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

Uma Proposta de Controle da Adaptação Dinâmica ao Contexto na Computação Ubíqua

por
Nelsi Warken

Dissertação apresentada como
requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Orientador: Prof. Dr. Adenauer Corrêa Yamin

DMII-2010/1-007

Pelotas, março de 2010

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 cotidianamente, 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, amiga e parceira, harmonizando e qualificando, diariamente, todos seus ambientes de contexto.

Agradeço a Embrapa Clima Temperado, chefia, aos colegas da Área de TI 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 aos integrantes do G3PD, Grupo de Processamento Paralelo Distribuído, em especial à Amanda, pela colaboração e apoio no desenvolvimento das aplicações empregadas nos estudos de caso.

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.

Agradeço às amigas Ariana e Iolanda pelo suporte, motivação, amizade e carinho diários.

Por último, agradeço a todos aqueles que não foram mencionados, mas que de alguma forma também contribuíram para a elaboração deste trabalho.

Uma longa viagem começa com um único passo.
LAO-TSÉ

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AMS	Arrythmia Monitoring System
API	Application Program Interface
ASCII	American Standard Code for Information Interchange
BNF	Backus-Naur Form
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
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ISAM	Infraestrutura 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
JVM	Java Virtual Machine
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 ao Processamento Semântico e Sistemas Autônomos na concepção de mecanismos de controle da adaptação ao contexto na Computação Ubíqua. 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 adaptativas, considerando o contexto em que estão inseridas. Ontologias são especificações formais dos conceitos de um determinado domínio, possibilitando que as aplicações semânticas possam interpretar o significado dos dados, e estabelecer relacionamentos e inferências entre os mesmos. A Computação Autônoma, por sua vez, 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 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. O mecanismo proposto chama-se EXEHDA-DA, EXEHDA-*Dynamic Adaptation*, serviço de controle da adaptação dinâmica ao contexto para o *middleware* EXEHDA. O EXEHDA-DA foi avaliado por dois estudos de caso, apresentando resultados satisfatórios quanto ao atendimento das demandas de ubiquidade dos mesmos.

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

TITLE: "A PROPOSAL FOR CONTROL OF DYNAMIC ADAPTATION TO CONTEXT IN UBIQUITOUS COMPUTING"

ABSTRACT

The main purpose of this study is to assess the use of concepts and technologies for the Semantic Processing and Autonomics Systems in the design of mechanisms to control the adaptation to the context in Ubiquitous Computing. In ubiquitous computing, the various computational systems interact with human beings all the time, regardless of the current position, 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 specific field. Applications should interpret the semantic meaning of the data, allowing the establishment of the 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 or programmers. Considering all these aspects, the proposed research has as focus on creating a proposal 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. The proposed mechanism is called EXEHDA-DA, EXEHDA-Dynamic Adaptation, service of dynamic adaptation control to the context for the middleware EXEHDA. The proposed model was evaluated by two case studies, presenting satisfactory results regarding the fulfillment of their demands of ubiquity.

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

1 INTRODUÇÃO

Este capítulo irá caracterizar o tema escolhido para o desenvolvimento do trabalho, as motivações para escolha do seu foco, assim como os objetivos e metas perseguidos no andamento das diferentes atividades de estudo e pesquisa que foram desenvolvidas.

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 dos aspectos referentes ao acesso ao ambiente computacional do usuário, isto é, ao espaço ubíquo do usuário, em qualquer lugar, todo o tempo com qualquer dispositivo. Nesta perspectiva 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 (FRAINER et al., 2007).

Na Computação Ubíqua as aplicações precisam monitorar 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 das modernas infraestruturas computacionais 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, continua um desafio de pesquisa em aberto (COSTA; YAMIN; GEYER, 2008). 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).

A avaliação de elementos do contexto, como localização, tempo e atividades do usuário, permite o desenvolvimento de aplicações sensíveis ao mesmo, bem como os respectivos procedimentos adaptativos (HENRICKSEN; INDULSKA; RAKOTONIRAINY, 2006).

Os esforços de estudo e pesquisa desta dissertação estão inseridos nas atividades

desenvolvidas pelo G3PD (Grupo de Pesquisa de Processamento Paralelo e Distribuído). O G3PD participa de um consórcio de pesquisa formado por diversas universidades do Rio Grande do Sul, o qual promove o desenvolvimento do *middleware* EXEHDA. O EXEHDA é um *middleware* adaptativo ao contexto, baseado em serviços, que tem por objetivo criar um ambiente ubíquo e gerenciar a execução de aplicações sobre o mesmo. As aplicações deste ambiente são distribuídas, móveis e adaptativas ao contexto.

O modelo de controle da adaptação proposto denomina-se EXEHDA-DA (*Execution Environment for Highly Distributed Applications - Dynamic Adaptation*, e tem a premissa de ser integrado ao *middleware* EXEHDA. Para sua avaliação foram desenvolvidos dois estudos de caso, cada um contemplando particularidades específicas do controle da adaptação 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

O paradigma introduzido pela Computação Ubíqua é uma tendência mundial, em função do rápido avanço tecnológico dos dispositivos móveis, redes sem fio, redes de sensores, sistemas e dispositivos inteligentes, sistemas distribuídos, grades computacionais, Sistemas Autônomos e outras tecnologias relacionadas a integração de sistemas. São tecnologias convergentes, que se integram sinergicamente no cotidiano dos usuários. Tendo em vista a dinamicidade, decorrente desta integração, a adaptação ao contexto torna-se um componente importante e fundamental na Computação Ubíqua (HILERA; RUIZ, 2006).

A concepção de ambientes de desenvolvimento para aplicações que se adaptem ao espaço ubíquo constitui um significativo desafio de pesquisa (COSTA; YAMIN; GEYER, 2008). Por sua vez, a disponibilização de modelos de programação que facilitem o desenvolvimento de novas aplicações, baseado em técnicas de reuso, também constitui uma frente de pesquisa em aberto.

Assim, motivado por estas idéias, este trabalho propõe-se a explorar o uso de processamento semântico baseado em Ontologias, bem como Sistemas Autônomos para Controle da Adaptação ao Contexto na Computação Ubíqua, contribuindo para que seja facilitado o desenvolvimento de aplicações *context-aware*.

Ontologia, além de ser um elemento importante no processamento semântico, se mostra um oportuno 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 contempla 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,

preceitua que os sistemas computacionais devem desempenhar funções automáticas de configuração, tratamento, otimização e proteção, substituindo a execução rotineira de tarefas complexas por políticas descritas em alto nível por usuários e administradores.

Apesar dos dois paradigmas apresentarem focos distintos, é possível identificar características comuns a ambos. 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 definem o termo de Computação Ubíqua Autônoma, nos casos em que os modelos computacionais apresentam características dos dois paradigmas (O'DONNELL; LEWIS; WADE, 2005; RANGANATHAN; CAMPBELL, 2004).

1.3 Objetivos

O objetivo central do trabalho é avaliar a exploração das tecnologias de processamento semântico 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.

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

- caracterizar as frentes de pesquisa relativas à proposição de controle de adaptação dinâmica ao contexto na Computação Ubíqua;
- definir premissas para a concepção de um serviço de controle para adaptação dinâmica ao contexto;
- caracterizar, definir e instanciar o modelo ontológico a ser utilizado pelo *middleware* para processamento semântico;
- propor um *framework* para instanciar a política de adaptação da aplicação;
- definir os comandos adaptativos para as aplicações ubíquas;
- propor um modelo computacional para o EXEHDA-DA, comprometido com as demais funcionalidades do *middleware* EXEHDA, para promover a adaptação de aplicações ubíquas;
- tomar decisões automáticas de adaptação, com nenhuma ou mínima intervenção do usuário;
- prover adaptação dinâmica, à nível de componente da aplicação, decidindo a melhor ação adaptativa em função da mudança de contexto;
- conceber um modelo para o controle da adaptação que possa ser reusável e customizado para diferentes aplicações e suas respectivas demandas de adaptação;

- prover e/ou selecionar ferramentas que viabilizem o emprego do modelo de controle proposto;

- fornecer subsídios para a elaboração de relatórios, artigos e trabalhos futuros, relacionados com o tema pesquisado.

A proposta consta de um modelo de adaptação genérico que será utilizado por diferentes 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 serviço 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 também pretende possibilitar uma evolução incremental das especificações de políticas, regras, ações e parâmetros de adaptação. Permitindo a reutilização e a customização destas para o desenvolvimento de novas aplicações *context-aware*. Além disso, o modelo 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.

O atingimento destes objetivos está inserido na estrutura do texto comentada na seção a seguir.

1.4 Estrutura do Texto

O texto é composto por sete capítulos e dois anexos.

No capítulo 1, que é esta introdução, é abordado o tema do trabalho, a motivação para escolha do mesmo e os objetivos perseguidos no desenvolvimento deste trabalho.

No capítulo 2 são tratados os fundamentos científicos e tecnológicos do trabalho, sendo explorados conceitos em Computação Ubíqua e Computação Autônoma. Também neste capítulo, são tratados aspectos pertinentes a Ontologias e ferramentas para seu processamento.

No capítulo 3 é apresentado o escopo do trabalho, o qual resume conceitos pertinentes ao Controle da Adaptação na UbiComp, discute projetos fortemente relacionados ao modelo proposto e, caracteriza as premissas consideradas para a concepção do EXEHDA-DA.

No capítulo 4 é tratada a modelagem do EXEHDA-DA, a qual contempla a definição do modelo semântico previsto, uma análise das funcionalidades do *framework* proposto FWADAPT, a discussão da estrutura proposta para o Serviço de Controle da Adaptação Dinâmica ao Contexto e os Comandos Adaptativos para uso das aplicações.

No capítulo 5 são tratadas as tecnologias de software empregadas na concepção do EXEHDA-DA.

No capítulo 6 são discutidos os resultados obtidos com dois estudos de caso, desenvolvidos com o objetivo de avaliar as funcionalidades do EXEHDA-DA.

No capítulo 7, são apresentadas as considerações finais, as quais compreendem uma discussão dos trabalhos relacionados ao EXEHDA-DA, um resumo das contribuições decorrentes do trabalho desenvolvido, um relato das principais conclusões, uma listagem das publicações realizadas e uma previsão de trabalhos futuros.

Por sua vez, o Anexo A, contempla uma descrição de 18 trabalhos relacionados ao controle da adaptação na UbiComp em geral.

O Anexo B, com o intuito de caracterizar em uma ordem cronológica os esforços dispendidos na realização do trabalho, apresenta um resumo da metodologia empregada para a concepção do mesmo.

2 FUNDAMENTOS CONCEITUAIS DO TRABALHO

Este capítulo contempla na sua primeira seção os principais conceitos pertinentes a Computação Ubíqua e a Computação Autônoma, apresentando resumos da fundamentação teórica, aspectos históricos, funcionalidades e exemplos de uso relativos a ambas. Esta seção também resume projetos de grande porte da Computação Ubíqua. Na segunda seção, são exploradas as tecnologias para processamento semântico, com destaque em Ontologias. Neste sentido são apresentados os principais conceitos, relacionados a ontologias, sua classificação em categorias, modelos ontológicos direcionados a Computação Ubíqua, e as linguagens consideradas mais importantes para o tratamento de ontologias.

2.1 Computação Ubíqua Autônoma

A Computação Ubíqua idealiza a existência de ambientes carregados de dispositivos computacionais que se integram ao cotidiano dos usuários de forma transparente, adaptando-se automaticamente de acordo com as situações observadas no contexto de seu interesse. Por sua vez, a Computação Autônoma preceitua que os sistemas computacionais devem desempenhar funções automáticas de configuração, tratamento, otimização e proteção, substituindo a execução rotineira de tarefas complexas por políticas descritas em alto nível por usuários, administradores ou desenvolvedores.

2.1.1 Computação Ubíqua

O termo Computação Ubíqua, foi introduzido 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. Deste modo, a computação não seria exclusividade de um computador, mas de diversos dispositivos conectados entre si.

A Computação Ubíqua, nesta perspectiva, não significa um computador que possa ser transportado para diferentes lugares. Mesmo o mais funcional *notebook*, com acesso a Internet ainda foca a atenção do usuário num único equipamento. Comparando à escrita, carregar este poderoso *notebook* é como carregar um livro muito importante. Por mais significativo que seja este livro, não significa capturar o real poder da Literatura (WEISER, 1991).

Uma analogia oportuna seria o surgimento da 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, constituindo 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 e, como todas as interfaces consolidadas, ele tende a ser imperceptível. 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. Sob esta premissa, o uso das interfaces passa a tornar-se inconsciente (WEISER, 1993).

A Computação Ubíqua é entendida como sendo o terceiro grande paradigma computacional, precedido 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 concebidos 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 de uso pessoal e exclusivo. Por fim, na Computação Ubíqua, veremos a tecnologia formando uma malha de dispositivos inteligentes em torno de cada indivíduo.

As aplicações ubíquas, em uma visão mais ampla, devem 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).

Para tal, 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, 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. Um desafio central na programação deste tipo de aplicação é possibilitar que as mesmas se adaptem continuamente ao ambiente e permaneçam funcionando mesmo quando o indivíduo se movimentar ou trocar de dispositivo (GRIMM; BERSHAD, 2003; COSTA; YAMIN; GEYER, 2008).

A Computação Ubíqua considerando esta perspectiva, constitui um ambiente altamente distribuído, heterogêneo, dinâmico, móvel, mutável e com forte interação entre homem e máquina (AUGUSTIN, 2003).

Ainda existe muita discussão quanto aos 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 resumidamente *UbiComp*.

2.1.1.1 Projetos de Grande Porte em Computação Ubíqua

A seguir são relacionados alguns projetos para Computação Ubíqua. Estes projetos são considerados de grande porte em função dos diferentes aspectos que contemplam quando do atendimento às demanda da área, proporcionando uma infraestrutura base para a concepção de aplicações para ambientes ubíquos, integrando várias funcionalidades. Sua inclusão nos estudos desta dissertação teve por finalidade resgatar suas motivações de pesquisa, resumidas a seguir.

Projeto Aura (GARLAN, 2001): tem como objetivo a 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 infraestruturas 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 ênfase na dimensão pessoal.

Projeto Gaia (ROMAN; AL., 2002): contempla como objetivo central o desenvolvimento de uma infraestrutura 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 infraestrutura de software para operar com os recursos, observar propriedades do contexto, assistir o desenvolvimento e execução das aplicações. É utilizado um *middleware* usado para prototipar aplicações, prover 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 ubíquo. Seu objetivo é permitir, através de uma arquitetura pré-estabelecida, que diferentes dispositivos possam se integrar ao sistema ubíquo, relacionando 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*, mecanismo de sensoreamento e aplicações.

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. Neste sentido, propõe ambientes que reajam à alteração do comportamento dos diferentes componentes que o integram. Utiliza um *middleware* programável, onde a arquitetura é

composta por módulos executáveis e por bibliotecas de programação, as quais contribuem para usabilidade do mesmo.

Projeto ISAM (YAMIN et al., 2005b): Infraestrutura de Suporte às Aplicações Móveis Distribuídas, iniciado na UFRGS, e em desenvolvimento por um consórcio de pesquisa formado por universidades gaúchas, tem como uma de suas atividades, o desenvolvimento do **EXEHDA** (*Execution Environment for Highly Distributed Applications*). O EXEHDA é um *middleware* direcionado às aplicações distribuídas, móveis e conscientes do contexto da computação ubíqua (YAMIN, 2004). Este *middleware* é baseado em serviços, e 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.

2.1.2 Computação Autônoma

A necessidade de integração de sistemas computacionais heterogêneos (KEPHART; CHESS, 2003) tem dificultado as operações de gerência e manutenção nos sistemas de maior porte.

Em 2001, Paul Horn, vice-presidente da área de pesquisas da IBM, introduziu 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 seres humanos, a computação autônoma apresenta uma visão na qual 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 usuários, 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 diferentes da visão inicial, a definição mais aceita na comunidade, classifica a computação autônoma em quatro aspectos principais resumidos 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, considerando as mudanças ocorridas no ambiente em que tem interesse. Entre outras coisas, isto faculta a possibilidade 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 minimizar o impacto da mesma no sistema.

3. Auto-otimização: infraestruturas computacionais modernas, como exemplo destacaríamos sistemas de banco de dados como Oracle e DB2, que 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 gerenciando. Em computação autônoma, na perspectiva ideal, 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, sob diferentes condições de uso.

4. Auto-proteção: apesar da existência de *firewalls* e ferramentas de detecção de intrusão, em muitas situações os usuários 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 tem interesse, 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 (LIN; MACARTHUR; LEANEY, 2005).

2.1.2.1 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 usuários 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 usuários ou outros elementos estabeleceram. Para dar suporte aos elementos autônomos e suas interações, faz-se necessária uma infraestrutura distribuída e orientada a serviços (KEPHART; CHESS, 2003; KHARGHARIA; HARIRI; YOUSIF, 2008).

Um elemento autônomo irá tipicamente consistir de um ou mais elementos gerenciados e um gerente de autonomia que os controla, como ilustrado na Figura 2.1. O elemento gerenciado é 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 uma aplicação em particular. De forma geral, elementos autônomos serão dotados de ciclos de vida complexos, compreendendo normalmente várias *threads*, continuamente monitorando e respondendo ao ambiente computacional.

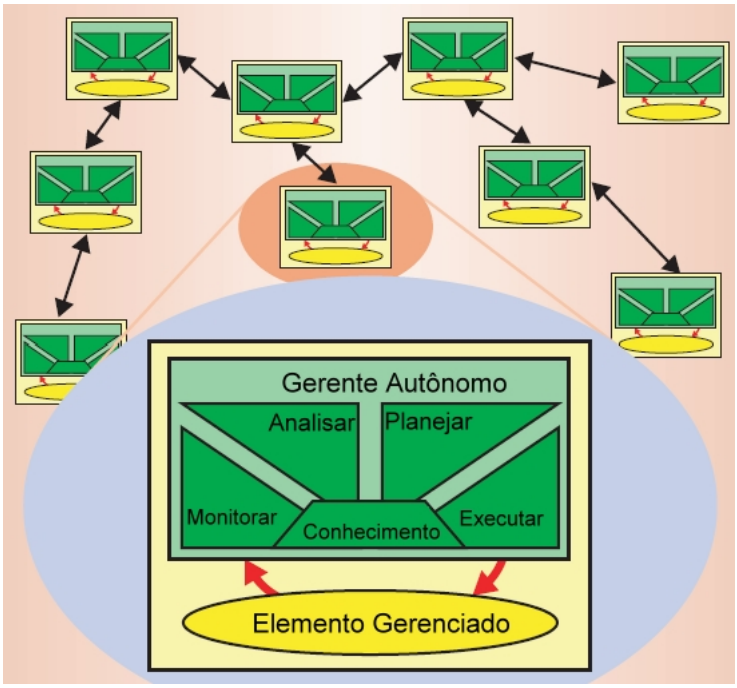


Figura 2.1: Gerente e Elementos Autônomos (KEPHART; CHESS, 2003)

2.1.2.2 Políticas

Os comportamentos dos elementos autônomos 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 desejado e, também, das restrições associadas a este comportamento. Duas questões são centrais quando da concepção de políticas em Sistemas Autônomos:

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 usuário ou outro elemento

autônomo pode dar direções para outro elemento sem que se faça necessário um conhecimento detalhado das atividades internas do mesmo.

- 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, as mais populares 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, especificando objetivos a serem perseguidos e, deixando a aplicação ou o *middleware* determinar as ações necessárias para alcançar aqueles objetivos (KEPHART; CHESS, 2003).

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.

Abordagens situação-ação exigem a descrição explícita de cada situação. Quanto mais elementos de contexto, mais condições e suas respectivas regras devem ser especificadas. Além disso, estas abordagens utilizam regras simples que falham para expressar algumas dependências entre adaptação e contexto. A priori, elas não são, portanto, apropriadas para o emprego no complexo contexto inerente a Computação Ubíqua. Usando abordagens orientadas a objetivos, é possível expressar políticas em alto nível. No entanto, abordagens orientadas a objetivos não conseguem capturar conflitos entre os objetivos. Por exemplo, um objetivo para a otimização do uso da CPU pode ser alcançado através da suspensão de aplicações de menor prioridade, mas tal suspensão teria impacto sobre o objetivo 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 um objetivo. Funções de utilidade utilizam a lógica para decisões de adaptação de uma forma precisa, e são, portanto, mais adequadas do que políticas de objetivos quando a adaptação desencadeia efeitos e interferência, ou quando existem objetivos em conflito.

A Computação autônoma possui um formalismo bastante sólido, tendo uma grande atuação em modelos aplicados à área da Engenharia Elétrica (KHARGHARIA; HARIRI; YOUSIF, 2008).

Na modelagem do mecanismo de adaptação proposto, EXEHDA-DA, discutido no capítulo 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 preferências dos usuários. A escolha da abordagem baseada em função de utilidade, teve como critério a complexidade das adaptações previstas.

2.2 Processamento de Ontologias: Principais Conceitos

A definição mais aceita e citada pelos autores da área de Computação é a que caracteriza ontologia como uma especificação formal e explícita de uma conceituação compartilhada. Esta definição foi introduzida pela referência (FENSEL, 2000), onde:

- conceituação - se refere a modelagem abstrata 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.

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.

Entre as potencialidades apresentadas pelo uso de ontologias, 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.

Ontologia pode ser considerado um bom instrumento para especificar os conceitos da computação ubíqua, além de ser o elemento central na emergente Web Semântica, novo paradigma que se apresenta como evolução da atual Web (Web Sintática). Web Semântica pode ser caracterizada pelo uso de interpretações, inferências e relações entre conceitos (BERNERS-LEE; HENDLER; LASSILA, 2001).

2.2.1 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.

A seguir alguns modelos de ontologias concebidos para uso na 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. SOUPA é composta de um conjunto de ontologias. 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*. Se mostra flexível quanto a extensibilidade, sendo possível 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. Contempla 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. Esta ontologia dividiu as descrições das dimensões do modelo do usuário em três partes: auxiliar, predicado e alcance de valores.

Através da representação dos dados utilizando ontologias, a informação pode ser estruturada de maneira que os componentes de software possam tratar o significado da informação. O uso de ontologias como artefato de software pode ocorrer em tempo de desenvolvimento ou manutenção, bem como em tempo de execução. No desenvolvimento, auxiliam nas especificações de requisitos, na modelagem conceitual de sistemas, em atividades de suporte, de gerenciamento de projeto, e reuso de conhecimento.
Por sua vez, em tempo de execução podemos ter dois tipos de ontologias (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. Neste sentido temos aplicações orientadas a ontologia (*Ontology-driven software*), nas quais a arquitetura de *software* é caracterizada por usar uma ou mais ontologias como elementos centrais do sistema. Uma aplicação típica, 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. Este tipo de software que consulta ontologias, em uma perspectiva de base de dados, são denominadas Aplicações Cientes da Ontologia (*Ontology-aware Software*).

2.2.2 Linguagens para Tratamento das Ontologias

As linguagens para ontologias devem permitir a seus usuários escreverem 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.

W3C, *World Wide Web Consortium*, (http://www.w3.org/), é o detentor dos padrões XML, RDF, e OWL. Em meados da década de 1990 o W3C começou a trabalhar em uma linguagem de marcação que combinasse a flexibilidade da SGML (*Standard Generalized Markup Language*) com a simplicidade da HTML (*HyperText markup Language*). O principio do projeto era criar uma linguagem que pudesse ser tratada por software, e integrar-se com as demais linguagens de programação existentes.

2.2.2.1 XML

XML, (*eXtensible Markup Language*) é uma recomendação do W3C para gerar linguagens de marcação para necessidades especiais. É um subtipo de SGML,

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.

2.2.2.2 OWL

Segundo a W3C (http://www.w3.org), 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 (ítem 5.2, 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, (*DARPA Markup Language + Ontology Inference Layer*), a qual é uma sintaxe, criada sobre RDF e XML, que pode ser utilizada para descrever conjuntos de fator criando uma ontologia.

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

- *OWL Lite*: atende 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 linguagens análogas mais expressivas. Por isso tudo ela constitui um caminho de migração mais rápido de tesauros e outras taxonomias. Bem como a 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 (é garantido 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 constitui a base formal da OWL.

- *OWL Full*: é direcionada àqueles usuários que querem a máxima expressividade e a liberdade sintática introduzida pelo RDF, porém nenhuma garantia computacional quanto a computabilidade ou tempo de processamento. 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 do RDF ou OWL. Atualmente não se tem registro de

software de inferência que suporte completamente os diferentes recursos 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.

Neste capítulo foram abordados aspectos sobre a Computação Ubíqua, sua origem, conceitos e características considerados mais importantes, assim como, alguns de seus projetos. Este capítulo tratou também 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. A seguir foram tratados os fundamentos teóricos sobre processamento semântico com seus principais conceitos, classificação em categorias, alguns modelos específicos de ontologias e as tecnologias para tratamento das mesmas, mostrando as principais linguagens envolvidas.

A partir dos fundamentos já introduzidos, o próximo capítulo discute o escopo do trabalho, a visão geral e as premissas de pesquisa consideradas para o desenvolvimento do mesmo. Com destaque para a discussão de tópicos referentes ao controle da adaptação na UbiComp e de trabalhos relacionados.

3 ESCOPO DO TRABALHO, VISÃO GERAL E PREMISSAS DE PESQUISA

Neste capítulo estão sintetizados os principais desafios inerentes ao controle da adaptação na Computação Ubíqua. Também serão descritos, comparados e avaliados trabalhos relacionados a este tema.

O estudo realizado indicou que o desenvolvimento de aplicações *context-aware*, ainda contempla muitos aspectos a serem considerados, constituindo uma área de pesquisa com diferentes focos de pesquisa a serem explorados.

3.1 Escopo do Trabalho: Controle da Adaptação na UbiComp

Como um dos requisitos para que um sistema ubíquo promova 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 computacional que as abriga e, através destes dados, adaptar seu comportamento, da melhor forma possível, buscando a satisfação das expectativas de seus usuários.

3.1.1 Principais Conceitos

Uma vez sendo possível capturar 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 (COSTA; YAMIN; GEYER, 2008).

Podemos diferenciar as adaptações em estáticas e dinâmicas (GEIHS et al., 2009). Adaptação estática diz respeito à redefinição e reconfiguração de arquiteturas e componentes das aplicações em tempo de concepção e/ou geração de código, tendo em vista a alteração em requisitos funcionais e não-funcionais da aplicação. 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.

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). Para efetivação de um procedimento adaptativo é feita a avaliação de elementos do contexto, como localização, tempo e ati-

vidades do usuário, permitindo o desenvolvimento de aplicações sensíveis ao mesmo. O tratamento dinâmico da adaptação à medida que as aplicações são executadas ainda é uma frente de pesquisa bastante ativa na Computação Ubíqua (HENRICKSEN; INDULSKA; RAKOTONIRAINY, 2006). 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 que tem capacidade de adaptar-se ao contexto, são chamadas **aplicações *context-aware***.

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 infraestrutura de adaptação são as principais preocupações. Geralmente, a adaptação é um ponto crucial na concepção de tais sistemas. Como premissa, para atingir escalabilidade, o número de variáveis que devem ser avaliadas em tempo de execução, em reação a uma mudança de contexto deve ser mantido tão baixo quanto possível, some-se a isto a preocupação de eliminar 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 de localização do usuário 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, preferências do usuário e outros. 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.

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; AUGUSTIN et al., 2006).

Nesta perspectiva, 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 assíncronas 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:

- variedade de filtros (compressão, omissão, conversão de formato) inseridos entre o servidor e o cliente;
- alteração da fidelidade dos dados de saída;
- alterações nas dimensões da interface;
- migração de componentes para máquinas mais adequadas (com maior poder computacional, com maior memória, com rede de comunicação mais veloz, etc.).

Como premissa motivacional, entende-se que as abstrações dos sistemas operacionais/linguagens de programação existentes não são suficientes, nem apropriadas para tratar as diferentes necessidades de adaptação das aplicações ubíquas (COSTA; YAMIN; GEYER, 2008).

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 de software a ser utilizado em um determinado contexto de execução, isto é, atua na seleção do código de execução do serviço de uma aplicação (YAMIN, 2004).

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, heterogêneas, móveis e adaptativas ao contexto, conforme demonstrado na figura 3.1, Premissas da Computação Ubíqua.

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

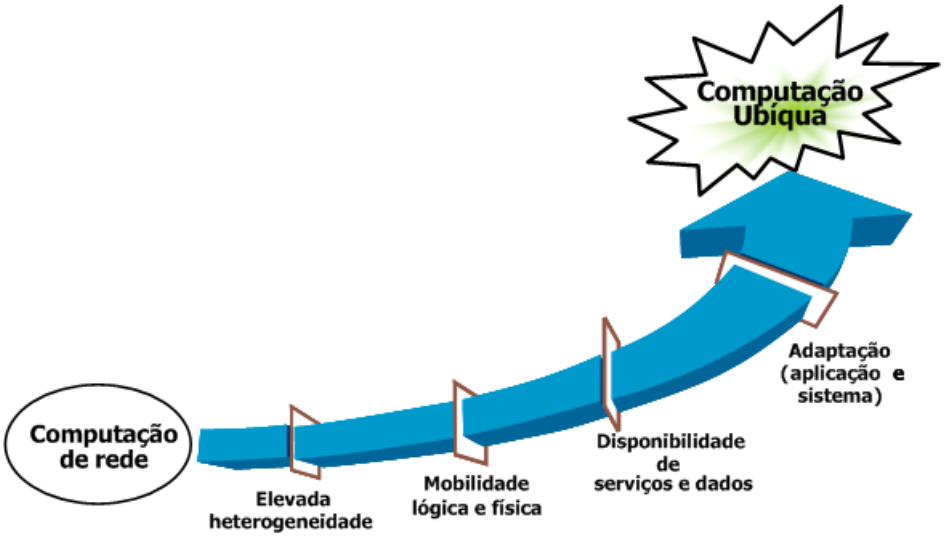


Figura 3.1: Premissas da Computação Ubíqua (YAMIN, 2004)

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 enquanto conhecedor do comportamento da aplicação pode otimizar o processo de adaptação, orientando a estratégia a ser empregada em relação a componentes específicos da aplicação.

3.1.2 Projetos Relacionados

Considerando o crescimento das pesquisas na área de Computação Ubíqua nos últimos anos (COSTA; YAMIN; GEYER, 2008), tem surgido na literatura especializada diversos trabalhos relacionados ao controle da adaptação e mecanismos afins. Em função disso, foi realizada uma busca em abrangência destes trabalhos. O produto deste estudo está organizado nesta dissertação em dois resumos:

- um primeiro resumo está no anexo A, onde estão relacionados 18 projetos referentes ao controle da adaptação numa perspectiva de atender requisitos da Computação Ubíqua. Para estes projetos são apresentadas sua descrição, comparação e avaliação, tendo como parâmetro os 12 aspectos considerados importantes para o controle da adaptação na UbiComp. Esta comparação é resumida na tabela comparativa disponível no Anexo A.1, e;

- um segundo resumo composto por outros 7 trabalhos com maior grau de similaridade com relação ao modelo proposto para o EXEHDA-DA. Estes trabalhos estão sintetizados na presente seção, e uma discussão comparando-os com o EXEHDA-DA é feita nas considerações finais do trabalho, (vide seção 7.1, Discussão dos Trabalhos Relacionados ao EXEHDA-DA).

Esta síntese contempla aspectos de identificação, autoria, uma descrição e uma referência bibliográfica entendida como significativa no escopo do referido projeto:

1. **Carisma** (CAPRA; EMMERICH; MASCOLO, 2003), Sistema baseado em um *Middleware* Reflexivo *Context-aware* para Aplicações Móveis. Consiste em um *middleware* para computação móvel *peer-to-peer* que explora o princípio de reflexão computacional para suportar a construção de aplicações móveis adaptativas e sensíveis ao contexto. O *middleware* oferece primitivas aos engenheiros de software para descrever como as mudanças de contexto devem ser tratadas usando políticas. Estas políticas podem entrar em conflito. O *middleware* classifica os diferentes tipos de conflitos que possam surgir em computação móvel e argumenta que os conflitos não podem ser resolvidos estaticamente, em tempo de concepção das aplicações, mas precisam de ser resolvidos em tempo de execução. O planejamento no Carisma consiste em definir entre um conjunto de políticas de adaptação, baseadas em regras pré-estabelecidas (a que se refere o perfil da aplicação), usando funções de utilidade e um método com abordagem micro-econômica que baseia-se em um tipo particular de lance de leilão selado, para resolver conflitos entre as políticas. Esse método é implementado na própria arquitetura do *middleware*. Os desenvolvedores do Carisma, após diferentes avaliações de desempenho defendem que a resolução de conflitos não implica em custos excessivos. Trabalha apenas com adaptações funcionais e prevê um número fixo de políticas de adaptação.

2. **Chisel** (KEENEY; CAHILL, 2003), é um *framework* para adaptações dinâmicas de serviços utilizando reflexão controlada por políticas, de maneira consciente ao contexto. É baseado na decomposição de aspectos extra-funcionais (não-funcionais) de uma aplicação em comportamentos alternativos. Então, um algoritmo orientado a políticas é usado para dinamicamente selecionar a alternativa mais adequada, com base no estado do contexto. Mudanças de contexto desencadeiam um processo que determina a adaptação a ser realizada. Chisel para instruir o comportamento adaptativo das aplicações emprega políticas de adaptação, utilizando linguagens declarativas.

3. **QUO** (Objetos de Qualidade) (HEINEMAN; LOYALL; SCHANTZ, 2004; SHARMA et al., 2004), é um projeto de *middleware*, que embora não trate todos os aspectos associados a dispositivos portáteis e redes sem fio, teve um impacto importante entre os sistemas adaptativos e projetos de desenvolvimento de *middleware QoS-aware*. Quo é uma arquitetura que está centrada em fornecer, em tempo de concepção e em tempo de execução, suporte para aplicações adaptativas do tipo cliente-servidor. A abordagem QUO permite que aplicações adaptem as suas funcionalidades dinamicamente alterando parâmetros de 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 considerando a flutuação dos recursos.

4. **Rainbow** (GARLAN et al., 2004), esta abordagem consiste em um *framework*, uma linguagem e um processo incremental de engenharia de auto-adaptação. O *framework* é constituído por duas partes: (i) uma genérica, composta por mecanismos de adaptação reutilizáveis, que podem ser aplicados a diferentes classes de aplicações, e (ii) um conjunto de pontos de personalização, explicitamente definidos, que permitem adaptar uma classe específica de aplicações. A linguagem

contempla conceitos que permitem a definição de políticas de adaptação baseadas em objetivos. O processo incremental de adaptação permite que as decisões de adaptação, possam ser desenvolvidas de forma fragmentada. As técnicas de adaptação propostas são separadas das funções lógicas da aplicação.

5. **Madam** (GEIHS et al., 2009), é um projeto europeu de pesquisa com parceiros distribuídos em indústrias e universidades. É uma solução abrangente para o desenvolvimento e execução de aplicações adaptativas conscientes de contexto. Suas principais características são: (i) um *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. Seu foco é a adaptação dinâmica de aplicações *context-aware* em dispositivos de computação móvel. O *middleware* desempenha três funções principais relacionadas com a adaptação: gerenciamento do contexto, gerenciamento da adaptação e gerenciamento da configuração. Dois componentes desempenham um papel vital, o gerenciador de recursos, que fornece informações sobre o estado dos mesmos, e o núcleo do *middleware* (*core*) que provê o suporte às operações. MADAM é uma abordagem ampla para o desenvolvimento de aplicações adaptativas para cenários ubíquos, com suporte tanto para dispositivos fixos, como móveis.

6. **Proteus** (TONINELLI et al., 2007), emprega um modelo semântico na gerência da adaptação dos acessos permitidos aos usuários. Trata o contexto como a classe principal quando da definição da adaptação de políticas. Contempla uma abordagem baseada em descrições semânticas e lógicas. O modelo considera três tipos de adaptação: (i)adaptação de políticas: consiste em instruir o sistema para reagir a um contexto que mudou e 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 permitidas/obrigatórias poderão ser realizadas. Faculta que as operações de acesso nos recursos sejam controladas baseados na visibilidade do contexto, através de tecnologias semânticas que permitem descrição de alto nível e inferência sobre o contexto e políticas.

7. **SECAS** (CHAARI; LAFOREST, 2007), *Simple Environment for Context Aware Systems*, é um projeto que trata com a adaptação das aplicações ao contexto, considerando as preferências do usuário, do ambiente e os dispositivos envolvidos. Uma plataforma que tem como propósito fazer os serviços, dados e interfaces de aplicações serem adaptativas às situações de contexto. Trabalha com cinco facetas (dimensões): perfil da rede, descrição/preferências do usuário, características dos dispositivos do usuários, localização e ambiente. A troca de um valor de parâmetro em uma faceta, assinala uma nova situação contextual. Mostra uma nova definição do contexto, separando dados da aplicação dos parâmetros do contexto. A arquitetura é baseada em quatro subsistemas: código da aplicação, camada de adaptação, sistema de gerenciamento de contexto e sistema do lado do cliente. Emprega a representação das redes Petri enquanto método formal para modelar

as funcionalidades da aplicação e suas dependências. Emprega a tecnologia Java XML-RPC para compor as funcionalidades das aplicações distribuídas, bem como para o disparo das adaptações. Suporta adaptações funcionais e utiliza expressões lógicas para configurar a situação de contexto para os diferentes adaptadores. Foi implementado no ambiente médico, objetivando validar a sua estratégia de adaptação.

3.2 EXEHDA-DA: Premissas de Pesquisa e Visão Geral

Mesmo os dispositivos e recursos de rede tornando-se cada vez mais poderosos, o projeto de aplicações móveis ainda se mostra condicionado por limitações físicas. Os dispositivos móveis continuarão a ser dependentes da bateria e os usuários são relutantes em transportar dispositivos pesados. Recursos de rede continuarão a basear-se em comunicação com estações base, com flutuações de largura de banda, dependendo das localizações físicas. Enfim para proporcionar qualidade aceitável do serviço para seus usuários, e, consequentemente, melhorar a sua satisfação, as aplicações precisam apresentar sensibilidade ao contexto, e serem capazes de adaptarem-se às suas mudanças, tais como as variações de largura de banda da rede, o esgotamento de energia da bateria ou a manutenção da acessibilidade dos serviços quando da troca de dispositivos. Neste sentido, a consulta periódica à sensores físicos heterogêneos, a fim de obter informações atualizadas de contexto, detectar as configurações de contexto de interesse para a aplicação, e se adaptar, seria uma tarefa para o desenvolvedor das aplicações extremamente trabalhosa e propensa a erros. A utilização de um *middleware* entre o sistema operacional e as aplicações, com abstrações e mecanismos para gerenciamento ubíquo, como o EXEHDA, possuem como um de seus objetivos, liberar os desenvolvedores de tratar detalhes de baixo nível. Por exemplo, aplicações devem ser capazes de especificar, de forma padronizada as suas necessidades de adaptação ao contexto. O *middleware*, então, atualiza informações de contexto, detecta as alterações de interesse para a aplicação, e reage de acordo.

3.2.1 Premissas para a Concepção do EXEHDA-DA

As tecnologias orientadas a serviços, cujos serviços podem ser selecionados de acordo com as suas características funcionais, podendo ser integrados e requisitados remotamente, afetam o nível de dinamismo e flexibilidade das aplicações. *Middlewares* para publicação e utilização de serviços, assim como Computação em Grade, facilitam a virtualização e independência de recursos, seja do ponto de vista de quem será o fornecedor, seja quanto a sua localização física. Estas são tecnologias que lidam com a complexidade, heterogeneidade e incertezas de sistemas e ambientes de software modernos, os quais devem ser considerados no âmbito da UbiComp.

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 as seguintes premissas de estudo e pesquisa:

- as decisões para adaptação da aplicação serão o mais automático possível, conside-

rando preferências do usuário, contexto, dispositivos e recursos;

- 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;
- o controle de adaptação acontecerá de forma colaborativa entre a aplicação e o *middleware*;
- aplicação deve se manter consistente independentemente da ocorrência de procedimentos adaptativos;
- adaptação pode acontecer a qualquer momento, e sem intervenção do usuário, em reação às mudanças de contexto;
- qualquer contexto factível de ser monitorado pode ser empregado como parâmetro de tomada de decisão de adaptação;
- o controle da adaptação é tratado separadamente da computação das aplicações;
- prover suporte para que os serviços do *middleware* também possam ser adaptativos ao contexto;
- potencializar a capacidade de manutenção, reaproveitamento de informações contextuais, bem como de regras para controle da adaptação;
- permitir a definição por componente da aplicação de alternativas de adaptação, e das respectivas ações a serem tomadas pelo gerenciador de adaptação;
- contemplar a possibilidade de inferências semânticas no procedimento para a tomada de decisões de adaptação.

A figura 3.2 mostra uma visão geral do controle da adaptação, identificando os extremos, *middleware* ou aplicação, como responsáveis pela adaptação.

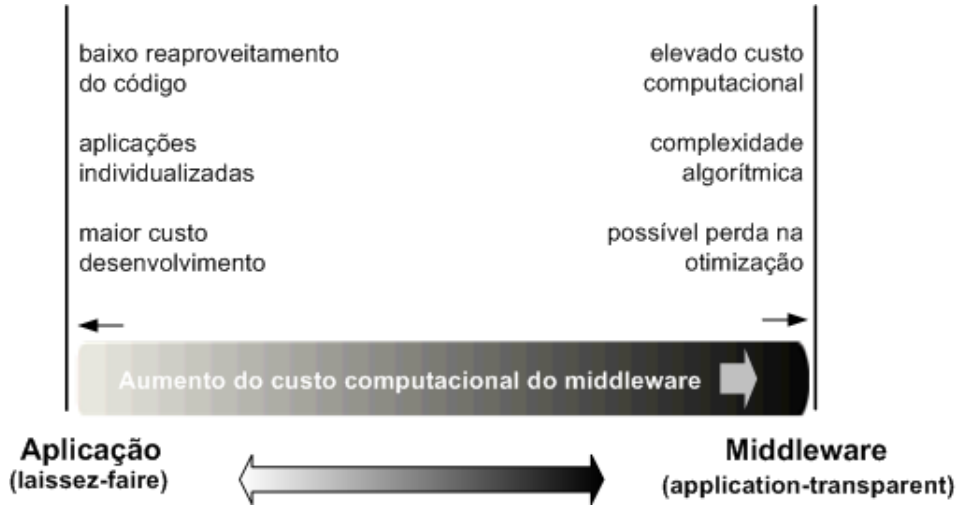


Figura 3.2: Distribuição da Responsabilidade pela Adaptação

Um dos desafios na concepção de um 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*. 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.

Na Figura 3.3 é destacado o módulo no EXEHDA que trata da Adaptação ao Contexto. Esta figura mostra os serviços do EXEHDA, organizados em quatro grandes subsistemas: Subsistema de Acesso Ubíquo, Subsistema de Comunicação, Subsistema de Execução Distribuída e Subsistema de Adaptação e Reconhecimento de Contexto. 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.

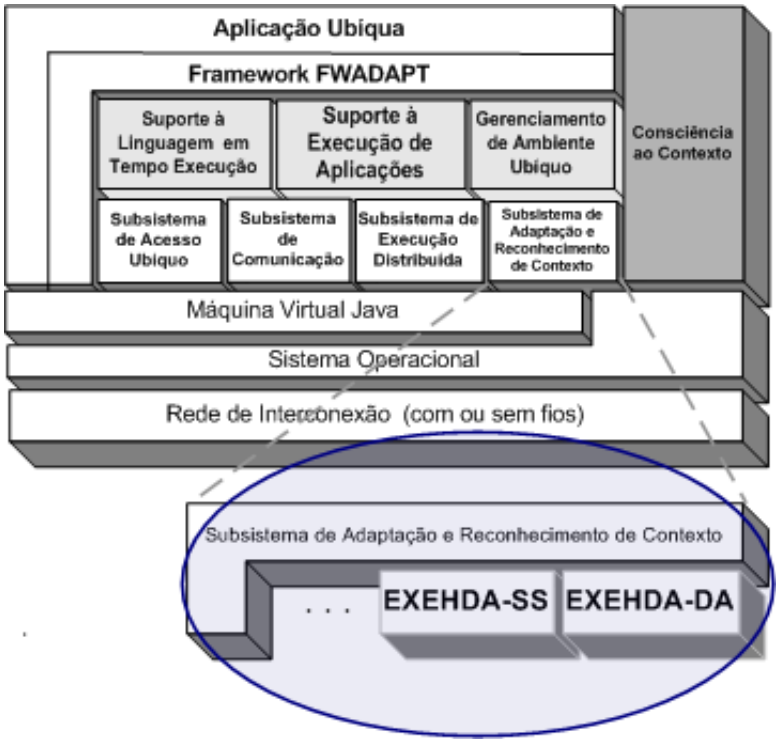


Figura 3.3: Subsistema de Adaptação e Reconhecimento de Contexto do *Middleware* EXEHDA

A Adaptação Multi-nível Colaborativa, a partir do EXEHDA-DA, fornece os seguintes níveis de atividades de adaptação:

1. No nível da aplicação:
   - em tempo de desenvolvimento:
     - definição e instanciação das ontologias de Política de Adaptação da Aplicação e Contexto de Interesse da Aplicação, através da utilização do *framework* FWADAPT, subseção 4.1.2;

  - programação dos comandos adaptativos nos códigos dos componentes, seção 4.3;
- em tempo de execução: ativação dos comandos adaptativos, e sua decorrente interatividade com o *middleware* EXEHDA;

2. No nível do Serviço de Controle da Adaptação Dinâmica do *middleware*, EXEDHA-DA: inferência e decisão de adaptação ao contexto, individualizada por componente da aplicação, seção 4.2.

3.2.2 Visão Geral do Controle da Adaptação para o EXEHDA-DA

São dois os tipos de adaptação dinâmica considerados no modelo:

- Adaptação Não-Funcional: atua sobre a gerência da execução distribuída, através de operações de mapeamento, reescalonamento, instanciação remota, migração;
- Adaptação Funcional: adaptação do código da aplicação, isto é, atua na seleção do componente de software que deve ser implementado.

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.

Assim, na monitoração observa-se que os múltiplos elementos de contexto devem ser considerados. Além disso, 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 evoluam 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 decorrência de uma melhor compreensão do problema a ser resolvido, ou da inclusão de novas funcionalidades na aplicação. Estas possibilidades levam as seguintes considerações arquiteturais: (i) a monitoração do contexto deverá ser mantida separada da aplicação e realizada através de elementos reutilizáveis pelo *middleware*; (ii) o tratamento de 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 (VENECIAN, 2010).

Além da reusabilidade e extensibilidade dos elementos do contexto, a separação da execução da aplicação, do controle das suas adaptações, entende-se que deverá simplificar o desenvolvimento de aplicações adaptativas.

No controle da adaptação, observamos múltiplas relações entre contexto e mecanismos de adaptação, e destes com 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. Quanto mais elementos de

contexto introduzidos, um número maior de relações entre eles precisam ser consideradas. Com a generalização dos cenários de uso, sabemos que é muito difícil captar todas essas relações (FLOCH; STAV; HALLSTEINSEN, 2006). Além disto, novas relações poderão ser introduzidas, envolvendo as aplicações e os elementos de contexto, a medida que estes evoluam.

Em (AUGUSTIN; YAMIN; SILVA, 2008) foram examinados dois tipos de abordagens para a 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.

Assim, na proposta do EXEHDA-DA os mecanismos de adaptação serão tratados 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 2.1.2.2, 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 cômputo da adaptação.

A função utilidade calcula a prioridade das alternativas de adaptação da aplicação como uma função dos parâmetros previstos para as diferentes adaptações e do contexto atual. O objetivo é selecionar a adaptação da aplicação que maximize a função de utilidade, satisfazendo as limitações dos recursos fornecidos pelo ambiente.

A 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 parâmetros representando a interação entre a aplicação e seu contexto.

A função de utilidade considera os parâmetros de adaptação da aplicação e do contexto de execução, bem como as preferências do usuário. A função de utilidade calcula a soma ponderada das diferenças entre os parâmetros ofertados pelo ambiente e os necessários pela aplicação, onde os pesos refletem as prioridades das necessidades do usuário. Dentro deste modelo, o objetivo do Serviço 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.

Duas técnicas se mostram mais usuais quando da composição de mecanismos de adaptação das aplicações: adaptação por parâmetros e adaptação composicional (POLADIAN et al., 2004). Adaptação por parâmetros de aplicações propõe ajustes na aplicação através da instanciação de variáveis e/ou pelo ajuste de 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 (POLADIAN et al., 2004).

Portanto, focaremos o modelo do EXEHDA-DA na adaptação composicional de componentes.

Em resumo, para o EXEHDA-DA foi adotada uma abordagem externa de adaptação onde as capacidades de adaptação de aplicações são realizadas por um serviço do *middleware* independentemente da aplicação. As políticas de adaptação serão expressas utilizando funções de utilidade, e a adaptação é baseada em adaptação composicional de componentes. 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.

No próximo capítulo, tendo como ponto de partida as discussões feitas quanto ao escopo do trabalho, será detalhada a proposta para o controle da adaptação dinâmica ao contexto. Será detalhado o modelo ontológico proposto. Também serão apresentadas as potencialidades deste modelo.

4 EXEHDA-DA: MODELAGEM

Este capítulo tem como premissa central discorrer sobre as decisões pertinentes à modelagem do EXEHDA-DA enquanto um Serviço para Controle Dinâmico da Adaptação.

O Serviço de Controle de Adaptação Dinâmica, EXEHDA-DA, enquanto um serviço do *middleware* EXEHDA, foi concebido com a perspectiva de potencializar seu uso com diferentes aplicações, cada uma com seu contexto de interesse e sujeitas a políticas de adaptação, personalizadas para cada um dos componentes de software, que fazem parte das aplicações. Para tanto, são registradas em uma ontologia de política de adaptação da aplicação, todas as regras, políticas e ações globais e ações específicas de cada aplicação. Deste modo, o EXEHDA-DA, a partir desta política de adaptação da aplicação, das mudanças do contexto e das preferências do usuário, infere a ação de adaptação a ser executada, definindo os códigos e os nodos envolvidos na mesma. Este modelo de controle de adaptação ao contexto possibilita uma evolução incremental das especificações de políticas, regras, parâmetros, restrições e ações de adaptação (configuração do perfil da aplicação), permitindo a reutilização e a customização destas no desenvolvimento de novas aplicações adaptativas.

A premissa operacional é adaptar ao contexto as aplicações da computação ubíqua, sem ou com mínima intervenção explícita do usuário, durante a execução das mesmas.

O Serviço EXEHDA-DA é responsável pela avaliação do impacto produzido pelas mudanças de contexto nas aplicações, adaptando os componentes da aplicação em execução, selecionando o componente adaptador que atenda as demandas da respectiva mudança contextual. Para a seleção do componente, quando da aplicação da regra de adaptação, é necessário considerar, além das variações de contexto, as especificações dos parâmetros referentes à adaptação em pauta.

O emprego de ontologias na concepção do EXEHDA-DA se deve ao fato das mesmas 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. Outro fator analisado foi a maior integração e padronização com o Serviço de Reconhecimento de Contexto que também faz uso de ontologia para representação de contexto. Por outro lado, é observado um crescimento na utilização de ontologias no desenvolvimento de aplicações ubíquas, especialmente como artefatos de software, com o objetivo de modelar as informações gerenciadas por essas aplicações.

A escolha do modelo fundamenta-se na utilização combinada de ontologias, Sistemas Autônomos e Computação Ubíqua, onde a sua integração poderá ser uma

alternativa utilizada no desenvolvimento de aplicações em espaços ubíquos, além de poder proporcionar facilidades para a concepção, manutenção e reutilização destas aplicações, e como consequência um atendimento mais ágil às demandas dos usuários.

4.1 Estrutura do Modelo Semântico do EXEHDA-DA

Com o intuito de ampliar a possibilidade de especificação dos diferentes tipos de adaptação, o EXEHDA-DA suporta que sejam definidos requisitos por aplicação. A premissa é que vários elementos de contexto podem ser relevantes para uma mesma aplicação na UbiComp, e que a composição de várias formas de adaptação pode contribuir para melhorar ou manter a qualidade do serviço quando o contexto muda.

A aplicação depende de uma série de elementos de contexto. No modelo ontológico proposto, o contexto é representado pela ontologia denominada OntUbi, onde serão descritos todos os elementos de contexto de interesse para as aplicações. Nesta ontologia, os elementos de contexto ou entidades, chamadas de Classes no OWL, possuem três categorias básicas de informações:

- recursos de hardware: dispositivos, periféricos e informações dos nodos e sensores, tais como: localização, memória, capacidade bateria, largura banda, carga CPU, entre outros;
- recursos de software: sistema operacional, *middleware* e aplicações, política de adaptação da aplicação;
- recursos de usuário: reflete o perfil e, as preferências e necessidades do usuário.

Em relação às categorias de ontologias, (ítem 2.2.1), o modelo semântico proposto contempla a seguinte estrutura:

- **Ontologia de Domínio do EXEHDA-DA**
  - Geral: **OntUbi** - Ontologia básica para Computação Ubíqua: ontologia com todas as entidades (classes), seus atributos e relacionamentos de interesse das aplicações ubíquas. As ontologias específicas fazem parte da ontologia geral.
  - Específica 1: **OntContext** - Ontologia da Situação do contexto: representa os contextos coletados, contextos notificados e os contextos de interesse da aplicação.
  - Específica 2: **OntAdapt** - Ontologia da de Adaptação: regras, parâmetros, operações e preferências, restrições e ações de adaptação para os componentes das aplicações.
  - Específica 3: **OntHistAdapt** - Ontologia prevista para o registro das decisões de Adaptação, histórico das adaptações realizadas.
- **Ontologia como Artefato de Software** no EXEHDA-DA é empregada em dois momentos:

- Em tempo de Desenvolvimento de Aplicações: criação e instanciação da ontologia OntUbi e da OntAdapt. A OntUbi foi modelada e instanciada pelo G3PD, utilizando o Protégé, versão 3.4, vide seção 5.6. Esta ontologia é também utilizada pelos trabalhos de pesquisa direcionadas à Sensibilidade ao Contexto e à Descoberta de Recursos. Por sua vez, a OntAdapt foi modelada utilizando o Protége e, sua instanciação foi realizada utilizando o *framework* FWADAPT.
- Em tempo de de Execução das Aplicações: Aplicações e serviços do *middleware* cientes das ontologias básicas: controle de adaptação ao contexto: consultas e instanciações na OntUbi, OntContext, OntAdapt e OntHistAdapt.

4.1.1 Estrutura das Ontologias Propostas

A Figura 4.1 apresenta a ontologia OntUbi, com destaque das ontologias OntContext e OntAdapt, que estão contidas na mesma.

Na OntUbi, que é a ontologia base do modelo, estão descritas as classes que compõem os elementos do contexto de interesse das aplicações ubíquas, com seus atributos e relacionamentos.

A Figura 4.2 mostra as classes com seus atributos da OntContext, bem como os seus relacionamentos. Esta ontologia representa os contextos coletados, os contextos notificados e os contextos de interesse da aplicação, sendo instanciada pelo Serviço de Sensibilidade ao Contexto. Em particular, a Classe Contexto de Interesse da Aplicação e seus relacionamentos é instanciada e mantida pelo *framework* FWADAPT (vide ítem 4.1.2).

A Figura 4.3 mostra as classes, os atributos e os relacionamentos entre as classes da Ontologia OntAdapt , Política de Adaptação da Aplicação, a qual tem a finalidade de registrar os perfis dos componentes das aplicações para a adaptação. As classes Aplicações, Componentes, Adaptações, Tipos de Parâmetros e Valores de Parâmetros são subclasses da superclasse Software. Esta ontologia também é instanciada e mantida pelo desenvolvedor, através do *framework* FWADAPT.

Na OntAdapt podem ser definidas políticas para várias aplicações. Cada aplicação é composta por diversos componentes, instâncias do relacionamento Aplicacao_Componente. Cada componente pode ter várias adaptações, instâncias do relacionamento Componente_Adaptador. Cada adaptação de cada componente da aplicação pode ter vários parâmetros, instâncias do relacionamento Adaptador_ParamTipo. Um Parâmetro pode ter várias ocorrências de valor inferior, valor superior e valor de utilidade, instâncias do relacionamento ParamTipo_ParamValor.

Parâmetros são, funcionalmente, dependentes dos elementos de contexto. Desta forma, é possível representar as dependências entre os componentes das aplicações e o contexto. Parâmetros previstos podem ser implementados de várias formas, por exemplo, para expressar que a execução de um componente específico requer mais do que 500 MB de memória, ou como expressões funcionais de outros parâmetros do componente em si. Por exemplo, o tempo de resposta oferecido por um componente pode depender da largura de banda disponível para esse componente.

Consideramos a interdependência dos parâmetros, quando um determinado valor de um tipo de parâmetro depende do valor de um outro tipo de parâmetro.

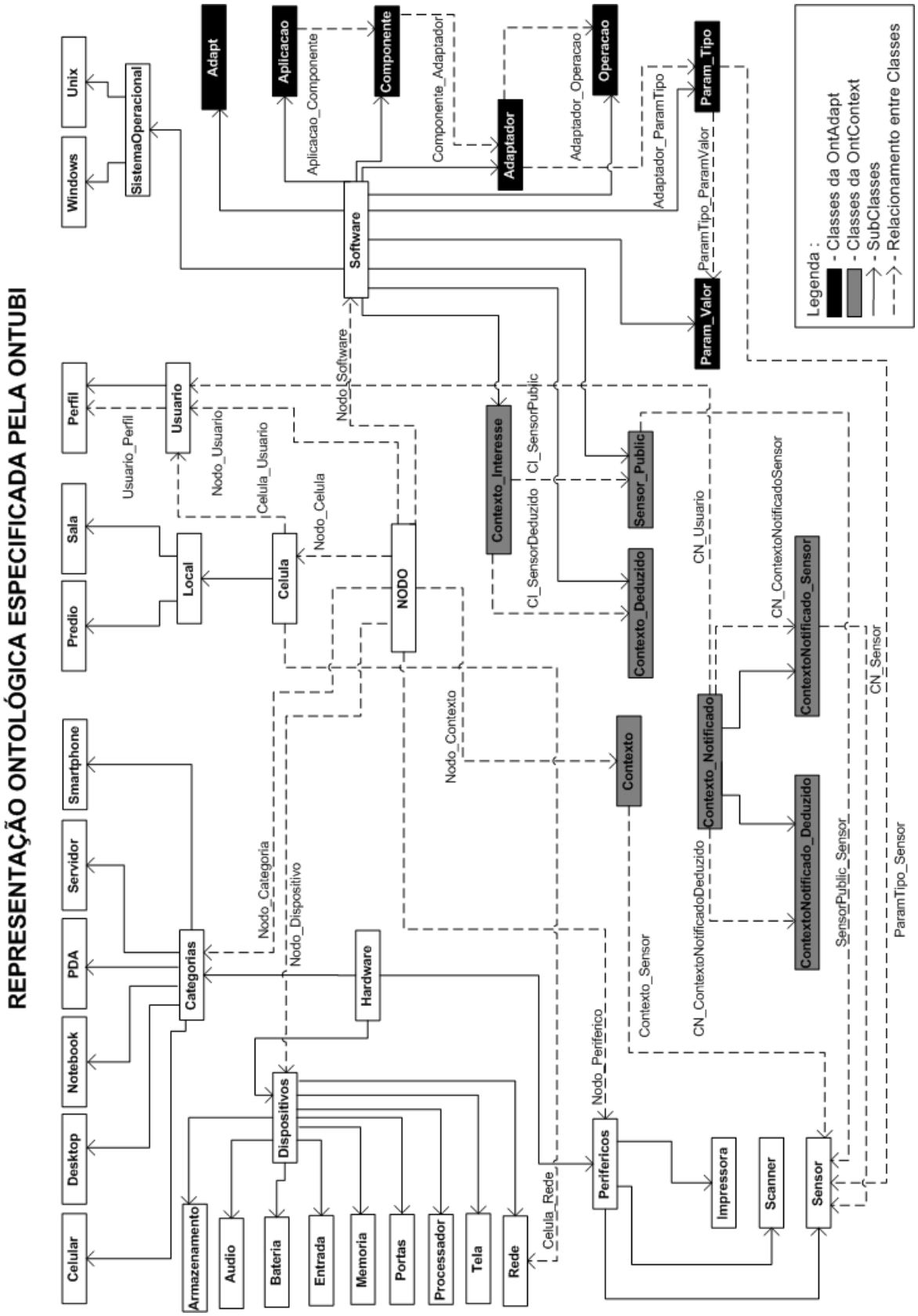


Figura 4.1: Ontologia Geral Proposta do Contexto Ubíquo - OntUbi

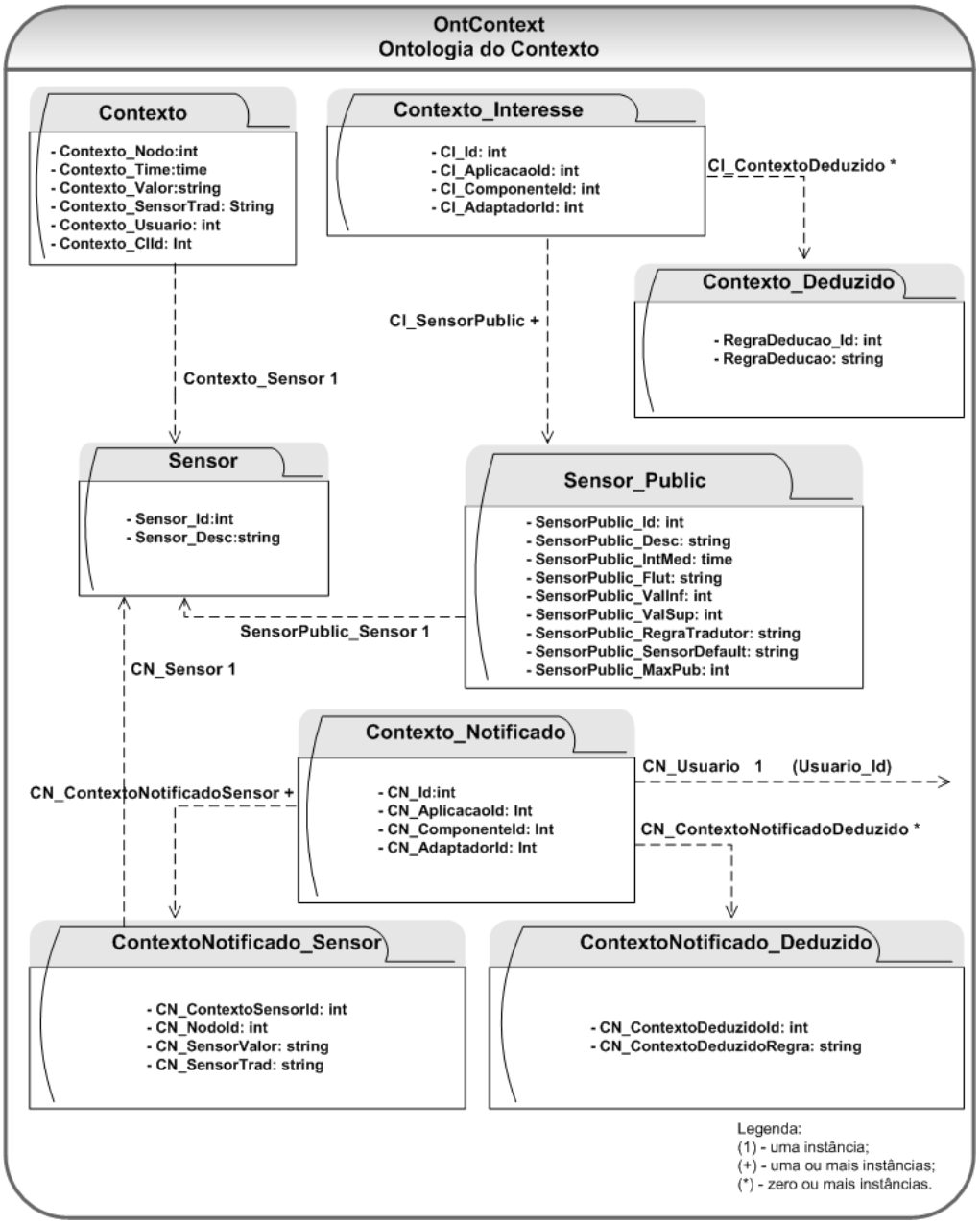


Figura 4.2: Ontologia da Situação de Contexto - ONTCONTEXT.

Utilidade é um valor definido na classe Parâmetros. O valor utilidade é utilizado em conjunto com os valores de parâmetros, com valores provenientes da notificação de contexto, bem como nas preferências ou necessidades do usuário. A utilidade é o peso que reflete as prioridades do parâmetro ou das necessidades do usuário. O emprego de utilidade, como fator de variabilidade de parâmetros de uma adaptação no EXEHDA-DA, está está citado na seção 3.2.2, Visão Geral do Controle da Adaptação Proposto.
O gerenciador do serviço de adaptação calcula as variantes de utilidade, selecionando a variante de utilidade especificada, para cada parâmetro dentro de cada tipo de adaptação.

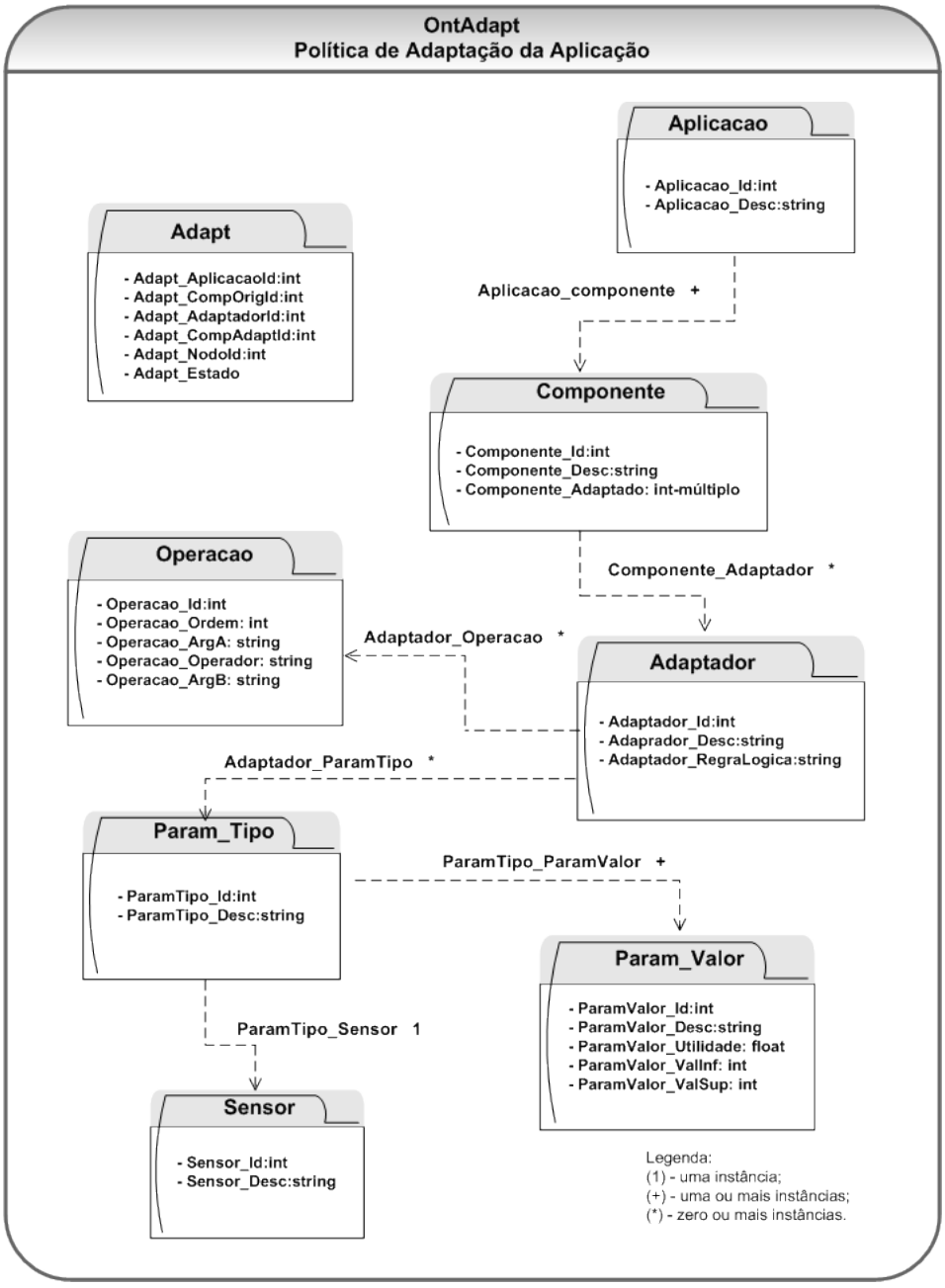


Figura 4.3: Ontologia da Adaptação - ONTADAPT - Política da Aplicação para Adaptação.

4.1.2 *Framework* FWADAPT

FWADAPT, *Framework* para definição da Política de Adaptação Dinâmica dos Componentes da Aplicação, apresentado na figura 4.4, é um *framework* com o objetivo principal de instanciar a Política da Aplicação para adaptação.

A seguir, são relacionadas as principais premissas consideradas para concepção do *Framework* FWADAPT:

- ser geral, permitindo à qualquer aplicação que siga o modelo de programação previsto para o EXEHDA, subseção 4.1.1;

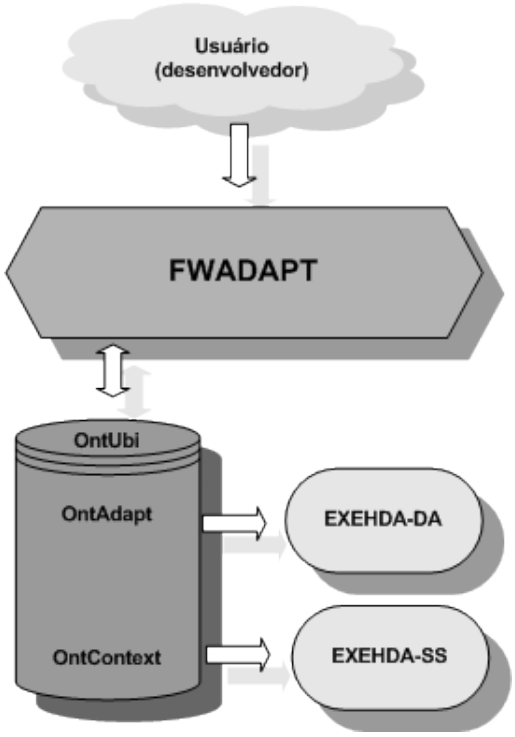


Figura 4.4: FWADAPT - instanciação da OntAdapt e OntContext.

- facultar procedimentos de edição e/ou extensão das definições previamente estabelecidas para as aplicações;
- Contemplar recurso que facilite a reutilização de especificações já existentes no que diz respeito a adaptações, regras e operações.

O FWADAPT também instanciará a Ontologia com os interesses da aplicação no que diz respeito ao Serviço de Reconhecimento de Contexto. Estas informações que integram o Contexto de Interesse da Aplicação seriam: identificação da aplicação, identificação do componente, identificação do adaptador, regra de dedução. por sua vez, para cada sensor utilizado pelo adaptador, teríamos os seguintes dados: identificação do sensor, tempo de intervalo de medição, valor de flutuação, valor inferior, valor superior, valor *default* e regra para tradução (VENECIAN, 2010).

Este *framework* é utilizado em tempo de desenvolvimento da aplicação, pelo desenvolvedor, para facilitar a configuração da política de adaptação dos componentes da aplicação. Após o FWADAPT instanciar a OntAdapt, Política de Adaptação da Aplicação, a ontologia é utilizada em tempo de execução pelo servidor de adaptação dinâmica para orientar as adaptações ao contexto dos componentes.

4.1.3 Especificação da OntAdapt

Classes em uma ontologia são conjuntos que contem os indivíduos ou os objetos de um determinado conceito ou domínio de interesse. Os indivíduos ou instâncias de classes representam objetos no domínio de interesse da classe. As propriedades em OWL representam relacionamentos entre dois indivíduos. Existem dois tipos principais

de propriedades: (i) propriedades de objeto, as quais conectam um indivíduo a outro indivíduo, denominadas **Relacionamentos** no EXEHDA-DA; (ii) propriedades de tipos de dados, que por sua vez conectam o indivíduo a um valor de tipo de dado ou a um literal. No modelo proposto, propriedades de tipos de dados são denominadas de **Atributos** da Classe.

A Figura 4.3 apresenta a composição da OntAdapt, suas classes, atributos e relacionamentos, os quais estão descritos a seguir:

- Aplicacao: são instanciadas todas as aplicações do ambiente ubíquo;
  - Aplicacao_Id: (int): identificação da aplicação;
  - Aplicacao_Desc: (string): descrição da aplicação;
- Componente: possui as instâncias de todos os componentes do ambiente ubíquo;
  - Componente_Id: (int): identificação do componente;
  - Componente_Desc: (string): descrição do componente;
  - Componente_Adaptado: (int-multiplo): se for igual a zero, é o componente que sofre adaptação. Se for diferente de zero, é o componente após adaptação. Se Componente_Id = Componente_Adaptado, permite apenas adaptação não-funcional;
- Adaptador: possui as instâncias das adaptações suportadas no ambiente ubíquo;
  - Adaptador_Id: (int): identificação da adaptação;
  - Adaptador_Desc: (string): descrição da adaptação;
  - Adaptador_RegraLogica: (string): Regra de adaptação com operadores matemáticos e lógicos para visualização do usuário desenvolvedor da aplicação;
- Operacao: possui as instâncias das operações que irão comportar o cálculo da regra de adaptação;
  - Operacao_Id (int): identificação da operação de cálculo;
  - Operacao_Ordem (int): ordem da operação dentro do cálculo da regra;
  - Operacao_ArgA (string): primeiro argumento da operação (A);
  - Operacao_Operador (string): operador entre os argumentos A e B;
  - Operacao_ArgB (string): segundo argumento da operação (B);
- Param_Tipo: possui as instâncias de todos os tipos de parâmetros do ambiente ubíquo;
  - ParamTipo_Id: (int): identificação do tipo de parâmetro;
  - ParamTipo_Desc: (string): descrição do tipo de parâmetro;
- Param_Valor: possui as instâncias dos valores para os tipos de parâmetros;
  - ParamValor_Id: (int): identificação do valor de parâmetro;

  - ParamValor_Desc: (string): descrição do valor de parâmetro;
  - ParamValor_Utilidade: (float): valor de utilidade do valor de parâmetro. Zero = sem valor de utilidade;
  - ParamValor_ValInf: (int): valor inferior do parâmetro;
  - ParamValor_ValSup: (int): valor superior do parâmetro;

- Sensor: possui as instâncias de sensores do ambiente ubíquo (classe da OntUbi);
  - Sensor_Id: (int): identificação do sensor;
  - Sensor_Desc: (string): descrição do sensor;

- Adapt: possui as instâncias dos componentes e nodos a serem executados pelo serviço Executor do *middleware* EXEHDA;
  - Adapt_AplicacaoId: (int): identificação da aplicação a ser executada;
  - Adapt_CompOrigId: (int): identificação do componente chamador do comando adaptativo;
  - Adapt_AdaptadorId: (int): identificação do adaptador processado pelo EXEHDA-DA;
  - Adapt_CompAdaptId: (int): identificação do componente adaptado que será executado. Se Adapt_CompOrigId e Adapt_CompAdaptId forem iguais o componente sofreu adaptação não-funcional, sem mudança de código;
  - Adapt_NodoId: (int): identificação do nodo em que o código adaptado deve ser executado.
  - Adapt_Estado: (int): estado da instância. zero = utilizada por outra adaptação; 1 = adaptação completa.

A seguir a descrição dos **Relacionamentos** entre as classes na OntAdapt:

- Aplicacao_Componente: instancia os componentes da aplicação, no mínimo uma instância (+);

- Componente_Adaptador: instancia as adaptações do componente, zero ou mais instâncias (*) ;

- Adaptador_Operacao: instancia as operações que compõe o cálculo da regra de adaptação do adaptador, zero ou mais instâncias (*);

- Adaptador_ParamTipo: instancia os tipos de parâmetro para a adaptação, uma ou mais instâncias (*);

- ParamTipo_ParamValor: instancia os valores para o tipo de parâmetro, uma ou mais instâncias (*);

- ParamTipo_Sensor: instancia o sensor para o tipo de parâmetro, uma instância (1);

A classe Adapt não é instanciada pelo FWADAPT, mas sim pelo módulo Disponibilização da Adaptação Inferida da estrutura do EXEHDA-DA, pois são informações geradas pelo serviço de adaptação em tempo de execução.

A tabela 4.1 resume os objetos e propriedades de objeto da ontologia OntAdapt.

Tabela 4.1: Classes e Relacionamentos da Ontologia da Política de Adaptação da Aplicação

CLASSE	**INSTÂNCIAS**
Aplicacao	Aplicações do Contexto Ubíquo
Componente	Componentes do Contexto Ubíquo
Adaptador	Adaptações previstas para o ambiente ubíquo
Operacao	Operações que podem ser usadas por adaptadores
Param_Tipo	Parâmetros que podem ser utilizados em adaptações
Param_Valor	Valores e Utilidade possíveis para Parâmetros
Adapt	Adaptações Inferidas
RELACIONAMENTO	**INSTÂNCIAS**
Aplicacao_Componente	Componentes das aplicações
Componente_Adaptador	Adaptações suportadas pelos componentes
Adaptador_Operacao	Operações utilizadas pelas regras de adaptação
Adaptador_ParamTipo	Tipos de parâmetros utilizados nas adaptações
ParamTipo_ParamValor	Valores e utilidade de cada parâmetro
ParamTipo_Sensor	Sensor usado no tipo de parâmetro

A classe Componente foi definida no mesmo nível da classe Aplicacao com a finalidade de tornar o modelo mais geral, possibilitando que um componente possa ser instanciado em mais de uma aplicação, através do relacionamento Aplicacao_Componente. O mesmo procedimento foi adotado entre as classes: Componente e Adaptador, Adaptador e Param_Tipo, Param_Tipo e Param_Valor, Param_Tipo e Sensor.

A Política de Adaptação da Aplicação é composta de especificações de regras que determinam o comportamento dos componentes da aplicação. A Política constitui a forma na qual os desenvolvedores definem seus objetivos em relação à adaptabilidade ao contexto de interesse da aplicação.

4.2 Estrutura do Serviço de Controle da Adaptação Dinâmica Proposto

O modelo de controle da adaptação dinâmica de aplicações em ambiente ubíquo, quando da tomada de decisão de adaptação, considera as categorias de informações: (i) o contexto, com base em informações monitoradas, informações semânticas e inferências a partir destas mesmas informações; (ii) política de adaptação da aplicação; (iii) preferências do usuário.

Para isto, o EXEHDA-DA comunica-se, através de interfaces definidas, tanto com as aplicações (comandos de adaptação) quanto com os outros serviços do *middleware*

(notificações, dados contextuais e decisões de adaptação), facultando que o mesmo possa ser desenvolvido, modificado, e estendido de forma independente.

A figura 4.5 mostra os relacionamentos entre os serviços Controle da Adaptação e Reconhecimento de Contexto. Também a relação destes serviços com o ambiente ubíquo e as aplicações, identificando o que acontece em tempo de desenvolvimento e o que acontece em tempo de execução.

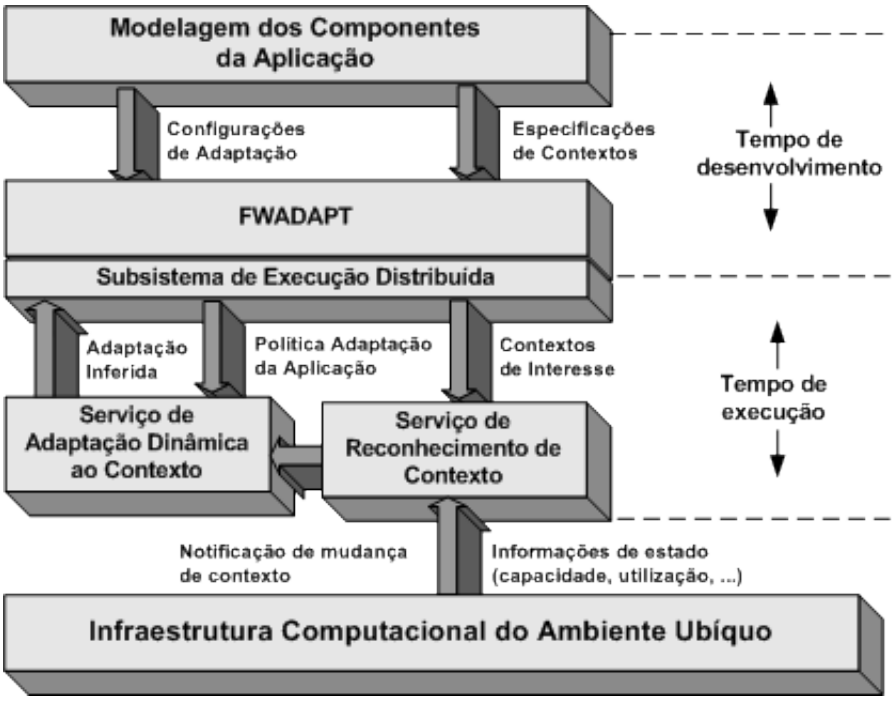


Figura 4.5: Visão Geral do Serviço de Adaptação Dinâmica ao Contexto

A figura 4.6 apresenta a arquitetura de software do Serviço de Adaptação Dinâmica ao Contexto, EXEHDA-DA, com seus módulos, interfaces entre eles, e a comunicação com os outros serviços do *middleware*.

O EXEHDA-DA é composto pelos módulos:

- **Tratamento do Contexto Notificado**: recebe do Serviço de Reconhecimento de Contexto a identificação do contexto notificado (CN_Id). Através da identificação da notificação de mudança de contexto, será acessada a classe Contexto_Notificado e os relacionamentos CN_ContextoNotificadoSensor e CN_Usuario da OntContext, instanciada pelo servidor de Reconhecimento de Contexto. Esta classe fornece as informações de mudança de contexto de interesse do componente da aplicação, as informações computacionais do contexto para a adaptação, valores dos sensores do ambiente computacional. O mecanismo de tratamento de contexto notificado também consulta a política da aplicação, através do processamento semântico, e acessa as informações necessárias para o cálculo da regra de adaptação em questão. Estas informações são os tipos, valores e utilidades de parâmetros necessários para o cálculo da regra de adaptação, configurados na OntAdapt, ítem 4.1.3 para a adaptação do componente da aplicação. Neste me-

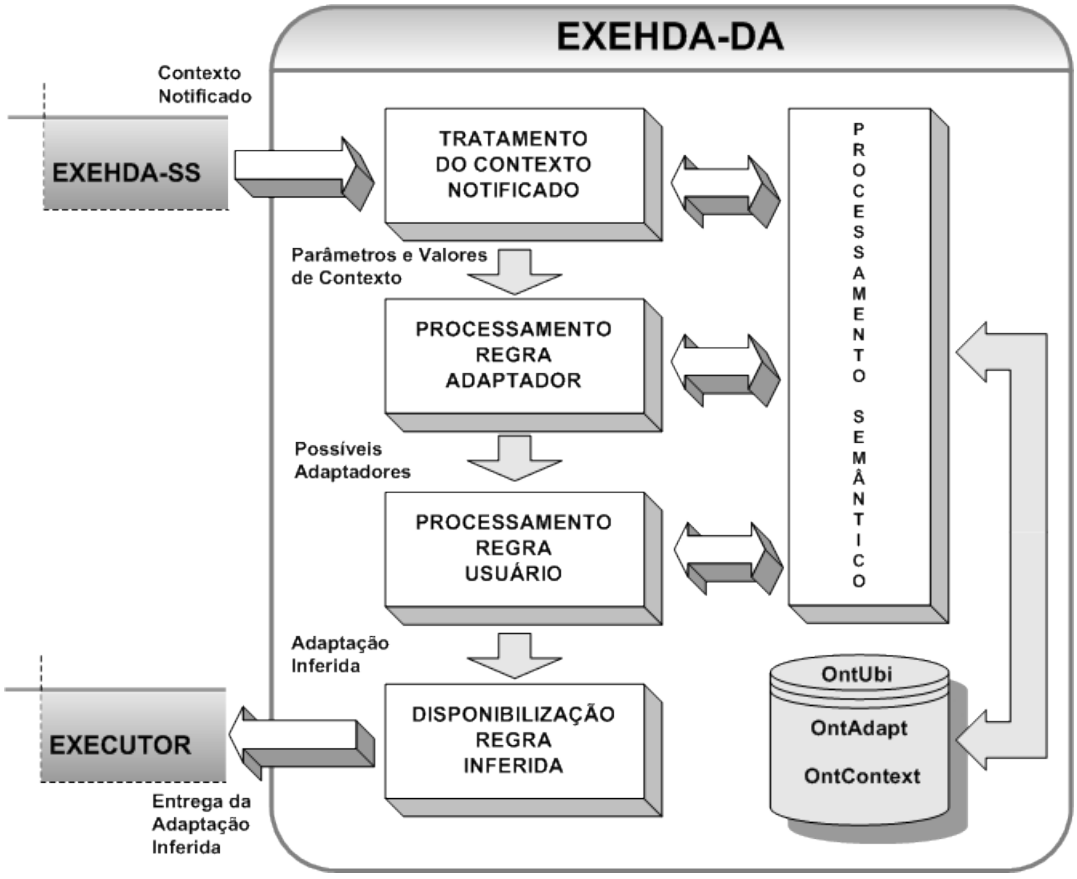


Figura 4.6: Arquitetura de Software do EXEHDA-DA

canismo são preparadas todas as informações necessárias para a tomada de decisão da adaptação. O Contexto Notificado possui as informações de contexto que são de interesse da aplicação e que sofreram mudanças, que devem ser utilizadas em um processo de adaptação como recursos oferecidos pelo ambiente computacional. As informações de contexto notificado são: CN_Id (identificação da notificação), CN_AplicacaoId (identificação da aplicação), CN_ComponenteId (identificação do componente), CN_AdaptadorId (identificação da adaptação), Usuario_Id (identificação do usuário) e múltiplos valores de Sensor_Id (identificação do sensor) e CN_SensorValor (valor do sensor), CN_SensorTrad (valor traduzido do sensor) e CN_NodoId (identificação do nodo do sensor). A classe Contexto_Notificado contém os contextos notificados, relacionando-se com ContextoNotificado_Sensor, que chama-se CN_ContextoSensor que possui as instâncias dos sensores e respectivos valores destes. Outro relacionamento do Contexto_Notificado é com a classe Usuario da OntUbi, denominado CN_Usuario.

- **Processamento da Regra do Adaptador**: este processamento produz a lista com uma ou mais ações de adaptação, classificada por critério de utilidade. Através da API Jena, são inferidas as possíveis opções para ação adaptativa, utilizando a regra de adaptação e os valores de parâmetros necessários e oferecidos para calculá-la. Utilizando a função de utilidade serão classificadas as possíveis ações, em ordem de mais alta utilidade de ação adaptativa para a mais baixa. A adaptação funcional e a adaptação não-funcional são computadas da mesma maneira, através de regra

de adaptação.
A regra de adaptação, utilizada pelo EXEHDA-DA para inferir ações de adaptação, é composta por uma sequência de operações (instâncias da Classe Operacao), que estão relacionadas à classe Adaptador. As informações fornecidas por este módulo são uma ou mais instâncias na Classe Adapt da OntAdapt, subseção 4.1.3.

- **Processamento da Regra do Usuário**: são acessadas as preferências do usuário para a Aplicação, na classe Usuario da OntUbi. Através destas preferências classificadas por valor de utilidade, será selecionada, dentre as possíveis ações adaptativas, inferidas no módulo anterior, aquela que tiver maior valor de utilidade para o usuário, fornecendo-lhe um maior nível de satisfação. Se não for configurada a preferência do usuário para este tipo de adaptador e aplicação, será selecionada a ação de maior valor de utilidade inferida no mecanismo anterior. As informações fornecidas por este módulo são do mesmo tipo que as fornecidas pelo módulo anterior. Porém, constituem apenas uma instância na Classe Adapt, e com o valor do atributo Adapt_Estado igual a 1.

- **Disponibilização da Adaptação Inferida**: neste módulo são retidas as informações necessárias para a ação de adaptação, inferida no módulo anterior, pelo *middleware* EXEHDA, através do serviço Executor. No momento da execução da adaptação, solicitada pelo comando adaptativo que está inserido no código do componente da aplicação em execução (Adapt_CompOrigId), o serviço Executor irá pesquisar pelo Adapt_AplicacaoId e Adapt_CompOrigId, na classe Adapt da OntAdapt, para acessar as informações para execução da adaptação inferida. Neste momento, esta adaptação é excluída da classe Adapt.
  Se Adapt_CompOrigId for diferente Adapt_CompAdaptId, significa uma adaptação funcional e que deverá ser trocado o código do componente. O acesso ao repositório de códigos de componentes deve ser feito pelo conteúdo dos atributos Adapt_AplicacaoId + Adapt_CompAdaptId. Se Adapt_CompOrigId e Adapt_CompAdaptId forem iguais, significa uma adaptação não-funcional, sendo mantido o mesmo código. A informação Adapt_Nodo informa a identificação do nodo onde o componente deverá ser executado.
  O atributo Adapt_Estado com valor igual a zero, indicará que a inferência não deverá ser notificada para a aplicação, pois é um resultado intermediário que será usada por outra decisão de adaptação.

- **Processamento Semântico**: neste mecanismo é utilizada a linguagem SPARQL, e a API Jena para acessar, instanciar e inferir informações necessárias para computar as regras de adaptação. Os produtos destas regras são entregues para o mecanismo que faz a retenção das decisões de adaptação ao contexto para os componentes das aplicações. Estas informações estão nas ontologias utilizadas neste serviço: OntUbi, OntContext e OntAdapt.

O serviço EXEHDA-DA, assim como o repositório com as informações semânticas, estão localizados no nodo base das células que compõem o ambiente ubíquo.

4.3 Comandos Adaptativos para Uso nas Aplicações

O *middleware* EXEHDA (YAMIN, 2004) fornece comandos para modelar estratégias de adaptação mais usuais em aplicações ubíquas e distribuídas. A implementação dos comandos é gerenciada pelo Serviço de Adaptação Dinâmica, EXEHDA-DA, com consulta e inferência às políticas de adaptação definidas pelo programador da aplicação, através do *framework* FWADAPT, conforme arquitetura apresentada na seção anterior. Os comandos fornecidos são: `ativa`, `move`, `rescalona`, `desconecta`, `reconecta`, `noContexto`.

`ativa`
O comando `ativa` implementa a ativação de componentes.

Sintaxe: *

`ativa` (Componente_Id, Nodo_Id | Adaptador_Id);

Por exemplo,
`ativa` (000021, 001);

indica que o componente 000021 será disparado, utilizando a adaptação 001 configurada na Ontologia da Política da Aplicação (OntAdapt).

Se o Componente_Id do comando `ativa` possui um valor de Componente_Adaptado na classe Componente da OntAdapt, a próxima informação do comando `ativa` é o Adaptador_Id, identificação da adaptação da OntAdapt que possui a respectiva regra de adaptação para ativação do componente.
Caso o Componente_Id não tenha Componente_Adaptado na OntAdapt, a próxima informação do comando `ativa` é Nodo_Id com a identificação do nodo onde deverá criado o componente. Nesta alternativa, o comando `ativa` não dispara o processo adaptativo, o componente será instanciado no Nodo_Id indicado.

`move`
Este comando indica que o componente irá ser submetido a um procedimento de migração.

Sintaxe: `move` (Componente_Id, Nodo_Id, Adaptador_Id);

Se neste comando, Componente_Id e Adaptador_Id forem iguais a zero, significa que o programador deseja uma migração do componente para o equipamento identificado por Nodo_Id, sem passar pelo processo de adaptação.

`move` (0, 00010, 0);

Se Nodo_Id e Adaptador_Id forem iguais a zero, significa que o programador quer a migração do componente em execução para o nodo aonde está o Componente_Id sem

*A sintaxe segue a notação BNF estentida utilizada pelo JAVACC - software para gerar compilador em Java. As notações utilizadas indicam:
| –> alternativas, ? –> opcional, * –> zero ou mais ocorrências.

passar pelo processo de adaptação.

`move` (000015, 0, 0);

Se Adaptador_Id for diferente de zero, significa que o componente deverá seguir os procedimentos configurados na OntAdapt de acordo com o que estiver especificado em Adaptador_Id

`move` (0, 0, 008);

Esta operação é realizada, em geral, para otimização dos aspectos de comunicação e/ou otimização no aproveitamento dos recursos computacionais.

`rescalona`

Permite ao programador influenciar na distribuição dos componentes de software sobre os recursos computacionais. Através deste comando, sob certas condições de contexto, definidas pelo programador na OntAdapt, a aplicação pode forçar um reescalonamento para melhorar aspectos não-funcionais da execução, dentre eles, seu desempenho.

Sintaxe: `rescalona` (Componente_Id, Adaptador_Id);

onde, Adaptador_Id indica a regra e os respectivos parâmetros que serão utilizados a partir da execução do comando `rescalona`. As regras já estão pré-definidas na Política da Aplicação.

`desconecta`

Este comando promove a desconexão voluntária do componente do ambiente ubíquo provido pelo *middleware*. O componente pode continuar a executar, embora a comunicação com os componentes residentes em outros nodos fique postergada.

Sintaxe: `desconecta` Adaptador_Id;

O comando desconecta promove uma chamada de adaptação, na qual o serviço EXEHDA-DA conduz um protocolo de duas fases para implementação da desconexão: (i) inicialmente o estado do elemento de contexto é definido como "desconexão em curso", permitindo que os componentes realizem as operações necessárias para operarem no modo desconectado, (ii) a seguir, o elemento de contexto é definido como "desconectado", indicando aos componentes registrados que o processo de conexão foi efetivado.

`reconecta`

Sintaxe: `reconecta` Adaptador_Id;

O comando indica a necessidade de conexão ao ambiente ubíquo. Este comando é uma chamada de procedimento adaptativo ao serviço EXEHDA-DA, de forma análoga e complementar ao comando `desconecta`. A reconexão dispara a troca de estado do componente para "conectado".

seleciona

Sintaxe: `seleciona` Aplicacao_Id, Componente_Id, Adaptador_Id;

Este comando é uma chamada para o processamento de uma regra de adaptação. Seu produto é a seleção da ação adaptativa (funcional ou não-funcional), considerando o contexto notificado. Como este comando não promove ação nos componentes da aplicação, via de regra, o mesmo é complementado por outro comando adaptativo.

noContexto

O comando `noContexto` agenda um bloco de comandos que serão executados quando um determinado estado de contexto vir a ocorrer. A implementação deste comando está associada ao registro de ações na OntAdapt realizado pelo serviço EXEHDA-DA.

O próximo trecho de código exemplifica a criação do componente Componente_Id quando um processador se tornar disponível. Neste exemplo, o comando `ativa` ficará bloqueado até receber a notificação de processador disponível.

```
context processor::idle;
.....
noContexto Aplicacao_Id, Componente_Id, Adaptador_Id {
ativa (Componente_Id, Nodo | Adaptador_Id);
}

noContexto 012 {
ativa (000035, 00001); }
```

012: identificação de adaptação de contexto da classe Adaptador na OntAdapt: processador disponível.

000035: identificação de componente da classe Componente na OntAdapt.

00001: identificação de adaptação da classe Adaptador na OntAdapt: ativação de componente.

O qualificador *sync*, colocado antes do comando `noContexto`, indica uma semântica síncrona, ou seja, o componente ficará bloqueado a espera da informação do contexto.

A figura 4.7 é um Diagrama de Sequência do comando adaptativo `ativa`. Este comando estará contemplado dentro do componente e este componente terá configurada uma adaptação na OntAdapt.

Uma vez definido o modelo semântico do EXEHDA-DA, o *framework* FWADAPT, utilizado para instanciar a Política de Adaptação Dinâmica das aplicações, bem como a estrutura do EXEHDA-DA, o próximo capítulo irá discorrer sobre as tecnologias de software utilizadas na implementação do modelo proposto.

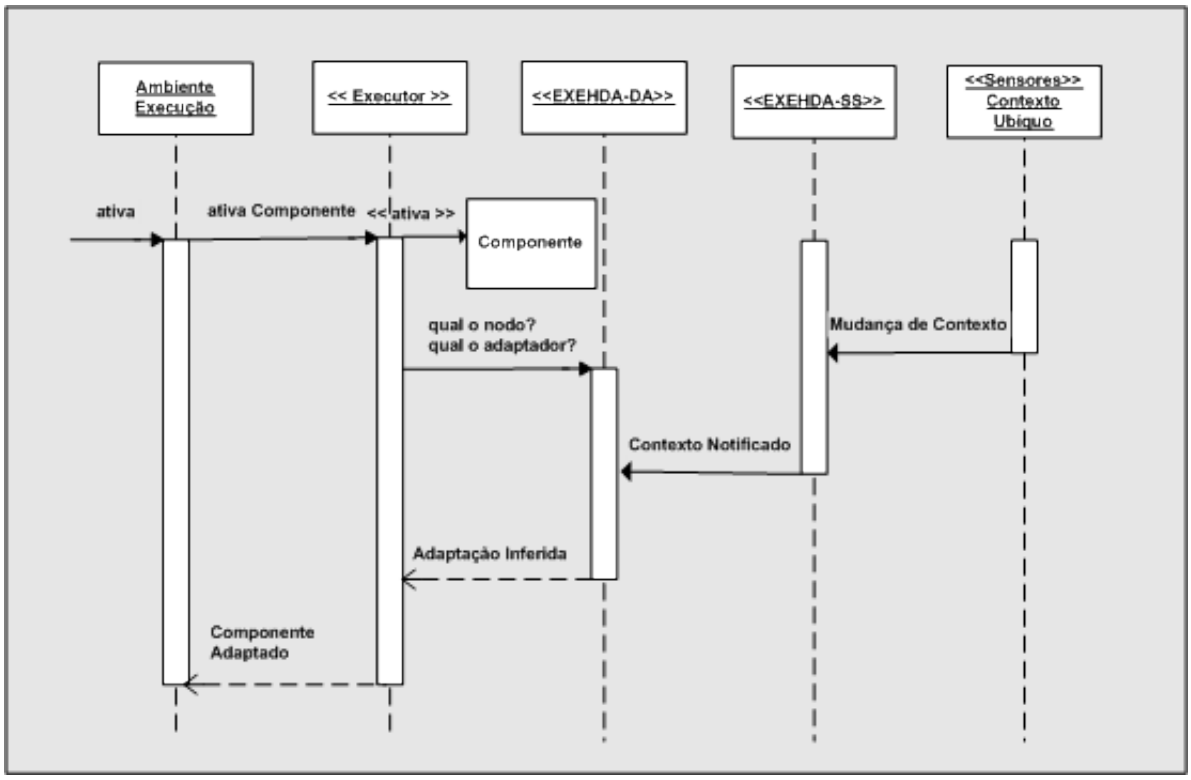


Figura 4.7: Diagrama de Sequência do comando adaptativo `ativa`.

5 EXEHDA-DA: TECNOLOGIAS DE SOFTWARE UTILIZADAS

Neste capítulo são tratados os aspectos referentes à implementação do EXEHDA-DA. Sendo caracterizada a seleção das tecnologias empregadas quando da implementação do Serviço de Controle de Adaptação Dinâmica, EXEHDA-DA, e do desenvolvimento dos estudos de caso trabalhados, tratados no capítulo 6.

As tecnologias adotadas na implementação:

- Linguagem Java no desenvolvimento do EXEHDA-DA e do FWADAPT;
- RDF como modelo de descrição e representação dos dados e metadados;
- Linguagem OWL para criação, manutenção e instanciação das ontologias;
- API Jena no EXEHDA-DA e FWADAPT;
- SPARQL na API Jena;
- Prótegé na criação e instanciação da OntUbi.

5.1 Linguagem Java

A linguagem Java vem sendo utilizada no desenvolvimento dos serviços do EXEHDA, opção definida quando da concepção do middleware (YAMIN, 2004). Este foi o fator central para o emprego desta linguagem no desenvolvimento do serviço EXEHDA-DA. Dentre as características de Java, consideradas importantes para o EXEHDA-DA, podemos destacar (LINDHOLM; YELLIN, 1997):

- portabilidade: na Computação Ubíqua podemos ter processamento distribuído e heterogeneidade tanto de software quanto de hardware. Portanto, a independência de plataforma é uma característica importante da linguagem de programação Java para estes ambientes. Esta característica decorre da disponibilização de uma vasta API padronizada e da utilização de uma representação de código executável baseada em um formato de instrução portável, denominado *bytecode*. Java é orientada a objetos, onde os programas são compostos por um conjunto de classes. Compilando estas classes, o código fonte é convertido para o formato de *bytecodes*. Essa sequência de *bytecodes* é interpretada e executada através de um elemento

intermediário que apresente uma interface uniforme sobre as diferentes arquiteturas de hardware existentes. Tal elemento, no *framework* Java, é a Máquina Virtual Java (JVM). A abordagem de máquina virtual, transfere a manutenção da portabilidade das aplicações para os implementadores da JVM, que deve ser implementada e compilada para cada plataforma alvo a ser suportada. Atualmente, versões da JVM são encontradas nas mais diversas plataformas, tais como: Windows, Linux, Solaris, MacOS, além de versões desenvolvidas especificamente para dispositivos embarcados, como Smartphones, PDAs, entre outras;

- suporte à carga dinâmica de código: o processo de carga de classes é feito de forma dinâmica na JVM, sendo controlado pelo Class Loader. Este, é um objeto plugável do sistema, podendo ser personalizado de forma a implementar a carga de classes, por exemplo, a partir do sistema de arquivos da máquina, ou de um servidor HTTP na Internet, ou mesmo através de um banco de dados;

- segurança: a máquina Java (JVM) executa as aplicações realizando uma forte verificação dinâmica sobre as classes Java, evitando os acessos indevidos à memória para evitar possíveis problemas de segurança. Na plataforma Java, também são controlados os acessos aos recursos do sistema operacional hospedeiro (sistema de arquivos, periféricos, *sockets*), sendo gerenciados pela JVM e não acessados diretamente pela aplicação. Estes aspectos favorecem a distribuição de código em um sistema com grande heterogeneidade de hardwares, sem que comprometa a segurança dos nodos;

- suporte à concorrência e sincronização: o *framework* Java disponibiliza de forma nativa em sua API um controle da concorrência através da classe *Thread*, que serve para disparar linhas de execução concorrentes dentro da aplicação. Este aspecto reforça o uso da linguagem nos ambientes heterogêneos da Computação Ubíqua por favorecer a portabilidade das aplicações que necessitam deste recurso. Outras linguagens, como C e C++ muitas vezes necessitam de bibliotecas para dar suporte à concorrência, e estas em muitos casos são específicas para determinadas plataformas. Aplicações desenvolvidas usando as *threads* Java poderão ser executadas sem recompilação;

- produtividade no desenvolvimento estruturado de software: em Java vários aspectos tendem a beneficiar o processo de desenvolvimento de software de um modo geral. A independência de plataforma reduz os custos de desenvolvimento e manutenção por não necessitar versões do código específicas para cada plataforma. A susceti-bilidade de erros é reduzida devido à um gerenciamento automático de memória, à inexistência de aritmética de ponteiros e a um tratamento estruturado de exceções. Por ser uma linguagem orientada a objetos, o Java favorece o encapsulamento de dados e componentes. Somado a isso, o Java apropriou-se de características interessantes de outras linguagens de programação como C e C++, o que resultou em uma linguagem com pouca ocorrência de erros e de fácil aprendizado;

- suporte ao desenvolvimento de aplicações distribuídas através de objetos RMI (*Remote Method Invocation*): os objetos Java passam a estar habilitados a receberem invocações de métodos advindas de outros nodos, ou seja, existe todo um suporte à execução de invocações remotas de método. A abstração provida por RMI

mostra-se oportuna, pois preserva, no âmbito de sistemas distribuídos, o paradigma orientado a objetos nativo de Java, uniformizando o processo de desenvolvimento de software distribuído.

5.2 RDF e RDF Schema

O modelo RDF, (*Resource Description Framework*)(http://www.w3.org), apesar de comumente confundido com uma linguagem, é na verdade um modelo para escrever metadados. Sua característica está na forma com que o modelo de dados utiliza metadados para descrever recursos, apresentando um significado ao recurso. RDF representa a relação entre os conceitos, indo além de um modelo para a descrição de metadados. O desenvolvedor da aplicação RDF, através do RDF Schema, define o vocabulário usado, sua semântica, e que propriedades podem ser aplicadas a que tipos de recursos, e outras informações importantes sobre o domínio. O objetivo do RDFS é determinar o vocabulário usado nos modelos RDF.

Este modelo visualiza os dados como recursos, identificados através de URIs, sendo utilizado em linguagens do tipo XML ou OWL. O recurso pode ser caracterizado como um objeto, 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. A descrição das relações entre os objetos é feita através de triplas do tipo <objeto, atributo, valor>. 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.

5.3 OWL

A OWL é uma linguagem para definir e instanciar ontologias. 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 usuários. 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 padrão para o processamento de conteúdo semântico da informação. 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 entendida pelos usuários, porém esta questão resolve-se com o emprego 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.

A linguagem OWL foi escolhida para construção do modelo ontológico do EXEHDA-DA, utilizando a sub-linguagem OWL-DL, ítem 2.2.2.2. Esta versão de OWL fornece suporte a metadados RDF, abstrações de classes, generalização, agregação, relações de transitividade, simetria e detecção de inconsistências.

5.4 API JENA

Na implementação também é utilizada a API (*Application Program Interface*) Java do toolkit Jena, desenvolvida pela HP, Hewlett-Packard Company, (MCBRIDE, 2008). Jena é utilizado no desenvolvimento do EXEHDA-DA e do *framework* FWADAPT, pois a API permite manipulação dinâmica de modelos ontológicos, o desenvolvimento de aplicações baseadas em ontologias, e suporta as linguagens de consulta de dados RDF e SPARQL. A inferência em ontologias é obtida a partir de diferentes informações ou a partir de mais de uma instância de determinada informação, com objetivo de uma tomada de decisão.

Jena é uma ferramenta de grande relevância não só para o desenvolvimento de aplicações RDF, mas também para aplicações que utilizam ontologias. É um *framework* para o desenvolvimento de aplicações com enfoque semântico, que provê um ambiente de programação para RDF, RDFS e OWL. JENA provê uma série de classes Java para construir modelos RDF/RDFS, manipulá-los, consultá-los através de linguagem tipo SQL, lê-los e gravá-los em arquivo, além de capacidade de raciocínio a qual que pode ser usada para inferir conhecimento a partir de uma ontologia.

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

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

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 irã 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. Onde *statements* é uma expressão que pode dar origem a uma nova tupla, representada por num objeto novo (VERZULLI, 2001).

A arquitetura base da API de ontologias do Jena é composta por 3 camadas, mostradas na Figura 5.1:

- *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 graph*: OWL está definido sobre RDFs. O Jena usa a API de RDF para manipular as ontologias.

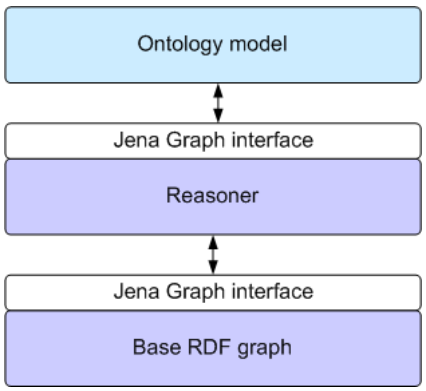


Figura 5.1: Arquitetura Base da API Jena (DICKINSON, 2008)

Em todas as camadas existem padrões que são usadas para permitir que a API seja mais genérica. É usada a *Factory Pattern* para a construção das classes 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 implementações da superclasse que representa o conceito.
A Figura 5.2 mostra os módulos do motor de inferências da API Jena.

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

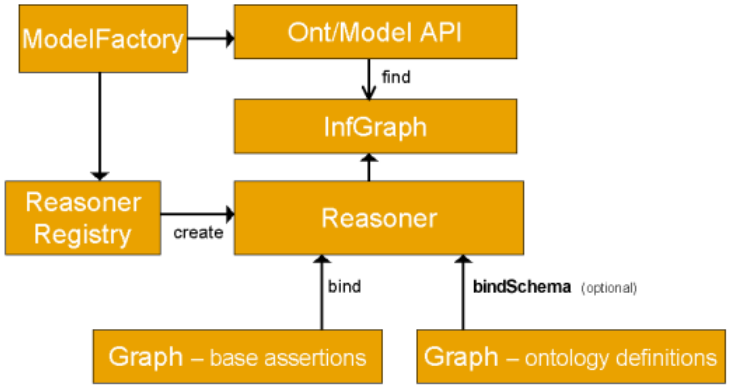


Figura 5.2: Motor Inferência da API Jena (REYNOLDS, 2008)

de inferência, que vêm incluídos na distribuição do Jena, são básicos e são usados para realizar tarefas comuns. Também 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. As regras estão definidas sobre BNF (*Backus-Naur Form*) 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. A OWL está definida sobre RDFS, logo o Jena usa as suas *APIs* de *RDF* para poder manipular as ontologias. Dai a existência do 3º módulo.

O Jena vem com um *reasoner* genérico que depois pode ser personalizado 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, DL (*Description Logics*) *Implementation Group*: o *Pellet*, o *Racer* e o *FaCT*.

Jena possui classes para executar *queries* sobre modelos ontológicos. Essas *queries* são feitas em RDQL ou a linguagem mais avançada para executar *queries*, SPARQL.

À nível de 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.

Na nossa implementação para fins de persistência utilizamos MySQL. Jena possui um driver em sua biblioteca, mysql-connector-java, permitindo ao Jena importar uma ontologia em OWL para o Banco de Dados MySQL. As atualizações das ontologias também foram feitas na base MySQL.

5.5 SPARQL

Em 2007 a W3C (http://www.w3.org) lançou como recomendação o SPARQL. *Protocol and RDF Query Language*, SPARQL, é a tecnologia de buscas para ontologias. 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.

Uma consulta é representada pela estrutura SELECT FROM WHERE, onde, assim como em SQL, SELECT identifica as variáveis, que são iniciadas por um ? (ponto de interrogação) a serem retornadas para a aplicação, FROM especifica o modelo que será consultado através de um URI, WHERE impõe restrições aos dados que devem ser recuperados através de uma lista de padrões de triplas. Adicionalmente temos o AND que especifica uma expressão booleana e serve de filtro para a consulta e o USING onde são definidos prefixos para URIs a fim de tornar a consulta mais legível.

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. As *queries* podem ser executadas sobre modelos RDF e também em modelos descritos em OWL, uma vez que estes são descritos sobre RDF. O SPARQL 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, 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 <>.

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

O conceito de preferência fazem parte do nosso cotidiano. Às vezes preferências são expressas em termos de desejos ou objetivos, mas nem sempre elas podem ser totalmente satisfeitas. Essa é a principal característica de preferências: quando não são atendidas totalmente, as pessoas geralmente estão preparadas para alternativas diferentes ou um pouco piores do que a ideal. A partir desta premissa, o EXEHDA-DA utiliza SPARQL *Preference* para processar as preferências de usuário.

SPARQL *Preference*

SPARQL Preference é uma uma extensão para a linguagem de consulta SPARQL com noção de preferências (SIBERSKI; PAN; THADEN, 2006) e (ABEL et al., 2007). Foi formulada uma linguagem de consulta de dados semânticos com o conceito de preferências através da inclusão do PREFERRING, com duas expressões atômicas de preferências e duas formas de combinação de preferências. As duas expressões de preferências definidas são: (i) preferências booleanas: especificadas como uma condição booleana onde os resultados que satisfazem a condição são preferidos sobre os resultados que não satisfazem; (ii) preferências pontuadas (*scoring preferences*): são especificadas por dois operadores (*HIGHEST*, *LOWEST*) seguidos de uma expressão numérica. Assim, os resultados desta expressão que levam ao maior (HIGHEST) ou menor (LOWEST) valor são preferidos aos menores ou maiores.

Essas preferências atômicas podem ser combinadas em dois tipos de preferências multidimensionais: (i) preferência composta de Pareto, *Pareto Composed Preference*: consiste de duas expressões de preferências conectadas por um operador *AND* e cada uma das expressões é avaliada independentemente. Assim, um objeto é preferido se este é melhor em uma das preferências e pelo menos igualmente bom na outra preferência; (ii) Preferência em cascata, *Cascading Preference*: consiste de duas expressões de preferências conectadas pelo operador *CASCADE*, onde a primeira preferência é avaliada antes, enquanto que a segunda preferência somente é considerada com os objetos dos quais são igualmente importantes com relação à primeira preferência. Com isso, foi estendido o motor de busca ARQ para SPARQL que faz parte do *framework* com o construtor e as expressões de preferência do SPARQL *Preference*.

Um exemplo de uso do SPARQL *Preference*, seria a seguinte tomada de decisão. "Eu gosto mais de carros pretos do que carros brancos. Se possível que não custe mais que 50.000 e que tenha os menores valores de quilômetros rodados e a maior quantidade de itens opcionais". Na figura 5.3 temos um exemplo de uma sentença SPARQL *Preference* contendo as expressões de preferência já citadas. Na linha 10 temos o construtor modificador indicado que a partir daquela linha se tem expressões de preferência. Nas linhas 11 e 12 temos a primeira expressão de preferência (Eu gosto mais de carros pretos do que carros brancos) formulada por uma combinação de preferência em cascata de duas preferências booleanas (cor preta mais importante que cor branca). Na linha 14 temos uma preferência atômica booleana representando os preços menores que 50.000. Nas linhas 16 e 18 temos dois exemplos das preferências pontuadas onde são valorizadas as menores quilometragens e a maior quantidade de itens opcionais (LOWEST e HIGHEST, respectivamente). Por fim, nas linhas 13, 15 e 17 temos o operador de preferência composta de Pareto (AND) combinando todas essas preferências independentes para que o princípio do ótimo de Pareto seja aplicado. Em síntese, o princípio do ótimo de Pareto reza que certos objetos dominam outros se eles são considerados melhores em uma das preferências e pelo menos equivalentes em todas as outras preferências.

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX car: <http://www.unifor.br/mia/carro.owl#>
3 SELECT *
4 WHERE
5 {   ?id rdf:type car:Carro
6     ?id car:temCor ?cor
7     ?id car:temPreco ?preco
8     ?id car:temKM ?km
9     ?id car:numeroItensOpcionais ?numOpcionais }
10 PREFERRING
11    ?cor= PRETO.
12         CASCADE ?cor= BRANCO.
13 AND
14    ?preco<= 50000.
15 AND
16    LOWEST(?km)
17 AND
18    HIGHEST(?numOpcionais)
```

Figura 5.3: Exemplo SPARQL Preference (SIBERSKI; PAN; THADEN, 2006)

5.6 PROTÉGÉ

Protégé (http://protege.stanford.edu/) 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.

No Protégé podemos editar graficamente ontologias, possuindo uma vasta quantidade de *plugins*, importando e exportando 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.

O editor de ontologias e bases de conhecimento Protégé oferece um amplo suporte à linguagem OWL. Através dele podemos carregar e salvar ontologias em formato RDF ou OWL, editar e visualizar as classes e propriedades, executar motores de inferência como o Racer e o Fact++, para checagem de consistência e classificação (KNUBLAUCH; MUSEN; RECTOR, 2004). Também é possível editar instâncias OWL para marcação voltada para a Web Semântica. O Protégé destaca-se por ser uma ferramenta que oferece um vasto suporte a cada uma das camadas da Web Semântica. Além de ser um poderoso editor de ontologias, o Protégé possui uma série de plug-ins que permitem trabalhar com bases de conhecimento em formato XML, RDF e OWL, além de possuir suporte a ferramentas de raciocínio automático como o Algernon e o Jess e consultas a bases de conhecimento.

O modelo ontológico OWL da OntUbi com todas as classes e propriedades de dados e de objetos foi criado através do Protégé. Para os estudos de caso, a OntAdapt e a OntContext foram instanciadas utilizando o *framework* FWADAPT, ítem 4.1.2. As demais classes e relacionamentos da OntUbi foram instanciadas utilizando o Protégé.

O próximo capítulo irá explorar estudos de caso para avaliação do EXEHDA-DA. As aplicações que integram estes estudos de caso foram implementadas utilizando as tecnologias descritas neste capítulo.

6 EXEHDA-DA: ESTUDOS DE CASO

O modelo de controle de adaptação do EXEHDA-DA por ser programável por regras personalizáveis por aplicação, permite a sua utilização para vários tipos de cenários de uso, em diferentes domínios de contexto. Nesta perspectiva o EXEHDA-DA contribui para que as premissas operacionais da Computação Ubíqua sejam disponibilizadas na rotina de trabalho de diferentes profissionais, tais como médicos, pesquisadores, agrônomos, veterinários. O EXEHDA-DA na sua operação contempla a definição de um contexto de interesse, de uma política de adaptação, bem como pressupõe um modelo de programação com aplicações baseadas em componentes.

Neste capítulo são tratados dois estudos de caso, um explorando o controle da adaptação funcional e outro o controle da adaptação não-funcional.

6.1 Estudo de Caso 1: Acompanhamento Ubíquo de Pacientes (AUP)

Nesta seção é apresentado o primeiro estudo de caso, o qual consiste de uma aplicação sintética direcionada à área médica, cujas funcionalidades destacadas nesta seção tem por objetivo explorar o controle da adaptação ao contexto promovido pelo EXEHDA-DA, quando do atendimento das demandas introduzidas pelo comportamento ubíquo da mesma.

As publicações de dados dos sensores foram produzidas por softwares que instanciaram valores pré-definidos no servidor de contexto do EXEHDA. Tanto os valores praticados, quanto os níveis de alerta, bem como as regras e parâmetros de adaptação são aproximações médias sem compromisso de traduzir uma realidade médica precisa.

6.1.1 Objetivos da AUP

A premissa buscada é qualificar o acompanhamento de pacientes, que não estejam internados em Unidades de Tratamento Intensivo (UTI). Particularmente, com o intuito de aumentar o período de tempo sob cuidado dos agentes de saúde é prevista a operação da AUP também em dispositivos móveis. Nesta perspectiva, os objetivos contemplados na aplicação AUP são:

- exibir dados de pacientes adquiridos dinamicamente por mecanismo de sensoreamento de sinais;

- emitir, de forma automatizada, diferentes níveis de alertas, em função dos dados sensoreados, para os agentes de saúde;
- integrar o serviço de alertas da aplicação a rede aberta de comunicação *Google Talk*;
- prover possibilidade de ter acesso, tanto a partir de dispositivo móveis, como de mesa;
- permitir acesso ubíquo ao histórico dos dados sensoreados dos pacientes por agentes de saúde.

A aplicação AUP está concebida para vir a ser integrada ao Sistema de Informações do Hospital. Nesta etapa de teste foi utilizada uma base de dados privada, sobre a qual foi possível ter liberdade no que diz respeito à manipulação dos dados utilizados.

As informações cadastrais são de três naturezas: (i) dados cadastrais de pacientes; (ii) dados de agentes de saúde; (iii) dados aquisitados dinamicamente pela rede de sensores de sinais vitais de pacientes.

Na versão da AUP, utilizada para avaliação do EXEHDA-DA, os agentes de saúde podem ser tanto enfermeiros quanto médicos. Os mesmos podem disparar dois procedimentos:

- módulo de captura dinâmica dos dados correspondentes aos sinais vitais de pacientes. Os sinais considerados nesta versão são batimentos cardíacos, pressão arterial e temperatura. Em função dos valores dos sinais coletados, são produzidos diferentes níveis de alertas aos agentes de saúde, por sua vez, dependendo do nível de alerta é disponibilizada a opção de envio de mensagem ao agente de saúde ou, ainda, no caso de alerta máximo, o envio incondicional de mensagem;
- módulo de exibição de histórico das últimas 48 horas de alertas dos pacientes. Este histórico contempla a data e hora do alerta, seu nível e os valores dos sinais vitais, subsidiando a tomada de decisão do médico em relação ao estado clínico do paciente.

A estratégia de implementação da aplicação AUP teve por finalidade reduzir e otimizar as atividades desenvolvidas na prototipação deste estudo de caso. Esta estratégia contemplou os seguintes aspectos:

- reduzir o esforço de desenvolvimento para PDA;
- possibilitar o uso por diferentes dispositivos móveis;
- utilizar Java Server Faces (JSF) para disponibilizar ao usuário, via *browser*, o componente de software/adaptador inferido pelo EXEHDA-DA;
- enquanto tecnologias acessórias foi utilizado o Glassfish como container Web (http://glashfish.dev.java.net/, e JAXB - Java Architecture for XML Binding, como API para manipulação de XML (http://jaxb.dev.java.net/).

Procedendo desta forma, foi possível avaliar as funcionalidades do EXEHDA-DA, sem necessitar fazer um porte de outros serviços do *middleware* para operação em dispositivos móveis, em particular os serviços do Subsistema de Execução Distribuída.

6.1.2 Organização em Componentes da AUP

A aplicação Acompanhamento Ubíquo de Pacientes, contempla quatro principais funcionalidades:

- seleção de procedimentos e de paciente;
- exibição de histórico de dados sensoreados do paciente nas últimas 48 horas;
- envio de mensagens;
- acompanhamento dinâmico de pacientes.

A organização em componentes da AUP está caracterizada na figura 6.1. Nesta figura estão identificados os três componentes não adaptativos 000200, 000300 e 000400, e o componente 000500 que pode se adaptar do 000501 ao 000514, em função dos sinais vitais coletados, bem como em função do tipo dispositivo utilizado pelo agente de saúde.

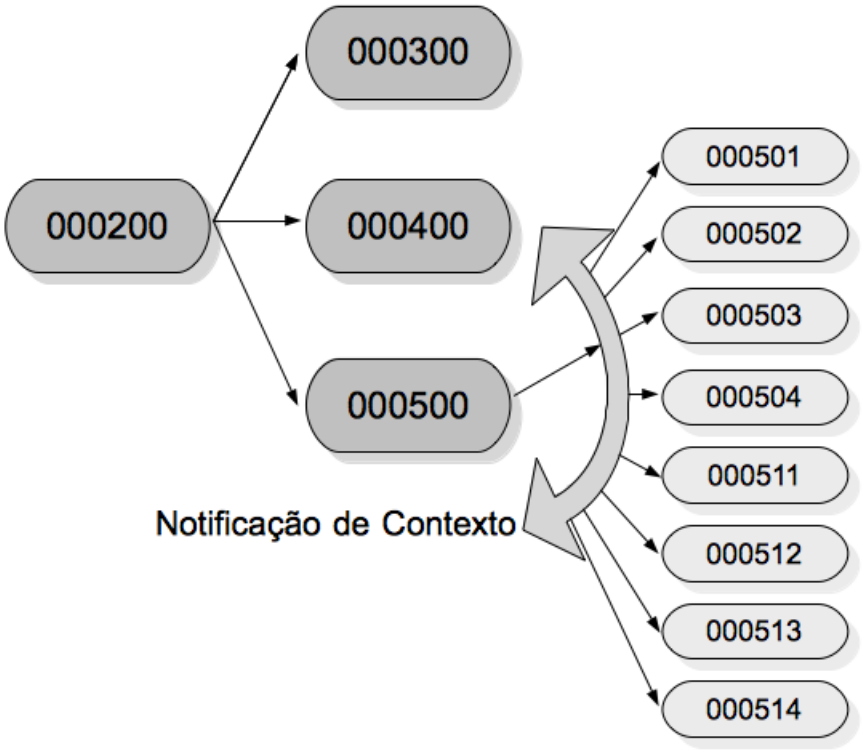


Figura 6.1: Visão da aplicação AUP organizada em Componentes

Por sua vez, a tabela 6.1 resume a instanciação da classe Componente da OntAdapt. Nesta classe são instanciados todos os componentes da aplicação, sendo caracterizadas por componente suas identificações, descrições, e possíveis componentes adaptadores.

O modelo semântico do EXEHDA-DA contempla que através do *Framework* FWADAPT (subseção 4.1.2) sejam instanciadas as classes Aplicacao e Componente da OntAdapt. Particularmente no caso da aplicação AUP, a classe Aplicacao recebe os seguintes valores:

Tabela 6.1: Componentes da aplicação Acompanhamento Ubíquo de Pacientes

Componente_Id	**Componente_Desc**	**Componente_Adaptado**
000200	Seleção de Funcionalidades e de Paciente	0
000300	Envio de Mensagens	0
000400	Histórico de Alertas de Pacientes	0
000500	Monitoramento de Pacientes	000501 000502 000503 000504 000511 000512 000513 000514
000501	Alerta Nível 1 para Interface Desktop	0
000511	Alerta Nível 1 para Interface PDA	0
000502	Alerta Nível 2 para Interface Desktop	0
000512	Alerta Nível 2 para Interface PDA	0
000503	Alerta Nível 3 para Interface Desktop	0
000513	Alerta Nível 3 para Interface PDA	0
000504	Alerta Nível 4 para Interface Desktop	0
000514	Alerta Nível 4 para Interface PDA	0

Classe Aplicacao
Aplicacao_Id = 100
Aplicacao_Desc = Acompanhamento Ubíquo de Pacientes

Considerando que na OntAdapt as classes Aplicacao e Componente são de um mesmo nível (figuras 4.1 e 4.3), o relacionamento entre ambas é feito de forma explicita. No caso da AUP os componentes 000200, 000300, 000400, 000500, 000501, 000502, 000503, 000504, 000511, 000512, 000513 e 000514, são instanciados para a aplicação 100.

Os dispositivos tratados no processo adaptativo são do tipo desktop e PDA. Quando o enfermeiro desloca-se no hospital, carrega consigo o PDA. Por outro lado, tanto o enfermeiro como o médico utilizam um desktop e/ou notebook, no consultório ou no plantão do hospital.
O PDA utilizado nos testes foi o Sharp Zaurus 5600 com resolução de 320x240 pixels, por sua vez, o desktop contemplou uma resolução mínima de 800x600 pixels.

As figuras 6.2 e 6.3 apresentam o menu de abertura para a aplicação AUP para Desktop e para PDA respectivamente.

Por sua vez, as figuras 6.4 e 6.5 exemplificam as interfaces para exibição do Histórico de Alertas do Paciente.

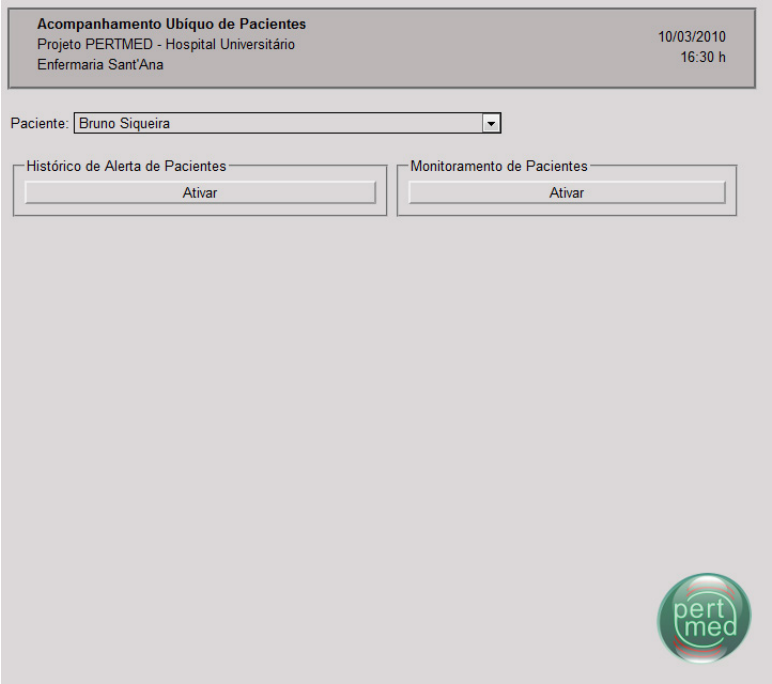
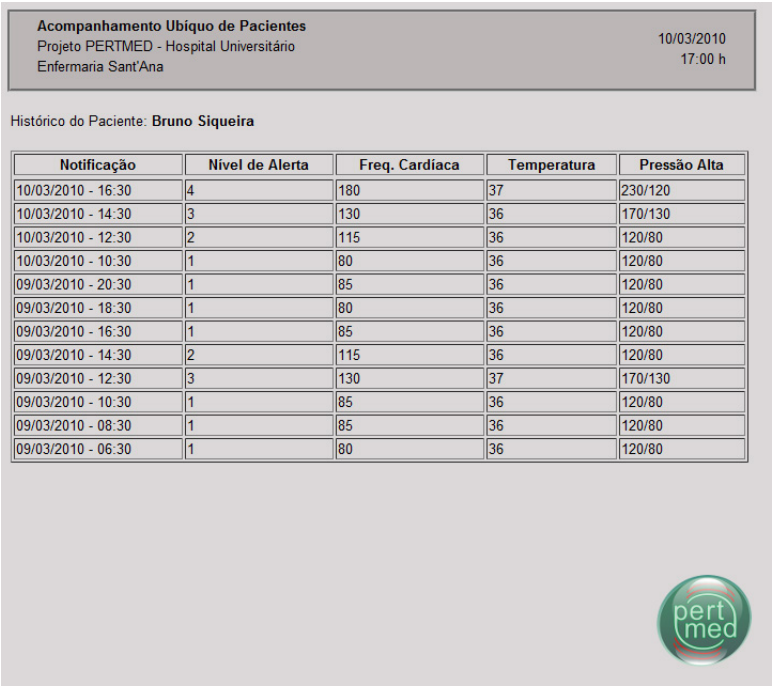


Figura 6.2: AUP - funcionalidades (Desktop)

Figura 6.3: AUP 1 (PDA)



Notificação	Nível de Alerta	Freq. Cardíaca	Temperatura	Pressão Alta
10/03/2010 - 16:30	4	180	37	230/120
10/03/2010 - 14:30	3	130	36	170/130
10/03/2010 - 12:30	2	115	36	120/80
10/03/2010 - 10:30	1	80	36	120/80
09/03/2010 - 20:30	1	85	36	120/80
09/03/2010 - 18:30	1	80	36	120/80
09/03/2010 - 16:30	1	85	36	120/80
09/03/2010 - 14:30	2	115	36	120/80
09/03/2010 - 12:30	3	130	37	170/130
09/03/2010 - 10:30	1	85	36	120/80
09/03/2010 - 08:30	1	85	36	120/80
09/03/2010 - 06:30	1	80	36	120/80

Figura 6.4: AUP Histórico de Alertas (Desktop)

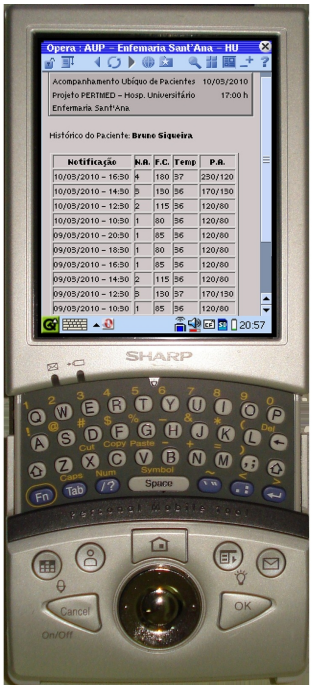


Figura 6.5: AUP 2 (PDA)

6.1.3 Adaptações na AUP

A AUP foi concebida para explorar adaptações funcionais de duas naturezas. Estas adaptações são especificadas através do FWADAPT na OntAdapt. No modelo do EXEHDA-DA, a eventual inclusão de novos procedimentos adaptativos na AUP, pode ser feito de forma incremental sem prejuízo das já existentes.

6.1.3.1 Adaptações previstas para a aplicação AUP

As adaptações previstas são: (vide ítem 3.2.2)

1. Adaptação funcional, em função dos dados de contexto sensoreados do paciente (sinais vitais), determinando o componente de software empregado na exibição do nível de alerta.

2. Adaptação funcional, mudança de interface em função de dispositivo que está sendo utilizado pelo usuário, considerando a perspectiva de mobilidade do agente de saúde.

O desenvolvedor da AUP, a partir das especificações fornecidas pelos agentes de saúde, utilizando o *framework* FWADAP (seção 4.1.2), constrói a Política de Adaptação da Aplicação. Neste sentido são instanciadas as classes e relacionamentos da OntAdapt com os respectivos parâmetros e regras de adaptação, conforme mostrado no ítem 6.1.3.3.

6.1.3.2 Instâncias de Classes e Relacionamentos para as adaptações previstas

As instâncias da classe Adaptador estão apresentadas nas tabelas 6.2 e 6.3. Estas instâncias são utilizadas no processamento da regra de adaptação, no ítem 6.1.4. A tabela 6.2 está relacionada a parte do código fonte do EXEHDA-DA mostrado na figura 6.17. Assim como a tabela 6.3 está relacionada ao código fonte mostrado na figura 6.19. Por sua vez, as classes Param_Tipo e Param_Valor da OntAdapt são descritas

Tabela 6.2: Classe Adaptador para determinação do nível de Alerta na AUP

Propriedade	**Valor**
Adaptador_Id	001
Adaptador_Desc	Alerta Automático
Adaptador_RegraLogica	ROUND ((CN_ContextoNotificadoSensor.CN_SensorTrad (CN_Sensor.Sensor_Id = 100) * ParamTipo_ParamValor.ParamValor_Utilidade (ParamTipo_ParamValor.ParamTipo_Id = 001) + CN_ContextoNotificadoSensor.CN_SensorTrad (CN_Sensor.Sensor_Id = 101) * ParamTipo_ParamValor.ParamValor_Utilidade (ParamTipo_ParamValor.ParamTipo_Id = 002) + CN_ContextoNotificadoSensor.CN_SensorTrad (CN_Sensor.Sensor_Id = 102) * ParamTipo_ParamValor.ParamValor_Utilidade (ParamTipo_ParamValor.ParamTipo_Id = 003) + 0,9) + Contexto_Notificado.CN_ComponenteId

na tabela 6.4. As instâncias destas classes são utilizadas no processamento da regra de adaptação do nível de alerta, ítem 6.1.4.

Tabela 6.3: Classe Adaptador para determinação da interface do tipo de dispositivo na AUP

Propriedade	**Valor**
Adaptador_Id	002
Adaptador_Desc	Tipo de Dispositivo
Adaptador_RegraLogica	(Adapt.Adapt_CompAdaptId (Adapt.Adapt_AplicacaoId = Contexto_Notificado.CN_AplicacaoId, Adapt.Adapt_CompOrigId = Contexto_Notificado.CN_ComponenteId, Adapt.Adapt_AdaptadorId = 001) + CN_ContextoNotificadoSensor.CN_SensorTrad (CN_Sensor.Sensor_Id = 103))

Tabela 6.4: Tipos e Valores de Parâmetros de Adaptação por Sensores para o Adaptador de Nível de Alerta

Propriedades	**Batimentos Cardíacos**	**Temperatura**	**Pressão Arterial**
Param_Tipo			
ParamTipo_Id	001	002	003
ParamTipo_Desc	Batimentos Cardíacos	Temperatura	Pressão Arterial
Param_Valor			
ParamValor_Id	001	002	003
ParamValor_Desc	Utilidade BC	Utilidade T	Utilidade PA
ParamValor_Utilidade	0.9	0.8	1.8
ParamTipo_Sensor			
Sensor_Id	100	101	102

Na figura 6.6 estão os Relacionamentos da OntAdapt considerados no processo de adaptação do EXEHDA-DA. estes relacionamentos são utilizados para o processamento das regras de adaptação da aplicação AUP, sendo empregados os valores dos atributos das classes que estão relacionadas.

6.1.3.3 *Framework* FWADAPT instâncias na OntAdapt para as adaptações previstas da AUP

A seguir as principais interfaces do FWADAPT para instanciação da Política de Adaptação para a aplicação AUP. As informações para alimentar a OntAdapt são fornecidas pelo desenvolvedor da aplicação.

A figura 6.7 mostra as aplicações instanciadas na Classe Aplicacao da OntAdapt, por sua vez, a figura 6.8 mostra a mesma interface com a aplicação Acompanhamento Ubíquo de Pacientes selecionada.

Figura 6.6: Relacionamentos da OntAdapt considerados no processo de adaptação do EXEHDA-DA:

Relacionamento Componente_Adaptador:
Componente_Id = 000500 e Adaptador_Id = 001
Componente_Id = 000500 e Adaptador_Id = 002
Relacionamento Adaptador_ParamTipo:
Adaptador_Id = 001 para ParamTipo_Id = 001, 002, 003
Relacionamento ParamTipo_Sensor:
ParamTipo_Id para Sensor_Id: 001 para 100, 002 para 101, 003 para 102
Relacionamento ParamTipo_Paramvalor:
ParamTipo_Id para ParamValor_Id: 001 para 001, 002 para 002, 003 para 003

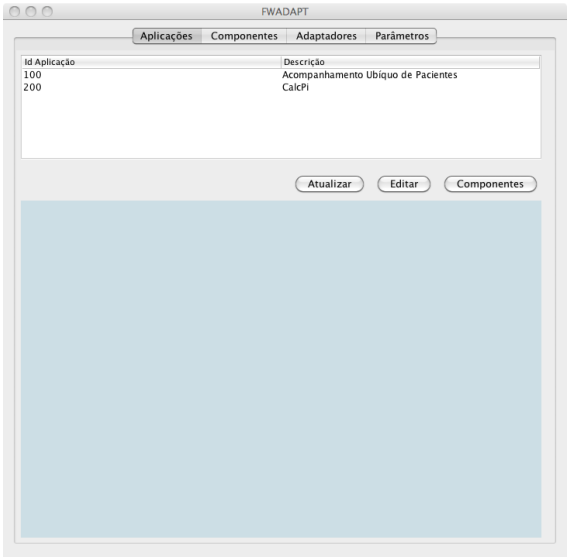


Figura 6.7: FWADAP - Aplicações

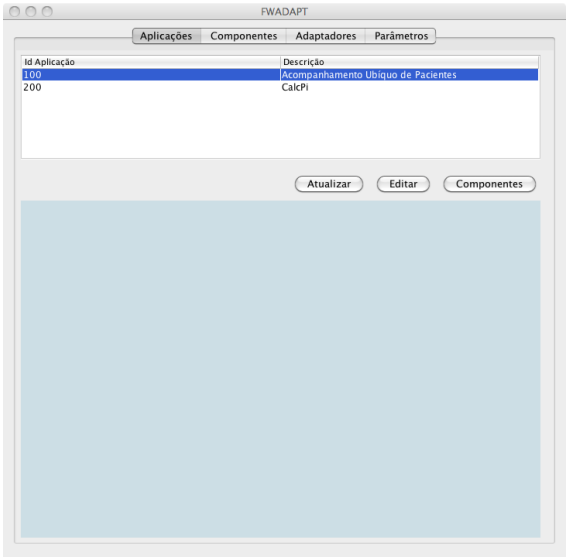


Figura 6.8: Seleção da Aplicação AUP

A figura 6.9 mostra a aplicação 100 (AUP) com a relação de seus componentes, instâncias do Relacionamento Aplicacao_Componente. Na figura 6.10 pode-se notar a seleção do componente 500 da aplicação 100, com o intuito de verificar os adaptadores desta seleção.

A figura 6.11 Mostra a lista de adaptadores para o componente Monitoramento de Pacientes (500) da aplicação 100 (AUP). A figura 6.12, além de mostrar as informações da figura 6.11, mostra o adaptador de Alerta Automático selecionado.

A figura 6.13 mostra os tipos de parâmetros relacionados ao adaptador Alerta Automático. Por sua vez, a figura 6.14 mostra os valores e sensores do Tipo de Parâmetro Batimentos Cardíacos.

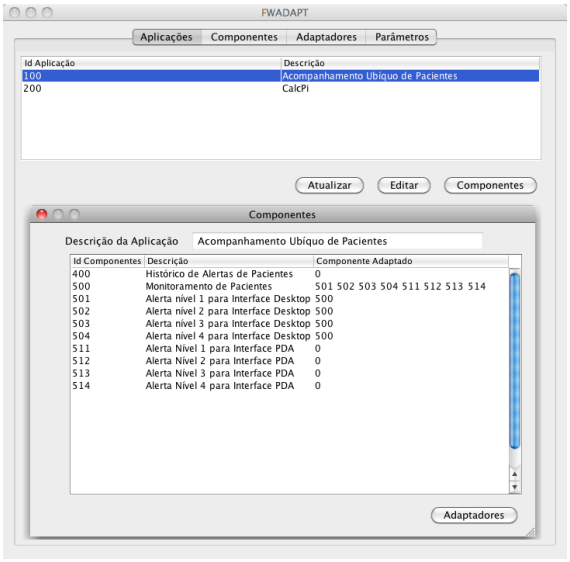


Figura 6.9: Componentes da Aplicação

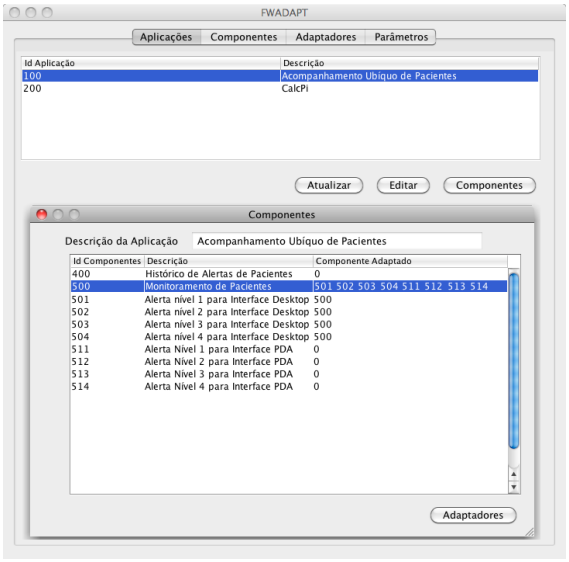


Figura 6.10: Componente selecionado

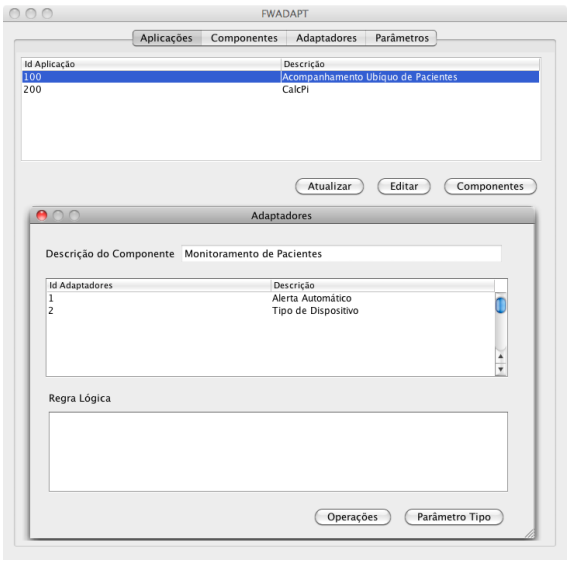


Figura 6.11: Adaptadores para o Componente 000500

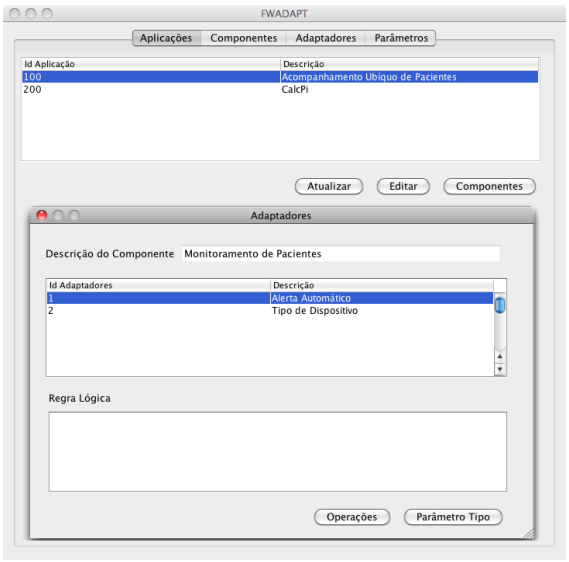


Figura 6.12: Seleção do Adaptador de Alerta Automático

6.1.3.4 Comandos Adaptativos

O comando adaptativo utilizado na aplicação AUP tem por objetivo central colocar a aplicação em estado de espera de uma notificação de adaptação. O EXEHDADA após ter concluído o processamento das regras de adaptação, pertinentes ao componente em questão, informará a aplicação, cuja execução está aguardando no comando `noContexto` que as decisões de adaptação a serem utilizadas pelos seus comandos associados, já foram disponibilizadas na classe Adapt da OntAdapt (vide tabela 6.9).

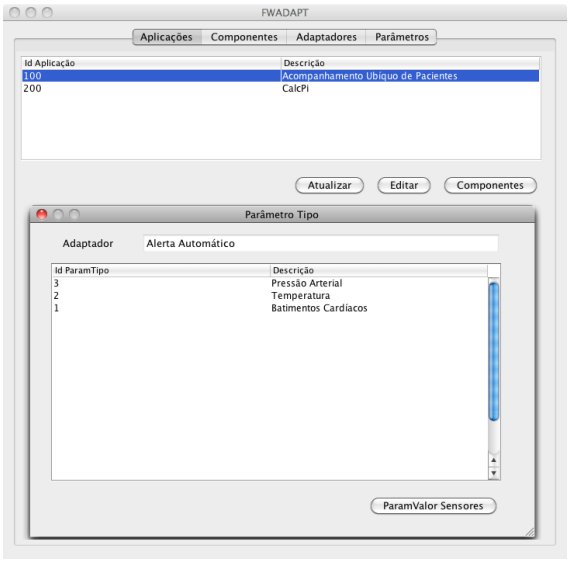


Figura 6.13: Tipos de Parâmetros para o Adaptador Alerta Automático

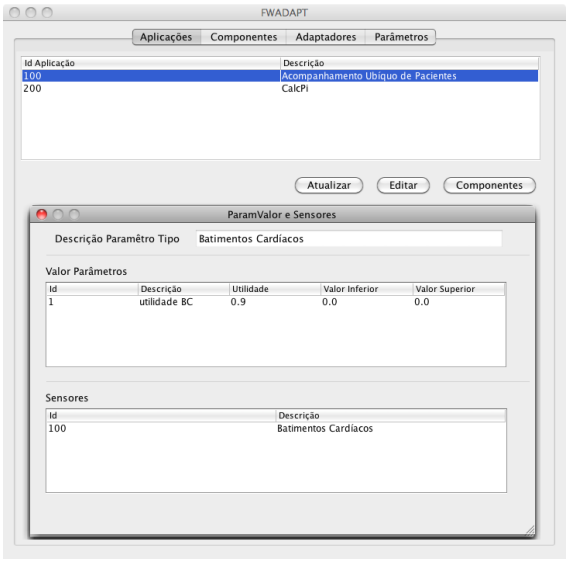


Figura 6.14: Valores e Sensores de Parâmetros do tipo Batimentos Cardíacos

A figura 6.15 mostra a composição do comando adaptativo que está presente no código do componente 000500 (Monitoramento de Pacientes) da aplicação 100 (`seleciona` 001). Esta composição determina que o resultado da decisão de adaptação do Nível de Alerta será utilizado na decisão de adaptação de tipo de dispositivo (`ativa` 002).

Figura 6.15: Composição do Comando Adaptativo no Componente 000500

```
sync
noContexto  100, 000500 {
     seleciona 001;
     ativa     002;
     }
```

6.1.4 Processamento de Regras de Adaptação para AUP

Esta subseção descreve como ocorre o processamento das duas regras de adaptação da aplicação AUP.

A associação entre os dados vitais sensoreados e os Níveis de Alertas Automáticos, produzidos pelo EXEHDA-DA, está resumida conforme a seguir:

- **Nível Alerta = 1**: sinais normais. Interface de nível 1 (verde) aplicação com

identificação e sinais do paciente: nome, especialidade, batimentos, pressão e temperatura. Sinais normais.

- **Nível alerta = 2**: início de problema. Batimentos Cardíacos ou Temperatura fora do normal. Interface de nível 2 (amarelo) com aviso de cuidado.
- **Nível alerta = 3**: problema médio. Apenas Pressão Arterial fora do normal, ou Batimentos Cardíacos e Temperatura fora do normal. Interface de nível 3 (laranja) com mensagem de cuidado maior que nível 2, mensagem gtalk para enfermeira.
- **Nível alerta = 4**: alerta máximo. Pressão Arterial fora do normal e Temperatura e/ou Batimentos cardíacos fora normal. Interface nível 4 (vermelho) com mensagem de emergência, mensagem gtalk para enfermeira e opção de envio de mensagem para médicos.

No que diz respeito a adaptação que irá produzir os diferentes níveis de alertas automáticos, o servidor de contexto produz um contexto notificado para o EXEHDA-DA, o qual representa a mudança de estado dos sensores de sinais vitais. As classes, propriedades e valores do contexto notificado estão descritos nas tabelas 6.5 e 6.6.
O valor traduzido dos sensores igual a zero indica que os sinais são normais. Se o valor traduzido for igual a 1, indica que os sinais do sensor estão fora da faixa de normalidade.

Tabela 6.5: Classe Contexto_Notificado

Propriedade	**Valor**
CN_Id	001
CN_AplicacaoId	100
CN_ComponenteId	000500
CN_AdaptadorId	001
Usuario_Id	300

Tabela 6.6: Relacionamento CN_ContextoNotificadoSensor

Propriedades	**Batimentos Cardíacos**	**Temperatura**	**Pressão Arterial**
CN_ContextoSensorId	001	002	003
Sensor_Id	100	101	102
CN_SensorValor	200	37	180,100
CN_SensorTrad	1	0	1
CN_NodoId	1	1	1

Por sua vez, a tabela 6.7 caracteriza o processamento desta associação, através da regra de adaptação, considerando as diferentes possibilidades de valores de entrada dos sensores de sinais vitais do paciente. A última coluna traduz o nível de alerta produzido pela regra.

A linha em destaque desta tabela, indica os valores traduzidos dos sensores (tabela 6.5 e os valores de utilidade dos sensores, tabela 6.4).

Tabela 6.7: Regra de definição do nível de alerta automático.

Trad. T	Util. T	Trad. BC	Util. BC	Trad. PA	Util. PA	Total	+ 0.9	Alerta
0	0	0	0	0	0	0	0.9	1
0	0	1	0.9	0	0	0.9	1.8	2
0	0	0	0	1	1.8	1.8	2.7	3
0	0	1	0.9	1	1.8	2.7	3.6	4
1	0.8	0	0	0	0	0.8	1.7	2
1	0.8	1	0.9	0	0	1.7	2.6	3
1	0.8	1	0.9	1	1.8	3.5	4.4	4
1	0.8	0	0	1	1.8	2.6	3.5	4

A figura 6.16 mostra um segmento do código do EXEHDA-DA que processa a regra do Adaptador_Id = 001, Nível Automático de Alerta. Nesta figura foi exibida a saída do processamento da regra para melhor entendimento do processo de adaptação. A regra configurada pelo desenvolvedor, através do FWADAPT, está apresentada na tabela 6.2.

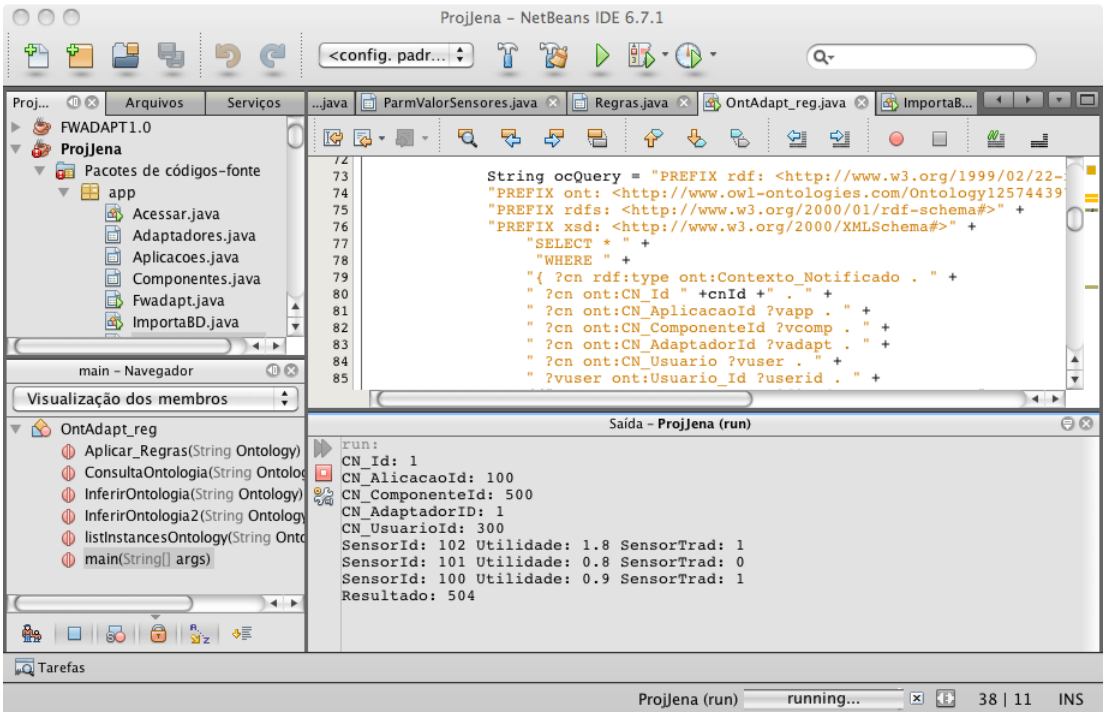


Figura 6.16: Processamento da regra do Adaptador 001 - Nível Automático Alerta

A regra lógica que está instanciada na classe Adaptador, atributo Adaptador_RegraLogica, empregada no processamento da adaptação, na versão atual do FWADAPT, é traduzida de forma manual para trechos SPARQL, de modo que possa ser computada pelo Jena. Neste sentido, a Classe Operacao oferece suporte para composição multi-linguagem da regra de adaptação. Na aplicação AUP a Classe Operacao não foi empregada.

Na figura 6.17 temos parte do código, que corresponde a tradução para Jena/SPARQL, da regra lógica da classe Adaptador, apresentada na tabela 6.2. O resultado produzido por este código eatá na figura 6.16.

A adaptação referente ao tipo de dispositivo e sua respectiva interface, considera o contexto notificado descrito na tabela 6.8.

Tabela 6.8: Contexto Notificado para Adaptador Tipo de Dispositivo

Propriedade	**Valor**
Contexto_Notificado	
CN_Id	002
CN_AplicacaoId	100
CN_ComponenteId	000500
CN_AdaptadorId	002
Usuario_Id	300
CN_ContextoNotificadoSensor	
CN_ContextoSensorId	004
Sensor_Id	103
CN_SensorValor	0
CN_SensorTrad	10
CN_NodoId	1

Na notificação de Contexto do tipo de dispositivo, CN_Id = 002, o valor traduzido dos sensor igual a zero indica que o dispositivo é da categoria Desktop. Se o valor traduzido for igual a 10, indica que o dispositivo é do tipo PDA.

Figura 6.17: Regra para o adaptador 001

```
...
String ocQuery = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
"PREFIX ont: <http://www.owl-ontologies.com/Ontology1257443971.owl#>" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
"PREFIX xsd: <http://www.w3.org/2000/XMLSchema#>" +
"SELECT * " +
"WHERE " +
"{ ?cn rdf:type ont:Contexto_Notificado . " +
" ?cn ont:CN_Id " +cnId +" . " +
" ?cn ont:CN_AplicacaoId ?vapp . " +
" ?cn ont:CN_ComponenteId ?vcomp . " +
" ?cn ont:CN_AdaptadorId ?vadapt . " +
" ?cn ont:CN_Usuario ?vuser . " +
" ?vuser ont:Usuario_Id ?userid . " +
"} ";
QueryExecution ocQe = QueryExecutionFactory.create(ocQuery,infmodel);
...
String ocQuery2 = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
"PREFIX ont: <http://www.owl-ontologies.com/Ontology1257443971.owl#>" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
"PREFIX xsd: <http://www.w3.org/2000/XMLSchema#>" +
"SELECT ?sensorid ?utilidade ?sensorTrad " +
"WHERE " +
"{ ?app rdf:type ont:Aplicacao .  " +
" ?app ont:Aplicacao_Id " + cnAppId + " . " +
" ?app ont:Aplicacao_Componente ?apcomp ." +
" ?apcomp  ont:Componente_Id " + cnCompId + " . " +
" ?apcomp ont:Componente_Adaptador ?adapcomp . " +
" ?adapcomp ont:Adaptador_Id " + cnAdaptId + " . " +
" ?adapcomp ont:Adaptador_ParamTipo ?adpamtp . " +
" ?adpamtp ont:ParamTipo_Id ?partpid . " +
" ?adpamtp ont:ParamTipo_ParamValor ?ptppvl . " +
" ?ptppvl ont:ParamValor_Utilidade ?utilidade . " +
" ?adpamtp ont:ParamTipo_Sensor ?sensor . " +
" ?sensor ont:Sensor_Id ?sensorid . " +
" ?cn ont:CN_Sensor ?sensor . " +
" ?cn ont:CN_SensorTrad ?sensorTrad . " +
//" ?cn ont:CN_NodoId ?nodoid . " +
"}";
QueryExecution ocQe2 = QueryExecutionFactory.create(ocQuery2,infmodel);
//ResultSet ocResults2 = ocQe2.execSelect();
//ResultSetFormatter.out(ocResults2);
String appDados[][] = new String[20][20];
int pos = 0;
for(ResultSet rs = ocQe2.execSelect() ; rs.hasNext() ; rs.getRowNumber())
{
QuerySolution binding = rs.nextSolution();
appDados[pos][0] = fwadapt.clearStringQuery(binding.get("sensorid").toString());
appDados[pos][1] = fwadapt.clearStringQuery(binding.get("utilidade").toString());
appDados[pos][2] = fwadapt.clearStringQuery(binding.get("sensorTrad").toString());
// System.out.println("SensorId: "+appDados[pos][0]+" Utilidade: "+appDados[pos][1]+"
//SensorTrad: "+appDados[pos][2]);
pos++;
}
ocQe2.close();
double regra = Math.round((Integer.parseInt(appDados[0][2]) *
Double.parseDouble(appDados[0][1]))
+ (Integer.parseInt(appDados[1][2]) * Double.parseDouble(appDados[1][1]))
+ (Integer.parseInt(appDados[2][2]) * Double.parseDouble(appDados[2][1]))
+ 0.9 ) + cnCompId;
// System.out.println("Resultado: "+(int)regra);
return (int)regra;
}
...
```

A regra que especifica o processamento da adaptação em função do tipo de dispositivo está apresentada na tabela 6.3. Esta regra contempla a seguinte especificação:

Componente adaptado ao dispositivo = componente nível de alerta + categoria do dispositivo

A figura 6.18 mostra um segmento do código do EXEHDA-DA que processa a regra do Adaptador_Id = 002, Tipo de dispositivo. Nesta figura foi exibida a saída do processamento da regra para melhor entendimento do processo de adaptação.

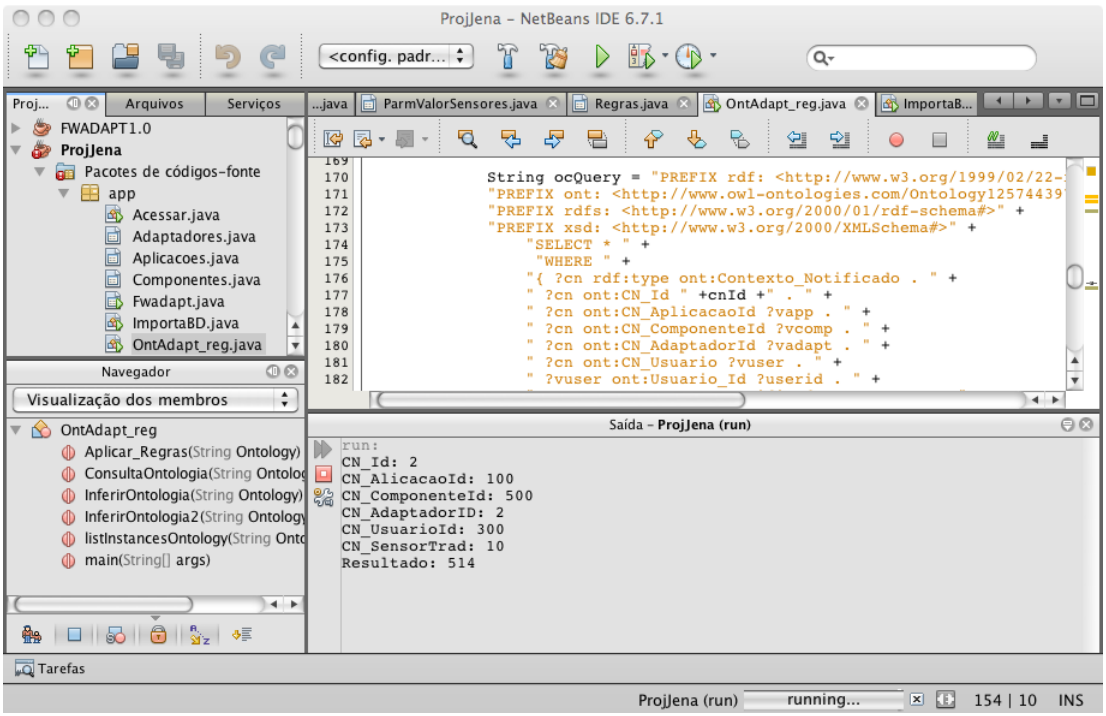


Figura 6.18: Processamento da regra do Adaptador 002 - Tipo de dispositivo.

Na figura 6.19 é mostrado parte do fonte do programa em Jena e SPARQL para produção do resultado da figura 6.18.

Figura 6.19: Regra para o adaptador 002

```
...
public static void InferirOntologia2(String Ontology, Integer result) {
  String Ontology1 = Ontology;
Integer cnId = 2;
Integer cnAppId = null, cnCompId = null, cnAdaptId = null, cnUserId = null, cnSensorTrad = null;
  Fwadapt fwadapt = new Fwadapt();
OntModel dados = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
dados.read(Ontology1);
Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
InfModel infmodel = ModelFactory.createInfModel(reasoner, dados);
String ocQuery = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
"PREFIX ont: <http://www.owl-ontologies.com/Ontology1257443971.owl#>" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
"PREFIX xsd: <http://www.w3.org/2000/XMLSchema#>" +
"SELECT * " +
"WHERE " +
"{ ?cn rdf:type ont:Contexto_Notificado . " +
" ?cn ont:CN_Id " +cnId +" . " +
" ?cn ont:CN_AplicacaoId ?vapp . " +
" ?cn ont:CN_ComponenteId ?vcomp . " +
" ?cn ont:CN_AdaptadorId ?vadapt . " +
" ?cn ont:CN_Usuario ?vuser . " +
" ?vuser ont:Usuario_Id ?userid . " +
" ?cn ont:CN_ContextoNotificadoSensor ?cns . " +
"?cns ont:CN_SensorTrad ?sensortrad . " +
"} ";
QueryExecution ocQe3 = QueryExecutionFactory.create(ocQuery,infmodel);
//ResultSet ocResults = ocQe.execSelect();
//ResultSetFormatter.out(ocResults);
for(ResultSet rs = ocQe3.execSelect() ; rs.hasNext() ; rs.getRowNumber())
{
QuerySolution binding = rs.nextSolution();
cnAppId = Integer.parseInt(fwadapt.clearStringQuery(binding.get("vapp").toString()));
cnCompId = Integer.parseInt(fwadapt.clearStringQuery(binding.get("vcomp").toString()));
cnAdaptId = Integer.parseInt(fwadapt.clearStringQuery(binding.get("vadapt").toString()));
cnUserId = Integer.parseInt(fwadapt.clearStringQuery(binding.get("userid").toString()));
cnSensorTrad = Integer.parseInt(fwadapt.clearStringQuery(binding.get("sensortrad").toString()));
}
ocQe3.close();
System.out.println("CN_Id: "+cnId);
System.out.println("CN_AlicacaoId: "+cnAppId);
System.out.println("CN_ComponenteId: "+cnCompId);
System.out.println("CN_AdaptadorID: "+cnAdaptId);
System.out.println("CN_UsuarioId: "+cnUserId);
System.out.println("CN_SensorTrad: "+cnSensorTrad);
Integer regra2 = result+cnSensorTrad;
System.out.println("Resultado: "+regra2);
}
...
```

A seguir mostramos, através das figuras 6.20, 6.21, 6.22, 6.23, 6.24, 6.25, 6.26 e 6.27 as interfaces para os diferentes níveis de alerta, tanto para Desktop, como para PDA.

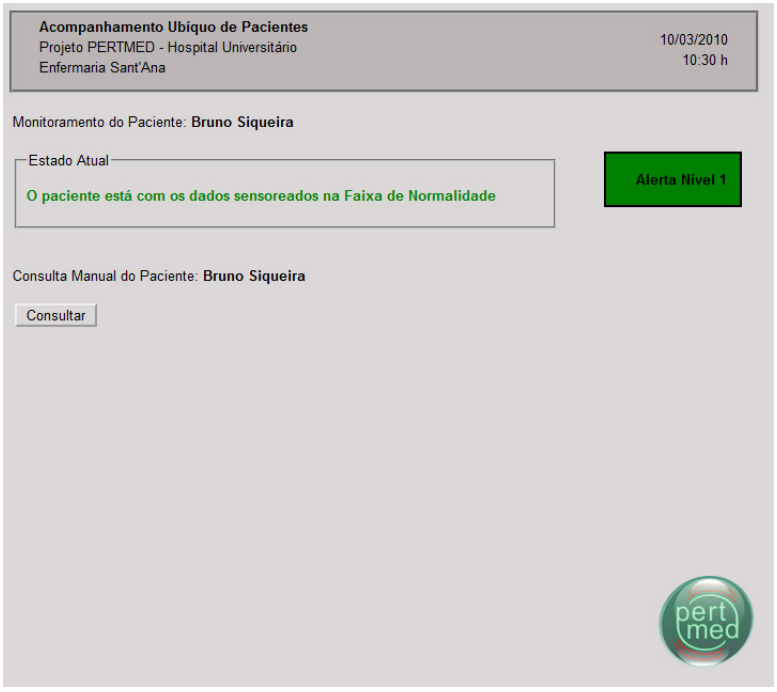


Figura 6.20: AUP - Alerta Nível 1 (Desktop)

Figura 6.21: AN 1 (PDA)

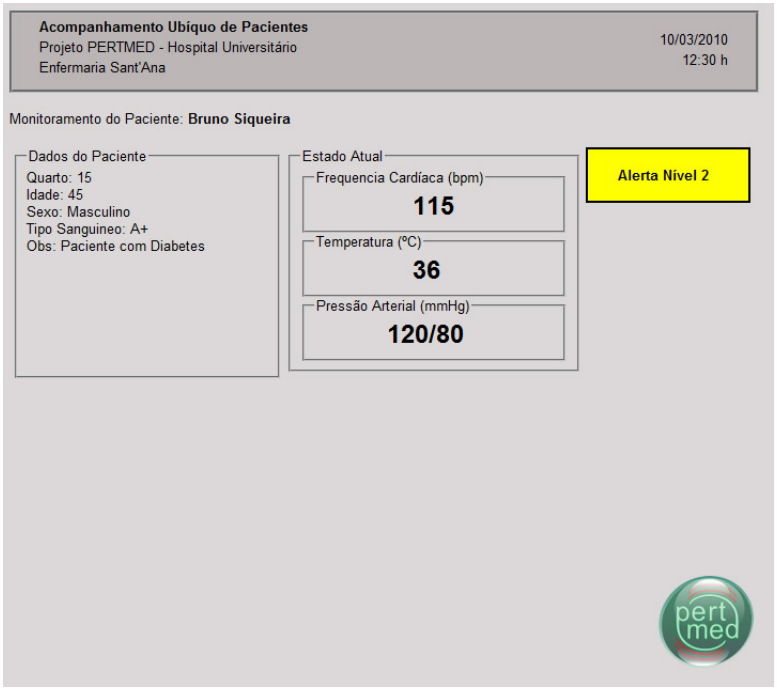


Figura 6.22: AUP - Alerta Nível 2 (Desktop)

Figura 6.23: AN 2 (PDA)

É importante ressaltar que as adaptações da aplicação AUP são encadeadas, isto é, a tomada de decisão do adaptador 002 (tipo de dispositivo) utiliza a decisão previ-

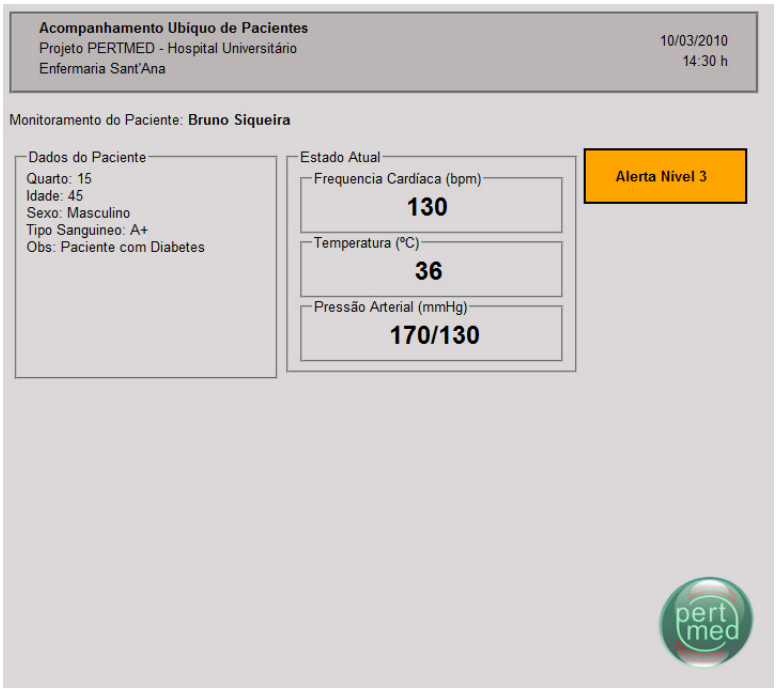


Figura 6.24: AUP - Alerta Nível 3 (Desktop)

Figura 6.25: AN 3 (PDA)

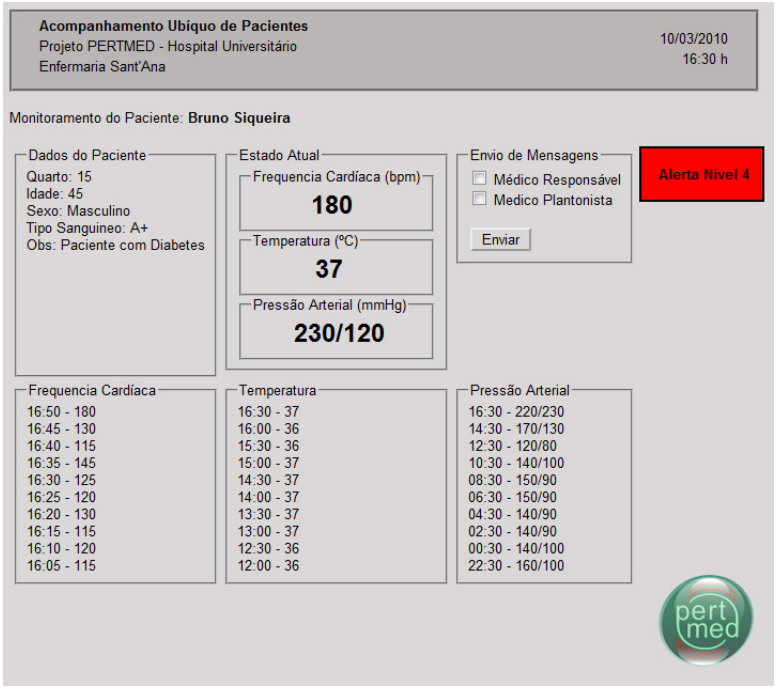


Figura 6.26: AUP - Alerta Nível 4 (Desktop)

Figura 6.27: AN 4 (PDA)

amente tomada com relação ao adaptador 001 (nível de alerta). A computação deste encadeamento aparece na tabela 6.3 na propriedade Adaptador_RegraUsuario. Os valores utilizados na regra do adaptador 002 estão relacionados nas tabelas 6.8 e 6.9.

As decisões de adaptação pelo EXEHDA-DA são mantidas em uma classe específica para esta finalidade, denominada Adapt. A tabela 6.9 apresenta a instanciação desta classe para a aplicação AUP, considerando uma adaptação para o alerta de nível 4 e

dispositivo do tipo PDA.

De modo mais específico é possível dizer que o EXEHDA-DA inferiu e sinalizou a seguinte adaptação para para o componente 000500 (Alerta adaptado por nível e dispositivo) da aplicação 100 (Acompanhamento Ubíquo de Pacientes): ativação do componente 000514 (Alerta Nível 4 para Interface PDA, figura 6.28) da aplicação 100 no nodo 001.

Tabela 6.9: Instâncias criadas pelo EXEHDA-DA na classe Adapt

Propriedade	**Adaptação 1**	**Adaptação 2**
Adapt_AplicacaoId	100	100
Adapt_CompOrigId	000500	000500
Adapt_AdaptadorId	001	002
Adapt_CompAdaptId	000504	000514
Adapt_NodoId	001	001
Adapt_Estado	0	1

Figura 6.28: AUP - Alerta Automático - Nível 4 (PDA Zaurus)

6.2 Estudo de Caso 2: Alocação Dinâmica de Recursos (ADR)

Nesta seção é discutido estudo de caso do EXEHDA-DA, no qual é caracterizado o uso do Controle da Adaptação Dinâmica, enquanto suporte para o escalonamento de recursos em grades computacionais, nas quais os recursos são heterogêneos e sujeitos à taxas de ocupação dinâmicas.

A aplicação, cujas computações serão escalonadas, é do tipo sintético, e tem por objetivo central permitir a exploração das funcionalidades do EXEHDA-DA quando do controle das adaptações não-funcionais (vide ítem 3.2.2).

6.2.1 Objetivos da ADR

O objetivo central da ADR é explorar de modo oportunista os recursos computacionais em uma infraestrutura distribuída. Nesta perspectiva as tarefas da aplicação serão escalonadas a medida que os recursos se tornarem disponíveis.

O ambiente de testes do EXEHDA-DA utilizado na ADR contempla o emprego de quatro tipos de software:

- uma aplicação paralela do tipo *bag-of-tasks*, denominada CalcPi. Esta aplicação tem como finalidade matemática calcular o número π (pi) utilizando o método de Monte Carlo (PRESS et al., 1992). O volume de processamento de cada tarefa a ser disparada é decorrente da precisão desejada para o valor do π;

- uma aplicação denominada CpuSteal, cujo objetivo é atuar como um gerador de cargas, baseado em uma distribuição de propriedade exponencial. Este software opera em ciclos de ativação e desativação. Na ativação são geradas operações de ponto flutuante ocupando o processador. Quando inativo, ele entra em repouso. O controle do nível de ocupação média do processador é feito através da determinação de quanto tempo ele permanece ativo e quanto tempo fica em repouso em cada um dos ciclos (REAL, 2004);

- uma aplicação denominada LoadGen, cuja finalidade é produzir um descritor de carga computacional. O LoadGen gera os tempos de ativação e desativação de processador, baseado em uma distribuição de probabilidades exponencial, para ser utilizada pelo CpuSteal (REAL, 2004);

- uma aplicação denominada MemSteal, cuja finalidade é promover a ocupação de memória. Esta aplicação se vale da instanciação de matrizes para gerar demanda de memória nos nodos processadores.

Os principais objetivos buscados quando da concepção da aplicação ADR podem ser resumidos como:

- ser possível selecionar o recurso computacional mais adequado, segundo um critério de adaptação, entre todos os disponíveis no momento;

- permitir que a distribuição das computações possa ser iniciada, independentemente de ter disponibilidade de processadores para acomodar todas as tarefas paralelas previstas;

- prover tomada de decisão de escalonamento em dois níveis: (i) um contemplando o estado na infraestrutura computacional e (ii) preferências do usuário quanto à origem do recurso;

- modelar a aplicação a ser escalonada de forma a oferecer granulosidade computacional adequada para processamento em grades computacionais;

A grade computacional que constitui a infraestrutura de teste é formada por duas arquiteturas paralelas do tipo *Cluster*. Os nodos destes *clusters* são heterogêneos e estão submetidos a regimes diferenciados de ocupação de memória e ocupação de processador.

O critério geral é disponibilizar para a aplicação CalcPi os nodos com maior poder computacional e menor nível de ocupação de processador e memória. Inicialmente serão alocados os nodos do *cluster* de preferência do usuário. Uma vez esgotados os mesmos serão escalonados os nodos melhor qualificados de outro *cluster*. Deste modo, o nodo que for inferido como de maior disponibilidade, a partir destes dois critérios, será, então, utilizado para instalação de um componente de software responsável por parte das computações paralelas.

No EXEHDA-DA, este tipo de adaptação denomina-se adaptação não-funcional (ítem 3.2.2). Neste tipo de adaptação, o componente a ser instalado é o mesmo, independentemente do equipamento (nodo) onde o processamento será disparado, consistindo a escolha do nodo o foco do procedimento adaptativo.

6.2.2 Organização em Componentes da Aplicação a ser Escalonada - CalcPi

A tabela 6.10 resume a instanciação da classe Componentes da OntAdapt para a aplicação CalcPi, cujo Aplicacao_Id é 200.

Tabela 6.10: Componentes da aplicação CalcPi

Componente_Id	**Componente_Desc**	**Componente_Adaptado**
000700	Distribui tarefas nos nodos	0
000800	Algoritmo de cálculo do π - Monte Carlo	0

Classe Aplicacao
Aplicacao_Id = 200
Aplicacao_Desc = Cálculo do Pi - CalcPi

Relacionamento Aplicacao_Componente: para a aplicação 200 são instanciados os componentes 000700 e 000800.

6.2.3 Adaptações na ADR

Na avaliação do EXEHDA-DA na tomada de decisão de escalonamento de recursos foram utilizados os *clusters*:

- H3P (http://h3p.g3pd.ucpel.tche.br), composto por 6 nodos;
- LANGTON (http://langton.ucpel.tche.br), composto por 10 nodos.

Os nodos dos *clusters* possuem sistema operacional Linux e são interligados por uma rede *FastEthernet*. Na tabela 6.11 estão descritas as configurações parciais de *hardware* com a distribuição das cargas computacionais no *cluster* H3P. Na tabela 6.12 são descritas as mesmas informações referentes ao *cluster* LANGTON. Os nodos H3P (10001) e LANGTON (20001) não serão utilizados para processamento, pois são os gerenciadores do sistema.

Tabela 6.11: Configuração do *cluster* H3P

Nodo	Frequência (MHz)	Memória (MB)	Ocupação CPU (%)	Ocupação Mem. (%)
10001	1680	185	xxxxx	xxxxx
10002	2020	185	55,22	22,45
10003	2020	185	80,12	30,15
10004	2019	185	40,02	10,10
10005	2019	185	89,30	20,06
10006	1680	185	20,05	30,08

Tabela 6.12: Configuração do *cluster* LANGTON

Nodo	Frequência (MHz)	Memória (MB)	Ocupação CPU (%)	Ocupação Mem. (%)
20001	1680	251	xxxxx	xxxxx
20002	1600	251	80,09	30,08
20003	1600	251	85,23	35,01
20004	1600	251	0,00	0,00
20005	1600	251	20,02	10,21
20006	1600	251	64,03	35,43
20007	1600	251	51,30	29,90
20008	1600	251	30,26	30,13
20009	1600	251	59,95	28,98
20010	1600	251	61,45	54,10

O tipo de adaptação a ser tratado pelo EXEHDA-DA para a aplicação Escalonamento de Tarefas é uma adaptação não-funcional (sem mudança de código) com o objetivo de determinar o nodo mais disponível para a execução do componente adaptativo 000800. A primeira etapa de decisão fornecerá a relação dos melhores nodos, em termos

de ocupação de CPU, ocupação de memória e frequência da CPU. A segunda etapa de decisão, definirá a escolha do melhor nodo da relação de acordo com as preferências do usuário. Neste estudo de caso, a preferência do usuário define o *cluster* de interesse do usuário, no caso o *cluster* H3P. Segundo o critério de utilidade do usuário, caso não tenha nodo disponível neste *cluster*, deverão ser escolhidos os melhores nodos disponíveis do *cluster* Langton.

A instância da classe Adaptador é descrita na tabela 6.13.

Tabela 6.13: Classe Adaptador para Escalonamento de nodos disponíveis

Propriedade	**Valor**
Adaptador_Id	050
Adaptador_Desc	Escalonador de nodos disponíveis
Adaptador_RegraLogica	**busca** ((Contexto_Notificado.CN_Id = notificacao) **e** (**preferencia** = (CN_ContextoNotificadoSensor.CN_NodoId ≥ CN_Usuario.Usuario_ClusterInf **e** ≤ CN_Usuario.Usuario_ClusterSup) **e menor** ((CN_ContextoNotificadoSensor.CN_SensorValor (CN_Sensor.Sensor_Id = 104) * ParamTipo_ParamValor.ParamValor_Utilidade (ParamTipo_ParamValor.ParamTipo_Id = 051)) + (CN_ContextoNotificadoSensor.CN_SensorValor (CN_Sensor.Sensor_Id = 105) * ParamTipo_ParamValor.ParamValor_Utilidade (ParamTipo_ParamValor.ParamTipo_Id = 052)) + (CN_ContextoNotificadoSensor.CN_SensorValor (CN_Sensor.Sensor_Id = 106) * ParamTipo_ParamValor.ParamValor_Utilidade (ParamTipo_ParamValor.ParamTipo_Id = 053))

As classes Param_Tipo e Param_Valor da OntAdapt para a aplicação CalcPi são descritas na tabela 6.14.

Tabela 6.14: Tipos e Valores de Parâmetros de Adaptação por Sensores para o escalonamento de tarefas

Propriedades	**Frequência de CPU**	**Carga de CPU**	**Memória Ocupada**
Param_Tipo			
ParamTipo_Id	051	052	053
ParamTipo_Desc	Frequência CPU	Carga CPU	Ocupação Memória
Param_Valor			
ParamValor_Id	151	152	153
ParamValor_Desc	Utilidade Freq CPU	Utilidade carga CPU	Utilidade Mem Ocupada
ParamValor_Utilidade	0.2	0.8	0.5
ParamTipo_Sensor			
Sensor_Id	104	105	106

Relacionamento Componente_Adaptador:
Componente_Id = 000800 e Adaptador_Id = 050
Relacionamento Adaptador_ParamTipo:
Adaptador_Id = 050 para ParamTipo_Id = 051, 052, 053
Relacionamento ParamTipo_Sensor:
ParamTipo_Id para Sensor_Id: 051 para 104, 052 para 105, 053 para 106
Relacionamento ParamTipo_Paramvalor:
ParamTipo_Id para ParamValor_Id: 051 para 151, 052 para 152, 053 para 153

6.2.3.1 Comandos Adaptativos

A aplicação prevista instala computações remotas em função da disponibilidade de recursos computacionais na célula de execução. A medida que surjam recursos que atendam os requisitos estabelecidos pelo desenvolvedor na Política de Adaptação (OntAdapt), no âmbito do contexto celular, o componente da aplicação responsável pela gerência da execução é notificado pelo EXEHDA-DA para que execute os procedimentos pertinentes (comando adaptativo `ativa`) que estão após o comando adaptativo `seleciona` no comando adaptativo `noContexto`, conforme a representação da figura 6.29.

Figura 6.29: Comando adaptativo do componente 000700

```
int contador = 0;
while (contador < NPP) { // NPP=Número Processos Paralelos
    sync
    noContexto  200, 000700 {
       ativa     000800, 050;
    }
    contador++;
}
```

A figura 6.30 mostra a sincronização do EXEHDA-DA e o escalonamento do componente 000800 da aplicação CalcPi nos nodos disponíveis, selecionados pelo serviço de controle da adaptação. A variável NPP contem o número de processos a serem paralelizados.

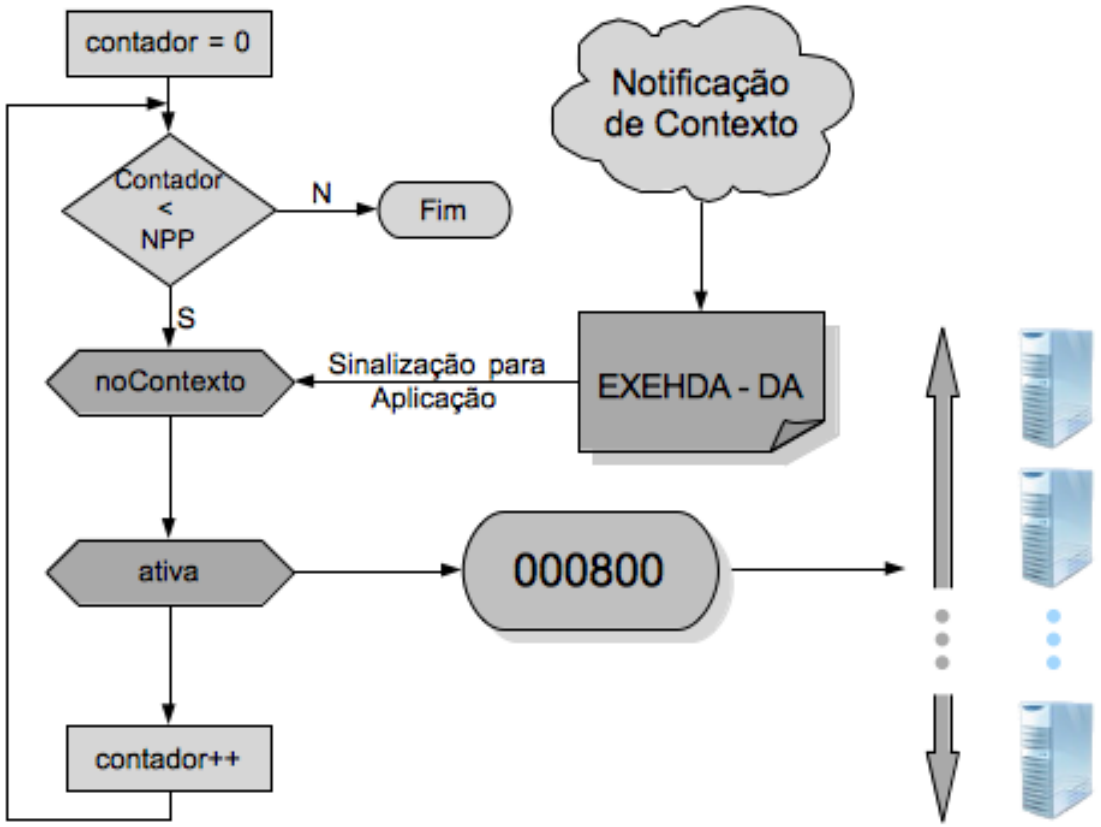


Figura 6.30: Escalonamento do Algoritmo do CalcPi

6.2.4 Processamento de Regra de Adaptação para a ADR

Esta subseção caracteriza como acontece a computação da regra de adaptação do estudo de caso ADR. A mesma considera 5 pressupostos:

- a meta é ativar em 5 nodos (NPP = 5) o algoritmo de cálculo π (CalcPi);
- os 5 nodos serão selecionados de um total de 14 distribuídos entre 2 clusters, tabelas 6.11 e 6.12;
- serão considerados válidos para o processamento, os nodos com ocupação de CPU inferior a 50%;
- os nodos considerados válidos, serão priorizados de acordo com os valores especificados na tabela 6.11;
- como resultado da aplicação desta regra de adaptação, foram selecionados 5 nodos, cuja descrição aparece na tabela 6.17.

A figura 6.31 mostra a regra de adaptação do adaptador de escalonamento do Cálculo do π utilizando a sintaxe da extensão SPARQL Preference.

Figura 6.31: Regra adaptador 050 em SPARQL Preference

```
"SELECT * " +
"WHERE " +
"{ ?app rdf:type ont:Aplicacao .  " +
" ?app ont:Aplicacao_Id " + cnAppId + " . " +
" ?app ont:Aplicacao_Componente ?apcomp ." +
" ?apcomp  ont:Componente_Id " + cnCompId + " . " +
" ?apcomp ont:Componente_Adaptador ?adapcomp . " +
" ?adapcomp ont:Adaptador_Id " + cnAdaptId + " . " +
" ?adapcomp ont:Adaptador_ParamTipo ?adpamtp . " +
" ?adpamtp ont:ParamTipo_Id ?partpid . " +
" ?adpamtp ont:ParamTipo_ParamValor ?ptppvl . " +
" ?ptppvl ont:ParamValor_Utilidade ?utilidade . " +
" ?adpamtp ont:ParamTipo_Sensor ?sensor . " +
" ?sensor ont:Sensor_Id ?sensorid . " +
" ?cn ont:CN_Sensor ?sensor . " +
" ?cn ont:CN_SensorTrad ?sensorTrad . " +
" ?cn ont:CN_NodoId ?nodoid . " + "}";

 PREFERRING
    ?nodoid < 20000
         CASCADE ?nodoid > 20000
 AND
    LOWEST (?sensorTrad * ?utilidade)
```

A adaptação referente ao escalonamento do algorítmo do cálculo do π, considera o contexto notificado descrito na tabela 6.15 e 6.16.

Tabela 6.15: Contexto Notificado para Adaptador de Escalonamento da CalcPi

Propriedade	**Valor**
Contexto_Notificado	
CN_Id	003
CN_AplicacaoId	200
CN_ComponenteId	000800
CN_AdaptadorId	050
CN_Usuario	
Usuario_Id	300
Usuario_ClusterInf	10002
Usuario_ClusterSup	10006

A tabela 6.17 demonstra os nodos para escalonamento em função de seus valores sensoreados para esta adaptação, informados na notificação de contexto, e os valores de

Tabela 6.16: Contexto Notificado dos sensores e nodos para Adaptador de Escalonamento da CalcPi

CN_ContextoNotificadoSensor

Propriedade	**Frequência**	**Carga CPU**	**Carga Mem.**
Nodo 1 Disponível			
CN_ContextoSensorId	005	006	007
Sensor_Id	104	105	106
CN_SensorValor	2019	40,02	10,10
CN_SensorTrad	0	0	0
CN_NodoId	10004	10004	10004
Nodo 2 Disponível			
CN_ContextoSensorId	008	009	010
Sensor_Id	104	105	106
CN_SensorValor	1680	20,05	30,08
CN_SensorTrad	0	0	0
CN_NodoId	10006	10006	10006
Nodo 3 Disponível			
CN_ContextoSensorId	011	012	013
Sensor_Id	104	105	106
CN_SensorValor	1600	0,00	0,00
CN_SensorTrad	0	0	0
CN_NodoId	20004	20004	20004
Nodo 4 Disponível			
CN_ContextoSensorId	014	015	016
Sensor_Id	104	105	106
CN_SensorValor	1600	20,02	10,21
CN_SensorTrad	0	0	0
CN_NodoId	20005	20005	20005
Nodo 5 Disponível			
CN_ContextoSensorId	017	018	019
Sensor_Id	104	105	106
CN_SensorValor	1600	30,26	30,13
CN_SensorTrad	0	0	0
CN_NodoId	20008	20008	20008

utilidade para cada tipo de sensor utilizado no cálculo da regra. A última coluna apresenta o tempo de processamento da CalcPi dos nodos. Considerando que a aplicação tem uma organização *bag-of-tasks*, o tempo total de processamento corresponde ao maior tempo de execução individual, no caso 3727.9 s. O valor aproximado obtido para o π foi 3,141592.

Tabela 6.17: Nodos selecionados para execução da CalcPi.

Nodo	Freq	Ut Freq	CPU	Ut CPU	Mem	Ut Mem	Total	Tempo Exec
20004	1600	0.2	0.00	0.8	0.00	0.5	320.00	2948.9
20005	1600	0.2	20.02	0.8	10.21	0.5	341.13	3284.4
20008	1600	0.2	30.26	0.8	30.13	0.5	359.28	3.491.5
10006	1680	0.2	20.05	0.8	30.08	0.5	367.08	3520.8
10004	2019	0.2	40.02	0.8	10.10	0.5	440.86	3727.9

O EXEHDA-DA registra a adaptação inferida na classe Adapt, e notifica a aplicação CalcPi para que a ação seja executada. Na tabela 6.18 mostramos a instância de um nodo selecionado para escalonamento da CalcPi.

Tabela 6.18: Instância criada pelo EXEHDA-DA na classe Adapt

Propriedade	**Adaptação Inferida**
Adapt_AplicacaoId	200
Adapt_CompOrigId	000800
Adapt_AdaptadorId	051
Adapt_CompAdaptId	000800
Adapt_NodoId	10006
Adapt_Estado	1

No EXEHDA-DA, a adaptação é inferida no momento em que recebe a notificação de mudança de contexto através do Serviço de Reconhecimento de Contexto. A ação resultante através desta inferência de adaptação é executada no momento em que é disparado o comando adaptativo, colocado pelo programador dentro do código do componente da aplicação, como sendo uma primitiva que dispara uma computação.

Este capítulo explorou duas aplicações de natureza bastante distintas, quando da avaliação do EXEHDA-DA no controle de adaptações funcionais e não-funcionais. Para tanto, foram sistematizados os diferentes procedimentos e a decorrente manipulação do modelo ontológico, quando da implementação das diferentes funcionalidades.

Por último, o próximo capítulo irá tratar das considerações finais do trabalho.

7 CONSIDERAÇÕES FINAIS

Neste capítulo está resumida a comparação do EXEHDA-DA com os trabalhos diretamente relacionados, são apresentadas as contribuições da pesquisa realizada, as principais conclusões obtidas, um registro das publicações realizadas, bem como uma previsão de trabalhos futuros.

7.1 Discussão dos Trabalhos Relacionados ao EXEHDA-DA

Nesta seção são discutidos os trabalhos relacionados ao EXEHDA-DA. O estudo de trabalhos relacionados foi desenvolvido em dois momentos: um primeiro, de âmbito mais geral, teve por finalidade identificar as tendências no foco da pesquisa. O resumo deste estudo está no anexo A, e compreende uma descrição de 18 trabalhos em adaptação ao contexto na UbiComp. Um segundo momento, após estarem definidas as linhas gerais para concepção do EXEHDA-DA, envolveu a avaliação de 7 outros trabalhos, cujo escopo contemplava um maior grau de similaridade com o EXEHDA-DA. Na continuidade estes 7 trabalhos terão suas funcionalidades discutidas em relação ao modelo proposto.

A tabela 7.1 apresenta uma comparação entre os trabalhos relacionados, tendo por base características consideradas na modelagem do EXEHDA-DA. Os campos marcados com ”+” indicam que o modelo possui a característica. Os campos sem marcação indicam que a ferramenta não possui a funcionalidade especificada. As características consideradas na comparação dos trabalhos relacionados com o EXEHDA-DA, foram:

- 01 - Adaptação funcional da aplicação e/ou do middleware;
- 02 - Adaptação não-funcional;
- 03 - Controle da adaptação externo à aplicação;
- 04 - Modelagem Semântica para a política da adaptação da aplicação;
- 05 - Dispositivos Móveis;
- 06 - Reutilização de políticas a partir de um catálogo;
- 07 - Tratamento autonômico da adaptação baseado em regras;
- 08 - Função de Utilidade.

Tabela 7.1: Comparativo dos Trabalhos relacionados ao modelo EXEHDA-DA.

MODELOS	01	02	03	04	05	06	07	08
CARISMA	+		+		+	+	+	+
CHISEL		+	+				+	
QUO	+		+			+		
RAINBOW	+		+			+	+	
MADAM	+	+	+		+		+	+
PROTEUS	+		+	+			+	
SECAS	+		+		+		+	
EXEHDA-DA	+	+	+	+	+	+	+	+

Embora as políticas baseadas em regras adotadas no Carisma se mostrem 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 de um projeto arquitetural a nível 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 para uma determinada adaptação, enquanto a abordagem do EXEHDA-DA com a função de utilidade é possível selecionar a melhor adaptação entre as alternativas, que são inferidas pelo servidor de adaptação.

Em Chisel as políticas são definidas com uma linguagem declarativa e essencialmente baseadas em regras, realizando adaptações não-funcionais. O EXEHDA-DA, além de regras, se vale de funções de utilidade no cômputo das decisões de adaptação, assim como, contempla adaptações funcionais e não-funcionais.

Uma desvantagem da arquitetura Quo é que ela só se concentra na adaptação dos aspectos funcionais da aplicação, em função do estado dos recursos do sistema, não considera adaptações não-funcionais quando do disparo de uma aplicação distribuída, bem como não tem suporte para dispositivos portáteis e redes sem fio, ao contrário do EXEHDA-DA. QUO, como o EXEHDA-DA, é também um dos poucos projetos que tentou abordar adaptação como um componente ou serviço reutilizáveis.

As técnicas de adaptação propostas pela abordagem Rainbow são semelhantes ao EXEHDA-DA, na medida em que separa a adaptação das funções lógicas da aplicação. No entanto, o Rainbow não foca nas exigências pertinentes aos ambientes com dispositivos móveis. Além disso, os seus desenvolvedores basearam as estratégias de adaptação em regras situação-ação, que especificam exatamente o que fazer em determinadas situações. Por sua vez, EXEHDA-DA usa políticas de objetivos estendidas expressas como funções de utilidade, que é um nível mais alto de especificação das estratégias de adaptação, definindo objetivos e implementando essas políticas, através de regras lógicas.

Madam, além de um *middleware*, contempla uma metodologia de desenvolvimento *model-driven*, que se baseia em modelos de adaptação e as correspondentes transformações modelo-para-código. EXEHDA-DA não tem foco em ferramentas de de-

senvolvimento de software, seu objetivo é prover suporte para adaptações que possam ser utilizadas por componentes das aplicações. Madam não utiliza modelagem semântica para política de adaptação da aplicação, assim como não explicita se as políticas de adaptação podem ser reutilizadas para novas adaptações ou para outros componentes de software.

Proteus realiza adaptações nas políticas de acesso à recursos, utilizadas pelas aplicações. Na gerência do processo adaptativo, de forma análoga ao EXEHDA-DA, contempla o uso de um modelo semântico. Por sua vez, o EXEHDA-DA prevê um espectro mais amplo de adaptações, estando sujeitas ao processo adaptativo diferentes funcionalidades de aplicações com naturezas diversas.

O projeto SECAS preocupa-se apenas com as adaptações funcionais para aplicações da área médica. Utiliza expressões lógicas para as configurações de contexto para os adaptadores. Diferentemente do EXEHDA-DA, não utiliza a modelagem semântica para as regras de adaptação e sim um formalismo baseado em Redes de Petri.

De modo geral, pode-se dizer que 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.
Além disso, o EXEHDA-DA permite, a qualquer momento, o desenvolvedor da aplicação incluir novas adaptações, novas condições, devido à política da aplicação ser mantida externamente, qualificando a configuração dos perfis das aplicações, através das regras, parâmetros e operações de sua política de adaptação.

Nesta seção o modelo proposto pelo EXEHDA-DA foi comparado com trabalhos relacionados, cujas características são próximas a sua modelagem. Verificamos que alguns dos trabalhos tratam apenas um tipo de adaptação, funcional ou não-funcional. Outros não fornecem a possibilidade de manutenção, incremental 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.

7.2 Contribuições da Pesquisa Realizada

A concepção do EXEHDA-DA compreendeu o estudo e a pesquisa de diversos aspectos relacionados ao controle da adaptação na Computação Ubíqua. A seguir estão destacadas as principais contribuições contempladas durante o desenvolvimento do trabalho.

As contribuições estão distribuídas em três frentes, que compreendem a concepção de: (i) modelo ontológico do ambiente ubíquo utilizado pelo *middleware*; (ii) *framework* FWADAPT e, (iii) arquitetura e funcionalidades de um serviço para controle da adaptação dinâmica ao contexto.

Como primeira frente, a concepção de um modelo ontológico para o ambiente ubíquo utilizado pelos serviços do *middleware* teve como principais contribuições:

- participar na criação da ontologia ONTUBI, que representa o ambiente ubíquo gerenciado pelo *middleware* EXEHDA;
- permitir a representação do ambiente ubíquo e da política de adaptação da

aplicação, em um nível elevado de abstração, através de uma linguagem de alto nível tipo OWL;

- possibilitar a definição em alto nível das adaptações previstas para as aplicações;
- contemplar a disponibilização e compartilhamento das informações do modelo ontológico entre diversas aplicações;
- permitir inferências e consultas lógicas em tempo de execução;
- possibilitar a integração a componentes de software tradicionais, através da utilização da linguagem OWL, que opera sobre estruturas similares às linguagens orientadas a objeto;
- possibilitar a validação parcial ou incremental do modelo;
- visão única do modelo para os componentes do G3PD.

Enquanto uma segunda frente, o FWADAPT consiste de um *framework* para criação do modelo ontológico de regras para a Política de Adaptação da Aplicação, facilitando a atividade da configuração das adaptações para o desenvolvedor, bem como, permitindo uma visão mais facilitada da Política de Adaptação já instanciada. A concepção do *framework* FWADAPT contribuiu com as seguintes funcionalidades:

- permite instanciação individualizada de aplicações e seus componentes;
- faculta associação gráfica dos adaptadores com os componentes com perfil adaptativo;
- instanciação na OntAdapt dos parâmetros para tomada de decisão de adaptação referente aos diversos componentes. Estes parâmetros são repassados para uso das operações e regra de adaptação;
- manipulação de parâmetros de adaptação existentes, associando-os ao tipo, operações, valores e critério de utilidade;
- construção de relações entre aplicação, componentes e adaptações;
- inclusão incremental de novas adaptações, parâmetros e operações;
- instanciação de diferentes níveis do modelo ontológico utilizado pelo EXEHDA-DA para gerência dos diferentes aspectos considerados pelos mecanismos de controle de adaptações dinâmicas definidas pelas aplicações.

Como terceira frente, a concepção da arquitetura e das funcionalidades do serviço EXEHDA-DA trouxeram as seguintes contribuições:

- processamento de contexto personalizado por componente da aplicação;
- definição de uma arquitetura de software para o serviço EXEHDA-DA baseada em regras de adaptação personalizáveis por aplicação;
- concepção de uma interface integrada aos demais serviços do *middleware*;

- identificação dos comandos adaptativos a serem utilizados pelo programador. Estes comandos são utilizados para definir as ações adaptativas na aplicação e estabelecer a ligação com o controle da adaptação dinâmica;

- gerência pró-ativa da adaptação, a qual pode ser disparada a qualquer momento, e sem intervenção do usuário, em reação às mudanças de contexto;

- faculta que seja explorada qualquer tipo de dependência de contexto, permitindo para que isto ocorra, a instanciação individualizada de políticas de adaptação associadas às diferentes aplicações;

- coordenação do processamento das funcionalidades da aplicação feito independentemente do controle de adaptação dinâmica ao contexto;

- contribuição para agilizar o desenvolvimento de aplicações adaptativas, no que se refere a independência da aplicação, em relação ao controle da adaptação, quanto à especificação da política de adaptação;

- permite que ações de elevada prioridade possam ser disparadas pelo *middleware* de forma automática em função do contexto;

- possibilidade de inferências lógicas baseadas em semântica para a tomada de decisões de adaptação;

- contempla descrição das diferentes políticas de adaptação em linguagem de alto nível do tipo OWL;

- emprego do conceito de utilidade como forma de priorizar as alternativas geradas pela política de adaptação;

- facilidade de manutenção e reutilização das diferentes informações necessárias a operação do modelo, onde destacam-se os adaptadores, as regras de adaptação com seus parâmetros e operações, descritos e instanciados em linguagem de alto nível, de forma independente do código das aplicações;

7.3 Principais Conclusões

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.

Um aspecto fundamental dos sistemas ubíquos é a sua disponibilidade a todo o momento nos ambientes computacionais 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 de software 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, podendo fazer com que suas necessidades, sejam atendidas com mais facilidade e, muitas vezes até, sem que ele mesmo tenha que solicitar procedimentos explicitamente.

Ontologias são especificações formais dos conceitos de um determinado domínio, com os quais 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 em rede 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 para manipulação de ontologias 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 paradigma, a Computação Autônoma. Este tipo de computação contempla 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. O EXEHDA-DA 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 proposta do EXEHDA-DA, tem por base proporcionar uma maior variabilidade de adaptação e uma maior adequação às necessidades do usuário. A utilidade é empregada com os valores dos parâmetros dos vários atributos, bem como com um peso que reflete as prioridades do atributo ou das necessidades do usuário.

Na tabela comparativa de Modelos *Context-aware*, tendo por base as características da Computação Ubíqua, apresentada no ítem A.0.1 do anexo A, constatou-se que 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 uma preocupação menor 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.

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, o trabalho desenvolvido, quando da concepção do modelo de controle da adaptação, considerou como questão central 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 proposta culminou em um mecanismo de adaptação genérico capaz de

ser utilizado por diferentes aplicações, em tempo de execução. Um outro aspecto significativo do modelo proposto, é que o mesmo 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*. Como perspectiva geral foi perseguida a premissa de adaptar as aplicações da computação ubíqua ao ambiente sem, ou com mínima, intervenção explícita do usuário.

Resumindo, podemos dizer que na Adaptação Multi-nível Colaborativa o Serviço de Controle da Adaptação processa a adaptação no momento em que recebe a notificação de contexto. Para tanto são utilizadas as informações de contexto, a política de adaptação da aplicação e as preferências do usuário. A seguir a decisão de adaptação inferida é instanciada na classe Adapt para retenção da decisão de adaptação ao contexto. A aplicação após receber a notificação, que aconteceu o cômputo de uma adaptação de seu interesse, solicita a execução do comando adaptativo ao *middleware*, o qual consulta a classe Adapt para promover a adaptação ao contexto.

7.4 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.

- CIC-UCPEL 2009: WARKEN, Nelsi, YAMIN, Adenauer C. EXEHDA-DA: Uma Proposta de Controle da Adaptação Dinâmica ao Contexto na Computação Ubíqua. Congresso de Iniciação Científica da Universidade Católica de Pelotas. Pelotas, RS. Mostra de Pós-Graduação, 2009. Prêmio Pesquisador em Pós-Graduação.

- ERAD 2010: WARKEN, Nelsi, YAMIN, Adenauer C. EXEHDA-DA: Uma Proposta de Controle da Adaptação Dinâmica ao Contexto na Computação Ubíqua. In: 10a Escola Regional de Alto Desempenho, 2010, Passo Fundo-RS. ERAD 2010

- Fórum de Pós-Graduação da 10a Escola Regional de Alto Desempenho, 2010. p.148-149.

- ERAD 2010: CARDOZO, Amanda Argou, WARKEN, Nelsi, YAMIN, Adenauer C. FWADAPT: Framework para Definição de Política de Adaptação Dinâmica de Aplicações na Computação Ubíqua. In: 10a Escola Regional de Alto Desempenho, 2010, Passo Fundo-RS. ERAD 2010 - Fórum de Iniciação Científica da 10a Escola Regional de Alto Desempenho, 2010. p.193-196.

7.5 Trabalhos Futuros

Diversas questões de fundo irão acompanhar a continuidade das pesquisas do EXEHDA-DA. 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 pesquisa em aberto: como validar e testar um sistema de software que é capaz de promover adaptações imprevistas em tempo de execução? Como fica a segurança e a confiança na informação de contexto distribuída?
É importante que estas questões sejam tratadas em trabalhos futuros. Com esta intenção, o EXEHDA-DA contempla os seguintes trabalhos futuros:

- possibilidade de ter regras de adaptação que modifiquem tanto aspectos funcionais como parâmetros de outras regras (capacidade reflexiva de adaptação do *middleware*), adaptação ao contexto do próprio EXEHDA-DA;

- expandir funcionalidades do *framework* FWADAPT. Dentre estas destacaríamos: a utilização da Classe Operacao na composição da regra lógica do adaptador; conversão automática da regra lógica em uma regra que possa ser usada diretamente pelo EXEHDA-DA, através da API Jena/SPARQL; mostrar a regra lógica num formato mais amigável para o usuário;

- aprofundar estudos das alternativas para especificação das operações que compõe a regra de adaptação;

- promover a aplicação do modelo de controle da adaptação dinâmica ao contexto em um ambientes com demandas potencias, como a Embrapa Clima Temperado;

- dar continuidade aos procedimentos de integração do EXEHDA-DA com o diferentes serviços e funcionalidades do *middleware* EXEHDA;

- viabilizar o uso do histórico das adaptações realizadas como subsídio para a tomada de decisões para novos procedimentos adaptativos, utilizando a OntHistAdapt (seção 4.1;

- revisar o atual modelo ontológico, documentando as suas classes, atributos e relacionamentos;

- revisar as taxonomias na área de aplicações ubíquas, sistematizando suas funcionalidades e propondo novos comandos adaptativos para atendimento das suas demandas;

- expandir o modelo de controle de adaptação dinâmica do EXEHDA-DA para que contemple tomada de decisão adaptativa considerando histórico de contexto e histórico de adaptações inferidas (trilhas de adaptações).

As diferentes produções referentes ao trabalho de estudo e pesquisa do EXEHDA-DA estão disponíveis no Wiki: http://olaria.ucpel.tche.br/nelsiw/.

REFERÊNCIAS

ABEL, F.; HERDER, E.; KÄRGER, P.; OLMEDILLA, D.; SIBERSKI, W. Exploiting Preference Queries for Searching Learning Resources. In: EC-TEL, 2007. **Anais...** Springer, 2007. p.143–157. (Lecture Notes in Computer Science, v.4753).

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.

AUGUSTIN, I.; YAMIN, A. C.; SILVA, L. C. da. Building a Smart Environment at Large-scale with a Pervasive Grid Middleware. **In: Jiannong Wong. (Org.). Grid Computing Research Progress**, New York, NY, USA, v.1, n.11-12, p.323–344, 2008.

AUGUSTIN, I.; YAMIN, A. C.; SILVA, L. C. da; REAL, R. A.; FRAINER, G.; GEYER, C. F. R. ISAMadapt: abstractions and tools for designing general-purpose pervasive applications: Experiences with Auto-adaptive and Reconfigurable Systems. **Softw. Pract. Exper.**, New York, NY, USA, v.36, n.11-12, p.1231–1256, 2006.

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.

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.

FRAINER, G.; LOPES, J. L. B.; SANDER, R. H.; AUGUSTIN, I.; GEYER, C. F. R.; YAMIN, A. C. Towards Pervasive Applications in a Grid Computing Environment. **XXXIII Conferencia Latinoamericana de Informática - CLEI 2007**, San José, 2007.

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. **ICUC 2006 Ubiquitous Computing**, Madrid, Spain, v.208, june 2006.

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.

KHARGHARIA, B.; HARIRI, S.; YOUSIF, M. S. Autonomic power and performance management for computing systems. **Cluster Computing**, [S.l.], v.11, n.2, p.167–181, 2008.

KNUBLAUCH, H.; MUSEN, M. A.; RECTOR, A. L. Editing Description Logic Ontologies with the Protégé OWL Plugin. In: DESCRIPTION LOGICS, 2004. **Anais...** CEUR-WS.org, 2004. (CEUR Workshop Proceedings, v.104).

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.

LINDHOLM, T.; YELLIN, F. **The Java Virtual Machine Specification**. [S.l.]: Addison-Wesley, 1997.

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.

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.

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

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.

PRESS, W. H.; FLANNERY, B. P.; TEUKOLSKY, S. A.; VETTERLING, W. T. Numerical Recipes in C: The Art of Scientific. 2nd ed. Cambridge, UK. **Cambridge University Press**, [S.l.], 1992.

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

REAL, R. A. **Tips**, **uma proposta de escalonamento direcionada à computação pervasiva**. Porto Alegre, RS: [s.n.], 2004. 115p.p. [s.n.].

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.

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).

SIBERSKI, W.; PAN, J. Z.; THADEN, U. Querying the Semantic Web with Preferences. **Proceedings of the 5th International SemanticWeb Conference (ISWC) 2006**, **Athens**, [S.l.], v.4273, n.3, p.612–614, nov 2006.

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. **Um Mecanismo para Sensibilidade ao Contexto com Suporte Semântico na UbiComp**. 2010. Tese de Mestrado em Ciência da Computação — PPGINF/Centro Politécnico/UCPEL, Pelotas-RS.

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, 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.; BROWN, J. S. The Coming Age of Calm Computing. **Copernicus**, New York, p.75–85, 1997.

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 TRABALHOS RELACIONADOS AO CONTROLE DA ADAPTAÇÃO

Neste anexo 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 resumido, através de uma tabela comparativa, a situação encontrada em relação a alguns requisitos, considerados importantes na Computação Ubíqua.

Dos diversos projetos estudados, foram relacionados 18, 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 internos; (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 *framework* em Java utilizado no desenvolvimento de 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. 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 infraestrutura de localização de serviços que permite as aplicações selecionarem serviços que são associados a uma determinada área espacial. A principal característica da arquitetura AROUND, como uma aplicação 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 *Top-Page*. 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. Basicamente, ele é constituído de duas partes: *Context-Awareness Programming Framework* e *Context-Awareness Runtime Infrastructure*. O *Context-Awareness Programming Framework* provê um conjunto de classes e interfaces Java para o desenvolvimento de aplicações 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 infraestrutura 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 infraestrutura, 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*. Apresenta uma infraestrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis. Trata 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 infraestrutura 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 infraestrutura 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. **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.

A.0.1 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 A.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.

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.

Tabela A.1: Comparativo de 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. CAPPUCINO		+[7]	+				+					

A.0.2 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 idio-

mas, 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 são carentes de uma solução genérica e padronizada para as aplicações ubíquas.

ANEXO B METODOLOGIA EMPREGADA

A seguir será detalhada a metodologia adotada para alcançar os objetivos propostos neste trabalho. Sua inclusão tem o objetivo de contribuir para o entendimento dos procedimentos de estudo e pesquisa realizados.

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: estruturação e criação das 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: implementação do 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 foram 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 foi 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 adaptativas ao contexto.

Escrita da Dissertação: redação progressiva do texto da dissertação. A redação foi feita a medida que as atividades foram sendo realizadas, de maneira incremental. Foi utilizado software para controle de versões (SVN) localizado no servidor "olaria.ucpel.tche.br".