EXEHDA-DA, além de regras, utiliza funções de utilidade para determinar a ação a ser utilizada e realiza adaptações funcionais e não-funcionais.

Objetos de Qualidade (Quo) (HEINEMAN; LOYALL; SCHANTZ, 2004; SHARMA et al., 2004) é um bem conhecido projeto de *middleware*. Embora Quo não resolva os problemas fundamentais associados a dispositivos portáteis e redes sem fio, teve um impacto importante em sistemas adaptativos e projetos de desenvolvimento de *middleware QoS-aware*. Quo é uma arquitetura que centra em fornecer, em tempo de concepção e em tempo de execução, suporte para aplicações cliente-servidor adaptáveis. A abordagem QUO permite que aplicações adaptem as suas funcionalidades dinamicamente alterando comunicação e recursos computacionais. Exige monitoração dos recursos em tempo de execução e uma descrição detalhada de todas as possíveis adaptações em cada componente da aplicação. Contratos Quo definem as possíveis adaptações da aplicação sob a mutação das características dos recursos. Uma desvantagem da arquitetura Quo é que ela só se concentra na adaptação da funcionalidade interna da aplicação baseada na disponibilidade de recursos de sistema, não considera adaptações na configuração da implantação de uma aplicação distribuída. Quo, como EXEHDA-DA, é também um dos poucos projetos que tentou abordar adaptação como um componente ou serviço reutilizáveis.

As técnicas de auto-adaptação propostas pela abordagem Rainbow (GARLAN et al., 2004) são semelhantes ao EXEHDA-DA na medida em que tenta separar adaptação das funções lógicas da aplicação. Rainbow estende adaptação baseada em arquitetura, adotando estilos arquiteturais. Estes estilos são utilizados para moldar a infra-estrutura da adaptação através da codificação dos conhecimentos específicos dos sistemas dos desenvolvedores, incluindo estratégias de adaptação e preocupações de sistema. No entanto, o Rainbow não faz qualquer tentativa para resolver as exigências de ambientes móveis. Além disso, os seus desenvolvedores parecem ter baseado as estratégias de adaptação em regras situação-ação, que especificam exatamente o que fazer em determinadas situações. EXEHDA-DA usa políticas de objetivos estendidas expressas como funções de utilidade, que é um nível mais alto de especificação de estratégia de adaptação, definindo objetivos comportamentais, os quais, em razão das ações necessárias, implementam essas políticas, além de permitir o uso de dispositivos móveis.

Madam (GEIHS et al., 2009) é uma solução abrangente para o desenvolvimento e operação de aplicações conscientes de contexto, aplicações auto-adaptáveis. As principais características são: (i) um sofisticado *middleware* que suporta a adaptação dinâmica de aplicações baseadas em componentes, e (ii) uma metodologia de desenvolvimento *model-driven*, que se baseia em modelos de adaptação e as correspondentes transformações modelo-para-código. Madam é uma abordagem geral e global para o desenvolvimento de aplicações auto-adaptáveis para cenários pervasivos. EXEHDA-DA não preocupa-se com o desenvolvimento de software adaptável, seu foco é permitir adaptações que podem ser utilizadas por componentes da aplicação ou do *middleware*, podendo serem reutilizáveis e dinâmicas, assim como no Projeto Madam.

As funções de utilidade proporcionam uma maior variabilidade de adaptação e uma maior adequação às necessidades do usuário. Mas é uma facilidade que não é considerada na maioria dos projetos.

Nesta seção o modelo proposto foi comparado com trabalhos relacionados, os quais possuem as características mais próximas à nossa modelagem. Verificamos que alguns dos trabalhos tratam apenas um tipo de adaptação, funcional ou não-funcional. Outros não fornecem a possibilidade de manutenção ou incrementos nas regras e políticas das aplicações, a qualquer momento. Quanto ao uso de modelo semântico, pela sua expressividade e possibilidade de reutilização e padronização, julgamos que proporciona uma maior facilidade ao desenvolvedor na definição do perfil da aplicação, mas ainda não é uma metodologia de uso geral.

6.4 Modelagem do EXEHDA-DA

O modelo proposto consta de um mecanismo de adaptação genérico que será utilizado, em tempo de execução, por todas aplicações e pelo próprio middleware, considerado pelo serviço EXEHDA-DA como uma aplicação, para suas adaptações funcionais e não-funcionais. Neste modelo teremos, no repositório de adaptação, todas as regras, políticas e ações globais e ações específicas de cada aplicação, utilizadas pelo servidor de adaptação, para a partir destes dados e das mudanças do contexto inferir o que e onde a ação de adaptação deverá ser executada. O modelo de controle de adaptação ao contexto proposto pretende tornar-se genérico, possibilitando uma

evolução incremental das especificações de políticas, regras, atributos, restrições e ações de adaptação (configuração do perfil da aplicação). Permitindo a reutilização e a customização destas no desenvolvimento de novas aplicações *contex-aware*. Além disso, deverá possuir mecanismos para administrar situações de conflito quando da tomada de decisões do servidor de adaptação. Assim sendo, o objetivo é o de adaptar os serviços da computação ubíqua ao ambiente sem, ou com mínima, intervenção explícita do usuário, utilizando a *inteligência* de dispositivos e de softwares.

O Servidor EXEHDA-DA é responsável pelo raciocínio sobre o impacto das mudanças de contexto nas aplicações, adaptando o conjunto de aplicações em execução, selecionando o componente das aplicações que atenda a respectiva mudança, além da configuração do dispositivo que melhor se encaixe ao contexto atual. Atividades do serviço de adaptação: coordenação das ações realizadas quando ocorre uma mudança de contexto e pela seleção de componentes para o conjunto de aplicações concorrentes. Para a seleção do componente é necessário determinar todas as variações possíveis para cada atributo e aplicar a respectiva regra de adaptação. Durante a avaliação da adaptação selecionada, verifica se os recursos necessários ao componente da aplicação podem ser satisfeitos.

6.4.1 Estrutura do Serviço de Adaptação Dinâmica Proposto

O Controle da Adaptação Dinâmica - EXEHDA-DA, pode ser desenvolvido, modificado, e estendido independentemente das aplicações. Por esta razão a proposta define EXEHDA-DA como sendo um serviço do *middleware* EXEHDA, sendo executado sob demanda.

A proposta do trabalho é criar um modelo de controle da adaptação dinâmica de aplicações em ambiente ubíquo. A idéia é controlar estas adaptações quando da tomada de decisões considerando: (i) o contexto, com base em informações monitoradas, informações semânticas e inferências a partir destas mesmas informações; (ii) perfil da aplicação; (iii) preferências e necessidades do usuário.

A estrutura do EXEHDA-DA é apresentada na Figura 6.4, mostrando a estrutura do sistema ubíquo em camadas, expandindo a camada do *middleware* EXEHDA com ênfase no Servidor de Adaptação. Apresenta também a relação direta com o Servidor de Contexto. A Figura 6.5 expande o Servidor de Adaptação da Figura 6.4, mostrando a visão geral do EXEHDA-DA.

Descrição das propriedades da estrutura em relação ao Servidor de Adaptação:

1. Adaptações: regras e ações para cada adaptação.

2. Aplicações/Componentes: repositório de componentes de cada aplicação e *middleware*.

3. Serviço *Executor*: após a adaptação, o componente da aplicação selecionado pelo Servidor CAD, é enviado para o *Executor* do EXEHDA (adaptações funcionais).

4. Serviço *Scheduler*: após a adaptação, o mesmo componente é enviado com a nova situação operacional para o *Scheduler* do EXEHDA (adaptações não-funcionais).

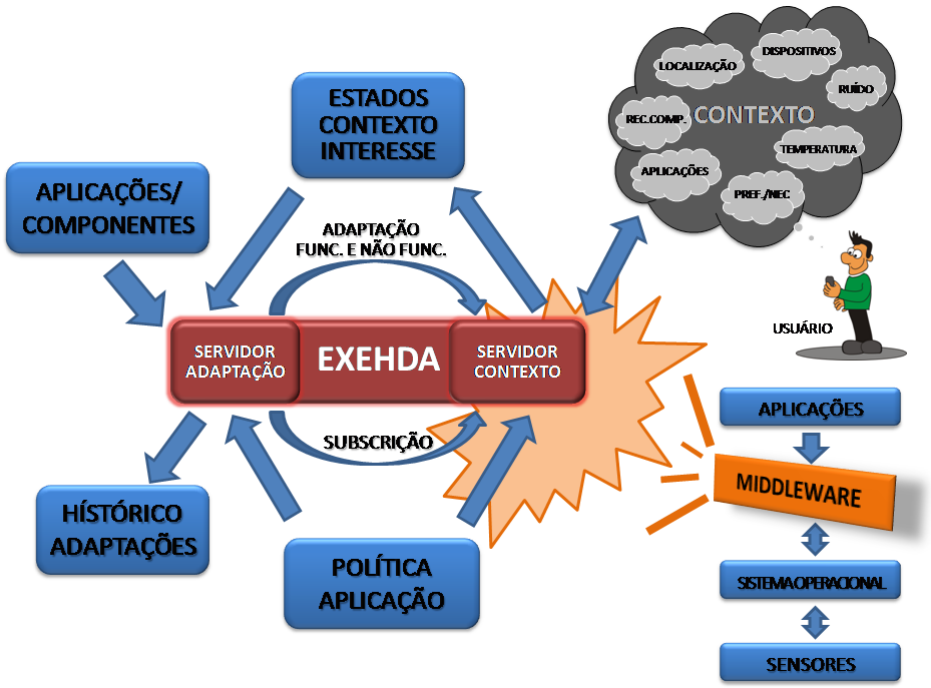

Figura 6.4: Relação entre os serviços do EXEHDA para o Controle da Adaptação

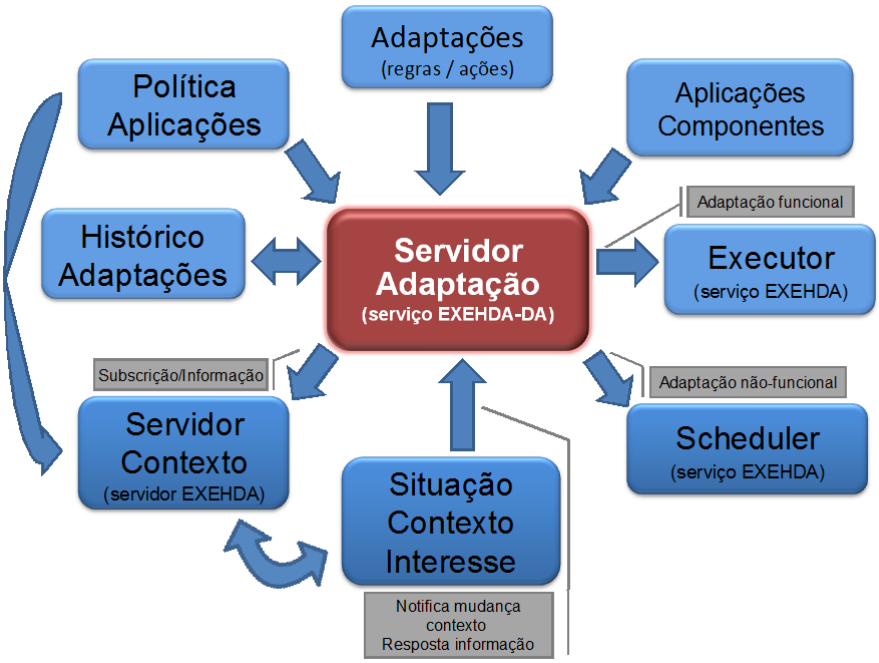

Figura 6.5: Visão Geral do EXEHDA-DA

5. Situação Contexto de Interesse: sempre que ocorre uma mudança de contexto, que seja de interesse das aplicações, o Servidor de contexto envia para o Servidor de Adaptação o contexto que mudou, a situação anterior e a nova situação do elemento de contexto. Também, através desta propriedade, o Servidor de Adaptação recebe as informações que foram solicitadas ao Servidor de Contexto, que são necessárias para o raciocínio da adaptação.

6. Servidor de Contexto: cria Situação Contexto de Interesse, através das mudanças de contexto e das informações subscritas pelo Servidor de Adaptação.

7. Histórico das Adaptações: ontologia com o registro das adaptações realizadas. Podem ser utilizadas em situações de conflito na decisão da adaptação.

8. Política das Aplicações: ontologia que contém a descrição de todas as adaptações suportadas por cada componente da Aplicação. Cada adaptação do Componente terá definido, pelo desenvolvedor da aplicação, os atributos, regras e ações de adaptação.

EXEHDA-SS, Serviço de Suporte Semântico, notifica a mudança de contexto para o serviço EXEHDA-DA.

Atributos e parâmetros são utilizados para qualificar os serviços oferecidos ou necessários dos componentes das aplicações. Por exemplo, os que podem descrever uma implementação de componente de interface de usuário suporta um modo *hands-free*, mãos-livres (comunicação via áudio). Atributos também podem definir as exigências de recursos do sistema, tais como a necessidade de memória ou largura de banda. Atributos são, funcionalmente, dependentes dos elementos de contexto. Desta forma somos capazes de representar as dependências entre os componentes das aplicações e o contexto. Atributos previstos podem ser implementados de várias formas, inclusive como constantes, por exemplo, para expressar que uma implementação de um componente específico requer mais do que 500 KB de memória, ou como expressões funcionais de outros atributos do componente em si. Por exemplo, o tempo de resposta oferecido por um componente pode depender da largura de banda disponível para esse componente.

A Figura 6.6 apresenta uma visão geral do processo do Controle de Adaptação no EXEHDA-DA.

O EXEHDA-DA é composto pelos módulos:

- **Preparação para adaptação**: recebe as informações de mudança de contexto do EXEHDA-SS e verifica quais informações são necessárias para o raciocínio da adaptação. Consulta a política da aplicação para a adaptação em questão, verificando os atributos e valores (atributos necessários); preferências e necessidades do usuário; informações computacionais para a adaptação (atributos oferecidos). Se faltarem informações para a tomada de decisões, estas são solicitadas ao EXEHDA-SS, recomeçando o processo.

- **Processamento regras adaptação**: a partir dos atributos necessários, dos atributos oferecidos e das preferências e necessidades do usuário, é computada a regra de adaptação. Neste módulo é selecionada a ação de adaptação de maior utilidade

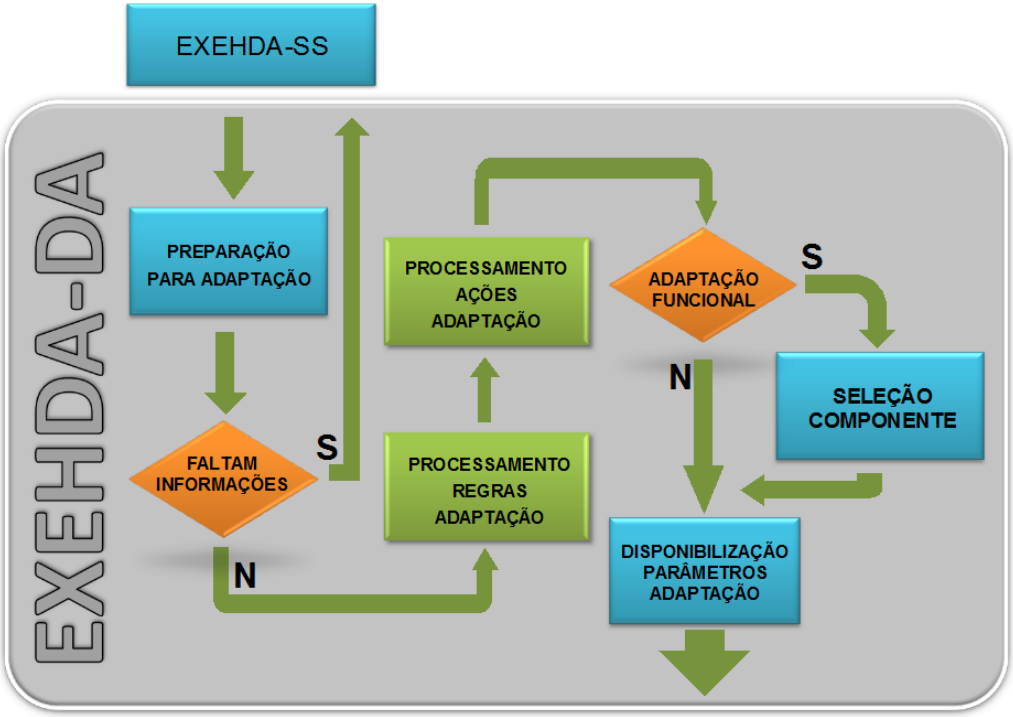

Figura 6.6: Processo do Controle da Adaptação no EXEHDA-DA

para o usuário. Também pode ser consultada ontologia de adaptações realizadas (OntHistAdapt) para resolver eventuais situações de conflito.

- **Processamento ações adaptação**: verifica a ação de adaptação a ser realizada, identificando o tipo de adaptação funcional e não funcional. E grava a adaptação e respectiva ação na ontologia de adaptações realizadas OntHistAdapt.

- **Seleção de componente**: caso seja adaptação funcional, isto é, mudança de código do componente da aplicação, este novo código deve ser acessado no repositório de Aplicações/Componentes.

- **Disponibilização dos parâmetros para o *middleware***: disponibiliza o componente a ser executado, usuário, dispositivo e localização para o *middleware* realizar a execução, através dos serviços *Scheduler* ou *Executor*.

6.4.2 Estrutura do Modelo Semântico do EXEHDA-DA

As características do modelo proposto devem prever variadas noções de adaptação, permitindo identificar os requisitos de propósito geral e específico para expressar adaptabilidade ao contexto, os quais podem ser:

- identificação dos elementos de contexto (entidades) que compõem o ambiente e modelam o contexto de interesse (classes no OWL): (i)recursos da arquitetura: dispositivos; contexto computacional: tais como, conectividade de rede, largura de banda, custos de comunicação, *loggings, databases*; (ii)recursos de software: coleções de

códigos das aplicações e *middleware*; (iii) recursos de usuário: perfil, preferências e necessidades do usuário;

- descrição do comportamento adaptativo: coleções de perfis de adaptação das aplicações (política da aplicação);
- descrição de políticas de adaptação: regras de propósito geral ou particular que orientam as decisões da adaptação realizadas pelo EXEHDA-DA;
- inferências a partir das informações das ontologias para subsidiar as tomadas de decisões do servidor de adaptação;
- possibilidade de usar histórico das adaptações realizadas para subsidiar a tomada de decisões de adaptação;
- estabelecimento de um padrão para descrições, características, configurações dos elementos do contexto e políticas de adaptação.

Vários elementos de contexto podem ser relevantes na UbiComp e várias formas de adaptação podem contribuir para melhorar ou manter a qualidade do serviço quando o contexto muda.

A aplicação tem um contexto, que consiste de uma série de elementos de contexto, cujos valores podem mudar durante o tempo de execução (contexto de interesse da aplicação).

No nosso modelo o Contexto é representado pela ontologia que chamamos de OntUbi, onde estão representados todos os elementos de contexto passíveis de interesse para as aplicações. Esta ontologia é composta pelas entidades, ou também podemos chamar de elementos de contexto ou entidades de interesse, possuindo possuindo três classes básicas:

- Recursos da arquitetura: reflete o estado dos recursos do sistema (memória, capacidade da bateria, largura de banda, conectividade de rede, entre outros); dispositivos (características dos dispositivos que podem ser utilizados no sistema); contexto ambiental (sensores físicos de luz, temperatura, ruído, localização);
- Recursos de software: sistema operacional, *middleware* e aplicações;
- Recursos de usuário: reflete o perfil, as preferências e as necessidades do usuário.

Categorias genéricas de ontologias: (i) Ontologias de Domínio que descrevem conhecimento sobre um domínio ou sub-domínio; (ii) Ontologias como Artefatos de Software: usadas como artefatos de diversos tipos no processo de desenvolvimento de aplicações ou durante a execução de uma aplicação (HILERA; RUIZ, 2006).

Abaixo será mostrada a versão inicial da estrutura do modelo semântico proposto:

- **Ontologia de Domínio**:
  - Básica: **OntUbi** - Ontologia Básica para Computação Ubíqua: ontologia com todas as entidades (classes), seus atributos e relacionamentos possíveis do Contexto Ubíquo.

  - Específica 1: **OntContext** - Ontologia da Situação do contexto e de seus elementos: estado (situação) anterior e estado novo, identificação do dispositivo, identificação da aplicação/componente serviço, identificação do usuário (login), localização, time (data/hora).
  - Específica 2: **OntAdapt** - Ontologia de Adaptação: regras, perfis e preferências, restrições e ações de adaptação para as aplicações.
  - Específica 3: **OntHistAdapt** - Ontologia das decisões de Adaptação, histórico das adaptações realizadas.

- **Ontologia como Artefato de Software** no Domínio em questão:

  - Em tempo de Desenvolvimento de Aplicações: criação e manutenção dos modelos ontológicos básicos: OntUbi e OntAdapt (aplicações orientadas à ontologias.
  - Em tempo de de Execução das Aplicações: Aplicações cientes da das ontologias básicas: controle de adaptação ao contexto: *querys* na OntUbi, OntContext, OntAdapt e OntHistAdapt; e registro em OntHistAdapt.

A Figura 6.7 mostra um exemplo de ontologia, criada no Protégé para demonstrar a primeira versão da ontologia OntUbi, Ontologia Básica do Contexto Ubíquo, que, nas próximas etapas do trabalho, deverá ser detalhada e ampliada.

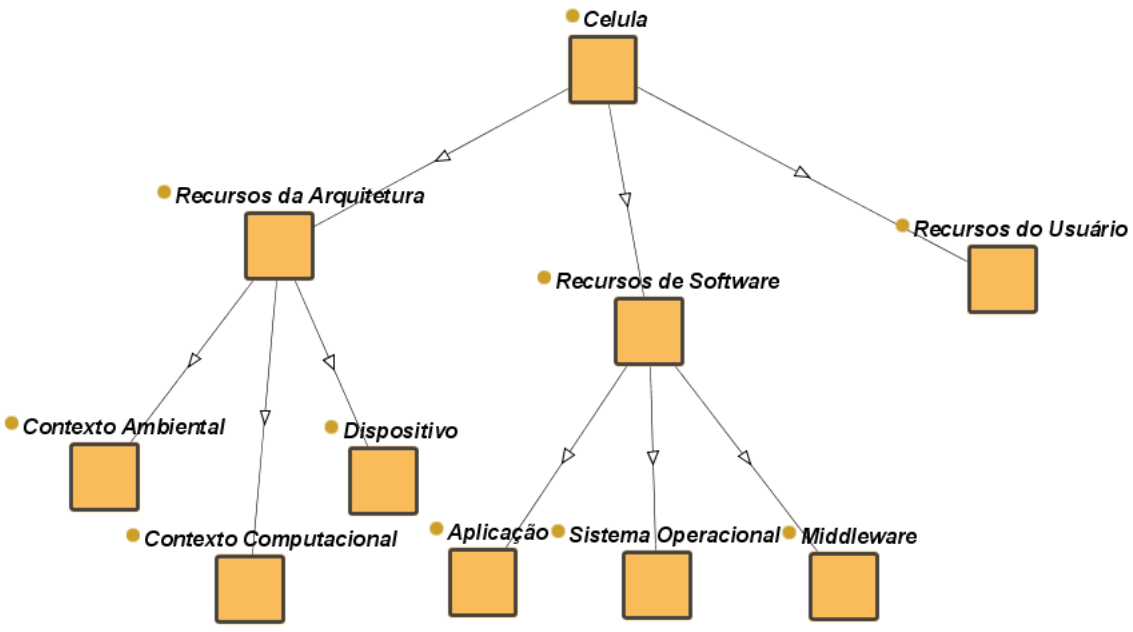

Figura 6.7: Ontologia Básica do Contexto Ubíquo - OntUbi

A ontologia chamada ONTUBI é a ontologia base do modelo, onde serão descritas todas as entidades, elementos do contexto ubíquo.

A Figura 6.8 mostra as classes e a Figura 6.9 mostra os relacionamentos entre as classes da Ontologia da Adaptação - OntAdapt - Política da Aplicação para adaptação,

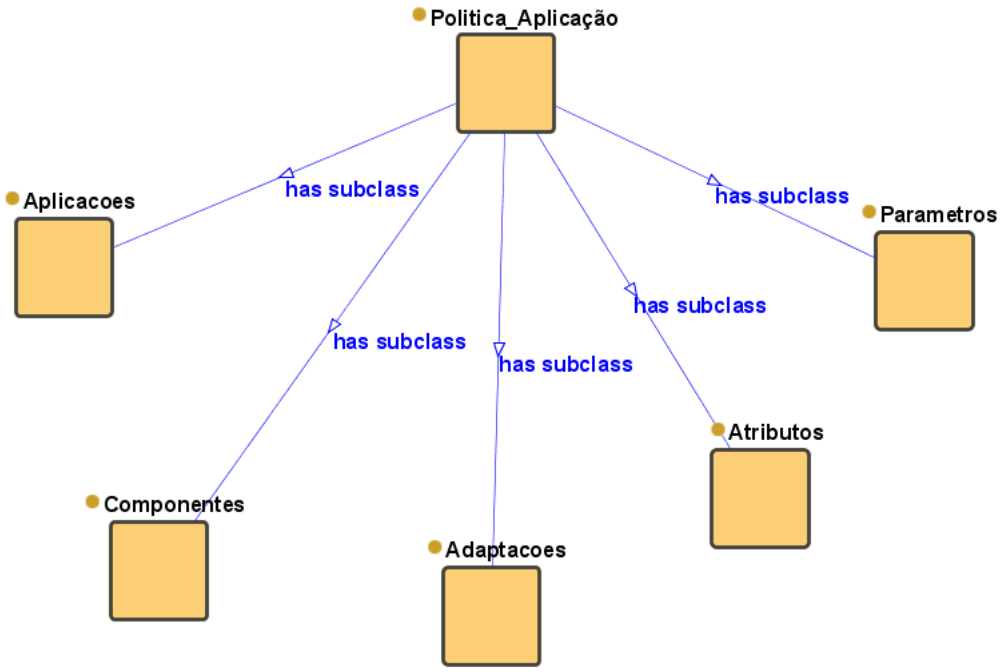

Figura 6.8: Ontologia da Adaptação - ONTADAPT - Política da Aplicação para Adaptação (Classes)

onde são descritos os perfis das aplicações para a adaptação. As classes Aplicações, Componentes, Adaptações, Atributos e Parâmetros são subclasses da superclasse Política da Adaptação. Estas figuras foram configuradas e exportadas do editor de ontologias Protégé.

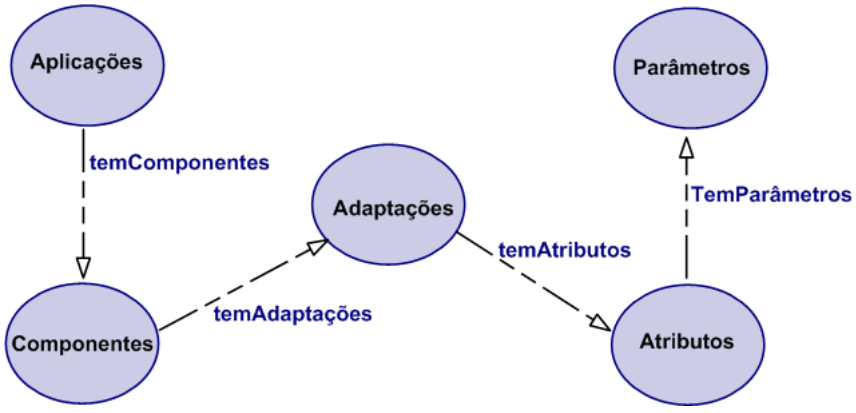

Figura 6.9: Ontologia da Adaptação - ONTADAPT - Política da Aplicação para Adaptação (Relacionamentos)

Na política da Aplicação, podem ser definidas várias aplicações. Cada aplicação é composta por vários componentes, instâncias do relacionamento temComponentes. Cada componente pode ter várias adaptações, instâncias do relacionamento temAdaptações. Cada adaptação de cada componente da aplicação pode ter vários atributos, instâncias

do relacionamento temAtributos. Um atributo pode ter vários parâmetros e respectivos valores de utilidade, instâncias do relacionamento temParâmetros.

O tipo de dados dos parâmetros depende do atributo a eles associados. Podem ser do tipo string, valores inteiros, valores flutuantes, intervalos de valores, valores mínimos ou valores máximos.

Consideramos a interdependência dos atributos, quando um determinado valor de parâmetro de um atributo pode implicar em um certo valor ou conjunto de valores permitidos ou negados de parâmetros para para outros atributos.

Utilidade é um valor definido na classe Parâmetros, podendo variar de zero até 1. A maior utilidade tem valor igual a 1. Utilidade é utilizado com os valores de parâmetros de atributos e nas preferências ou necessidades do usuário. A utilidade é o peso que reflete as prioridades do atributo ou das necessidades do usuário.

Para a criação, manutenção e utilização pelo Servidor de Adaptação do modelo ontológico serão utilizadas as tecnologias a seguir relacionadas:

- RDF - Resource Description Framework;
- OWL - Web Ontology Language
- SPARQL - Protocol and RDF Query Language
- PROTÉGÉ - editor de ontologias;
- API JENA - Toolkit Java - criação, manutenção e inferência dinâmica de ontologias.

A crescente proliferação de dispositivos móveis tem afetado as expectativas dos usuários. Usuários móveis agora esperam que seus dispositivos possam adaptar-se automática e inteligentemente às mudanças de contexto. No entanto, tal comportamento adaptativo pode ser muito difícil de desenvolver, especialmente quando a natureza das adaptações é complexa e difícil de descrever. Adicionando à complexidade, o fato de que diferentes conjuntos de informações de contexto estão disponíveis em diferentes dispositivos e, ainda, que usuários têm diferentes preferências relativas às adaptações desejadas.

Embora os tipos de adaptação possam parecer simples a nível individual, a programação de um sistema para atingir várias das adaptações em paralelo pode ser muito desafiador. Além disso, o processo de desenvolvimento de aplicações adaptativas envolve uma série de ações repetidas, tais como definir a forma de coletar a informação de contexto e como decidir sobre as adaptações. Assim, existe uma necessidade de uma abordagem mais metódica e automatizada, a fim de livrar os desenvolvedores deste trabalho complexo e repetitivo para aplicações adaptáveis ao contexto.

Foi analisado o uso de ontologias para a concepção deste modelo de controle de adaptação. Esta escolha se deve ao fato de ontologias estarem sendo utilizadas no desenvolvimento de aplicações ubíquas, especialmente como artefatos de software, com o objetivo de modelar as informações gerenciadas por essas aplicações, por terem uma maior expressividade na definição de contexto e adaptações, por possuírem a possibilidade de realizar inferências a partir das informações semânticas, facilidade de reutilização e extensibilidade por novas aplicações ou novas situações de contexto. A

escolha fundamenta-se na utilização combinada de ontologias, Sistemas Autônomos e Computação Ubíqua, onde a sua integração poderá tornar-se um padrão a ser utilizado no desenvolvimento de aplicações em espaços ubíquos, além de poder proporcionar facilidades para a concepção e manutenção destas aplicações, reutilização e um atendimento mais eficaz às demandas dos usuários.

A proposta pretende contribuir para o estado atual da pesquisa no G3PD, concebendo uma solução que atenda de forma abrangente o controle da adaptação, considerando os seguintes desafios de pesquisa:

- adaptação acontece ininterruptamente e sem intervenção do usuário, em reação às mudanças de contexto;
- as aplicações podem explorar qualquer tipo de dependências de contexto, desde que tenha informado adequadamente a política de adaptação;
- adaptatividade de aplicações é tratado como um assunto em separado da aplicação;
- além de adaptações para as aplicações podem, também, serem definidas adaptações para o *middleware*;
- facilidade de manutenção e reutilização das diferentes informações necessárias ao modelo, por exemplo, perfil de adaptação da aplicação;
- facilidade e maior agilidade no desenvolvimento de novas aplicações;
- descrição e definição dos estados de cada elemento ativo e das ações a serem tomadas pelo gerenciador de adaptação do ambiente ubíquo, fora da aplicação;
- possibilidade de inferências para a tomada de decisões de adaptação e descrição das diferentes políticas de adaptação necessárias para a adaptação dinâmica.

7 PLANO DE TRABALHO

A seguir será detalhada a metodologia que está sendo adotada para alcançar os objetivos propostos neste trabalho e que terão continuidade nos esforços de estudo e pesquisa da disciplina de Dissertação II.

7.1 Etapas de Trabalho

Revisão bibliográfica

Revisão da literatura relacionada à Computação Ubíqua (UbiComp), Adaptação ao Contexto, Computação Autônoma e Ontologias. Particularmente, quanto à UbiComp e adaptação ao contexto, aprofundar aspectos relacionados às tecnologias associadas, bem como identificar e revisar os principais projetos relacionados, sistematizando os diferentes aspectos relativos aos mecanismos de controle da adaptação ao contexto utilizados. Na Computação Autônoma ou Autonômica, verificar e identificar as tecnologias que poderão ser utilizadas na concepção de um mecanismo de controle de adaptação genérico. Em Ontologias, estudar tecnologias para geração de um modelo ontológico para controle da adaptação em Sistemas Ubíquos.

Estudar o *middleware* EXEHDA

Estudo do projeto EXEHDA enquanto ambiente de execução para Computação Ubíqua, revisando seus fundamentos e as decisões inerentes a concepção dos diversos módulos de sua arquitetura, além de aprofundar os aspectos relativos aos mecanismos responsáveis pela adaptação funcional e não-funcional. Avaliar também, a relação destes mecanismos com os outros componentes da arquitetura de EXEHDA.

Estudar as Políticas de Adaptação

Estudo das principais políticas de controle de adaptação ao contexto hoje utilizadas na Computação Autonômica e na Computação Ubíqua. Selecionar e/ou criar uma política a ser adotada na modelagem do mecanismo de adaptação.

As políticas de adaptação devem contemplar variadas noções de adaptação, permitindo identificar os requisitos de propósito geral e específico para expressar adaptabilidade ao contexto, os quais podem ser:

- identificação dos elementos de contexto (entidades) que compõem o ambiente e modelam o contexto de interesse (classes no OWL): usuários, dispositivos, serviços, localização, coleções de códigos, de interfaces para dispositivos e serviços;

- descrição do comportamento adaptativo: coleções de ações e o respectivo conjunto de restrições onde estas podem ocorrer;
- descrição de preferências de usuários para determinadas situações e aplicações;
- descrição de políticas: regras de propósito geral ou particular que orientam as decisões da adaptação realizadas pelo gerenciador de adaptação;
- inferências a partir das informações da ontologia para subsidiar as tomadas de decisões do gerenciador de adaptação;
- possibilidade de usar histórico das adaptações realizadas para subsidiar a tomada de decisões de adaptação;
- estabelecimento de um padrão para descrições, características, e configurações dos elementos do contexto.

Estruturar o Modelo Semântico para o Controle da Adaptação Dinâmica Estruturar e criar as ontologias de domínio e as ontologias como artefato de software, que irão compor o modelo semântico do mecanismo de controle de adaptação dinâmica ao contexto. Criar as classes, sub-classes e o relacionamento entre elas para cada ontologia do modelo. O modelo será concebido baseando-se nas políticas estabelecidas na etapa anterior.

Modelar o Mecanismo de Controle da Adaptação Dinâmica ao Contexto para o EXEHDA

Modelagem do mecanismo de adaptação que será utilizado tanto para adaptação das aplicações, quanto do próprio *middleware*, em tempo de execução. Modelagem do modelo semântico utilizado pelo serviço de adaptação. Estabelecer o relacionamento e a interação do serviço de controle da adaptação dinâmica com os demais serviços fornecidos pelo *middleware* EXEHDA.

Implementar e testar o mecanismo de adaptação no *middleware* EXEHDA

Implementar o mecanismo modelado e concebido, considerando as especificações e serviços do EXEHDA enquanto *middleware* para aplicações ubíquas. Implementar adaptação funcional e não-funcional. Ao longo do trabalho de implementação serão realizados testes periódicos.

Avaliação do Andamento do Trabalho

Avaliação e reconsideração sobre as atividades programadas e realizadas, tendo como parâmetro as proposições feitas no Seminário de Andamento.

O trabalho será avaliado ao longo de seu desenvolvimento, quanto ao atendimento dos requisitos impostos pela computação ubíqua, particularmente pelo modelo computacional do EXEHDA, e as demandas de aplicações adaptáveis ao contexto.

Escrita da Dissertação

Redigir, progressivamente, o texto da dissertação. A redação será feita a medida que as atividades forem sendo realizadas, de maneira incremental. As revisões terão periodicidade quinzenal. Será utilizado software para controle de versões (SVN) localizado no servidor "olaria.ucpel.tche.br".

7.2 Cronograma

Atividades/Meses	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
1	o	o	o	o								
2				o	o							
3					o	o						
4					o	o	o	x				
5						o	o	x	x			
6								x	x	x	x	
7									x	x	x	x
8	o	o	o	o	o	o	o	x	x	x	x	x
9								x				
10												x

Atividades (o = já realizado, x = a realizar):

1. Revisão bibliográfica sobre o escopo do trabalho: Computação Ubíqua, Controle da Adaptação, Computação Autonômica e Ontologias.
2. Estudo do *middleware* EXEHDA.
3. Estudo dos tipos de Políticas de Adaptação, e de metodologias de especificações das mesmas.
4. Definição do Modelo Semântico para o Controle da Adaptação Dinâmica.
5. Modelagem do Mecanismo de Controle da Adaptação Dinâmica ao Contexto para o EXEHDA.
6. Implementação e testes do mecanismo de adaptação no *middleware* EXEHDA.
7. Elaboração de artigos sobre o Tema da Dissertação.
8. Escrita da Dissertação.
9. Seminário de Andamento.
10. Defesa da Dissertação.

8 CONSIDERAÇÕES FINAIS

Um aspecto fundamental dos sistemas computacionais ubíquos é a sua disponibilidade a todo o momento nos ambientes físicos em que estiverem envolvidos. Para isso, as aplicações precisam ”entender” o ambiente e se adaptar ao mesmo, interagindo com o contexto de seu interesse (MACIEL; ASSIS, 2004). Essa nova classe de sistemas computacionais, sensíveis ao contexto, e que se adaptam continuamente ao ambiente, exploram a natureza dinâmica e a mobilidade do usuário, o que introduz novos desafios de desenvolvimento. Em contrapartida, aplicações que se adaptam ao espaço ubíquo, podem otimizar e automatizar atividades do usuário, tornando a sua vida mais confortável, podendo fazer com que suas necessidades, sejam atendidas com mais facilidade e, muitas vezes até, sem que ele mesmo tenha que solicitá-la. A computação ubíqua é centrada no usuário e é reconhecida como um novo paradigma, decorrente do rápido avanço tecnológico dos dispositivos móveis, redes sem fio, redes de sensores, dispositivos e sistemas inteligentes, sistemas distribuídos, grades computacionais, *Web Services* e *Web* Semântica.

Ontologias são especificações formais dos conceitos de um determinado domínio, constituindo o núcleo da Web Semântica. Aplicações semânticas buscam interpretar o significado dos dados, permitindo estabelecer relacionamentos e inferências entre os mesmos. As principais características das ontologias são: (i) otimizadas para a representação de conhecimento estruturado em um nível elevado de abstração, favorecendo o entendimento comum das informações compartilhadas; (ii) os modelos de domínio podem ser disponibilizados na Internet e compartilhados entre múltiplas aplicações; (iii) baseada em dialetos não ambíguos da lógica formal, permitindo a utilização de serviços inteligentes baseados no raciocínio (*reasoning*), possibilitando em tempo de execução, a definição dinâmica de classes, a reclassificação de instâncias e a execução de consultas lógicas complexas; (iv) as linguagens operam sobre estruturas similares às linguagens orientadas a objeto, e podem ser eficazmente integradas aos componentes de software tradicionais.

Também foi considerado no desenvolvimento do trabalho um outro novo paradigma, a Computação Autônoma. Este tipo de computação representa que os sistemas computacionais deveriam desempenhar funções automáticas de configuração, tratamento, otimização e proteção, substituindo a especificação destas tarefas complexas por políticas descritas em alto nível por usuários e administradores. A proposta que está sendo concebida explora um dos aspectos considerados centrais na definição de políticas de controle na Computação Autônoma, qual seja a função de utilidade. Esta decisão foi construída a partir de de uma revisão de literatura em Computação Autônoma. A função utilidade, utilizada na especificação do perfil da aplicação e na decisão de adaptação da nossa proposta, tem por base proporcionar uma maior variabilidade de adaptação e uma maior adequação

às necessidades do usuário. A utilidade é utilizada com os valores dos parâmetros dos vários atributos, bem como o peso que reflete as prioridades do atributo ou das necessidades do usuário. Será considerada a possibilidade de interdependência entre os atributos.

Na tabela comparativa de Modelos *Context-aware* versus características de ambiente ubíquo, apresentada na seção 5.2.1, constatou-se que as abordagens consideradas atendem alguns ou grande parte dos requisitos introduzidos pela UbiComp. Os modelos focam alguns aspectos e deixam lacunas em outros. Por exemplo, a maioria dos *middlewares* focam a etapa de aquisição da informação contextual, levando em conta questões como descoberta de nós e serviços em redes heterogêneas, porém não abordam a forma como a informação é representada e como é feita a inferência sobre a mesma. Já as abordagens que usam ontologias, lidam bem com a parte de representação da informação, porém tem menos preocupação com a forma de obtenção da informação, muitas vezes não provendo nenhum suporte para isso. Além disso, as soluções, de um modo geral, referem-se a um determinado domínio de aplicação, principalmente as soluções baseadas em ontologias, ou à determinada tecnologia de rede, mais comum nos *middlewares*.

O controle da adaptação ainda é, na sua quase totalidade, específico a determinado contexto ou a determinada aplicação. Os trabalhos existentes ainda não possuem uma solução genérica e padronizada para as aplicações ubíquas.

Considerando estes aspectos, a proposta do trabalho em andamento tem por objetivo criar um modelo de controle da adaptação dinâmica de aplicações em ambiente ubíquo. A idéia central é controlar estas adaptações quando da tomada de decisões considerando o contexto, com base em informações monitoradas, informações semânticas e inferências a partir destas mesmas informações. A sugestão consta de um mecanismo de adaptação genérico que possa ser utilizado por diferentes aplicações, em tempo de execução. Neste modelo temos, na ontologia de perfil da aplicação, todas as regras, políticas e ações globais e ações específicas de cada aplicação, utilizadas pelo servidor de adaptação, para a partir destes dados, das mudanças do contexto, das preferências e necessidades do usuário, inferir o que e onde a ação de adaptação deverá ser executada. O modelo de controle de adaptação ao contexto proposto possibilita uma evolução incremental das especificações de políticas, regras e ações de adaptação, permitindo a reutilização e a customização destas para o desenvolvimento de novas aplicações *context-aware*. Além disso, estão sendo previstos mecanismos para administrar situações de conflito quando da tomada de decisões do servidor de adaptação, através do uso de funções de utilidade e dos históricos das adaptações. Estes mecanismos estão sendo concebidos com a premissa de adaptar as aplicações da computação ubíqua ao ambiente sem, ou com mínima, intervenção explícita do usuário.

O mecanismo será concebido para integrar o Subsistema de Adaptação e Reconhecimento de Contexto do middleware EXEHDA, sendo denominado EXEHDA-DA (*Dynamic Adaptation*).

Diversas questões de fundo irão acompanhar o desenvolvimento deste trabalho. A usabilidade e interação humano-computador são importantes questões que precisam de mais compreensão. Será que o usuário aceitará sistemas que se adaptam automaticamente? Com que frequência pode uma aplicação adaptar-se sem se tornar um aborrecimento para o usuário? E a confiança do usuário em um sistema que muda automaticamente? Isso leva a outra área de problema em aberto: como validar e testar um sistema de software auto-adaptável que é capaz de lidar com adaptações imprevistas em tempo de execução? Como fica a segurança e a confiança na informação de contexto

distribuída? Muitas destas questões só podem ser respondidas através de experimentação.

8.1 Publicações Realizadas

No decorrer da concepção do EXEHDA-DA, seus resultados parciais foram divulgados através dos trabalhos a seguir:

- CIC-UCPEL 2008: WARKEN, Nelsi, PALAZZO, Luiz A. M., YAMIN, Adenauer C. Ontologias no Controle da Adaptação em Espaço Ubíquo.Congresso de Iniciação Científica da Universidade Católica de Pelotas. Pelotas, RS. Mostra de Pós-Graduação, 2008.

- ERAD 2009: WARKEN, Nelsi, PALAZZO, Luiz A. M., YAMIN, Adenauer C. Uma Contribuição ao Controle da Adaptação na Computação Ubíqua. In: 9a Escola Regional de Alto Desempenho, 2009, Caxias do Sul-RS. ERAD 2009 - Fórum de Pós-Graduação da 9a Escola Regional de Alto Desempenho, 2009. p.133-134.

- ERAD 2009: NEVES, Vilnei Marins de Freitas, YAMIN, Adenauer C., WARKEN, Nelsi. Explorando Sensibilidade ao Contexto com MoCA. In: 9a Escola Regional de Alto Desempenho, 2009, Caxias do Sul-RS. ERAD 2009 - Fórum de Iniciação Científica da 9a Escola Regional de Alto Desempenho, 2009. p.193-196.

- CSBC 2009: WARKEN, Nelsi, YAMIN, Adenauer, AUGUSTIN, Iara, GEYER,Cláudio. Controle da Adaptação na UbiComp. In: XXIX Congresso da Sociedade Brasileira de Computação, 2009, Bento Gonçalves-RS. SBCUP - I Simpósio Brasileiro de Computação Ubíqua e Pervasiva - 2009.

- WIKI de Trabalho: WARKEN, Nelsi. Produção no Curso de Mestrado em Ciência da Computação do PPGINF/UCPEL. Disponível: http://olaria.ucpel.tche.br/nelsiw/. 2009.

Adaptação em tempo de execução será um requisito importante para os futuros sistemas de software em ambientes de computação ubíqua. Tecnologias tais como a computação móvel, redes *ad-hoc*, infra-estruturas de computação ubíqua, descoberta dinâmica de serviços, claramente contribuem para esta tendência. A auto-adaptação levará não só a aplicações mais úteis e versáteis, mas também a aplicações mais robustas e mais confiáveis.

REFERÊNCIAS

ABOWD, G. D.; MYNATT, E. D. Charting past, present, and future research in ubiquitous computing. **ACM Transactions on Computer-Human Interaction**, [S.l.], v.7, n.1, p.29–58, Mar. 2000.

AHMED, S.; AHAMED, S. I.; SHARMIN, M.; HAQUE, M. M. **Self-healing for autonomic pervasive computing**. [S.l.]: ACM, 2007. 110-111p.

ALMEIDA, D. R. de. **Omnipresent-Um sistema ciente de contexto baseado em arquitetura orientada**. 2006. Tese de Mestrado em Informática — Centro de Engenharia Elétrica e Informática / Universidade Federal de Campina Grande, Campina Grande,PB.

ANDERSEN, K. V. B.; CHENG, M.; KLITGAARD-NIELSEN, R. **Online Aalborg Guide - Development of a Location-Based Service**. Aalborg Universitet: Technical report, 2003.

AUGUSTIN, I. **Abstrações para uma Linguagem de Programação Visando Aplicações Móveis Conscientes do Contexto em um Ambiente de Pervasive Computing**. 2003. Tese de Doutorado em Ciência da Computação — Instituto de Informática/UFRGS, Porto Alegre,RS.

AYED, D.; BELHANAFI, N.; TACONET, C.; BERNARD, G. **Deployment of Component-based Applications on Top of a Context-aware Middleware**. [S.l.]: IASTED/ACTA Press, 2005. 414–419p.

BARDRAM, J. E. **The Java Context Awareness Framework (JCAF) A Service Infrastructure and Programming Framework for Context-Aware Applications**. [S.l.]: Springer, 2005. 98–115p. (Lecture Notes in Computer Science, v.3468).

BERNERS-LEE; HENDLER, T.; LASSILA, J. The Semantic Web. **Scientific American**, [S.l.], May 2001.

BRICKLEY, D.; GUHA, R. V. **RDF Vocabulary Description Language 1.0**: RDF Schema. World Wide Web Consortium, Recommendation REC-rdf-schema-20040210.

BRUMITT, B.; MEYERS, B.; KRUMM, J.; KERN, A.; SHAFER, S. **Easyliving**: Technologies for intelligent environments. [S.l.]: Microsoft - Technical report, 2000.

CAHILL, V.; GRAY, E.; JENSEN, C.; SEIGNEUR, J.-M.; CHEN, Y.; SHAND, B.; DIMMOCK, N.; TWIGG, A.; BACON, J.; ENGLISH, C.; WAGEALLA, W.; TERZIS, S.;

NICON, P. Using Trust for Secure Collaboration in Uncertain Environments. **IEEE Pervasive Computing**, [S.l.], v.2, n.3, p.52–61, 2003.

CAPRA, L.; EMMERICH, W.; MASCOLO, C. CARISMA: Context-aware reflective middleware system for mobile applications. **IEEE Transactions on Software Engineering**, [S.l.], v.29, n.10, p.929–945, 2003.

CHAARI, T.; LAFOREST, F. Adaptation in context-aware pervasive information systems: the SECAS project. **International Journal of pervasive Computing and Communications**, [S.l.], v.3, n.4, p.400–425, 2007.

CHEN, H.; FININ, T.; JOSHI, A. **An Ontology for Context-Aware Pervasive**.

CHEN, H.; PERICH, F.; FININ, T. W.; JOSHI, A. **SOUPA**: Standard Ontology for Ubiquitous and Pervasive Applications. [S.l.]: IEEE Computer Society, 2004. 258-267p.

COSTA, C. A. da; YAMIN, A. C.; GEYER, C. F. R. Toward a General Software Infrastructure for Ubiquitous Computing. **IEEE Pervasive Computing**, Los Alamitos, CA, USA, v.7, n.1, p.64–73, 2008.

DEY, A.; ABOWD, G.; SALBER, D. **A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications**. [S.l.]: Human-Computer Interaction, 2001. 97-166p.

DEY, A. K.; ABOWD, G. D. **CybreMinder**: A Context-Aware System for Supporting Reminders. 172–186p.

DICKINSON, I. **Jena Ontology API**. Disponível em: <http://jena.sourceforge.net/ontology/>. Acesso em Novembro de 2008.

DOURISH, P. What we talk about when we talk about context. **Personal and Ubiquitous Computing**, [S.l.], v.8, n.1, p.19–30, 2004.

DÜRR, F.; HÖNLE, N.; NICKLAS, D.; BECKER, C.; ROTHERMEL, K. **Nexus–A Platform for Context-Aware Applications**. [S.l.]: Hagen: Informatik-Bericht der FernUniversität Hagen, 2004. 15–18p.

FENSEL, D. **Ontologies-Silver Bullet for Knowledge Management and Electronic Commerce**. Berlin: [s.n.], 2000.

FIPA. **FIPA - Foundation for Intelligent Physical Agents - Device Ontology Specification**. Geneva, Switzerland: [s.n.], 2001. Disponível em: <http://www.fipa.org/specs/fipa00091/XC00091C.pdf>. Acesso em Novembro de 2008.

FLOCH, J.; STAV, E.; HALLSTEINSEN, S. **Interfering effects of adaptation**: Implications on self-adapting systems architecture. [S.l.]: Springer, Berlin, 2006. 64-69p.

GARLAN, D. Aura: Distraction-Free Ubiquitous Computing. **Lecture Notes in Computer Science**, [S.l.], v.2254, p.1–??, 2001.

GARLAN, D.; CHENG, S.-W.; HUANG, A.-C.; SCHMERL, B.; STEENKISTE, P. Rainbow: Architecture-based self-adaptation with reusable infrastructure. **Computer**, [S.l.], v.37, n.10, p.46–54, 2004.

GEIHS, K.; BARONE, P.; ELIASSEN, F.; FLOCH, J.; FRICKE, R.; GJØRVEN, E.; HALLSTEINSEN, S.; HORN, G.; KHAN, M. U.; MAMELLI, A.; PAPADOPOULOS, G. A.; PASPALLIS, N.; REICHLE, R.; STAV, E. A comprehensive solution for application-level adaptation. **Software Practice & Experience**, [S.l.], v.39, n.4, p.385–422, 2009.

GRIMM, R.; BERSHAD, B. N. **System Support for Pervasive Applications**. [S.l.]: Springer, 2003. 212–217p. (Lecture Notes in Computer Science, v.2584).

GU, T.; PUNG, H. K.; ZHANG, D. A service-oriented middleware for building context-aware services. **J. Network and Computer Applications**, [S.l.], v.28, n.1, p.1–18, 2005.

HECKMANN, D.; SCHWARZKOPF, E.; MORI, J.; DENGLER, D.; KRÖNER, A. **The User Model and Context Ontology GUMO Revisited for Future Web 2.0 Extensions**. [S.l.]: CEUR-WS.org, 2007. v.298.

CRNKOVIC, I.; STAFFORD, J. A.; SCHMIDT, H. W.; WALLNAU, K. C. (Ed.). **Component Technology and QoS Management**. [S.l.]: Springer, 2004. 249–263p. (Lecture Notes in Computer Science, v.3054).

HELAL, S.; MANN, W.; EL-ZABADANI, H.; KING, J.; KADDOURA, Y.; ; JANSEN, E. The gator tech smart house: A programmable pervasive space. **IEEE Computer**, [S.l.], p.50–60, march 2005.

HENRICKSEN, K.; INDULSKA, J.; RAKOTONIRAINY, A. **Using context and preferences to implement self-adapting pervasive computing & applications**. [S.l.: s.n.], 2006.

HILERA, J. R.; RUIZ, F. **Ontologies in Ubiquitous Computing**. v.208.

HORN, P. **Autonomic Computing-IBM's Perspective on the State of Information Technology**. Disponível em: <http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf>. Acesso em Novembro 2008.

JOSÉ, R.; MOREIRA, A. J. C.; RODRIGUES, H.; DAVIES, N. The AROUND Architecture for Dynamic Location-Based Services. **MONET**, [S.l.], v.8, n.4, p.377–387, 2003.

KEENEY, J.; CAHILL, V. **Chisel**: A Policy-Driven, Context-Aware, Dynamic Adaptation Framework. [S.l.]: IEEE Computer Society, 2003. 3–14p.

KEPHART, J. O.; CHESS, D. M. Cover Feature: The Vision of Autonomic Computing. **IEEE Computer**, [S.l.], v.36, n.1, p.41–52, 2003.

KEPHART, J. O.; DAS, R. Achieving self-management via utility functions. **IEEE Internet Computing**, [S.l.], v.11, n.1, p.40–48, 2007.

KNUBLAUCH, H.; RECTOR, A. L.; DRUMMOND, N.; HORRIDGE, M.; ROGERS, J.; STEVENS, R.; WANG, H.; WROE, C. **OWL Pizzas-Practical Experience of Teaching OWL-DL**: Common Errors & Common Patterns. [S.l.]: Springer, 2004. 63–81p. (Lecture Notes in Computer Science, v.3257).

LIN, P.; MACARTHUR, A.; LEANEY, J. **Defining Autonomic Computing**: A Software Engineering Perspective. [S.l.]: IEEE Computer Society, 2005. 88–97p.

LISZKA, K. J.; YORK, D. W.; ROSENBAUM, D. S.; MACKIN, M. A.; LICHTER, M. J. **Remote Monitoring of a Heterogeneous Sensor Network for Biomedical Research in Space**. 829–833p.

LUTFIYYA, H.; MOLENKAMP, G.; KATCHABAW, M.; BAUER, M. Issues in managing soft QoS requirements in distributed systems using a policy-based framework. **Proceedings of 2nd International Workshop on Policies for Distributed Systems and Networks (POLICY 01)**, [S.l.], p.185–201, 2001.

MACIEL, R. S. P.; ASSIS, S. R. **Middleware**: Uma solução para o desenvolvimento de aplicações distribuídas. [S.l.]: In: Científico - Ano IV, 2004.

MCBRIDE, B. **An Introduction to RDF and the Jena RDF API**. Disponível em: <http://jena.sourceforge.net/tutorial/RDF_API/index.html>. Acesso em Novembro de 2008.

MCKINLEY, P. K.; SADJADI, S. M.; KASTEN, E. P.; CHENG, B. H. C. **Composing adaptive software**. [S.l.: s.n.], 2004. 56-64p. v.37, n.7.

MENASCE, D. A.; KEPHART, J. O. **Guest Editors' Introduction**: Autonomic Computing. [S.l.: s.n.], 2007. 18–21p. v.11, n.1.

MOREIRA BUBLITZ frederico. **Infra-estrutura para o Desenvolvimento de Aplicações Cientes de Contexto em Ambientes Pervasivos**. 2007. Tese de Mestrado em Informática — Centro de Engenharia Elétrica e Informática / Universidade Federal de Campina Grande, Campina Grande,PB.

NAKANISHI, Y.; TAKAHASHI, K.; TSUJI, T.; HAKOZAKI, K. ICAMS: A Mobile Communication Tool Using Location and Schedule Information. **Lecture Notes in Computer Science**, [S.l.], v.2414, p.239–??, 2002.

NORMAN, D. A. **The Invisible Computer**. Cambridge-MA: MIT Press, 1998.

O'DONNELL, T.; LEWIS, D.; WADE, V. **Intuitive Human Governance of Autonomic Pervasive Computing Environments**. [S.l.]: IEEE Computer Society, 2005. 532–536p.

OREIZY, P.; GORLICK, M.; TAYLOR, R.; HEIMHIGNER, D.; JOHNSON, G.; MEDVIDOVIC, N.; QUILICI, A.; ROSENBLUM, D.; WOLF, A. **Architecture-based approach to self-adaptive software**. [S.l.]: IEEE Intelligent Systems and Their Applications, 1999. 54-62p. v.14, n.3.

PARRA, C. A.; DUCHIEN, L. Model-Driven Adaptation of Ubiquitous Applications. **ECEASST**, [S.l.], v.11, 2008.

POLADIAN, V.; SOUZA, J.; GARLAN, D.; SHAW, M. Dynamic configuration of resource-aware services. **Proceedings of 26th International Conference on Software Engineering**, **Edinburgh**, **U.K.**, [S.l.], 2004.

RANGANATHAN, A.; CAMPBELL, R. H. **Autonomic Pervasive Computing Based on Planning**. [S.l.]: IEEE Computer Society, 2004. 80-87p.

REYNOLDS, D. **Jena 2 Inference support**. Disponível em: <http://jena.sourceforge.net/inference/index.html>. Acesso em Novembro de 2008.

ROMAN, M.; AL. et. Gaia: a Middleware Infrastructure to Enable Active Spaces. **IEEE Pervasive Computing**, New York, v.1, n.4, p.74–83, Oct 2002.

SAHA, D.; MUKHERJEE, A. Perspectives: Pervasive Computing: A Paradigm for the 21st Century. **Computer**, [S.l.], v.36, n.3, p.25–31, mar 2003.

SALBER, D.; DEY, A. K.; ABOWD, G. D. **The Context Toolkit**: Aiding the Development of Context-Enabled Applications. 434–441p.

SATYANARAYANAN, M. **Pervasive Computing**: Vision and Challenges.

MEERSMAN, R.; TARI, Z. (Ed.). **Component-Based Dynamic QoS Adaptations in Distributed Real-Time and Embedded Systems**. [S.l.]: Springer, 2004. 1208–1224p. (Lecture Notes in Computer Science, v.3291).

SORENSEN, C.-F.; WU, M.; SIVAHARAN, T.; BLAIR, G. S.; OKANDA, P.; FRIDAY, A.; DURAN-LIMON, H. A. **A context-aware middleware for applications in mobile Ad Hoc environments**. [S.l.]: ACM, 2004. 107–110p.

SOUZA, M. H. L. de. **Uma infra-estrutura para o desenvolvimento de aplicações pervasivas autoconfiguráveis**. 2008. Tese de Mestrado em Informática — Centro de Engenharia Elétrica e Informática / Universidade Federal de Campina Grande, Campina Grande,PB.

TONINELLI, A.; MONTANARI, R.; KAGAL, L.; LASSILA, O. **Proteus**: A Semantic Context-Aware Adaptive Policy Model. [S.l.]: IEEE Computer Society, 2007. 129–140p.

VÁZQUEZ, J. I.; IPIÑA, D. L. de; SEDANO, I. **SOAM-An Environment Adaptation Model for the Pervasive Semantic Web**. [S.l.]: Springer, 2006. 108–117p. (Lecture Notes in Computer Science, v.3983).

VENECIAN, L. **Revisando a Sensibilidade ao Contexto na Computação Ubíqua**. 2008. Trabalho Individual Mestrado Ciência Computação — UCPEL, Pelotas, RS. Disponível: http://olaria.ucpel.tche.br/luthiano/lib/exe/fetch.php?media=lvti.pdf.

VERZULLI, J. **Using the Jena API to process RDF**. Disponível em: <http://www.xml.com/pub/a/2001/05/23/jena.html>. Acesso em Novembro 2008.

WALSH, W. E.; TESAURO, G.; KEPHART, J. O.; DAS, R. **Utility Functions in Autonomic Systems**. [S.l.]: IEEE Computer Society, 2004. 70–77p.

WANG, X. H.; GU, T.; ZHANG, D. Q. **Ontology Based Context Modeling and Reasoning using OWL**.

WANT, R.; PERING, T. **System challenges for ubiquitous & pervasive computing**. [S.l.]: ACM, 2005. 9–14p.

WEISER; BROWN. **The Coming Age of Calm Technology**.

WEISER, M. The computer for the 21st century. **Scientific American**, [S.l.], v.265, n.3, p.94–104, 1991.

WEISER, M. Ubiquitous Computing. **IEEE Computer**, [S.l.], v.26, n.10, p.71–72, Oct. 1993.

WEISER, M. The Invisible Interface: Increasing the Power of the Environment through Calm Technology. **Lecture Notes in Computer Science**, [S.l.], v.1370, p.1–??, 1998.

WEISSENBERG, N.; VOISARD, A.; GARTMANN, R. **Using ontologies in personalized mobile applications**. [S.l.]: ACM, 2004. 2–11p.

WHITE, S. R.; HANSON, J. E.; WHALLEY, I.; CHESS, D. M.; KEPHART, J. O. **An Architectural Approach to Autonomic Computing**. [S.l.]: IEEE Computer Society, 2004. 2-9p.

YAMIN, A. C. **Arquitetura para um Ambiente de Grade Computacional Direcionada às Aplicações Distribuídas, Móveis e Conscientes de Contexto da Computação Pervasiva**. 2004. Tese de Doutorado em Ciência da Computação — Instituto de Informática/UFRGS, Porto Alegre-RS.

YAMIN, A. C.; AUGUSTIN, I.; BARBOSA, J.; GEYER, C. **EXEHDA-Adaptative Middleware for Building a Pervasive Grid Environment**. Amsterdam: [s.n.], 2005. 203-219p. v.135.

YAMIN, A. C.; AUGUSTIN, I.; BARBOSA, J.; GEYER, C. **ISAM**: A software architecture for pervasive computing. Arequipa-Peru: [s.n.], 2005. 1–13p. v.8, n.11.

ANEXO A GLOSSÁRIO

Neste glossário, os conceitos listados não necessariamente refletem a totalidade das propriedades requeridas à Computação Ubíqua (*UbiComp*), mas representam seus principais atributos. A computação ubíqua integra estes conceitos tendo como objetivo fundamental o aumento da qualidade de vida dos usuários. Desenvolve-se de forma onipresente e adaptável ao contexto, fazendo com que não seja mais o usuário que se desloca até a rede, mas a rede que passa a envolver os usuários e os objetos numa conexão generalizada.

Ad-hoc: tradicionalmente refere-se a arquiteturas de redes, independentes de infra-estrutura fixa, enquanto a idéia da espontaneidade agrega mais amplitude ao termo, numa noção não exclusivamente infra-estrutural, como também de serviços que dinamicamente se encontram, se conhecem e se interagem para a troca de informações.

Ambiente Inteligente: é o conjunto de tecnologias que, trabalhando de maneira integrada, permite o entendimento automático de certas situações, ativando instruções ou respondendo comandos pré-programados, mesmo sem instruções explícitas do usuário. Podemos exemplificar um Ambiente Inteligente quando computadores podem identificar a presença humana, desligando luzes e dispositivos na ausência dos mesmos, ou ativando o aquecedor ou ar-condicionado na temperatura ideal para cada usuário. Se estas atividades acontecem sem a intervenção do usuário, podemos chamá-la também de Computação Invisível.

Calm Computing: a automatização dos sistemas e dispositivos pode sobrecarregar o usuário com o excesso de informações e funcionalidades. A área que trata da solução para o problema de excesso de informações expostas aos usuários é chamada de *Calm Technology* (WEISER, 1998).

Computação Móvel: é a capacidade de um dispositivo computacional e os serviços associados ao mesmo serem móveis, permitindo este ser carregado ou transportado mantendo-se conectado a rede ou a Internet. Verifica-se este conceito hoje na utilização de redes sem fio, acesso à internet através de dispositivos celulares ou telefones celulares. Também podemos verificar o crescimento de aplicações *Bluetooth* seja através de fones de ouvido sem fios, impressoras fotográficas ou mouses sem fio.

Consciência de Contexto: indo além da interação física do usuário, sensores espalhados pelo meio (ambiente) poderão detectar o que acontece ao redor, o que as de-

mais pessoas em uma reunião estão fazendo ou dizendo e até mesmo outros dados como luminosidade, temperatura, expressões faciais, direção do olhar entre outros. Estes dados poderão ser disponibilizados para as aplicações e através de modelos computacionais, as interfaces poderão interpretar o que está acontecendo ao redor do usuário e adaptar-se ou tomar decisões, alterando e facilitando a comunicação com o usuário.

Contexto: é qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é qualquer pessoa, lugar ou objeto que é considerado relevante para a interação entre usuário e a aplicação, incluindo o próprio usuário e a própria aplicação (DEY; ABOWD; SALBER, 2001).

A riqueza de um contexto da computação ubíqua pode ser constituída de um conjunto de atributos físicos (e.g., localização de pessoas e dispositivos no espaço), fisiológicos (e.g., temperatura corporal e batimentos cardíacos), psicológicos (e.g., alegria, ansiedade, angústia, irritação), histórico de atividades, padrão de comportamento, entre outros (SATYANARAYANAN, 2001). Com esta bagagem de informações, assim como uma secretária executiva deve ser capaz de antecipar os desejos de seu chefe e, diante de um cenário de irritação, incomodá-lo apenas em situações de emergência, a computação ubíqua também deve buscar a coleta de dados para evitar ao máximo a necessidade de chamar a atenção de seus usuários. Estas atividades que objetivam diminuir o envolvimento do usuário, através da antecipação ou automatização de decisões, são premissas também preconizadas pela *Calm Computing*.

Espaços inteligentes (*smart spaces*): são naturalmente definidos pela colaboração intensa entre elementos computacionais e componentes do mundo físico.

Interfaces naturais: em *smart-spaces*, é preciso buscar técnicas para que os recursos normalmente utilizados no dia-a-dia de uma sociedade, como gestos, voz e mesmo olhares, permaneçam como meio de comunicação entre homem e máquina. Se a proposta é integrar sutilmente elementos digitais ao mundo real sem influenciar na normalidade de seu funcionamento, precisamos manter interfaces naturais de interatividade entre o homem e o mundo que o rodeia (ABOWD; MYNATT, 2000).

Interoperabilidade espontânea: um sistema ubíquo pode ser fragmentado em componentes de infra-estrutura, de natureza fixa em relação ao *smart space*, e componentes espontâneos, entendidos como unidades de software para abstração de serviços, recursos ou dispositivos em constante deslocamento entre ambientes distintos (KINDENBERG e FOX, 2002). A interoperabilidade espontânea é uma característica desejável nas implementações da computação ubíqua, observada pela possibilidade de integração dinâmica entre componentes móveis e de infra-estrutura do sistema. Por exemplo, um ambiente ubíquo implementado em uma sala de reuniões de uma empresa. A interoperabilidade espontânea pode ser observada se, um cliente ou um fornecedor da organização é capaz de transferir a apresentação, armazenada em seu PDA, ao projetor da sala, sem qualquer intervenção manual para configurar o dispositivo.

Invisibilidade: quanto mais presente em nossas vidas uma tecnologia estiver, menos perceptível ela deve ser. O computador torna-se fundamental nas atividades cotidianas, mas dilui-se no mundo físico, tornando-se presente e imperceptível. A

tecnologia não deve exigir mais que nossa atenção periférica.

Prédios Inteligentes ou Automação Residencial: prédios e/ou casas onde a fechadura da porta principal é acionada pela íris do dono, as cortinas e as luzes são acionadas pela voz, o aquecedor, o ar condicionado, banheira, podem ser ativados pelo celular. Esta tecnologia já é disponível, inclusive no Brasil.

Pró-atividade: significa a capacidade do sistema em antecipar a intenção do usuário, como exemplificado na situação hipotética de Luciana, na seção 2.

Redes *Wireless*: o termo mais comum utilizado na Computação Móvel é o *Wireless* (sem fio) ou Wi-Fi (*wireless fidelity*), utilizado para designar pequenos receptores e transmissores de rádio no Reino Unido no início de sua invenção e hoje, muito mais avançada, acoplada aos computadores ou PDAs transmitindo sinais digitais. Com a evolução e o aumento da disponibilidade do acesso à Internet, diversos provedores (ISP) utilizaram dessa tecnologia para prover serviços a seus clientes. Com o barateamento dos custos dos equipamentos, corporações, prestadores de serviços e principalmente estabelecimentos comerciais passaram a prover acesso à Internet através de redes sem fios à seus clientes, em sua grande maioria cafeterias, livrarias, *shopings*, universidades, empresas popularizam seu uso. Estes pontos de acesso à Internet que possibilitam acesso sem fio à rede foram chamados de *Hotspot Wi-Fi*, e deram suporte ao crescimento do uso de *notebooks*, dispositivos portáteis, telefones celulares, *I-Phone*.

Sensibilidade ao Contexto: para que um sistema ubíquo, tenha o mínimo de intrusão no mundo real, sua infra-estrutura de software deve ser adaptativa ao contexto, ou seja, possuir uma base extensa de informações a respeito das pessoas e do ambiente que as abriga e, através destes dados, adaptar seu comportamento da melhor forma possível para completa satisfação das expectativas de seus usuários (DOURISH, 2004). O sucesso da pró-atividade do sistema é alavancado pela completude das informações de que dispõe. No exemplo de Luciana, seção 2, observamos uma suave integração do conhecimento de sistemas de diferentes níveis computacionais, como softwares de base para a identificação do congestionamento da rede *wireless*, sensores físicos para captação das condições meteorológicas e aplicativos corporativos para a avaliação do cronograma de vôos. Toda esta cooperação, quando implementada de forma ágil e transparente, traz ao usuário final a antecipação e automação da tomada de decisões, que lhe reservam tempo hábil para dedicação prioritária à sua atividade fim.

Sistemas RFID (*Radio-Frequency Identification*): esta tecnologia permite que um objeto dotado de um pequeno sensor com seu formato cadastrado previamente, seja passível de reconhecimento pelo computador. As etiquetas RFID podem substituir os códigos de barra, informando localização e diversas propriedades dos produtos.