Controle da Adaptação na Computação Ubíqua

Nelsi Warken[1], **Adenauer C. Yamin**[1], **Luiz A. M. Palazzo**[1]

[1]Escola de Informática – Universidade Católica de Pelotas (UCPel)
Rua Félix da Cunha, 412 – 96.010-000 – Pelotas – RS – Brasil

Resumo. *Espaço ubíquo refere-se ao ambiente computacional do usuário, disponível todo o tempo, de qualquer lugar e a partir de qualquer dispositivo. Na computação ubíqua, os diversos sistemas interagem com o ser humano a todo o momento, não importando onde esteja, constituindo um ambiente altamente distribuído, heterogêneo, dinâmico, móvel e mutável. As aplicações devem ser adaptáveis ao ambiente, considerando o contexto em que estão inseridas. Ontologias são especificações formais dos conceitos de um determinado domínio, constituindo o núcleo da Web Semântica. Aplicações semânticas buscam interpretar o significado de textos e outros formatos de dados, permitindo estabelecer relacionamentos e inferências entre os dados. A Computação Autônoma indica que os sistemas computacionais deveriam desempenhar funções automáticas de configuração, tratamento, otimização e proteção, substituindo tarefas complexas por políticas descritas em alto nível por usuários e administradores. O objetivo central do trabalho é o estudo de ontologias e sistemas autônomos como mecanismos de controle da adaptação ao contexto na computação ubíqua, considerando suas principais características e desafios de pesquisa.*

1. Introdução

As tecnologias mais profundas são aquelas que desaparecem, elas se integram na vida cotidiana até se tornarem indistinguíveis da mesma [Weiser 1991]. Esta frase retirada do clássico artigo sobre a Computação para o século 21, sintetiza um pouco do que é esperado com a Computação Ubíqua. O termo trata do acesso ao ambiente computacional do usuário, isto é, ao espaço ubíquo do usuário, em qualquer lugar, todo o tempo com qualquer dispositivo. A computação e seus diversos sistemas interagem com o ser humano a todo o momento, não importando onde ele esteja, em casa, no trabalho e na rua, constituindo um ambiente altamente distribuído, heterogêneo, dinâmico, móvel, mutável e com forte interação entre homem e máquina [Augustin 2003].

Contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é qualquer pessoa, lugar ou objeto que é considerado relevante para a interação entre usuário e a aplicação, incluindo o próprio usuário e a própria aplicação [Dey et al. 2001]. Através da adaptação é feita a avaliação de elementos do contexto, como localização, tempo e atividades do usuário, permitindo o desenvolvimento de aplicações sensíveis ao mesmo [Henricksen et al. 2006].

O G3PD, grupo de pesquisa em que estou inserida, participa de um consórcio de pesquisa formado pela Universidade Católica de Pelotas (UCPEL), Universidade Federal de Pelotas (UFPEL), Universidade Federal de Santa Maria (UFMS) e Universidade Federal do Rio Grande do Sul (UFRGS). O G3PD está desenvolvendo o *middleware*

EXEHDA, *Execution Environment for Highly Distributed Applications*. O EXEHDA é um *middleware* adaptativo ao contexto, baseado em serviços, que tem por objetivo criar e gerenciar um ambiente ubíquo. As aplicações deste ambiente são distribuídas, móveis e adaptativas ao contexto.

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

Assim, motivado por estas idéias, este trabalho propõe-se a utilizar Ontologias, Serviços Web e Sistemas Autônomos para Controle da Adaptação ao Contexto na Computação Ubíqua, facilitando e otimizando o desenvolvimento de aplicações *context-aware*. Pode-se tornar uma tendência natural, a utilização combinada das áreas de ontologias, computação ubíqua e sistemas autônomos, pelo avanço rápido tanto da Web Semântica como dos ambientes ubíquos, além de facilitar a análise e a reutilização destas novas aplicações e a sua integração. A padronização para o desenvolvimento de aplicações ubíquas poder ser interessante e desejável, em função das demandas crescentes dos usuários. [Hilera and Ruiz 2006]. Outra motivação é a possibilidade de sugerir um mecanismo de controle de adaptação, funcional e não funcional, para o *middleware* EXEHDA, foco de desenvolvimento do G3PD.

O objetivo central do trabalho é o estudo de ontologias e sistemas autônomos como mecanismos de controle da adaptação ao contexto na computação ubíqua, considerando suas principais características e desafios de pesquisa. 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.

2. Computação Ubíqua

Mark Weiser, no início dos anos 90, já previa um aumento nas funcionalidades e na disponibilidade de serviços de computação para os usuários finais, entretanto com a visibilidade destes serviços sendo a menor possível. A computação não seria exclusividade de um computador, mas de diversos dispositivos conectados entre si. A Computação Ubíqua, também conhecida como Computação Invisível [Norman 1998], é definida como sendo o terceiro grande paradigma computacional, precedido pelo império dos mainframes e pela onda da computação pessoal [Weiser and Brown 1997]. Marcante no passado e hoje presente em cenários específicos, os computadores de grande porte foram definidos sobre uma arquitetura onde, poucas máquinas de imenso poder de processamento, são compartilhadas simultaneamente por inúmeros usuários. Com o domínio dos microcomputadores, observamos a máquina como peça íntima de uso pessoal e exclusivo. Na Computação Ubíqua, veremos a tecnologia discretamente nos envolvendo e agindo nos bastidores de nossas vidas, formando uma malha de dispositivos inteligentes em torno de cada indivíduo, sem contudo impactar profundamente seu modo de viver.

O panorama de aplicações sensíveis ao contexto deve prever a mobilidade de

equipamentos e usuários, denominada mobilidade física, e também dos componentes da aplicação e serviços, chamada de mobilidade lógica. Para isso, as aplicações devem ter o estilo siga-me, facultando que o usuário possa acessar seu ambiente computacional independente da localização, do tempo e do dispositivo utilizado [Yamin 2004, Yamin et al. 2005].

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

Ainda existe muita discussão e controvérsia nos termos Computação Pervasiva (*Pervasive Computing*) e Computação Ubíqua (*Ubiquitous Computing*). O termo *pervasivo* não existe na língua Portuguesa e em inglês significa espalhado, integrado, universal. O termo *ubíquo* significa onipresente. Os dois termos também são tratados sem distinção por alguns pesquisadores. Neste trabalho, para fins de padronização, será adotado Computação Ubíqua ou simplesmente *UBICOMP*.

A computação ubíqua tem como objetivo fundamental o aumento da qualidade de vida dos usuários. Desenvolve-se de forma onipresente e adaptável ao contexto, fazendo com que não seja mais o usuário que se desloca até a rede, mas a rede que passa a envolver o usuários e os objetos numa conexão generalizada. A Computação Ubíqua trata do acesso ao ambiente computacional do usuário, em qualquer lugar, todo o tempo com qualquer dispositivo, constituindo um ambiente altamente distribuído, heterogêneo, dinâmico, móvel, mutável e com forte interação entre homem e máquina [Augustin 2003].

3. Principais Conceitos em Ontologias

A definição mais aceita e citada pelos autores da área de Computação é a que define ontologia como uma especificação formal e explícita de uma conceituação compartilhada [Fensel 2000], onde: conceituação - se refere ao modelo abstrato do mundo real; explícita - significa que os conceitos e seus requisitos são definidos explicitamente, definições de conceitos, instâncias, relações, restrições e axiomas; formal - indica que a ontologia é processável por máquina, permite raciocínio automático e possui semântica lógica formal; compartilhada - significa que uma ontologia captura o conhecimento apresentado não apenas por um único indivíduo, mas por um grupo, sendo uma terminologia comum da área modelada ou acordada entre os desenvolvedores. Uma ontologia não se resume somente a um vocabulário, também possui relacionamentos e restrições (axiomas) entre os conceitos definidos pelo vocabulário e, através de regras de inferência, é possível derivar novos fatos baseando-se em fatos existentes.

Ontologia pode ser considerado um bom instrumento, embora não o único, para especificar os conceitos da computação ubíqua, além de ser o elemento central na emer-

gente Web Semântica, novo paradigma como evolução da atual Web (Web Sintática). Web Semântica se caracteriza pela análise de páginas da Web indexados por motores de busca, que permitindo anotar documentos e recursos na Web semântica com informações baseadas em ontologias [Berners-Lee et al. 2001].

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

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

XML, *eXtensible Markup Language*, é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais. É um subtipo de SGML, *Standard Generalized Markup Language*, ou Linguagem Padronizada de Marcação Genérica, capaz de descrever diversos tipos de dados. Seu propósito principal é a facilidade de compartilhamento de informações através da Internet. 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.

RDF (*Resource Description Framework*) não é uma linguagem, mas um modelo, sendo descrito com a utilização do XML. Sua característica está na forma com que o modelo de dados utiliza metadados para descrever recursos, apresentando um significado ao recurso. O modelo é apresentado como tuplas (predicado, sujeito, objeto), permitindo uma visualização através de grafos rotulados. A viabilidade para representar a interligação das propriedades e os outros recursos mostra-se limitada. Com isso, surgiu o **RDF Schema** [Brickley and Guha 2004] que é uma extensão de RDF e 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, a fim de serem reaproveitadas em outros modelos.

Segundo a Segundo a W3C, a maneira mais completa de representar ontologias é sobre **OWL** (*Web Ontology Language*), esta linguagem é uma extensão do RDFS e permite obter um maior nível de expressividade. A linguagem OWL fornece suporte a metadados RDF, abstrações de classes, generalização, agregação, relações de transitividade, simetria e detecção de inconsistências. A W3C lançou a OWL como um padrão no uso de ontologias em 2008. É uma linguagem para definir e instanciar ontologias na Web.

Jena é uma API (*Application Program Interface*) Java desenvolvida pela HP, Hewlett-Packard Company, [McBride 2008], utilizada para desenvolver aplicações baseadas em Semântica Web. A Jena permite manipulação dinâmica de modelos ontológicos e o desenvolvimento de aplicações baseadas em Semântica Web. O Jena vem com um *reasoner* genérico que depois pode ser implementado por terceiros. Jena possui classes para executar *queries* sobre modelos ontológicos. Essas *queries* são feitas em RDQL. Existe

também outra linguagem mais avançada para executar *queries*, **SPARQL**. Permite armazenar as ontologias em bases de dados relacionais. Atualmente o Jena suporta as bases de dados: MySQL, Oracle e PostgreSQL. A W3C também lançou como recomendação o SPARQL em 2007.

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

Entre as vantagens apresentadas por essas linguagens, destacam-se [Knublauch et al. 2004]: i) são otimizadas para a representação de conhecimento estruturado em um nível elevado de abstração; ii) os modelos de domínio podem ser disponibilizados na Internet e compartilhados entre múltiplas aplicações; iii) é baseada em dialetos não ambíguos da lógica formal, permitindo a utilização de serviços inteligentes baseados no raciocínio (reasoning), possibilitando assim, em tempo de execução, a definição dinâmica de classes, a reclassificação de instâncias e a execução de consultas lógicas complexas; iv) essas linguagens operam sobre estruturas similares às linguagens orientadas a objeto, e podem ser eficazmente integradas aos componentes de software tradicionais.

4. Computação Autônoma

A necessidade de integração de sistemas computacionais heterogêneos [Kephart and Chess 2003] tem dificultado as operações de gerência e manutenção nos sistemas corporativos. As milhares de linhas de código desses sistemas tem apresentado níveis de complexidade que desafiam a capacidade da compreensão humana [Ahmed et al. 2007]. Em 2001 foi definido o termo *Computação Autônoma* para descrever uma solução para a crescente complexidade envolvida nos sistemas de software [Horn 2001]. Inspirado no sistema nervoso autônomo dos humanos, a computação autônoma apresenta uma visão onde os sistemas de software possuem a capacidade de auto-gerência baseados em informações contextuais e guiados por políticas descritas em alto nível por seus administradores [Menasce and Kephart 2007]. Assim, as complexidades de baixo-nível seriam abstraídas dos humanos, deixando com que os administradores de software deixem de lado as rotineiras operações de manutenção, concentrando-se em políticas descritas em alto nível. Nos últimos anos a computação autônoma tem chamado a atenção da comunidade científica que, por sua vez, tem incentivado as pesquisas nessa área em busca de uma padronização de conceitos, modelos e aplicações que estejam inseridos na visão de [Horn 2001].

Apesar de existirem algumas interpretações e adaptações da visão inicial, a definição mais aceita na comunidade, classifica a computação autônoma em quatro aspectos principais detalhados a seguir [Kephart and Chess 2003, Lin et al. 2005]:

1. **Auto-configuração**: o aspecto de auto-configuração define que os sistemas de

software devem se adaptar automaticamente de acordo com as mudanças ocorridas no ambiente em que estão inseridos. Entre outras coisas, isto permite que o sistema continue funcionando normalmente quando uma nova entidade de software se integra ao sistema.

2. **Auto-tratamento**: em computação autônoma, a capacidade de auto-tratamento é responsável por detectar, diagnosticar e reparar problemas resultantes de *bugs* ou falhas de *hardware* e *software*. Aplicações com esta capacidade se baseiam em dados de *log* e monitores que indicam falhas no sistema. Uma vez diagnosticado o problema, o sistema deve aplicar as mudanças necessárias para reparar a falha ou indicar o tratamento apropriado para sanar o problema.
3. **Auto-otimização**: sistemas de banco de dados como Oracle e DB2 apresentam centenas de parâmetros de configuração que devem ser ajustados para otimizar o funcionamento. Para isso, os administradores devem ter o conhecimento das responsabilidades de cada parâmetro e verificar qual a melhor combinação para utilizar no sistema em que está administrando. Em computação autônoma, tais ajustes deveriam ser desempenhados de forma automática, tanto em tempo de instalação, quanto em tempo de execução, com a finalidade de melhorar o desempenho do sistema.
4. **Auto-proteção**: apesar da existência de *firewalls* e ferramentas de detecção de intrusão, em muitas situações os humanos devem estar presentes para decidir como proteger o sistema a partir de ataques maliciosos. Por exemplo, ao identificar um grande número de requisições com um determinado IP em um curto espaço de tempo, servidores de nomes poderiam caracterizar uma situação de ataque e bloquear as requisições originadas desta máquina automaticamente. Neste sentido, aplicações com a capacidade de auto-proteção preservam o funcionamento do sistema se defendendo de ataques maliciosos e falhas em cascata, além de antecipar problemas com base em relatórios de *log*, requisições, histórico.

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

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

Em [White et al. 2004] são apresentados três tipos de políticas na concepção de sistemas autônomos: i) 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 presen-

tes nas condições e deve executar as ações sempre que as condições são satisfeitas; ii) 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; iii) 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.

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

Apesar dos dois paradigmas apresentarem focos distintos, é possível perceber que eles possuem características em comum. Por exemplo, a capacidade de adaptação ao contexto de forma transparente para os usuários está presente nos Sistemas de Computação Ubíqua e de Computação Autônoma. Com isso, alguns autores já definem o termo de Computação Ubíqua Autônoma, nos casos em que as tecnologias desenvolvidas apresentam características dos dois paradigmas [O'Donnell et al. 2005, Ranganathan and Campbell 2004].

5. Controle da Adaptação na UBICOMP

O desenvolvimento de aplicações *context-aware*, tem muitos aspectos a serem considerados e desafios a serem vencidos. Poucas linguagens e ferramentas estão disponíveis para a programação destes *softwares* [Want and Pering 2005]. As aplicações precisam ‘entender’ e se adaptar ao ambiente, compreendendo o contexto em que estão inseridas [Maciel and Assis 2004]. Essa nova classe de sistemas computacionais, adaptativas ao contexto, abre perspectivas para o desenvolvimento de aplicações mais ricas, elaboradas e complexas, explorando a natureza dinâmica e a mobilidade do usuário. Portanto, o desenvolvimento de aplicações que se adaptem continuamente ao ambiente e permaneçam funcionando mesmo quando o indivíduo se movimentar ou trocar de dispositivo [Grimm and Bershad 2003], continua um sendo um trabalho de pesquisa interessante e com muitos focos de pesquisa a serem estudados.

Para que um sistema ubíquo tenha o mínimo de intrusão no mundo real, sua infraestrutura de software deve ser sensível ao contexto [Dourish 2004], ou seja, possuir uma base extensa de informações a respeito das pessoas e do ambiente que as abriga e, através destes dados, adaptar seu comportamento, da melhor forma possível, para completa satisfação das espectativas de seus usuários. Uma vez que é possível perceber o contexto, é necessário usar essa informação e agir de forma pró-ativa. Gerência de contexto é a ação de responder a uma mudança detectada. A idéia é tomar decisões baseada em informações sentidas, monitoradas ou inferidas.

O tratamento dinâmico da adaptação à medida que as aplicações são executadas ainda não estão resolvidas para a computação ubíqua. As condições de contexto são próativamente 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. 2005].

O contexto caracteriza informações sobre o estado dos dispositivos envolvidos no processamento, sobre os recursos de rede, sobre a localização do usuário, dentre outros. Deste modo, o monitoramento do contexto, a modelagem das informações provenientes do contexto e a notificação das mesmas são aspectos centrais para as aplicações. Faz-se necessária à 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 et al. 2001].

Etapas da adaptação: i) detecção de alterações: engloba monitoração, interpretação e notificação, visa observar o ambiente para detectar e notificar alterações significativas. Esta pode ser realizada de duas formas: *pull* - a informação é solicitada por uma requisição, o cliente busca a informação sempre que é preciso, ou *push* - a informação é enviada ao cliente que se inscreveu no serviço, sem que ele a tenha explicitamente requisitado. A informação é entregue ao cliente sem que este tenha controle de quando isto ocorre; ii) escolha da ação: engloba a seleção da ação adaptativa entre alternativas pré-definidas pelo programador. A seleção pode ser realizada periodicamente, ou quando ocorre o evento de notificação de alteração; iii) ativação da ação: refere-se à execução da ação adaptativa selecionada.

Considera-se que as abstrações dos sistemas operacionais/linguagens de programação existentes não são suficientes, nem apropriadas para tratar as necessidades de adaptação das aplicações ubíquas. O controle da adaptação deve prever: i) adaptação dinâmica de aspectos não-funcionais; adaptação dinâmica de aspectos funcionais; iii) política cooperativa com a aplicação nas decisões de adaptação. A adaptação não-funcional consiste na capacidade do sistema atuar sobre a localização física dos componentes das aplicações, seja no momento de uma instanciação do componente, seja, posteriormente, via migração do mesmo. Ambas operações demandam a existência de mecanismo para instalação sob demanda do código, assim como mecanismos para descoberta e alocação dinâmicas de recursos e acompanhamento de seu estado. Por sua vez, a adaptação funcional consiste na capacidade do sistema atuar sobre a seleção da implementação do componente a ser utilizado em um determinado contexto de execução, isto é, atua na seleção do código de execução do serviço de uma aplicação.

O EXEHDA: *Execution Environment for Highly Distributed Applications* é um *middleware* direcionado às aplicações distribuídas, móveis e conscientes do contexto da computação ubíqua [Yamin 2004, Yamin et al. 2005]. Este *middleware* é baseado em serviços, que tem por objetivo criar e gerenciar um ambiente ubíquo. As aplicações deste ambiente são distribuídas, móveis e adaptativas ao contexto. 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 ambi-

ente 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.

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

6. Projetos Relacionados ao Controle da Adaptação na UbiComp

Nesta seção serão relacionados alguns projetos adaptativos ao contexto, sob a ótica da Computação Ubíqua. São descritas, de maneira sucinta, algumas características de cada modelo e após é demonstrado, através de uma tabela comparativa, a situação encontrada em relação a alguns requisitos, considerados importantes na Computação Ubíqua.

Relação dos Projetos:

1. **Omnipresent** - Um Sistema Ciente de Contexto baseado em Arquitetura Orientada a Serviço [Almeida 2006] - arquitetura baseada em Web *services* que proporciona várias funcionalidades, como: um serviço de mapas, com os pontos de interesse de acordo com o perfil do usuário; um serviço de rota, com o qual o usuário pode encontrar o menor caminho entre dois pontos no mapa; 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; um serviço que monitora o contexto do usuário e do ambiente ao seu redor; um modelo de representação do contexto usando ontologias.

2. **SOCAM**: *Service-Oriented Context-Aware Middleware* [Gu et al. 2005], 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.

3. **CybreMinder**: *Context-Aware System for Supporting Reminders* [Dey and Abowd 2000] - 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. Para perceber o contexto associado com a lembrança, o CybreMinder usa o Context Toolkit.

4. **Context Toolkit** [Salber et al. 1999] - é um software que ajuda a construir aplicações cientes de contexto. Ele promove três principais conceitos para construir aplicações cientes de contexto: separar a aquisição de contexto do uso de contexto; agregação de contexto e interpretação de contexto. Agregação e interpretação facilitam a aplicação a obter o contexto requerido. *Framework* em Java para programação de aplicações. Fornece classes para programar sensores, agregadores, tradutores e notificadores.

5. **AROUND** [José et al. 2003] provê uma infra-estrutura de localização de serviços que permite as aplicações selecionarem serviços que são associados a uma determinada área espacial. Utiliza dois modelos distintos de seleção de serviços: modelo baseado em distância, modelo baseado em escopo. 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 comunicação entre os componentes é baseada em CORBA.

6. **CoBrA** [Chen et al. 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, 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: o suporte ao recurso limitado dos dispositivos móveis e preocupações com privacidade e segurança do usuário. A ontologia CoBrA define um conjunto de vocabulários para descrever pessoas, agentes, lugares e apresentação de eventos.

7. **Online Aalborg Guide** [Andersen et al. 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.

8. **Flame2008** [Weissenberg et al. 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.

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. A plataforma Nexus monitora o espaço físico, 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. 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** (*Arrhythmia Monitoring System*) [Liszka et al. 2004] é um trabalho na área de telemedicina com o objetivo de prever ataques cardíacos conhecidos como arritmia. 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. 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, passando a localização do usuário.

12. **SOAM**: *An Environment Adaptation Model for the ubique Semantic Web* [Vázquez et al. 2006]. A mais importante entidade da arquitetura é o que foi denominado *smobject* como 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. As 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 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**: *Java Context Awareness Framework* [Bardram 2005] 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*. Um aspecto interessante a ser observado sobre a modelagem de contexto no JCAF é que novas entidades e items de contexto podem ser adicionados a um serviço de contexto sempre que necessário, mesmo em tempo execução. Porém, toda a modelagem da informação contextual fica a cargo do desenvolvedor da aplicação.

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. 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.

16. **LOTUS** [Bublitz 2007] é uma infra-estrutura projetada com o objetivo de abordar o problema da disponibilização da informação contextual às aplicações em uma abordagem integrada. Foi adotada uma solução que reúne as soluções baseadas em ontologias, desenvolvendo um módulo para representar a informação contextual, com as soluções baseadas em *middlewares*, possibilitando a descoberta de nós e serviços mesmo em redes heterogêneas.

17. **ADAPT!** [Souza 2008] - foi desenvolvida através da linguagem Java definindo modelo de entidades, conjunto de predicados, tradutores, engenhos de políticas e provedores de informação na busca da auto-configuração no ambiente físico do *Embedded*. Apresentada uma infra-estrutura para o desenvolvimento de aplicações pervasivas auto-configuráveis. Utiliza uma infra-estrutura de rede *UPnP*, tecnologia *Universal Plug and Play* define uma arquitetura que permite a conectividade em redes ubíquas ponto-a-ponto sem a necessidade de configurações. Foi implementado, a principio, para as atividades: iniciando e parando a reprodução de áudio, ligando e desligando a lâmpada de uma Sala. Não trabalhava com Ontologias.

18. **SECAS**: *Simple Environment for Context Aware Systems* [Chaari and Laforest 2007] é um projeto que trata com a adaptação das aplicações ao contexto (preferências do usuário e do ambiente, dispositivos). Uma plataforma que tem como propósito fazer os serviços, dados e interfaces de usuário de aplicações adaptáveis às situações de contexto. Trabalha com facetas (dimensões): perfil da rede, descrição/preferências do usuário, características dos dispositivos do usuários, localização, ambiente. Foi utilizada a representação das redes Petri para descrever os serviços da aplicação e suas dependências acoplados a serviços Java XML-RPC para estabelecer a adaptação. Foi implementado no ambiente médico, focando o impacto do contexto estático na aplicação, objetivando validar a sua estratégia de adaptação.

19. **PROTEUS** [Toninelli et al. 2007] - um modelo semântico de política adaptativa ao *context-aware*. Trata o contexto como a classe principal para definição e adoção de políticas. Abordagem baseada em descrições lógicas ontológicas e regras em linguagem de programação. O modelo considera adaptação de políticas, adaptação de ações e adaptação de contexto.

20. **CAPPUCINO** [Parra and 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.

7. 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 serviços;
8. Modelagem 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 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.
8:Apenas contexto computacional.

8. 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

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

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

móvel. No Omnipresent, o usuário pode criar regras para encontrar pontos de interesse, quando certas situações acontecem, por exemplo, quando estiver com fome, para listar as lanchonetes num raio de 1 Km. Por sua vez o Flame2008 analisa o histórico do usuário para deduzir novas informações sobre o seu perfil.

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

Em relação aos trabalhos relacionados na tabela comparativa de Modelos versus

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

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

9. Uma Contribuição ao Controle da Adaptação na UBICOMP

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

A proposta do trabalho é criar um modelo de controle da adaptação dinâmica de aplicações em ambiente ubíquo. A idéia é controlar estas adaptações quando da tomada de decisões considerando o contexto, com base em informações monitoradas, informações semânticas e inferências a partir destas mesmas informações. Tipos de adaptação que serão considerados: i) adaptação não-funcional: atua sobre a gerência da execução distribuída, operações de mapeamento, reescalonamento, instanciação remota, migração; ii) adaptação funcional: adaptação do código da aplicação, atua sobre a seleção da implementação em determinado contexto.

Questões a serem consideradas para que ocorra o controle da adaptação ao contexto:

- como tomar decisões automáticas para a adaptação da aplicação, considerando preferências e perfil do usuário, contexto, dispositivos e recursos?
- definição do que é relevante para a aplicação, elementos de contextos + estado + localização;
- como montar e entregar a informação de mudança de contexto em tempo de execução;

- o que fazer com a informação de contexto, onde fazê-lo e como a aplicação deve adaptar-se para que seja sensível ao contexto;
- qual o nível de colaboração entre a aplicação e o *middleware* e como acontece esta colaboração;
- realocação de recursos para manter a aplicação operacional;
- no EXEHDA o tratamento da adaptação funcional é realizado pela gerência da aplicação de modo autônomo;
- como resolver situações de conflito na tomada de decisões;
- como diminuir a complexidade e o tempo de desenvolvimento das aplicações adaptáveis ao contexto;
- considerando todos estes fatores, como deixar tanto a aplicação quanto o *middleware* eficientes?

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.

Quem fará o Controle da Adaptação, Aplicações ou *Middleware*? A figura 1 nos dá uma visão geral do controle da adaptação.

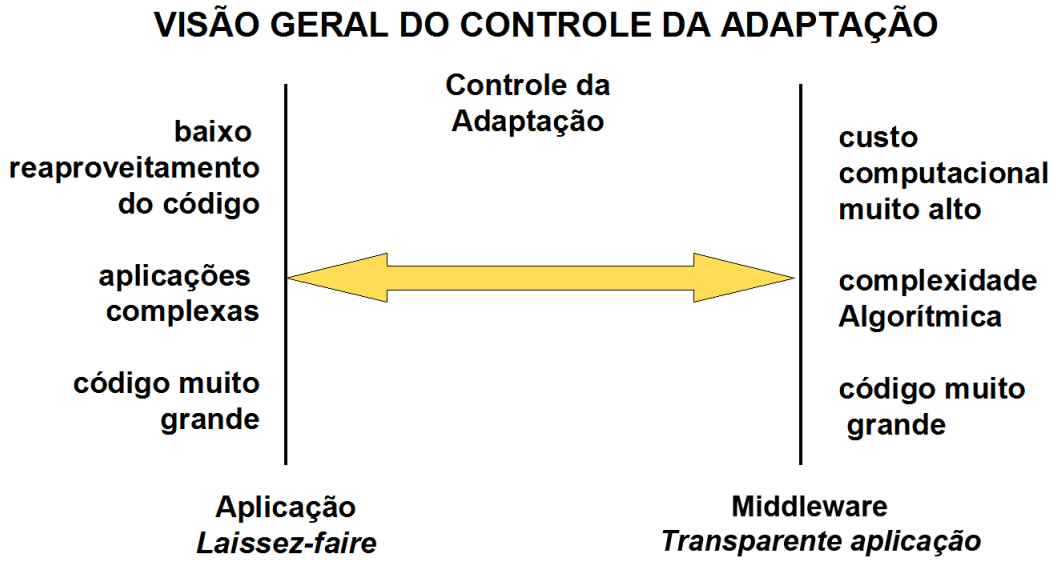

Figura 1. Responsabilidade pela Adaptação

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

A sugestão consta de um mecanismo de adaptação genérico que será utilizado por todas aplicações, em tempo de execução. Neste modelo teremos, no repositório de adaptação, todas as regras, políticas e ações globais e ações específicas de cada aplicação, utilizadas pelo servidor de adaptação, para a partir destes dados e das mudanças do contexto inferir o que e onde a aplicação deverá ser executada. Para que a aplicação seja de alto nível, ser genérica e com bom desempenho, pode-se pensar em permitir uma inserção, por parte do desenvolvedor da aplicação, no modelo de políticas de adaptação (regras, ações e restrições), utilizando a programação e a meta-programação. O modelo de controle de adaptação ao contexto proposto, pretende possibilitar uma evolução incremental das especificações de políticas, regras e ações de adaptação. Permitindo a reutilização e a customização destas para o desenvolvimento de novas aplicações *contex-aware*. Além disso, deverá possuir mecanismos para administrar situações de conflito quando da tomada de decisões do servidor de adaptação. Assim sendo, o objetivo é o de adaptar os serviços da computação ubíqua ao ambiente sem, ou com mínima, intervenção explícita do usuário, utilizando a *inteligência* de dispositivos e de softwares.

Este mecanismo será uma sugestão de um módulo, Controle de Adaptação funcional e não funcional, para o *middleware* EXEHDA. Na figura 2 é mostrado o módulo no EXEHDA que trata da Adaptação ao Contexto. Como áreas de aplicações pretende-se explorar soluções para Medicina e Agropecuária.

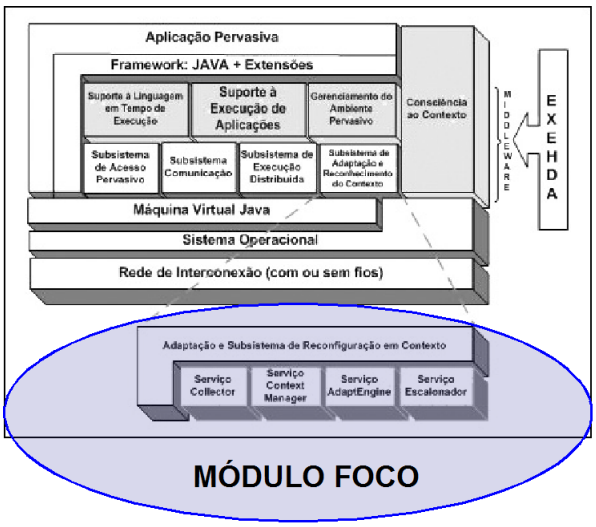

Figura 2. Serviço de Adaptação no *Middleware* EXEHDA [Yamin 2004]

As características do modelo proposto devem ter 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.

10. Considerações Finais

Está sendo analisado o uso de ontologias para a concepção deste modelo de controle de adaptação. Esta escolha se deve ao fato de ontologias estarem sendo utilizadas no desenvolvimento de aplicações ubíquas, especialmente como artefatos de software, com o objetivo de modelar as informações gerenciadas por essas aplicações, por terem uma maior expressividade na definição e descrição de contexto e adaptações, por possuírem a possibilidade de realizar inferências a partir das informações semânticas, facilidade de reutilização e extensibilidade para novas aplicações ou novas situações de contexto. Até o momento, a escolha esta fundamentando-se na utilização combinada de ontologias, Sistemas Autônomos e Computação Ubíqua, onde a sua integração poderá tornar-se um padrão a ser utilizado no desenvolvimento de aplicações em espaços ubíquos, além de poder proporcionar facilidades para a concepção e manutenção destas aplicações, reutilização e um atendimento mais eficaz às demandas dos usuários.

Referências

Ahmed, S., Ahamed, S. I., Sharmin, M., and Haque, M. M. (2007). *Self-healing for autonomic pervasive computing*. ACM.

Almeida, D. R. (2006). Omnipresent-um sistema ciente de contexto baseado em arquitetura orientada. 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., and Klitgaard-Nielsen, R. (2003). *Online Aalborg Guide Development of a Location-Based Service*. Aalborg Universitet.

Augustin, I. (2003). Abstrações para uma linguagem de programação visando aplicações móveis conscientes do contexto em um ambiente de pervasive computing. Tese de doutorado em ciência da computação, Instituto de Informática/UFRGS, Porto Alegre,RS.

Ayed, D., Belhanafi, N., Taconet, C., and Bernard, G. (2005). Deployment of component-based applications on top of a context-aware middleware.

Bardram, J. E. (2005). The java context awareness framework (jcaf) a service infrastructure and programming framework for context-aware applications.

Berners-Lee, Hendler, T., and Lassila, J. (2001). The semantic web. *Scientific American*.

Brickley, D. and Guha, R. V. (2004). Rdf vocabulary description language 1.0: Rdf schema. World Wide Web Consortium, Recommendation REC-rdf-schema-20040210.

Bublitz, F. M. (2007). Infra-estrutura para o desenvolvimento de aplicações cientes de contexto em ambientes pervasivos. Tese de mestrado em informática, Centro de Engenharia Elétrica e Informática / Universidade Federal de Campina Grande, Campina Grande,PB.

Chaari, T. and Laforest, F. (2007). Adaptation in context-aware pervasive information systems: the secas project. *International Journal of pervasive Computing and Communications*, 3(4):400–425.

Chen, H., Finin, T., and Joshi, A. (2003). An ontology for context-aware pervasive.

Dey, A., Abowd, G., and Salber, D. (2001). *A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications*. Human-Computer Interaction.

Dey, A. K. and Abowd, G. D. (2000). Cybreminder: A context-aware system for supporting reminders.

Dourish, P. (2004). What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8(1):19–30.

Dürr, F., Hönle, N., Nicklas, D., Becker, C., and Rothermel, K. (2004). Nexus–A platform for context-aware applications.

Fensel, D. (2000). Ontologies-silver bullet for knowledge management and electronic commerce.

Grimm, R. and Bershad, B. N. (2003). *System Support for Pervasive Applications*, volume 2584 of *Lecture Notes in Computer Science*. Springer.

Gu, T., Pung, H. K., and Zhang, D. (2005). A service-oriented middleware for building context-aware services. *J. Network and Computer Applications*, 28(1):1–18.

Henricksen, K., Indulska, J., and Rakotonirainy, A. (2006). *Using context and preferences to implement self-adapting pervasive computing & applications*.

Hilera, J. R. and Ruiz, F. (2006). Ontologies in ubiquitous computing.

Horn, P. (2001). Autonomic computing-ibm's perspective on the state of information technology.

José, R., Moreira, A. J. C., Rodrigues, H., and Davies, N. (2003). The around architecture for dynamic location-based services. *MONET*, 8(4):377–387.

Kephart, J. O. and Chess, D. M. (2003). Cover feature: The vision of autonomic computing. *Computer*, 36(1):41–50.

Knublauch, H., Rector, A. L., Drummond, N., Horridge, M., Rogers, J., Stevens, R., Wang, H., and Wroe, C. (2004). Owl pizzas-practical experience of teaching owl-dl: Common errors & common patterns.

Lin, P., MacArthur, A., and Leaney, J. (2005). Defining autonomic computing: A software engineering perspective.

Liszka, K. J., York, D. W., Rosenbaum, D. S., Mackin, M. A., and Lichter, M. J. (2004). Remote monitoring of a heterogeneous sensor network for biomedical research in space.

Maciel, R. S. P. and Assis, S. R. (2004). Middleware: Uma solução para o desenvolvimento de aplicações distribuídas.

McBride, B. (2008). An introduction to rdf and the jena rdf api.

Menasce, D. A. and Kephart, J. O. (2007). *Guest Editors' Introduction: Autonomic Computing*, volume 11.

Nakanishi, Y., Takahashi, K., Tsuji, T., and Hakozaki, K. (2002). Icams: A mobile communication tool using location and schedule information. *Lecture Notes in Computer Science*, 2414:239–??

Norman, D. A. (1998). *The Invisible Computer*. MIT Press, Cambridge-MA.

O'Donnell, T., Lewis, D., and Wade, V. (2005). Intuitive human governance of autonomic pervasive computing environments.

Parra, C. A. and Duchien, L. (2008). Model-driven adaptation of ubiquitous applications. *ECEASST*, 11.

Ranganathan, A. and Campbell, R. H. (2004). Autonomic pervasive computing based on planning.

Salber, D., Dey, A. K., and Abowd, G. D. (1999). The context toolkit: Aiding the development of context-enabled applications.

Sorensen, C.-F., Wu, M., Sivaharan, T., Blair, G. S., Okanda, P., Friday, A., and Duran-Limon, H. A. (2004). A context-aware middleware for applications in mobile ad hoc environments.

Souza, M. H. L. (2008). Uma infra-estrutura para o desenvolvimento de aplicações pervasivas autoconfiguráveis. 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., and Lassila, O. (2007). Proteus: A semantic context-aware adaptive policy model.

Vázquez, J. I., de Ipiña, D. L., and Sedano, I. (2006). *SOAM-An Environment Adaptation Model for the Pervasive Semantic Web*, volume 3983 of *Lecture Notes in Computer Science*. Springer.

Want, R. and Pering, T. (2005). *System challenges for ubiquitous & pervasive computing*. ACM.

Weiser and Brown (1997). The coming age of calm technology.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265(3):94–104.

Weissenberg, N., Voisard, A., and Gartmann, R. (2004). Using ontologies in personalized mobile applications.

White, S. R., Hanson, J. E., Whalley, I., Chess, D. M., and Kephart, J. O. (2004). An architectural approach to autonomic computing.

Yamin, A. C. (2004). Arquitetura para um ambiente de grade computacional direcionada às aplicações distribuídas, móveis e conscientes de contexto da computação pervasiva.

Tese de doutorado em ciência da computação, Instituto de Informática/UFRGS, Porto Alegre-RS.

Yamin, A. C., Augustin, I., Barbosa, J., and Geyer, C. (2005). Exehda-adaptative middleware for building a pervasive grid environment.