UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

Uma Abordagem baseada em Aspectos e Composição Dinâmica para a Construção de Aplicações Adaptativas Cientes ao Contexto

Isanio Lopes Araújo Santos

Natal - RN
2008

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

Uma Abordagem baseada em Aspectos e Composição Dinâmica para a Construção de Aplicações Adaptativas Cientes ao Contexto

Isanio Lopes Araujo Santos

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade do Rio Grande do Norte como requisito para a obtenção do grau de Mestre em Sistemas e Computação.

Profª. Dra. Flávia Coimbra Delicato
Orientadora

NATAL-RN
2008

Dedico este trabalho à minha mãe e ao meu pai, que sempre me ensinaram a valorizar o estudo, e a compreender que o meu conhecimento é uma conquista que jamais poderá ser tirada de mim.

AGRADECIMENTOS

Mais uma etapa se encerra, foram muitos desafios durante este período de Mestrado, um período de crescimento profissional e pessoal, mas com fé e perseverança consegui superá-los para alcançar mais essa vitória. Assim, gostaria de agradecer algumas pessoas que me apoiaram e me incentivaram nessa jornada.

Primeiramente gostaria de agradecer aos meus pais, Isaias e Vera, pelo incentivo, pela confiança e pelo esforço de me proporcionar uma boa formação, mesmo que para isso necessitasse estar longe de casa. Agradeço pelos valores passados, pelo apoio, pelo espelho que sempre foram para mim e por sempre me abrirem os olhos para as dificuldades a serem vencidas no dia-a-dia.

Ao meu irmão, Isaias Junior, pelo companheirismo, pela amizade, pelo incentivo e por tudo mais que sempre fez ajudando a mim e aos meus pais para que eu pudesse ter essa oportunidade, sempre lhe serei grato e estarei ao seu lado para o que precisar.

À Débora, pelo amor, pelo carinho, pelo incentivo, pelo companheirismo, pela compreensão, por sua força e caráter e por sempre estar ao meu lado me ajudando e me fazendo crescer e por nunca ter se omitido nos momentos em que mais precisei.

À Professora Flávia pela atenção nas orientações, pela disponibilidade, pelos ensinamentos, pela paciência em revisar o texto, pelos incentivos para se fazer um bom trabalho, pelas eventuais broncas que faziam com que eu me dedicasse mais e mais para conseguir cumprir os objetivos.

À minha tia, Eunice, pelo apoio, pelo carinho, pelo acolhimento e pelos constantes ensinamentos que sempre me motivaram e me ajudaram a lidar com as mais difíceis situações.

A toda a equipe que participou da elaboração dos artigos produzidos com base nessa dissertação, cujos membros são: Professora Flávia, Professor Paulo Pires, Professora Thaís Batista, Professora Luci Pirmez e Ana Liz.

Agradeço também àqueles que durante esse período me deram oportunidades, me abrindo portas para iniciar minha carreira profissional e também a todos os professores, amigos e familiares que contribuíram e me incentivaram a conquistar este objetivo. Para todos vocês eu dedico os mais sinceros agradecimentos. Obrigado.

"A perseverança é a mãe da boa sorte."

Miguel de Cervantes

RESUMO

Aplicações para a computação ubíqua operam em ambientes onde a disponibilidade de recursos muda significativamente durante a sua operação. Tal característica demanda que aplicações sejam adaptativas e cientes do seu contexto de execução. Visando atender esses requisitos, é proposto o PACCA (Projeto de Aplicações Ciente ao Contexto e Adaptativas), um arcabouço para desenvolvimento e execução de aplicações adaptativas cientes de contexto. O paradigma de orientação a aspectos é usado no PACCA para modularizar o comportamento adaptativo e dissociá-lo da lógica da aplicação. Para prover maior flexibilidade o PACCA utiliza o conceito de aspecto abstrato para permitir a extensão e adição de novos interesses adaptativos, além de um modelo de aspectos *default* que contempla interesses adaptativos comuns a grande parte das aplicações ubíquas. A orientação a aspectos aliada à composição dinâmica de software oferece suporte para adaptação ciente ao contexto, guiada por políticas previamente definidas e tem por objetivo permitir a carga de módulos de software sob demanda possibilitando melhor utilização dos recursos limitados de um dispositivo móvel. Um Processo de Desenvolvimento para a construção de aplicações ubíquas também é proposto e visa demonstrar um conjunto de atividades a serem executadas para a concepção de aplicações ubíquas. Por fim, é realizado um estudo quantitativo com o intuito de avaliar com base em métricas a abordagem baseada em aspectos e composição dinâmica para a construção de aplicações ubíquas.

Palavras-Chave: PACCA, computação ubíqua, ciência de contexto, adaptação de software, composição dinâmica, programação orientada a aspectos.

ABSTRACT

Ubiquitous computing systems operate in environments where the available resources significantly change during the system operation, thus requiring adaptive and context aware mechanisms to sense changes in the environment and adapt to new execution contexts. Motivated by this requirement, a framework for developing and executing adaptive context aware applications is proposed. The PACCA framework employs aspect-oriented techniques to modularize the adaptive behavior and to keep apart the application logic from this behavior. PACCA uses abstract aspect concept to provide flexibility by addition of new adaptive concerns that extend the abstract aspect. Furthermore, PACCA has a default aspect model that considers habitual adaptive concerns in ubiquitous applications. It exploits the synergy between aspect-orientation and dynamic composition to achieve context-aware adaptation, guided by predefined policies and aim to allow software modules on demand load making possible better use of mobile devices and yours limited resources. A Development Process for the ubiquitous applications conception is also proposed and presents a set of activities that guide adaptive context-aware developer. Finally, a quantitative study evaluates the approach based on aspects and dynamic composition for the construction of ubiquitous applications based in metrics.

Keywords: PACCA, ubiquitous computing, context-aware, software adaptation, dynamic composition, aspect-oriented programming.

SUMÁRIO

Lista de Figuras

Lista de Tabelas

Lista de Siglas

API - Application Programming Interface

CAE - Coupling on Advice Execution

CASA - Contract-based Adaptive Software Architecture

CA-IDL - Context-Aware Interface Definition Language

CBM - Coupling Between Modules

CFA - Coupling on Field Access

CIS - Context Information Service

CMC - Coupling on Method Call

CORBA - Common Object Request Broker Architecture

CRS - CASA Runtime System

CS - Configuration Service

DAO – Data Access Object

DNS - Domain Name Server

DS - Discovery Service

DSOA - Desenvolvimento de Software Orientado a Aspectos

GUI – Graphic User Interface

IDE - Integrated Development Environment

IEEE - Institute of Electrical and Electronic Engineers

Java SE - Java Standard Edition

Java ME - Java Micro Edition

JCAF - Java Context-Aware Framework

JNDI - Java Naming and Directory Interface

JVM - Java Virtual Machine

LCO - Lack of Cohesion in Operations

LDAP - Lightweight Directory Access Protocol

LIS - Location Inference Service

LOC - Lines of Code

LTS - Labelled Transition Systems

MAC – Media Access Control

MOA - Modelagem Orientada a Aspectos

MOCA - Mobile Collaboration Architecture

MVC - Model-View-Controller

ORB - Object Request Broker

PACCA - Projeto de Aplicações Cientes ao Contexto Adaptativas

PDA - Personal Digital Assistant

POA - Programação Orientada a Aspectos

POO - Programação Orientada a Objetos

RFM - Response for Module

RMI – Remote Method Invocation

RMI-IIOP - Remote Method Invocation – Internet Inter-ORB Protocol

R-ORB – Reconfigurable Object Request Broker

RSCM - Reconfigurable Context-Sensitive Middleware

SPI - Service Provider Interface

SRM - Symbolic Region Manager

Triple-DES - Triple Data Encryption Standard

UI - User Interface

UML – Unified Modeling Language

WOM - Weighted Operations in Module

XML - Extensible Markup Language

1. Introdução

A mobilidade é uma das principais tendências atuais na área de computação distribuída. Os recentes avanços na comunicação sem fio, na miniaturização de dispositivos e no aumento da duração das suas baterias têm criado uma demanda crescente de aplicações e infra-estruturas que explorem e dêem suportem a mobilidade de terminais que se movem no espaço enquanto mantêm suas conexões com outros terminais [LIMA JR; CALSAVARA, 2008]. Esses avanços têm alterado a forma como usuários usam a computação, como eles interagem com dispositivos computacionais, e têm trazido profundas mudanças a todos os setores da vida humana.

Neste contexto de mobilidade, segundo [PIRES et al., 2005] a Computação Ubíqua é uma forma de computação onde o processamento está espalhado no ambiente através de vários dispositivos, que executam tarefas bem definidas, e são interligados de forma que essa estrutura de computação e comunicação torna-se invisível para o usuário. A definição de Computação Ubíqua [WEISER, 1991] propõe a centralização da computação no próprio usuário e nas atividades deste, de forma transparente, com sistemas operando proativamente a partir da percepção do ambiente e de modo independente do controle humano. Tipicamente, aplicações ubíquas são distribuídas, sensíveis ao contexto e móveis.

Dessa forma, aplicações que operam em cenários de Computação Ubíqua caracterizam-se por constantes mudanças em seu estado de execução, causadas, dentre outros fatores, pela mobilidade de seus usuários, por suas diferentes preferências e características de seus dispositivos, e pela variabilidade dos recursos disponíveis (rede, energia, processamento). Esse caráter dinâmico do ambiente de execução demanda da aplicação a capacidade de adaptar-se em função desses fatores. Ou seja, tais aplicações devem levar em conta, nas suas tomadas de decisão, em seu comportamento e em seus processamentos, não apenas as entradas de dados explícitas dos usuários, mas também entradas implícitas referentes ao contexto físico e computacional onde executam [COSTA; STRZYKALSKI; BERNARD, 2007, GRAY; SALBER, 2001].

Em face dessas características, aplicações para a Computação Ubíqua possuem dois importantes requisitos: (i) capacidade de perceber dinamicamente as características do ambiente ao seu redor, conhecida como *ciência de contexto* [HOH; TAN; HARTLEY, 2006]; e (ii) capacidade de adaptar-se para atender às mudanças no ambiente, selecionando e incorporando novos comportamentos, conhecida como

adaptação dinâmica [CANAL; MURILLO; POIZAT, 2006]. Aplicações adaptativas cientes ao contexto monitoram seu ambiente de execução e exploram a natureza contextual provocada pelas mudanças nesse ambiente, com o intuito de fornecer serviços adaptáveis e centrados no usuário.

Nos últimos anos, várias aplicações, de diversas áreas, foram estudadas e desenvolvidas no âmbito da Computação Ubíqua, podendo-se citar como exemplos:

- Guias turísticos eletrônicos que registram anotações de usuários sobre os pontos visitados [HAGEN; MODSCHING; KRAMER, 2005] e disponibilizam essas informações automaticamente para outros usuários interessados, fornecendo um guia de navegação com base na localização atual do usuário.
- Sistemas de Informação ubíquos para ambientes de medicina [GOMES et al., 2005], onde os médicos podem ter acesso a informações de pacientes em dispositivos portáteis a todo instante, inclusive permitindo a paramédicos realizar consultas sobre pacientes durante o deslocamento para um atendimento de urgência.
- Sistemas de escritório que permitem ao usuário continuar uma tarefa, como a edição de uma apresentação, em outro dispositivo, pois ao acessar o sistema, o arquivo em uso é automaticamente transferido para esse outro dispositivo sem a interferência humana [SATYANARAYANAN, 2001].
- Sistemas educacionais de suporte ao professor em suas tarefas de produção de material didático, como projeto *Classroom 2000* [ABOWD, 1999], que apresenta dentre outros objetivos, o de capturar as aulas através de uma lousa eletrônica e disponibilizar essas aulas em conteúdo multimídia, nos dispositivos dos presentes.

1.1. Motivação

Além de lidar com ambientes dinâmicos de execução, aplicações ubíquas caracterizam-se pelo alto grau de heterogeneidade, tanto dos dispositivos computacionais em que executam quanto da infra-estrutura de comunicação, que pode ser composta por várias redes sem fio com tecnologias e características distintas [KANG et al., 2006]. O dinamismo e a heterogeneidade de seu ambiente de execução fazem com que aplicações ubíquas possuam requisitos e níveis de complexidade

diferenciados em relação às aplicações que executam em ambientes tradicionais, requerendo abordagens distintas de desenvolvimento [CANAL; MURILLO; POIZAT, 2006]. Adicionalmente, é necessária a existência de serviços e ambientes de execução que ofereçam suporte para que tais aplicações possam explorar e reagir a mudanças de contexto dentro de seu domínio dinâmico e heterogêneo, sendo imprescindível o uso de mecanismos para prover adaptação de software.

Um fator complicador no desenvolvimento de aplicações ubíquas deriva do fato de que o código de adaptação, que implementa o comportamento adaptativo ciente ao contexto, encontra-se em geral entrelaçado com o código funcional da aplicação, que implementa o comportamento referente à lógica do negócio. Como conseqüência desse entrelaçamento, o código responsável pela adaptação tende a espalhar-se por diversos componentes do sistema, caracterizando-se como um comportamento transversal (*crosscuting behavior*) [KICZALES et al., 1997]. Tal fato gera impactos negativos em diversos atributos desejáveis do sistema de software construído, como, por exemplo, sua modularização.

A fim de desacoplar o comportamento adaptativo ciente ao contexto da lógica de negócio das aplicações, uma possível solução consiste em adotar uma abordagem de Desenvolvimento Orientado a Aspectos (DSOA, ou AOSD, do inglês *Aspect Oriented Software Development* [AOSD, 2008]). Com o uso do paradigma DSOA, componentes que implementam funcionalidades representando a lógica das aplicações (código base) são separados dos componentes que implementam adaptação e manipulação de contexto, através de mecanismos de composição definidos pelo paradigma, visto que a adaptação pode ser analisada como um interesse transversal, pois intercepta diversos pontos da aplicação através de diferentes formas de adaptação. Dessa forma, todo o código que implementa adaptação pode ser encapsulado em um ou mais **Aspectos**, que são mecanismos disponibilizados pela programação orientada a aspectos para agrupar fragmentos de código referentes a interesses não inerentes ao negócio, por sua vez, as aplicações geradas podem ser melhor modularizadas, facilitando sua manutenção, extensão e contribuindo para o reuso. Como forma de prover maior flexibilidade são utilizados **Aspectos Abstratos** (são entidades que encapsulam conceitos transversais, porém possuem algumas características abstratas postergando a concretização para os aspectos que o estenderem, possibilitando a posterior definição de aspectos que tratam de detalhes específicos de uma aplicação) de modo a prover um maior nível de abstração e permitir a outros aspectos a reutilização de código, além do fato de aspectos

abstratos poderem ser estendidos através da adição de novos comportamentos. Dessa forma, os aspectos ajudam a tratar detalhes dinâmicos das aplicações provendo a injenção de comportamento adaptativo quando há o acontecimento de um determinado evento.

Além disso, com o intuito de prover aplicações com um maior grau de adaptabilidade, pode ser utilizada em conjunto com o DSOA, a técnica de composição dinâmica, a qual consiste em permitir a integração e a modificação de características na aplicação em tempo de execução, modificando dinamicamente a forma como são providos os serviços. [CANAL; MURILLO; POIZAT, 2006]

Portanto, faz-se necessário o desenvolvimento de uma infra-estrutura para dar suporte à criação de aplicações adaptativas cientes de contextos modulares e preparadas para lidar com os processos de manutenção e evolução das aplicações, além de lidar com os recursos limitados dos dispositivos inseridos em um ambiente altamente dinâmico, além disso problemas relacionados à falta de modelos e metodologias para o projeto de aplicações ubíquas também são enfrentados, portanto tal infra-estrutura deve também tratar detalhes relativos a todo processo de concepção de aplicações adaptativas cientes de contexto.

1.2. Objetivos

Este trabalho tem como principal objetivo propor o PACCA (**P**rojeto de **A**plicações **C**ientes ao **C**ontexto e **A**daptativas) [SANTOS et al., 2008, SANTOS et al., 2009 (*to appear*)], uma infra-estrutura para dar suporte à construção e execução de aplicações adaptativas cientes de contexto em ambientes de Computação Ubíqua, bem como um processo de desenvolvimento a ser utilizado para a construção de aplicações que executam na infra-estrutura proposta. Propõe-se a adoção de uma abordagem de desenvolvimento baseada no uso de composição dinâmica de software e de orientação a aspectos a fim de prover maior modularidade e flexibilidade para as aplicações desenvolvidas. No PACCA, as funcionalidades relativas à ciência de contexto são realizadas por um *middleware* para provisão de contexto [LE SOMMER; GUIDEC; ROUSSAIN, 2006, SACRAMENTO et al., 2004]. Assim, funções como: a aquisição (sensoriamento), o processamento e o armazenamento de informações contextuais, bem como serviços para: suporte à coordenação, notificação e composição de eventos, são providos pelo *middleware*.

O enfoque deste trabalho é apresentar a arquitetura do PACCA e os serviços providos pelos seus componentes, bem como o seu funcionamento e o processo de desenvolvimento de aplicações proposto. O principal componente do PACCA é um módulo de adaptação, o qual foi desenvolvido usando o paradigma DSOA, e que encapsula os requisitos ligados à adaptação ciente de contexto. A partir de informação contextual obtida dos componentes do *middleware* de provisão de contexto e de políticas de adaptação previamente definidas, o módulo de adaptação gera uma especificação de composição da aplicação. Ou seja, são geradas informações para a configuração arquitetural (conjunto de componentes e conexões entre os mesmos) a ser adotada na execução da aplicação. Em uma etapa posterior, realiza-se a composição dos blocos funcionais ligados à lógica da aplicação (componentes base) e dos blocos representando os aspectos ligados à adaptação e à manipulação de contexto, a fim de gerar uma aplicação plenamente executável, adaptativa e ciente de contexto. No PACCA, os requisitos de adaptação são especificados como um conjunto de aspectos genéricos, potencialmente reutilizáveis, identificados como sendo comuns a diversos tipos de aplicações para a Computação Ubíqua. Dessa forma, obtêm-se um alto nível de modularização e reuso da solução gerada.

1.3. Estrutura do Documento

Esta dissertação está estruturada em sete capítulos, na seguinte disposição. O Capítulo 2 apresenta os conceitos básicos, os quais foram utilizados no desenvolvimento do trabalho como: orientação a aspectos e as tecnologias utilizadas no desenvolvimento de aplicações orientadas a aspectos, sistemas de *middleware* de provisão de contexto e adaptação de software.

O Capítulo 3 apresenta o PACCA através da descrição da arquitetura e dos módulos que compõem esta arquitetura, do processo de desenvolvimento para a concepção de aplicações adaptativas cientes de contexto utilizado com o PACCA e dos detalhes de implementação do PACCA.

O Capítulo 4 apresenta o estudo de caso desenvolvido para a validação da proposta do PACCA, onde são descritas as características do cenário proposto, além de demonstrados todos os passos do processo de desenvolvimento da aplicação utilizando o PACCA.

O Capítulo 5 expõe a avaliação do PACCA através da apresentação dos resultados obtidos com a sua utilização no estudo de caso proposto comparados com

uma abordagem de desenvolvimento puramente orientada a objetos. O Capítulo 6 cita os trabalhos relacionados, descrevendo as suas semelhanças e diferenças com o PACCA. Por fim, o Capítulo 7 apresenta as conclusões e os trabalhos futuros.

2. Conceitos Básicos

Os conceitos e técnicas da Programação Orientada a Aspectos (POA) [KICZALES et al., 1997] foram usados para a construção do PACCA e a abordagem de Desenvolvimento de Software Orientado a Aspectos (DSOA) foi adotada como base para o processo de desenvolvimento a ser seguindo na construção de aplicações que executam sobre ele. Portanto, a Seção 2.1 apresenta os conceitos básicos do paradigma de orientação a aspectos, apresenta algumas das tecnologias de suporte ao paradigma e ainda descreve algumas das métricas mais comumente usadas para a avaliação de sistemas construídos segundo a abordagem DSOA.

Com relação à manipulação de informação contextual, o PACCA obtém informações providas por um *middleware* de provisão de contexto denominado MOCA [SACRAMENTO et al., 2004]. Portanto, a Seção 2.2 fornece uma visão geral de sistemas de *middleware* de provisão contexto e detalha o MOCA. A Seção 2.3 apresenta conceitos básicos para a compreensão de Adaptação de Software.

2.1. Orientação a Aspectos

O desenvolvimento de software orientado a aspectos (DSOA) visa aumentar a modularidade dos sistemas complementando o paradigma de Orientação a Objetos, tratando os interesses que não fazem parte dos propósitos básicos do sistema e tornando a estrutura do software mais concisa e, consequemente, conferindo maior facilidade de manutenção [KICZALES et al., 1997]. Tais interesses, denominados de interesses transversais (*crosscutting concerns*), em geral estão entrelaçados e espalhados nos elementos que representam os interesses básicos do sistema.

A Orientação a Aspectos estende outras técnicas como a Orientação a Objetos, propondo não apenas uma decomposição funcional, mas sistêmica do problema. Isso permite que a implementação de um sistemas seja separada em requisitos funcionais e não-funcionais, disponibilizando a abstração de aspectos para a decomposição de interesses não contemplados pelos recursos oferecidos pelas linguagens de componentes. [WINCK; GOETTEN JUNIOR, 2006]

Dessa forma, o desenvolvimento de software orientado a aspectos busca uma maior separação de interesses (*separation of concerns*) entre os módulos do sistema de software e, como conseqüência, busca minimizar a replicação de código e reduzir o acoplamento entre os módulos. Esses fatores somados aumentam o potencial de reuso e facilitam a evolução de sistemas complexos [FIGUEIREDO et al., 2005].

A fim de se obter os melhores resultados com o uso de DSOA, as aplicações devem ser modeladas seguindo alguns passos. A área de pesquisa conhecida como Modelagem Orientada a Aspectos (MOA) trata da definição de métodos, técnicas, artefatos e processos de modelagem orientados a aspectos. Há diversas propostas de trabalhos nessa área [BANIASSAD; CLARKE, 2004a, GRUNDY, 1999, SUTTON; ROUVELLOU, 2004], cada qual com uma abordagem distinta e produzindo um conjunto diferente de artefatos de modelagem. Porém, podem-se destacar os seguintes passos básicos necessários para modelar um sistema segundo a abordagem orientada a aspectos [SCHAUERHUBER; RETSCHITZEGGER, 2006]:

(i) Identificação dos interesses (*concerns*), os quais são elementos cruciais ao domínio da aplicação e incluem os requisitos dos *stakeholders*.

(ii) Definição do Modelo Base, onde se encontrarão os interesses que forem identificados como não-transversais, e que podem ser modelados utilizando as técnicas convencionais de orientação a objetos (OO).

(iii) Definição do Modelo de Aspectos, onde serão definidos os interesses transversais, ou seja, conceitos que se encontram dispersos por diferentes funcionalidades da aplicação e não podem ser adequadamente modelados usando técnicas OO.

(iv) Definições das regras de composição, que irão definir onde e como os interesses transversais (aspectos) irão atuar sobre os elementos do Modelo Base, e caso seja necessário definir precedência entre aspectos que interceptem um mesmo elemento.

(v) Identificação e resolução de conflitos entre aspectos.

A partir da modelagem, a programação orientada a aspectos (POA, ou do inglês, AOP - *Aspect-Oriented Programming*) introduz sobre o paradigma de orientação a objetos (POO, ou do inglês, OOP - *Object-Oriented Programming*) um conjunto de novas abstrações e mecanismos para facilitar o desenvolvimento de software. Assim, a POA complementa a POO por tratar uma nova dimensão para decomposição de responsabilidades no escopo do programa, que modularizam os conceitos transversais: os *aspectos*.

A implementação de aplicações utilizando POA emprega os seguintes elementos:

(i) Linguagem de componentes - linguagem de programação usada pelo programador para escrever programas que implementam as funcionalidades básicas do sistema (lógica do negócio), ou seja, que implementam o Modelo Base. O código que

implementa a lógica do negócio é conhecido como *código base* e exemplos dessas linguagens são Java, C#, C++, etc;

(ii) Linguagem de aspectos - fornece construções básicas para o programador criar as estruturas necessárias para descrever o comportamento dos aspectos (ou seja, implementar o Modelo de Aspectos) e definir os critérios ou situações que estabelecem o ponto em que um aspecto será executado. Por exemplo, o programador pode definir no aspecto critérios que possibilitem a interceptação de um método ou construtor de uma classe. Exemplos de linguagens de aspectos são AspectJ [KICZALES et al., 2001, ASPECTJ, 2008], PHPAspect [CANDILLON; VANWORMHOUDT, 2007], AspectLua [FERNANDES; BATISTA, 2004], AspectC++ [SPINCZYK; LOHMANN; URBAN, 2005], etc.

(iii) Compositor de aspectos (*aspect weaver*) - responsável por compor os programas escritos em linguagem de componentes (ou seja, o código base) com os programas escritos em linguagem de aspectos. O processo encarregado de possibilitar ao aspecto interceptar o código base é conhecido como *weaving*, e pode ser estático ou dinâmico. No *weaving* estático, o código a ser injetado no código base é carregado em tempo de compilação. Por outro lado, no *weaving* dinâmico, tal código pode ser adicionado e removido em tempo de execução. [JBOSS AOP, 2008, WINCK; GOETTEN JUNIOR, 2006].

Segundo os conceitos de orientação a aspectos, a composição de aspectos com o código base é feita através da definição de **pontos de junção** (*join point*). Pontos de junção representam pontos bem definidos na execução de um programa onde um determinado aspecto pode ser aplicado como, por exemplo, a chamada ou execução de um método, o acesso a um atributo, a ocorrência de uma exceção, entre outros. Os pontos de junção são utilizados na criação das regras que darão origem aos pontos de atuação. **Pontos de atuação** (*pointcuts*) são expressões da linguagem para definir os pontos de junção, ou seja, o ponto de atuação torna possível a inserção de comportamento no ponto de junção. Pontos de atuação podem expor determinados valores no contexto de execução de cada ponto de junção para serem utilizados pelos adendos. **Adendos** (*advices*) são constituídos de código a ser injetado na aplicação quando acontece um determinado evento, ou seja, o código definido no *advice* é executado quando o ponto de junção correspondente ao ponto de atuação for capturado. Um aspecto pode conter métodos além das definições citadas acima, sendo assim, um

aspecto deve ser declarado como abstrato sempre que possuir métodos ou *pointcuts* abstratos.

Outros elementos presentes na POA são as **declarações intertipos** (*intertype declarations)*. Declarações intertipos são declarações referentes à estrutura de um programa, por exemplo, um aspecto poderia ser utilizado para adicionar novos métodos e atributos a uma classe, ou declarar que uma classe estende uma nova superclasse. [WINCK; GOETTEN JUNIOR, 2006]

2.1.1. Tecnologias de Suporte à Programação Orientada a Aspectos

Nesta Seção serão apresentadas as características de duas das mais usadas tecnologias para o desenvolvimento de software orientado a aspectos: AspectJ e JBoss AOP.

2.1.1.1. AspectJ

AspectJ é uma extensão para a linguagem Java, incorporando à linguagem uma nova unidade de abstração, o aspecto, além dos demais elementos que caracterizam o paradigma orientado a aspectos, como por exemplo, os *advices, pointcuts e join points.*

No projeto de AspectJ foram levados em consideração alguns aspectos importantes em relação à compatibilidade com a linguagem Java. AspectJ apresenta compatibilidade total com Java, ou seja, todo programa Java puro é um programa AspectJ válido, e todo programa AspectJ pode ser executado em uma JVM (*Java Virtual Machine*). [KICZALES et al., 2001]

Em AspectJ, os aspectos são definidos por declarações similares às declarações de uma classe. As declarações de aspectos podem incluir pontos de atuação, adendos, além de outros tipos de declarações permitidas nas declarações de classes, como métodos e atributos. Ainda é possível utilizar os conceitos de herança presentes na orientação a objetos (permitindo a hierarquização entre aspectos) e o conceito de aspectos abstratos de modo a aumentar o nível de abstração. [KICZALES et al., 2001]

Através das Figuras 1, 2 e 3 será apresentado um exemplo simples de AspectJ, onde primeiramente declara-se uma classe *HelloWorld*, a qual contém o método *hello()*, que recebe uma *String* como parâmetro e imprime essa *String*. Em seguida, na Figura 2 declara-se um aspecto em AspectJ, onde na linha 2, é definido um *pointcut* para capturar a execução do *join point* determinado (o método *hello* da classe *HelloWorld*). Na linha

3, declara-se o *advice* (o *advice* contém o código a ser injetado na aplicação quando o *pointcut* é capturado), onde através da palavra-chave *before* é indicado que o código definido no *advice* será executado antes do *join point.* Por outro lado, o uso do *after* no lugar de *before,* faria com que o advice fosse executado após o *join point.* Por fim, tem-se a classe *Main,* responsável por iniciar a execução da aplicação, que terá como saída a *String* "Olá Isanio".

```
1.  public class HelloWorld{
2.      public static void hello(String name){
3.          System.out.println(name);
4.      }
5.  }
```

Figura 1 – Declaração da classe HelloWorld

```
1.  public aspect AspectHello{
2.      pointcut helloWorld(): execution (* HelloWorld.hello(..));
3.      before () : helloWorld(){
4.          System.out.print("Olá");
5.      }
6.  }
```

Figura 2 – Declaração de um aspecto em AspectJ

```
1.  public class Main{
2.      public static void main(String args[]){
3.          HelloWorld.hello("Isanio");
4.      }
5.  }
```

Figura 3 – Classe que faz a invocação do método hello da classe HelloWorld

Para que o aspecto intercepte o código base é necessário que seja realizado o processo de combinação aspectual, o qual é um processo que antecede a compilação, gerando código intermediário na linguagem de componentes capaz de produzir a operação desejada, ou de permitir a sua realização durante a execução. As classes referentes ao código base não sofrem alterações para suportar POA, isso é feito no momento da combinação entre os componentes e os aspectos, conforme apresentado na Figura 4. Dessa forma, o combinador utiliza os aspectos e os componentes para gerar um novo código. [WINCK; GOETTEN JUNIOR, 2006]

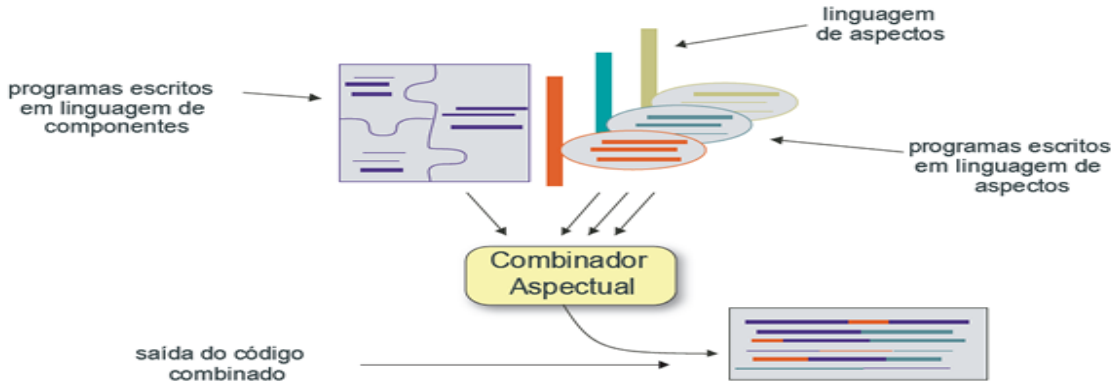

Figura 4 – Composição de um sistema orientado a aspectos [WINCK; GOETTEN JUNIOR, 2006]

A partir da versão 5 do AspectJ é suportado o *weaving* dinâmico e uso de reflexão, que pode ser definida como a capacidade que um processo computacional tem de manipular formalmente suas próprias operações e estruturas. [SMITH, 1982] A reflexão possui algumas peculiaridades possibilitando ao desenvolvedor observar o estado interno de um programa processo denominado de introspecção e alterar esse estado.

O uso de reflexão permite às aplicações muito mais flexibilidade e adaptabilidade permitindo que a mesma manipule suas estruturas internas de modo a adaptar-se dinamicamente de acordo com as suas necessidades e também pode ser utilizado em um mecanismo de tolerância a falhas. Ao manipular informações internas do estado de execução de um programa, tais informações podem ser transformadas em dados disponíveis para o programa em tempo de execução, sendo este processo chamado de reificação (*reification*). A reflexão computacional aplicada a um programa pode ser comportamental ou estrutural. A reflexão estrutural em uma aplicação orientada a objetos permite a modificação na classe refletida, por exemplo, criação de métodos e atributos, por sua vez, a reflexão comportamental pode modificar como uma classe reage ao recebimento de uma mensagem. [SILVEIRA, 2001]

Em AspectJ, o uso de reflexão é possível através da variável especial *thisJoinPoint*, a qual fornece informações sobre o *join point* corrente. [ASPECTJ, 2008]

2.1.1.2. JBoss AOP

O JBoss AOP [JBOSS AOP, 2008] é um arcabouço (*framework*) desenvolvido em Java que possibilita o desenvolvimento de aplicações orientadas a aspectos utilizando Java, porém não é uma extensão da linguagem, como o AspectJ. O arcabouço

pode ser tanto utilizado através do servidor de aplicações JBoss AS (*Application Server)* como através de sua versão *standalone* [JBOSS USER GUIDE, 2008].

No JBoss AOP, os adendos de AspectJ ganham um novo nome: interceptadores. Os interceptadores podem ser definidos nas classes de interceptadores ou nas classes de aspectos. As classes de interceptadores são limitadas, pois permitem definir apenas um interceptador por classe. Para suprir tal limitação, o JBoss AOP propõe a idéia de classe de aspecto. A classe de aspecto é uma classe Java que agrupa diversos métodos de interceptação, ou seja, têm-se diversos interceptadores definidos em uma mesma classe de aspecto. Essas duas formas de codificação dos interceptadores definem o código que deve ser executado antes e depois dos pontos de junção.

Os pontos de atuação no JBoss AOP podem ser definidos em um arquivo XML [XML, 2008] ("jboss-aop.xml') ou com anotações adicionadas às classes de aspectos e de interceptadores. No arquivo XML, a definição dos pontos de atuação fica separada das classes de interceptadores e de aspectos. Conseqüentemente, essas classes são independentes de qualquer ponto de atuação e podem ser reusadas mais facilmente. Com anotações, os pontos de atuação e os aspectos são definidos na mesma localização. A vantagem dessa situação é que ela permite um melhor entendimento (pois as anotações são definidas próximas às classes de aspectos) e uma programação mais simples. [JBOSS USER GUIDE, 2008]

Ademais, o JBoss AOP fornece o suporte à combinação dinâmica de aspectos (*weaving dinâmico*). Em outras palavras, os aspectos podem ser combinados em tempo de execução do programa, sem a necessidade de parar ou de recompilar a aplicação.

Para ilustrar a explanação sobre o JBoss AOP, será apresentado o mesmo exemplo utilizado na seção sobre AspectJ. Assim, os códigos exibidos nas Figuras 1 e 3 são válidos também com JBoss AOP. A diferença está na declaração do aspecto, conforme citado anteriormente. Na Figura 5 é declarada uma classe Java que possui um método interceptador definido na linha 2, o qual representa o *advice*. O código exibido na linha 4, antes do *invocation.invokeNext()* contido na linha 5 tem o mesmo resultado que o uso do *before* em AspectJ. Caso fosse necessário um resultado igual ao produzido pelo *after* de AspectJ, o código contido na linha 4 pode ser introduzido em uma cláusula *finally*, finalizando o bloco *try-catch.*

A Figura 6 apresenta um exemplo do arquivo de configuração jboss-aop.xml, onde verifica-se na linha 3, a indicação de que a classe *AspectHello* será um aspecto e na linha 4, a definição do *pointcut* para capturar o *join point* determinado em *expr*. Entre

as linhas 5 e 7, verifica-se a associação que determina qual o *advice* a ser executado quando houver a captura do *pointcut* indicado.

```
1.      public class AspectHello {
2.              public Object helloWorld(Invocation invocation){
3.                      try{
4.                              System.out.print("Olá ");
5.                              return invocation.invokeNext();
6.                      }catch (Throwable e){
7.                              e.printStackTrace();
8.                              return null;
9.                      }
10.             }
11.      }
```

Figura 5 – Declaração de um aspecto com JBoss AOP

```
1.  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2.  <aop>
3.      <aspect class="hello.AspectHello" scope="PER_VM"/>
4.      <pointcut name="teste" expr="execution(public * *.*->hello(..))"/>
5.      <bind pointcut="teste">
6.                      <advice name="helloWorld" aspect="hello.AspectHello"/>
7.      </bind>
8.  </aop>
```

Figura 6 – Exemplo do arquivo de configuração jboss-aop.xml

2.1.1.3. Métricas de Avaliação para AOP

Nesta Seção, serão apresentadas algumas métricas descritas em [MUNNELLY; FRITSCH; CLARKE, 2007], as quais são utilizadas para a avaliação de aplicações orientadas a aspectos. Essas métricas podem ser avaliadas com o uso de ferramentas, dentre as quais podemos citar a AOP Metrics [AOP METRICS, 2008].

(i) LOC (Linhas de Código - *Lines of Code*) – indica a quantidade de linhas de código da aplicação. Essa métrica ressalta que a retirada dos conceitos transversais do código base e sua concentração nos aspectos resulta na redução da quantidade de linhas de código escritas pelo desenvolvedor, através do reuso do código contido nos aspectos.

(ii) WOM (Peso das Operações em um Módulo - *Weighted Operations in Module*) – essa métrica mede o número de operações em um dado módulo. Ela captura a complexidade interna de um módulo em termos de

suas funcionalidades implementadas. Um grande número de operações em um módulo indica que o desenvolvimento e a manutenção deste módulo torna-se mais complexa e exige mais tempo, além de que módulos com um grande número de operações são mais específicos da aplicação, limitando o reuso.

(iii) CAE (Acoplamento na Execução de *Advices* - *Coupling on Advice Execution*) - essa métrica indica o número de aspectos contendo *advices* disparados pela execução de operações em um dado módulo. Se o comportamento de uma operação pode ser alterado por um *advice*, devido à interceptação do *pointcut*, existe uma dependência entre a operação e o *advice*. Tal acoplamento não existe em sistemas orientados a objetos.

(iv) CMC (Acoplamento na Invocação de Métodos - *Coupling on Method Call*) - essa métrica indica o número de módulos ou interfaces declarando métodos que são possivelmente chamados por um dado módulo. O uso de uma grande quantidade de métodos de vários módulos diferentes indica que a funcionalidade de um dado módulo não pode ser facilmente isolada dos outros e isso implica um alto acoplamento.

(v) CFA (Acoplamento no Acesso a Atributos - *Coupling on Field Access*) – essa métrica apresenta a dependência em termos de atributos. Em sistemas orientados a objetos, o valor de CFA é normalmente 0 (zero) devido ao encapsulamento. Por outro lado, em sistemas orientados a aspectos, os aspectos, através de seu comportamento intrusivo, podem depender de um determinado campo para realizar uma operação, provocando acoplamento a um determinado campo.

(vi) CBM (Acoplamento entre Módulos - *Coupling Between Modules*) - essa métrica indica o acoplamento entre os módulos, ou seja, a dependência entre os mesmos, e seu valor denota a dependência em relação a um determinado número de módulos. A presença de muitos acoplamentos entre os módulos reduz a possibilidade de reuso e dificulta a modificação da aplicação, tornando mais trabalhoso o processo de evolução do software.

(vii) RFM (Resposta de um Módulo - *Response for Module*) – essa métrica RFM refere-se à resposta de um módulo, e indica a possibilidade de

comunicação entre os módulos, ou seja, métodos e *advices* geralmente executam em resposta a uma mensagem recebida por um dado módulo. Essa métrica em sistemas orientados a aspectos deve levar em consideração as responsabilidades implícitas que são disparadas quando um *pointcut* intercepta uma operação de um dado módulo.

(viii) LCO (Falta de Coesão entre as Operações - *Lack of Cohesion in Operations*) - essa métrica representa a falta de coesão entre as operações dos módulos. A coesão é um forte indicador de boa modularidade, visto que em uma abordagem OO envolvendo conceitos transversais, os módulos acabam desempenhando tarefas sobre as quais não têm responsabilidade, devido aos interesses transversais estarem espalhados pelo código, fazendo com que as operações dos módulos tratem os mais diferentes interesses tornando o módulo menos coeso.

2.2. Middleware de Provisão de Contexto

Um dos requisitos das aplicações para Computação Ubíqua é a capacidade de adaptarem-se dinamicamente com base em um conjunto de fatores, incluindo o estado do ambiente, limitações dos dispositivos onde executam, mobilidade e outras características relativas ao usuário. Tais informações caracterizam o contexto [DEY, 2001], que pode ser definido como qualquer informação que pode ser usada para caracterizar a situação de uma entidade, onde entidade pode ser uma pessoa, um local, ou um objeto que é considerado relevante para a interação entre o usuário e a aplicação, incluindo os próprios usuário e aplicação.

Dessa forma, para o desenvolvimento de aplicações adaptativas cientes de contexto é necessário que haja uma infra-estrutura de serviços para a manipulação de contexto. Ela deve prover funcionalidades tais como monitoração e armazenamento de informações contextuais, além de suporte à coordenação e notificação de eventos contextuais.

Existem atualmente várias plataformas de *middleware* de provisão de contexto, com diferentes características, tais como: MOCA (*Mobile Collaboration Architecture*) [SACRAMENTO et al., 2004], *Java Context-Aware Framework* (JCAF) [BARDRAM, 2005], *Context Toolkit* [SALBER; DEY; ABOWD, 1999], dentre outros. No entanto, neste trabalho, o *middleware* de provisão de contexto MOCA será apresentado com

mais detalhes. Ele foi o escolhido para atuar em conjunto com a implementação atual do PACCA, devido à disponibilidade da ferramenta e de sua API, documentação e por ser bem aceito no auxílio ao desenvolvimento de aplicações ubíquas, sendo responsável por fornecer as informações contextuais necessárias à aplicação.

O MOCA é um *middleware* que provê suporte ao desenvolvimento e à distribuição de aplicações distribuídas cientes de contexto. O MOCA fornece meios de coletar, armazenar e processar dados relativos ao contexto de dispositivos móveis. Essencialmente, o MOCA consiste em um conjunto de API's (*Application Program Interface*) e um conjunto de serviços capazes de suportar a execução de aplicações sensíveis ao contexto. A Figura 7 mostra a arquitetura do MOCA e o relaciomento entre os seus serviços.

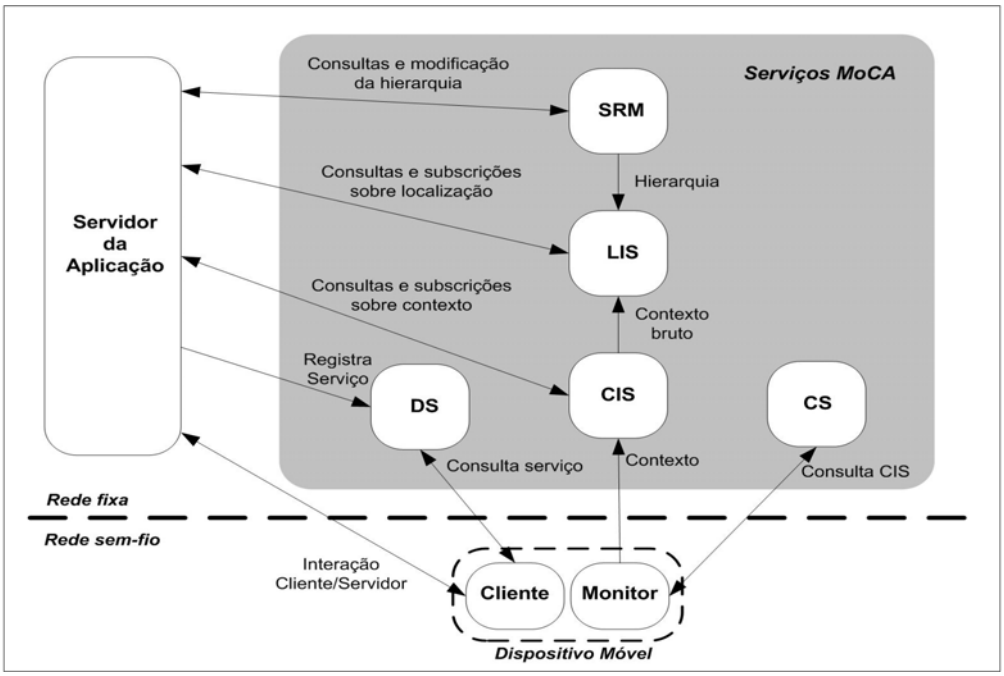

Figura 7 – Arquitetura do MOCA

O DS (*Discovery Service*) é um serviço de descoberta de outros serviços disponibilizados por um servidor de aplicação. Esse servidor registra os serviços no DS para posterior localização e utilização pelos clientes móveis.

O CS (*Configuration Service*) é um serviço de configuração com a responsabilidade de gerenciar as informações de configuração para que os dispositivos móveis possam utilizar o CIS.

O CIS (*Context Information Service*) é o serviço encarregado de notificar mudanças de contexto em um dispositivo móvel, recebendo informações sobre o

dispositivo e seu ambiente de execução, através de um componente denominado *Monitor* contido no dispositivo. O *Monitor* encarrega-se de repassar periodicamente para o CIS as informações sobre o contexto de execução do dispositivo. O intervalo de periodicidade temporal com o qual o Monitor repassa as informações para o CIS é um parâmetro de configuração.

No MOCA são modelados e implementados dois tipos de informação de contexto: o contexto local do dispositivo (nível de bateria, de memória, uso de CPU) e o contexto de conectividade da rede, que inclui todos os pontos de acesso ao alcance de um dispositivo e as correspondentes intensidades de sinais recebidas [ROCHA; CASANOVA; ENDLER, 2007]. Atualmente, as informações de contexto manipuladas pelo MOCA são descritas em um arquivo XML usando um modelo *par-valor*.

O LIS (*Location Inference Service*) é o serviço do MOCA responsável pela inferência da localização lógica aproximada. A posição aproximada de um dispositivo pode ser inferida através de comparações de sinais de radiofreqüência recebidos em relação a sinais medidos previamente em pontos de referência. Para auxiliar na inferência da localização podem ser definidas regiões simbólicas, que consistem em conjuntos hierárquicos de regiões definidos pelo desenvolvedor de acordo com o interesse da aplicação para designar regiões físicas como salas, prédios, etc. Para a definição das regiões simbólicas é utilizado o *Symbolic Region Manager* (SRM).

2.3. Adaptação de Software

Como mencionado anteriormente, aplicações para a Computação Ubíqua executam em um ambiente altamente dinâmico e devem estar preparadas para reagir a diferentes situações e contextos. As constantes mudanças ambientais e as decisões de reconfiguração afetam diferentes partes de uma aplicação impondo novos desafios para a construção das aplicações.

Segundo [CÁMARA; SALAUN; CANAL, 2007] a adaptação de software pode ser vista como uma disciplina caracterizada pela modificação ou extensão do comportamento dos componentes através do uso de componentes especiais chamados adaptadores (*adaptors*), os quais são responsáveis por modificar a aplicação e a forma como a mesma provê seus serviços.

Ainda em relação à adaptação de software existem também diferentes tipos de adaptação com relação ao momento em que a mesma ocorre, considerando as diversas

fases do ciclo de vida do software. Em [BULCÃO NETO; TEIXEIRA; PIMENTEL, 2005] são identificadas duas categorias, descritas abaixo:

(i) Adaptação Estática ou em Tempo de Projeto: inclui toda a adaptação feita antes do sistema estar executando. Pode se referir a adaptação tanto dos requisitos (modelos) quando de partes de código já desenvolvidas. A primeira é necessária quando a especificação de um sistema tem que ser estendida para atender novos requisitos, ou para modificar os requisitos antigos. Uma característica comum a todos os exemplos de adaptação estática é que os passos a serem realizados com a adaptação são conhecidos – foram planejados – antes do momento em que a adaptação de fato ocorre.

(ii) Adaptação Dinâmica ou em Tempo de Execução: esse é o caso quando partes de software já executando precisam ser adaptadas a fim de mudar o modo com o qual um serviço é fornecido. Aqui, os componentes a serem adaptados, assim como os passos para gerenciar a adaptação podem ser desconhecidos antes do instante em que a adaptação ocorre ou ser podem planejados para ocorrer em tempo de execução através da adoção de políticas que norteiam o processo de adaptação. Tal processo pode ser viabilizado através da composição dinâmica, fazendo com que os módulos carregados dinamicamente interajam de forma correta adaptando a forma como o serviço é provido.

3. PACCA (Projeto de Aplicações Cientes ao Contexto e Adaptativas)

Neste trabalho é proposta uma infra-estrutura para o desenvolvimento de aplicações ubíquas denominada PACCA (**P**rojeto de **A**plicações **C**ientes ao **C**ontexto e **A**daptativas) [SANTOS et al., 2008,SANTOS et al., 2009 (*to appear*)]. Na proposta são utilizadas as técnicas de desenvolvimento de software orientado a aspectos e composição dinâmica de software, a fim de prover comportamento adaptativo ciente ao contexto. Tal proposta tem como objetivo reduzir o grau de acoplamento entre as soluções de adaptação e a lógica de negócio das aplicações. A abordagem de adaptação proposta pode ser vista como sendo dinâmica, haja vista que o comportamento adaptativo é realizado durante a execução da aplicação, com base em políticas de adaptação previamente definidas. As estratégias de adaptação utilizadas no PACCA consistem em especificar o conjunto de módulos da aplicação a serem dinamicamente carregados e executados, a partir de regras pré-estabelecidas e em função dos diferentes contextos, os quais são monitorados por um *middleware* de provisão de contexto tais como: MOCA (*Mobile Collaboration Architecture*), *Java Context-Aware Framework* (JCAF) [BARDRAM, 2005], *Context Toolkit* [SALBER; DEY; ABOWD, 1999], etc. Tais estratégias, em vez de estarem espalhadas e entrelaçadas com o código de negócios da aplicação, encontram-se encapsuladas em **Aspectos**. Portanto, na presente proposta, a adaptação contextual da aplicação é caracterizada como um comportamento transversal e tratada como um ou mais aspectos.

Para a construção do PACCA e sua integração com um *middleware* de provisão de contexto, adotou-se uma abordagem na qual tanto os serviços de adaptação e ciência de contexto, quanto as aplicações a serem executadas, consistem em conjuntos de módulos de software (classes) interligados.

Este Capítulo está estruturado da seguinte forma: a Seção 3.1 discute as motivações existentes para se considerar a Adaptação Contextual como um Aspecto; a Seção 3.2 descreve a Arquitetura do PACCA, detalhando seus componentes estruturais; a Seção 3.3 detalha o Processo de Desenvolvimento a ser seguido pelas aplicações construídas para executar na infra-estrutura provida pelo PACCA e a Seção 3.4 provê os detalhes de implementação.

3.1. Adaptação Contextual como um Aspecto

Segundo [LOUGHRAN et al., 2006], em qualquer ambiente há uma grande quantidade de informações de contexto, as quais os seres humanos estão aptos a interpretar e empregar para enriquecer o seu entendimento de eventos correntes. No entanto, as habilidades de captura e compreensão do contexto não estão presentes nos computadores. As abordagens computacionais tradicionais usam informações contextuais pré-definidas providas pelo usuário, que são interpretadas de forma *hard-coded*, deixando a aplicação com uma visão empobrecida do ambiente, limitando sua habilidade para se adaptar e conseqüentemente sua utilidade.

O uso do contexto é um importante campo de investigação no âmbito do paradigma da Computação Ubíqua. Nesse novo paradigma, os desenvolvedores de aplicações têm passado a considerar em seus projetos o contexto do usuário, assim como o contexto do dispositivo, com o objetivo de aumentar a capacidade de adaptação sempre que houver mudanças no ambiente de execução.

Conforme mencionado na Seção 2.2, o termo *contexto* pode ser definido como "qualquer informação que possa ser usada para caracterizar a situação de uma entidade, podendo ser uma pessoa, um lugar e/ou objeto físico ou computacional". A captura e o uso de informações contextuais aumentam a utilidade das aplicações, permitindo que as mesmas se adaptem de acordo com as mudanças em seu ambiente de execução. Porém, os desenvolvedores de aplicações adaptativas cientes de contexto enfrentam problemas relacionados com a falta de modelos e metodologias para o projeto de suas aplicações e com a variedade de informações de contexto.

Com relação ao projeto de aplicações adaptativas cientes de contexto, um dos fatores complicadores é que, em geral, o código de adaptação, que implementa o comportamento adaptativo, encontra-se entrelaçado com o código funcional da aplicação, que por sua vez implementa o comportamento imperativo ligado a lógica do negócio. O código da adaptação, portanto, espalha-se por diversos componentes do sistema, cruzando suas fronteiras, e caracterizando um comportamento transversal (*crosscuting behavior*). O código da Figura 8 ilustra esse comportamento transversal. Tal código representa a lógica de negócio relativa a uma funcionalidade de Cadastro de Consulta Médica, a qual é executada quando o usuário clicar em certo botão na interface gráfica da aplicação. Na Figura 8, o código compreendido entre as linhas 8 e 27 representa um comportamento adaptativo, caracterizado por consultas ao contexto

corrente e a tomada de decisões com base no contexto (definidas nos blocos *if-else)*. As linhas mencionadas ainda demonstram a forma *hard-coded* como é geralmente implementada uma aplicação adaptativa ciente de contexto. Esse mesmo código que promove a adaptação encontra-se em outros componentes da aplicação, caracterizando a adaptação como um interesse transversal (*crosscutting concern*).

```
1  public void cadastrarButtonCadastroConsultaMouseClicked()
2  {
3      loginMedico = loginMedicoTextFieldCadastroConsulta.getText();
4      loginPaciente = loginPacienteTextFieldCadastroConsulta.getText();
5      diagnostico = diagnosticoConsultaTextAreaCadastroConsulta.getText();
6      data = dataTextFieldCadastroConsulta.getText();
7
8      //Com Rede usar Base de Dados Central
9      if(GerenteContexto.isOnLine())
10     {
11         //Ambiente Seguro
12         if(GerenteContexto.isCriptografia()==false)
13         {
14             //Código de conexão com a Base de Dados Central e realização do cadastro de consulta médica
15         }
16         //Ambiente Inseguro - Usar Criptografia
17         else
18         {
19             //Código de Criptografia dos dados e envio para o lado servidor do Módulo de Criptografia
20         }
21     }
22     //Sem Rede usar Base de Dados Local
23     else
24     {
25         //Código de conexão com a Base de Dados Local e realização do cadastro de consulta médica
26     }
27 }
```

Figura 8 - Trecho de código da aplicação entrelaçado com código de adaptação

```
1  public void cadastrarButtonCadastroConsultaMouseClicked()
2  {
3      loginMedico = loginMedicoTextFieldCadastroConsulta.getText();
4      loginPaciente = loginPacienteTextFieldCadastroConsulta.getText();
5      diagnostico = diagnosticoConsultaTextAreaCadastroConsulta.getText();
6      data = dataTextFieldCadastroConsulta.getText();
7
8  }
```

Figura 9 - Trecho de código da aplicação sem entrelaçamento com código de adaptação

Com a utilização do paradigma DSOA, o código referente à adaptação passa a ser removido do código implementando a lógica da aplicação, sendo encapsulado em um ou mais aspectos. A Figura 9, ilustra o mesmo código de negócio mostrado na Figura 8 sem a inserção do código adaptativo, tornando assim a aplicação livre de comportamento adaptativo transversal.

Portanto, modelar a adaptação contextual como um aspecto provê maior modularidade e separação de interesses, visto que o código base deixará de ter código de adaptação entrelaçado com a lógica de negócio e com o código das interfaces de usuário [DANTAS et al., 2004]. No processo de adaptação das aplicações, em especial das altamente dinâmicas, é necessário modificar um conjunto de pontos em seu código a

fim de realizar a adaptação. Conseqüentemente, se houver a necessidade de realizar qualquer alteração, tanto no comportamento adaptativo quanto nos locais afetados pela adaptação, isso exigirá do desenvolvedor um grande esforço para localizar todos os pontos no código base que serão afetados por tais alterações. Ao contrário, encapsulando a adaptação em um (ou mais) aspecto(s), o processo de alteração do comportamento adaptativo seria facilitado pela concentração de tal comportamento nos aspectos, e o processo de alteração dos locais afetados consistiria apenas na mudança das expressões dos *pointcuts*, facilitando assim a evolução e manutenção da aplicação.

Já com relação à grande variedade de informações de contexto, um método proposto em [LOUGHRAN et al., 2006] para tornar o espaço de contexto mais gerenciável consiste em dividir o espaço sucessivamente em subconjuntos, até chegar ao nível de granularidade desejado. Sob a perspectiva orientada a aspectos, pode-se selecionar um desses subconjuntos e modelar todo o código de ciência de contexto para tal subconjunto em um aspecto. Essa flexibilidade permite ao desenvolvedor escolher o nível de granularidade para cada aspecto, permitindo também agrupar em um único aspecto contextos similares e subdividir contextos diferentes em vários aspectos, aumentando a modularidade. Na Figura 10, pode ser visto um exemplo de subdivisão do espaço de contexto em vários subconjuntos. Porém, não é inteiramente possível subdividir o espaço de contexto em subconjuntos totalmente independentes, pois os subconjuntos podem compartilhar informações contextuais.

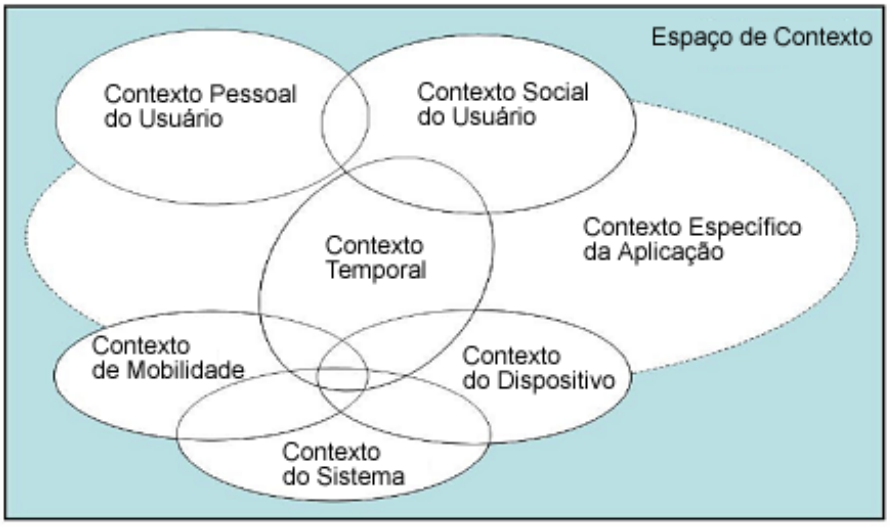

Figura 10 – Exemplo de subdivisão do espaço de contexto [LOUGHRAN et al., 2006]

Segundo [LOUGHRAN et al., 2006] o uso de orientação a aspectos para modularizar interesses contextuais pode introduzir algumas complicações quando se tem por objetivo criar um *framework* orientado a aspectos que contemple a adaptação com base nas mudanças de contexto. Isso ocorre, pois, para adaptar uma aplicação com base nas mudanças do contexto corrente, infere-se que a adaptação deve ser específica de um contexto e qualquer adaptação que é específica de um contexto, em geral, é específica da aplicação. Dessa forma, devem ser providos aspectos para tratar interesses mais gerais como a mobilidade, por exemplo, os quais devem ser mais comuns para a maioria dos sistemas adaptativos cientes de contexto e, a partir desse ponto, o desenvolvedor pode especializar tais aspectos de acordo com os detalhes de sua aplicação.

A seguir serão apresentados alguns interesses adaptativos identificados em [LOUGHRAN et al., 2006] que podem ser utilizados em aplicações cientes de contexto e estar encapsulados em aspectos.

(i) Contexto do usuário – está relacionado com todo o conhecimento pertencente ao usuário que é conhecido pelo sistema, o qual pode ser o nome, endereço, data de nascimento, além de informação relativa a seus papéis ou responsabilidades em cada aplicação. Essas responsabilidades dizem respeito às funcionalidades que o usuário pode realizar e definem seu *perfil*. Ainda, contexto do usuário inclui o nome de usuário (*login*), senha e questões que podem estar relacionadas à autenticação e autorização. Esse tipo de contexto permite que a aplicação possa adaptar-se como base nas características específicas de cada usuário.

(ii) Contexto do dispositivo – a variedade de dispositivos que o usuário pode utilizar para acessar uma aplicação ubíqua está crescendo rapidamente, exigindo que as aplicações estejam preparadas para lidar com esse tipo de informação contextual. Essas informações dizem respeito às diferentes interfaces de comunicação, diferentes tamanhos e configurações de tela, capacidades de processamento, armazenamento e fontes de energia dos dispositivos. O contexto nesse cenário também abrange as características das diferentes redes de acesso (em geral sem fio), as quais incluem, por exemplo, informações como largura de banda, atraso máximo, taxa de erros, dentre outras. Sendo assim, o comportamento da aplicação, incluindo o conteúdo a ser apresentado ao usuário, pode ser adaptado de acordo com as restrições impostas pelo dispositivo e pela rede em uso.

(iii) Contexto de localização – em ambientes distribuídos móveis, as aplicações necessitam dinamicamente obter informações e serviços que sejam relevantes para a localização corrente, pois a partir de determinada localização podem ser disparadas adaptações no comportamento da aplicação. Ainda em relação à localização pode-se considerar a proximidade entre dispositivos, a qual pode facilitar a comunicação entre diferentes componentes de um sistema e contribuir para a provisão de melhores serviços, como por exemplo, indicar a impressora mais próxima de um usuário conectado através de um dispositivo portátil.

(iv) Contexto do sistema – está associada à sua semântica arquitetural, isto é, ao contexto relativo aos componentes do sistema, como a sua configuração, suas interações e seu estado corrente. Uma aplicação pode reconfigurar-se automaticamente, como por exemplo, estar executando em modo cliente-servidor e posteriormente se adaptar para executar em modo *peer-to-peer.*

(v) Contexto temporal – pode-se identificar o contexto temporal dependente apenas de informação estática da aplicação, como por exemplo, o envio de um e-mail no dia do aniversário do usuário e outros subconjuntos de contexto que compreendem todos os dados dependentes do estado da aplicação em determinado momento, como por exemplo, uma adaptação ser iniciada no início ou ao término de uma ação específica do usuário.

(vi) Contexto específico da aplicação – incorpora contextos inerentes à aplicação, cujo uso está limitado ao escopo específico de cada aplicação.

(vii) Contexto do ambiente – representa informação relativa ao ambiente físico, como por exemplo, luminosidade e temperatura, que pode ser relevante para certos tipos de aplicações ubíquas, por exemplo, aplicações de Deteção de Reuniões [SCHULTZ et al., 2001].

Os interesses supracitados serviram de base para a definição dos interesses adaptativos tratados no presente trabalho, visto que no PACCA são identificados e tratados alguns comportamentos comuns às aplicações cientes de contexto adaptativas para o domínio da Computação Ubíqua.

3.2. Arquitetura

A Figura 11 apresenta o diagrama de componentes que representa a arquitetura do PACCA e sua integração com um *middleware* de provisão de contexto. Os componentes

do PACCA são responsáveis pela adaptação ciente de contexto e serão detalhados nas subseções a seguir.

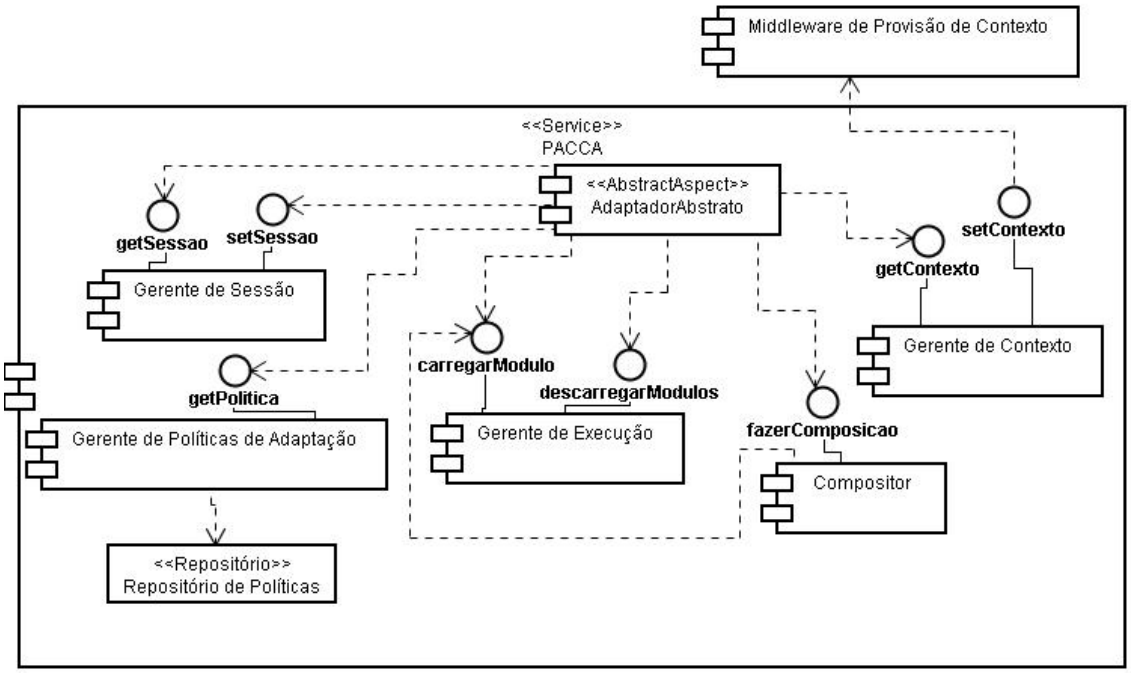

Figura 11 - Componentes do PACCA e sua integração com o *middleware* de provisão de contexto

O diagrama de classes da Figura 12 expõe uma visão estrutural do PACCA mais detalhada apresentando as características internas dos seus módulos, através das respectivas classes com os seus atributos e métodos.

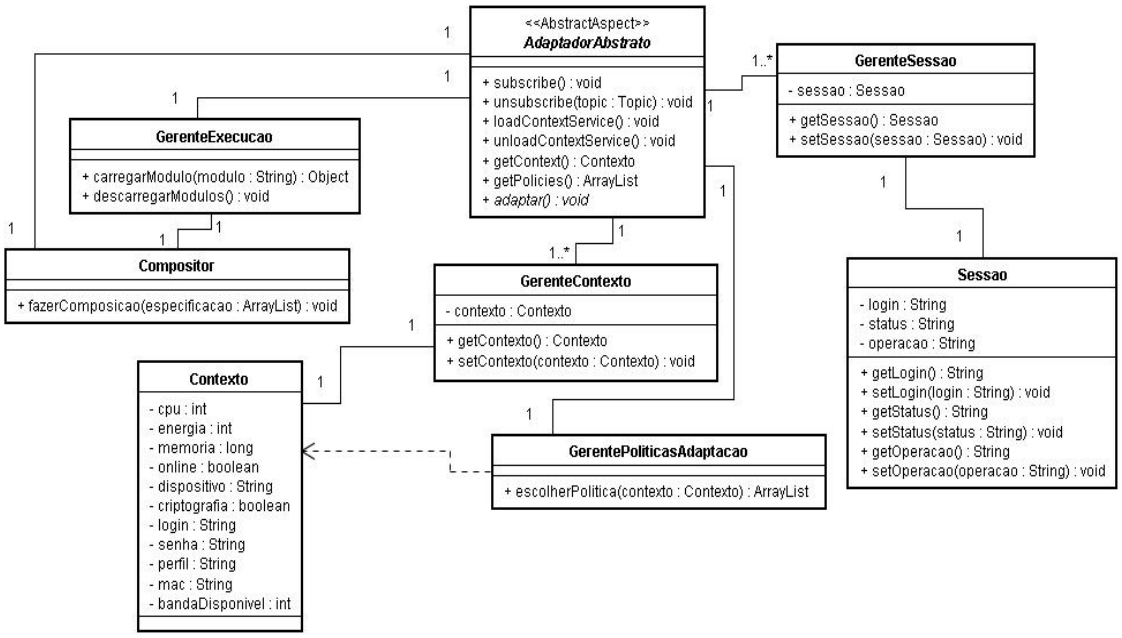

Figura 12 – Diagrama de Classes do PACCA

3.2.1. Gerente de Contexto

O *Gerente de Contexto* armazena as informações do contexto corrente, as quais são constantemente atualizadas pelas notificações recebidas do *middleware* de provisão de contexto, e caracterizam o estado atual do dispositivo e da rede. Dessa forma, toda decisão sobre a adaptação considera as informações contidas no *Gerente de Contexto*.

3.2.2. Gerente de Políticas de Adaptação e Repositório de Políticas

O *Repositório de Políticas* armazena as políticas de adaptação, as quais são definidas de forma declarativa utilizando XML. O *Gerente de Políticas de Adaptação* é responsável pela definição dos comportamentos adaptativos baseando-se no contexto corrente. Esse componente é responsável por analisar as políticas de adaptação armazenadas no *Repositório de Políticas*. Essa análise inclui um *parser* XML, ou seja, é feita a conversão das políticas definidas sob a forma declarativa para a forma imperativa/procedural. Ainda é de responsabilidade do componente verificar qual a política adequada ao contexto atual e retornar uma especificação com a composição de módulos da aplicação adequada ao contexto. Essa especificação será usada inicialmente pelo *Compositor* e depois pelo *Gerente de Execução*, para saber quais módulos devem ser dinamicamente carregados.

3.2.3. Compositor e Gerente de Execução

O *Compositor* é o responsável por realizar a composição dos módulos que serão carregados pelo *Gerente de Execução* em cada contexto. Ele possui a incumbência de interpretar a saída do *Gerente de Políticas de Adaptação* traduzindo-a para uma lista de nomes de classes totalmente qualificados. Também é sua responsabilidade fazer com que os módulos carregados com base na especificação se relacionem e funcionem corretamente, ou seja, validar a composição. Porém, não foram tratados detalhes relativos à questão de validação, assumindo-se que todas as composições geradas são válidas. A seguir, os nomes das classes identificadas pelo *Compositor* como necessárias para o contexto corrente são passadas para o *Gerente de Execução* que por sua vez, irá buscá-las através de um serviço de nomes (no trabalho está sendo utilizado o JNDI (*Java Naming and Directory Interface*) [JNDI, 2008]). Tais classes podem localizar-se tanto no espaço local de execução da aplicação quanto remotamente. O *Gerente de Execução* é o componente responsável pela carga dinâmica, no espaço de execução da

aplicação, dos módulos de software especificados pelas composições e já localizados pelo serviço de nomes. A importância do *Gerente de Execução* deve-se ao fato de que, em dispositivos móveis, a gerência de alocação de memória é um fator crítico. Então, com a carga dinâmica de componentes pode-se manter em memória somente os módulos que são realmente necessários em cada contexto de execução.

3.2.4. Gerente de Sessão

Outro componente do PACCA é o *Gerente de Sessão*, o qual é responsável pela criação, manutenção e resgate de sessões do usuário. Uma sessão é caracterizada pelo período em que o usuário interage com uma determinada aplicação. O gerenciamento de sessões em ambientes de Computação Ubíqua é extremamente importante, pois permite ao usuário maior produtividade no uso de uma aplicação, visto que ele pode migrar entre diversos dispositivos, "mantendo-se conectado à aplicação" e continuar a utilizá-la exatamente do ponto onde havia parado. A garantia de execução sem interrupção e a migração de dispositivos de forma transparente para o usuário são fundamentais para caracterizar o comportamento ubíquo de uma aplicação.

No PACCA, as sessões são armazenadas em um repositório quando o usuário efetua *login* na aplicação e removidas quando é realizado o *logoff*. No entanto, quando um usuário encerra a aplicação de forma inesperada, por exemplo, por falta de carga de bateria, ou quando realiza *login* utilizando outro dispositivo, é possível resgatar a sessão. O resgate da sessão permite ao usuário continuar suas operações ao migrar entre os dispositivos.

O Gerente de Sessão implementado na versão atual do PACCA está tratando apenas casos preliminares de migração de sessão entre dispositivos, pois tal problema é bastante complexo e problemas como o gerenciamento de sessões quando o usuário está *offline* em um dispositivo e depois acessa a aplicação em modo *online* a partir de outro dispositivo, há uma inconsistência entre as sessões, pois tal problema ainda não está sendo tratado.

3.2.5. Módulo de Adaptação

O *Módulo de Adaptação* é o componente que responde pelo processo de adaptação dinâmica, gerenciando todos os demais componentes do PACCA. Esse módulo foi concebido usando os conceitos de orientação a aspectos e foi desenvolvido

como um aspecto abstrato (*AdaptadorAbstrato*, ver Figura 13) no qual estão definidas operações comuns às aplicações adaptativas cientes de contexto, tais como: (i) carga e descarga do serviço de provisão de contexto, onde pode ser carregado e descarregado qualquer *middleware* de provisão de contexto; (ii) subscrição (*subscribe*) de tópicos de interesse da aplicação (contextos) e o cancelamento da subscrição (*unsubscribe*); (iii) consulta ao contexto corrente e (iv) consulta às políticas de adaptação. No PACCA, optou-se pela abordagem de especificar um aspecto abstrato com o intuito de possibilitar maior desacoplamento e flexibilidade, facilitando o reuso dos aspectos e tornando a arquitetura mais flexível e genérica.

O *AdaptadorAbstrato* provido pela implementação atual do PACCA possui comportamento padrão definido pra cada uma de suas operações. Por exemplo, na versão atualmente implementada, a funcionalidade padrão de carga e descarga de provisão de contexto utiliza o serviço de informação de contexto do MOCA denominado CIS [SACRAMENTO et al., 2004], o qual será detalhado na Seção 3.4. Porém, esse comportamento pode ser redefinido e incrementado através da implementação de um aspecto concreto.

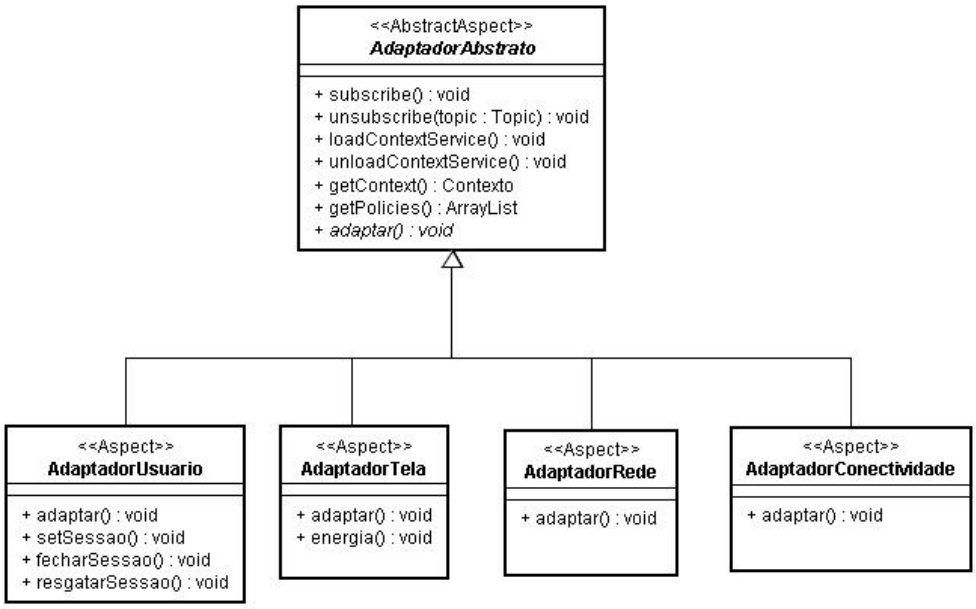

Figura 13 – Aspectos abstrato e concretos

O *AdaptadorAbstrato* é responsável por interceptar as situações onde existe a necessidade de adaptação. Ele possui um método *adaptar*() que deve ser redefinido para cada um dos diversos comportamentos adaptativos específicos (representados por aspectos concretos). Os aspectos concretos acrescentam também comportamentos específicos ao método *adaptar*(), através da definição de novas operações relativas aos

interesses do seu escopo. Porém, independente do comportamento específico, tal método sempre realiza as seguintes operações: (i) obtém o contexto corrente do *Gerente de Contexto*, (ii) repassa o contexto atual para o *Gerente de Políticas de Adaptação*, (iii) recebe uma especificação de módulos, (iv) envia a especificação ao *Compositor*, para que a política possa efetivamente ser aplicada, através da composição dinâmica da aplicação. O *Compositor*, por sua vez, invoca o *Gerente de Execução* para realizar a carga dinâmica dos módulos. A Figura 14 apresenta um Diagrama de Sequência para ilustrar melhor as interações dos componentes do PACCA citados.

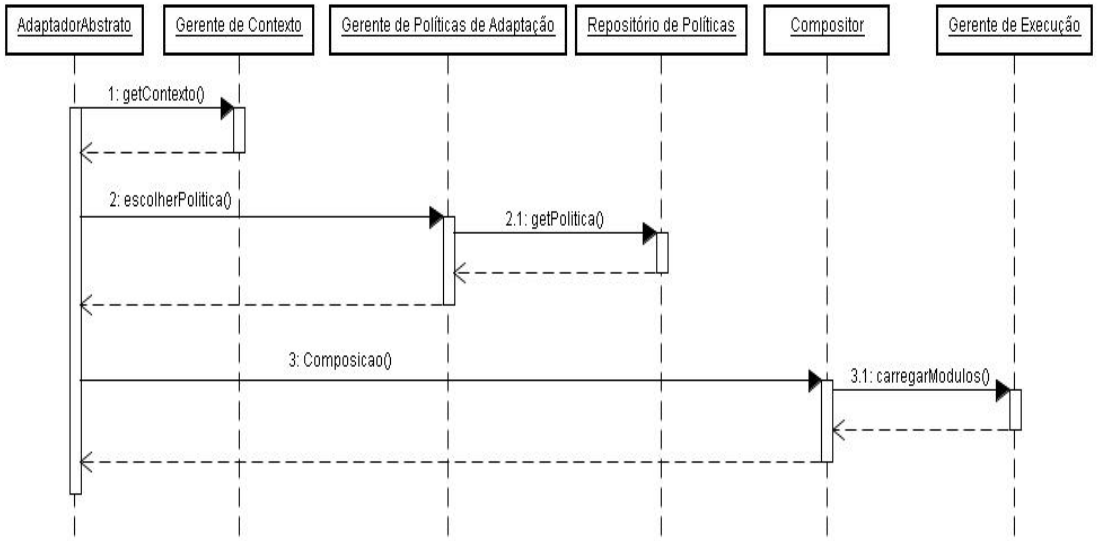

Figura 14 - Diagrama de Seqüência do processo de adaptação realizado pelo PACCA

Com relação aos aspectos concretos, para a sua definição foi utilizado um conjunto de *interesses adaptativos* (*adaptive concerns* identificados em [LOUGHRAN et al., 2006], ver Seção 3.1), os quais representam comportamento adaptativo transversal comum a vários tipos de aplicações ubíquas. Ao modelar os interesses adaptativos como um aspecto abstrato e um conjunto de aspectos concretos, potencialmente reutilizáveis, obtêm-se um alto grau de flexibilidade e reuso da solução. Assim, na maioria dos casos, as aplicações precisarão apenas usar os aspectos existentes devendo somente definir onde e quando, no código de negócio, eles deverão ser aplicados. Por outro lado, quando necessário, o modelo proposto pode ser facilmente estendido, bastando para isso incluir novos aspectos concretos representando o comportamento adaptativo adicional.

Na Figura 13 é apresentado um diagrama de classes onde pode ser vista a associação estática entre o *AdaptadorAbstrato* e os aspectos concretos definidos neste trabalho. Os aspectos concretos (interesses adaptativos) definidos no diagrama são: (i)

AdaptadorUsuario – responsável por adaptar detalhes específicos do perfil de cada usuário, inclusive considerando a autenticação e o gerenciamento de sessão dos usuários*;* (ii) *AdaptadorTela* – responsável pelos detalhes relativos à adaptação das telas para os dispositivos aceitos pela aplicação (e em função do nível de energia do dispositivo) e com bases nos respectivos perfis de usuário; (iii) *AdaptadorRede* – responsável por detalhes que dizem respeito à adaptação para diferentes tipos e condições de rede (por exemplo, pode ser necessário comprimir os dados enviados pela rede em caso de pouca disponibilidade de banda); (iv) *AdaptadorConectividade* – encarrega-se dos detalhes relativos às operações que envolvem acesso as Bases de Dados de uma aplicação que podem ser locais ou remotas.

3.3. Processo de Desenvolvimento de Aplicações

A construção de aplicações adaptativas cientes de contexto no PACCA requer que o desenvolvedor siga um conjunto de passos, desde a fase de levantamento, análise e modelagem de requisitos, até a implementação. Os passos propostos foram elaborados a partir de diversas propostas de processos de desenvolvimento existentes na área de DSOA, tais como [CHITCHYAN et al., 2005].

Durante a fase de análise, são necessários os passos listados a seguir.

(i) Especificação dos requisitos de negócio da aplicação, construindo o Modelo Base. Nessa etapa, são produzidos diagramas UML, como Diagramas de Classes, de Casos de Uso e de Interações.

(ii) Identificação dos interesses adaptativos relevantes para a aplicação. Tais interesses provavelmente já estarão cobertos pela lista de interesses adaptativos fornecida pelo PACCA, para o domínio de aplicações ubíquas.

(iii) Definição das políticas de adaptação, as quais descrevem o comportamento a ser adotado pela aplicação em cada contexto, através de regras declarativas.

(iv) Identificação dos componentes de negócio que serão afetados pelos interesses adaptativos. Nessa etapa, sugere-se a construção de uma matriz relacionando cada componente (classe) do Modelo Base com cada interesse adaptativo identificado.

(v) A partir da matriz construída no passo (iv) e dos interesses adaptativos identificados no passo (ii), construir o Modelo de Aspectos. Não há uma notação padrão para representar aspectos em nível de modelo, embora haja várias propostas, como Theme/UML [BANIASSAD; CLARKE, 2004b] e Aside [CHAVEZ, 2004]. No

presente trabalho optamos por representar aspectos como classes UML adicionadas de estereótipos (<<Aspect>> para os aspectos e <<AbstractAspect>> para o aspecto abstrato). As interceptações (relações entre os aspectos e as classes do Modelo Base) são representadas como associações estereotipadas com <<intercepta>>.

Na fase de análise não é considerada a plataforma de implementação, ou seja, a utilização de linguagens de programação específicas (tanto linguagem de componentes quanto de aspectos) e de *frameworks.* Tais detalhes são especificados na fase de projeto, onde são definidas as tecnologias e soluções a serem adotadas. Portanto, a fase de projeto concentra-se em como será implementado o sistema, fornecendo subsídios para a fase de implementação. Ademais, deve ser definido o projeto arquitetural da aplicação, o qual define quais os relacionamentos entre os componentes e suas respectivas interfaces e em quais nós de processamentos serão instalados os componentes.

Já para a fase de implementação, o processo proposto prevê as seguintes etapas:

(i) Implementação dos componentes do Modelo Base previamente especificados (interface gráfica, lógica de negócio da aplicação) usando uma abordagem modular, ou seja, separando as funcionalidades da aplicação em módulos de software (componentes) que devem ser compostos para originar uma aplicação completa.

(ii) Especificação em XML das políticas de adaptação definidas.

(iii) Especificação dos pontos de junção (*join points*), ou seja, dos pontos no código dos componentes de negócio que serão afetados pelos interesses adaptativos.

(iv) Encapsulamento do comportamento adaptativo definido em um ou mais aspectos, ou seja, definição dos *pointcuts* (conjunto de pontos de junção) e dos adendos (*advices*), código que contém o comportamento adaptativo a ser injetado na aplicação nos pontos determinados.

Cada aspecto a ser definido consiste de um conjunto de *advices* e *pointcuts,* onde se definem os comportamentos adaptativos e os pontos no modelo base onde serão injetados esses comportamentos.

Cabe ressaltar que, na etapa (iv), os aspectos a serem definidos pelo desenvolvedor da aplicação serão concretizações do aspecto abstrato definido no PACCA. Como o PACCA já considera um conjunto *default* de interesses adaptativos comuns a aplicações ubíquas, fornecendo os seus respectivos aspectos concretos, o desenvolvedor precisa apenas redefinir os respectivos métodos *adaptar()* de cada aspecto concreto de acordo com as necessidades específicas da aplicação. Em outras

palavras, no aspecto abstrato estão definidas operações comuns às aplicações adaptativas cientes de contexto. Por sua vez, o aspecto concreto estende essas operações com as específicas daquele comportamento adaptativo. Uma das especificidades identificadas consiste no tipo de informação de contexto relevante para cada comportamento adaptativo. Por exemplo, um interesse adaptativo definido pelo aspecto *AdaptadorTela* depende do contexto do dispositivo em uso e do perfil do usuário acessando a aplicação. Já o interesse adaptativo definido pelo aspecto *AdaptadorRede* depende apenas do contexto da rede.

Por fim, temos a fase de testes com o objetivo de validar e verificar a aplicação construída, a fim de constatar se o produto final está de acordo com as especificações.

O Diagrama de Atividades da Figura 15 exemplifica o processo de desenvolvimento de uma aplicação ciente de contexto adaptativa utilizando a abordagem proposta no presente trabalho, com as respectivas atividades definidas no processo e os artefatos a serem gerados e consumidos por cada atividade.

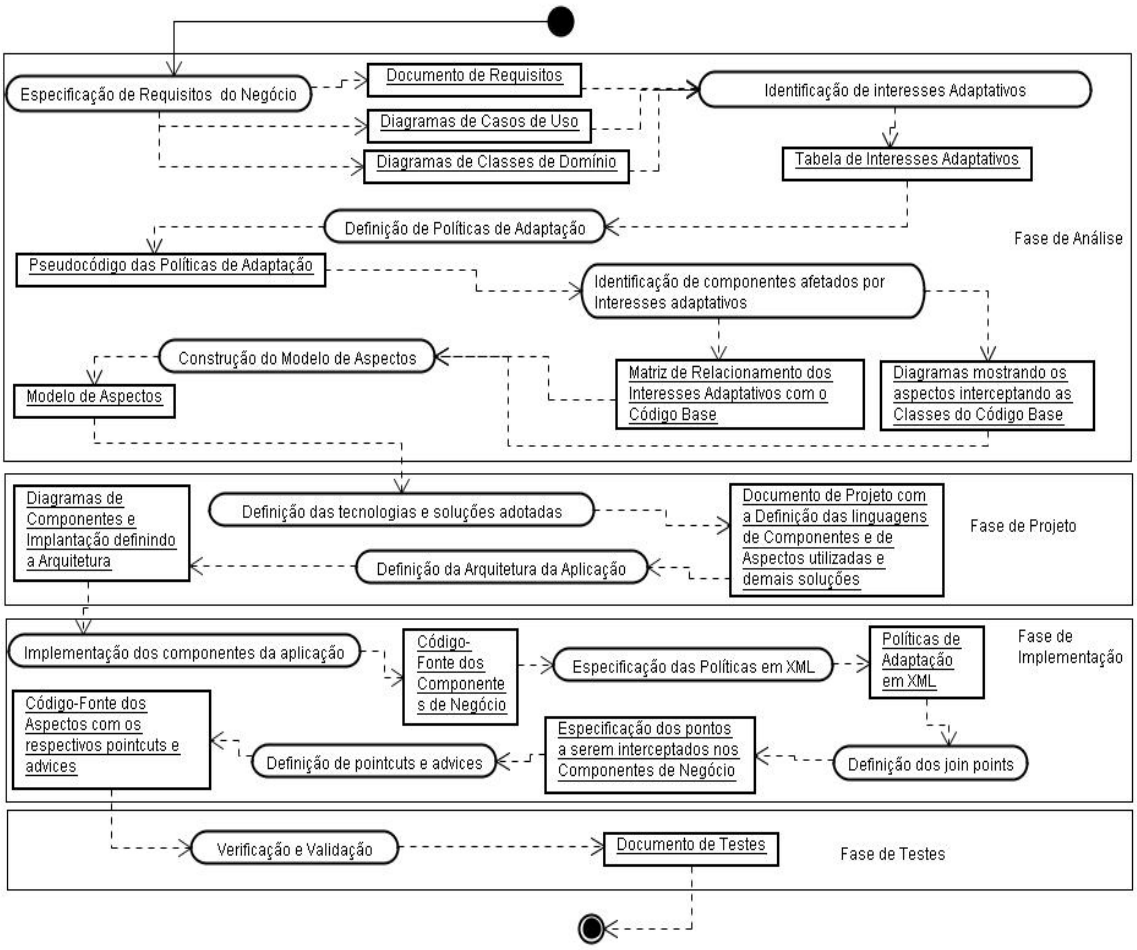

Figura 15 – Processo de Desenvolvimento de aplicações cientes de contexto adaptativas

3.4. Implementação

Nesta Seção serão descritos os aspectos relativos à implementação do PACCA. Conforme definido em sua arquitetura, o PACCA utiliza um *middleware* de provisão de contexto para realizar todas as funções de aquisição e manipulação de informações contextuais de baixo nível. Na atual implementação, adotou-se o MOCA [SACRAMENTO et al., 2004]. Tal escolha foi motivada pela disponibilidade do código do MOCA (e de sua API) e por ele ser um middleware já bastante utilizado pela comunidade na construção de aplicações ubíquas.

O PACCA utiliza alguns serviços providos por componentes do MOCA, como o *CIS*, *LIS* e *Monitor*, além do Modelo de Contexto do MOCA [ROCHA; ENDLER, 2006].

No MOCA, a fim de se obter notificações sobre o contexto corrente e mudanças de contexto do dispositivo, é necessária a subscrição do endereço *MAC* (*Media Access Control*) do dispositivo junto ao *CIS*. Por tratar-se de um requisito específico da plataforma de provisão de contexto, optou-se por realizar tal subscrição (*subscribe*) no *Modulo de Adaptação* do PACCA, dessa forma deixando-a transparente para a aplicação. No entanto, o dispositivo já deve ter sido previamente cadastrado com o seu respectivo endereço *MAC*. Assim, no processo de autenticação do usuário, o *Módulo de Adaptação* obtém o *MAC* do dispositivo de acesso a partir de uma Base de Dados e subscreve-o ao *CIS*.

Outra forma de receber notificações sobre o contexto no MOCA é a subscrição, junto com o endereço *MAC*, de tópicos específicos, como por exemplo, “notificar quando a quantidade de memória atingir determinado nível”. Na versão atual do PACCA, é usada apenas a primeira opção, onde o *CIS* notificará sobre qualquer mudança de contexto que afete o dispositivo com *MAC* subscrito. As subscrições são canceladas (*unsubscribe)* quando o usuário realiza o *logoff* da aplicação e o PACCA, quando finalizado, realiza a descarga do MOCA através da operação u*nloadContextService()*.

Para a implementação das funcionalidades do PACCA que usam POA, foram desenvolvidas duas versões. Uma versão foi construída utilizando o arcabouço orientado a aspectos JBOSS AOP [JBOSS AOP, 2008] e a outra foi implementada na linguagem AspectJ [ASPECTJ, 2008]. Ambas as versões foram desenvolvidas

utilizando *plugins* disponíveis para o ambiente integrado de desenvolvimento (IDE – *Integrated Development Environment*) Eclipse [ECLIPSE, 2008].

A implementação do PACCA foi realizada utilizando a versão Java SE 5.0 (*Java Standard Edition*). Em trabalhos futuros serão implementadas versões em Java ME (*Java Micro Edition*) para representar funcionalidades que poderiam rodar em celulares e PDA´s, visto que Java ME tem o seu foco no desenvolvimento de aplicações para dispositivos móveis.

Atualmente, a plataforma Java ME possui limitações em relação as suas capacidades reflexivas, não provendo suporte para a API Java de reflexão [CAPRA et al., 2002]. Essa API permite que os componentes sejam inspecionados em tempo de execução, obtendo-se dinamicamente informações sobre construtores, atributos e métodos das classes. Porém, já existem algumas propostas com o intuito de superar tais limitações, provendo soluções para implementar reflexão na plataforma Java ME, como por exemplo [LIBORIO; BIGONHA, 2004]. O *Gerente de Políticas de Adaptação* do PACCA faz uso de reflexão para obter do *Gerente de Contexto* o valor corrente, de forma dinâmica, dos atributos a serem considerados para a aplicação da política de adaptação. Portanto, o suporte a reflexão é um requisito fundamental para a implementação do PACCA.

Além disso, na atual versão da plataforma Java ME não é permitido ao programador reescrever o mecanismo de carga de classes (*ClassLoader*) [DANTAS; BORBA, 2003, RODRIGUES; RODRIGUES, 2007]. Tal limitação não afeta o uso do PACCA, pois ele baseia-se no mecanismo de carga de classes padrão de Java, não requerendo alterações de seu comportamento. Para fazer a carga dinâmica das classes, o *Gerente de Execução* do PACCA utiliza o método *forName* da classe *Class* do pacote java.lang. Tal método recebe como parâmetro o nome qualificado da classe e o uso de reflexão permite que a classe possa ser carregada a partir de diferentes construtores, os quais podem ser obtidos dinamicamente, assim como os seus parâmetros.

Em relação à implementação do *Compositor*, é importante ressaltar que ele faz uso do JNDI [JNDI, 2008], que funciona como uma ponte sobre diversos serviços de nomes e diretórios diferentes, permitindo que componentes (locais ou remotos) possam ser registrados e localizados a partir de seu nome. A vantagem do uso do JNDI está no fato de que é necessário aprender apenas uma API para acessar diferentes tipos de serviços de nomes e de diretório.

O JNDI apresenta um componente chamado SPI (*Service Provider Interface*), o qual permite que diversos serviços de nomes e diretórios, tais como Sistemas de Arquivos [JNDI, 2008], LDAP [LDAP, 2008], DNS [DNS, 2008], além de serviços de nomes providos em RMI [RMI, 2008] e CORBA [CORBA, 2008], sejam utilizados de forma transparente. No *Compositor* está sendo utilizado RMI-IIOP (*Remote Method Invocation – Internet Inter-ORB Protocol*) [RMI-IIOP, 2008], uma versão do RMI compatível com CORBA [CORBA, 2008].

O uso de JNDI associado ao RMI-IIOP exige a criação de um servidor RMI-IIOP para a publicação dos objetos, os quais devem ser registrados no JNDI para que possam ser localizados pelos clientes [RMI-IIOP, 2008].

Para um objeto ser localizado remotamente é necessário que ele implemente uma interface remota a qual declara os métodos a serem expostos pelo objeto, os quais correspondem aos serviços a serem providos. Para que haja a comunicação entre os clientes e os objetos remotos que provêem serviços é preciso que sejam criados os *stubs* e *skeletons.* Ambos funcionam como *proxies* entre o cliente e o objeto remoto e são criados a partir da compilação das classes relativas ao cliente e ao objeto remoto com o compilador *rmic*. Como neste caso está sendo utilizado RMI-IIOP, no momento da compilação é preciso ativar a opção de compilação RMI-IIOP do compilador *rmic*, utilizando o argumento –iiop. [RMI-IIOP, 2008]

O Capítulo 4 apresenta a aplicação utilizada como Estudo de Caso, a qual faz uso das funcionalidades providas pelo PACCA.

4. Estudo de caso: Sistema de Informação para Ambiente de Medicina Ubíqua

Para ilustrar o uso do PACCA, bem como do processo de desenvolvimento a ser adotado para a construção das aplicações sobre ele, foi idealizado um cenário médico no qual o PACCA fornece suporte a uma aplicação para consulta e troca de informações em um ambiente ubíquo.

Este Capítulo apresenta na Seção 4.1 uma descrição do cenário adotado no estudo de caso e em seguida aborda o processo de construção da aplicação usando o PACCA nas fases de Análise (Seção 4.2), Projeto e Implementação (Seção 4.3).

4.1. Descrição do Cenário

No cenário do estudo de caso desenvolvido, informações relativas a pacientes de um sistema de informações médicas são concentradas em um servidor e acessadas a partir de diferentes locais (hospital, consultório médico e residência do paciente). Os dispositivos utilizados para visualização, consulta, cadastro e atualização de tais informações, bem como o formato dos dados apresentados (textual ou gráfico, criptografado ou não) podem variar, dependendo do usuário e de sua localização. Todas essas variações representam o comportamento adaptativo da aplicação, e são manipuladas pelos serviços providos pelo PACCA.

A aplicação desenvolvida para automatizar o cenário descrito disponibiliza serviços como: visualização e prescrição de exames, medicamentos e tratamento, cadastro de diagnósticos, consultas ao histórico do paciente e gerência do tratamento do paciente. Ela ainda possibilita o monitoramento à distância de pacientes através de sensores físicos instalados nos mesmos, os quais enviam dados através de conexão de redes sem fio. Assim, combinando-se o suporte da infra-estrutura provida pelo PACCA com a aplicação implementada, é possível utilizar as informações médicas disponíveis de forma seletiva, a qualquer instante e lugar, através de qualquer dispositivo habilitado.

A aplicação pode ser acessada por profissionais, médicos ou enfermeiros, bem como pelos pacientes, todos esses cadastrados e enquadrados em diferentes perfis de usuários, os quais espelham as responsabilidades de cada grupo e limitam as funcionalidades do sistema baseadas nas atividades exercidas por eles. O perfil *Enfermeiro,* usado pelos profissionais de enfermagem, é responsável por cadastrar a administração de medicamentos e de procedimentos, como por exemplo, aferir pressão arterial, realizar exames e coletar material fisiológico. Esse perfil não possui permissão

de acessar as informações pessoais do paciente, nem pode prescrever medicamentos, sendo essas as atividades relativas ao perfil *Médico.* Porém, ele pode receber notificações de estados críticos de pacientes, as quais são enviadas pelos sensores de monitoramento dos pacientes. O perfil *Médico* pode prescrever medicamentos, procedimentos e exames, emitir diagnósticos, receber notificações do estado do paciente e, quando necessário, encaminhar o paciente a outro médico especialista. Ele está habilitado a acessar as informações pessoais do paciente, como por exemplo, seu histórico, resultado de exames, medicamentos em uso, dentre outros dados relevantes ao tratamento. O terceiro perfil, *Paciente*, é criado para os pacientes e seus acompanhantes consultarem o sistema a qualquer momento, a fim de obterem informações a respeitos dos exames solicitados e seus resultados, bem como os medicamentos e procedimentos prescritos.

Há também um perfil de *Administrador do sistema* encarregado de realizar o cadastro dos médicos, enfermeiros e pacientes aptos a utilizar a aplicação. Foi necessária a criação desse perfil para fins de segurança e gerenciamento, pois os dispositivos para usufruírem do acesso ao sistema precisam ser previamente cadastrados e autorizados pelo *Administrador*. Portanto, quando o usuário deseja acessar a aplicação, há um processo de *login* do qual faz parte verificar se o endereço *MAC* do seu dispositivo de acesso encontra-se cadastrado na Base de Dados Central e associado a um usuário habilitado a utilizar o sistema. Depois de autorizado o acesso, o usuário pode desenvolver suas atividades, de acordo com seu perfil e o dispositivo ao qual está conectado no momento.

Para ilustrar a adequação do emprego do PACCA no cenário da aplicação médica supracitada, um caso comum passível de acontecer aplica-se aos hospitais que oferecem atendimento *home-care* a pacientes. Eles podem utilizar a infra-estrutura do PACCA e a aplicação para possibilitar ao médico assistido pelo sistema, dispor de informações relativas ao paciente a ser atendido em seu dispositivo portátil. Nesse cenário, ao chegar à residência do paciente, o médico consulta o sistema para tomar conhecimento de quando foi realizada a última consulta, os resultados dos últimos exames, os medicamentos tomados, dentre outras informações disponíveis. Todos os dados prescritos pelo médico nessa residência em um momento de ausência de disponibilidade de sinal de rede são armazenados temporariamente no dispositivo portátil do médico, para tão logo o sinal de rede esteja disponível, esses dados sejam atualizados no sistema central.

Os comportamentos adaptativos considerados neste estudo de caso são baseados nas seguintes características contextuais: (i) *Perfil de Usuário*: identificado no momento do *login* e que permite a aplicação carregar os componentes funcionais adequados ao papel do usuário (responsabilidades) no sistema; (ii) *Tipo e Estado do Dispositivo de Acesso*: usados para adaptar a interface gráfica, ou seja, carregar uma interface adequada para ser visualizada no dispositivo específico (os dispositivos de acesso aceitos são Notebooks, Celulares e PDA's); (iii) *Estado da Rede*: com base nessa informação a aplicação é adaptada de forma que, quando o estado é *conectado*, a Base de Dados Central é usada e quando o estado é *desconectado*, a aplicação utiliza a Base de Dados Local para armazenar informações, repassando tais informações posteriormente para a Base de Dados Central com o restabelecimento do sinal de rede; (iv) *Nível de Energia*: quando o nível de energia do dispositivo for menor que um limiar definido pela política de adaptação, é carregada uma interface textual; (v) *Localização*: especificada por uma *região simbólica* [SACRAMENTO et al., 2004] que representa o hospital, a localização permite dividir o ambiente de execução em *externo* e *interno*; um ambiente *interno* representa o acesso a aplicação de um ponto dentro do hospital, e é considerado "seguro" na abordagem aqui apresentada, não havendo nesse caso necessidade de adoção de criptografia. Ambientes *externos*, definidos como qualquer região fora do hospital, são considerados "inseguros" e requerem que a aplicação adapte-se provendo criptografia (carregar o módulo de criptografia) para o tráfego de dados entre o dispositivo de acesso e a Base de Dados Central. É importante observar que tanto o estado da rede, quanto a localização e o nível de energia do dispositivo podem mudar ***enquanto*** a aplicação está em execução, e isso é tratado pela abordagem de adaptação dinâmica provida pelo PACCA.

4.2. Construção da Aplicação usando o PACCA – Fase de Análise

Esta Seção aborda a construção da aplicação definida no estudo de caso, cujo cenário foi descrito na Seção 4.1, segundo o processo de desenvolvimento proposto no presente trabalho (descrito na Seção 3.3).

4.2.1. Especificação de Requisitos de Negócio

Como visto na Seção 3.2, o primeiro passo necessário para a construção de uma aplicação usando o processo proposto para uso com o PACCA consiste em especificar os requisitos de negócio da aplicação, construindo assim o Modelo Base. Para o estudo

de caso considerado, os requisitos da aplicação identificados estão ilustrados nos Diagramas de Casos de Uso das Figuras 16, 17, 18 e 19, separados de acordo com os perfis de usuário. Esses requisitos irão definir as principais funcionalidades da aplicação a ser construída.

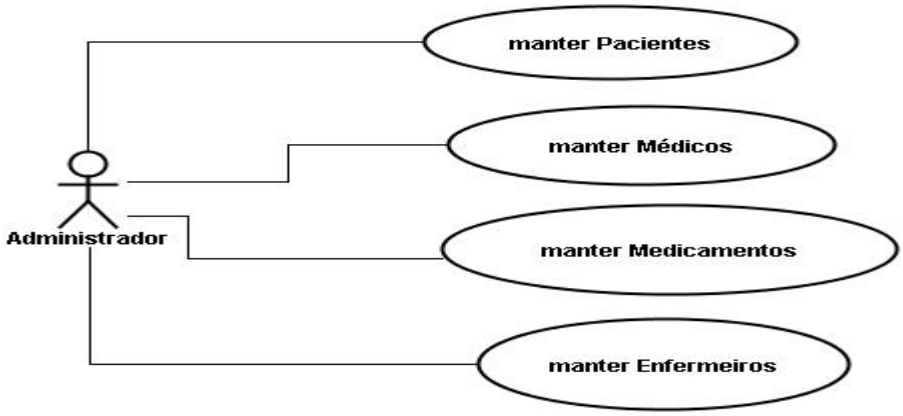

Figura 16 – Casos de Uso de Administrador

No diagrama de casos de uso referente às responsabilidades do perfil *Administrador*, é utilizado o verbo *manter*, o qual tem objetivo de referir-se ao conjunto de operações custodiais (cadastro, atualização e busca). Portanto, o caso de uso *manter Pacientes* refere-se às operações *cadastrar Paciente*, *atualizar Paciente* e *buscar Paciente*. O mesmo valendo para os demais casos de uso ilustrados.

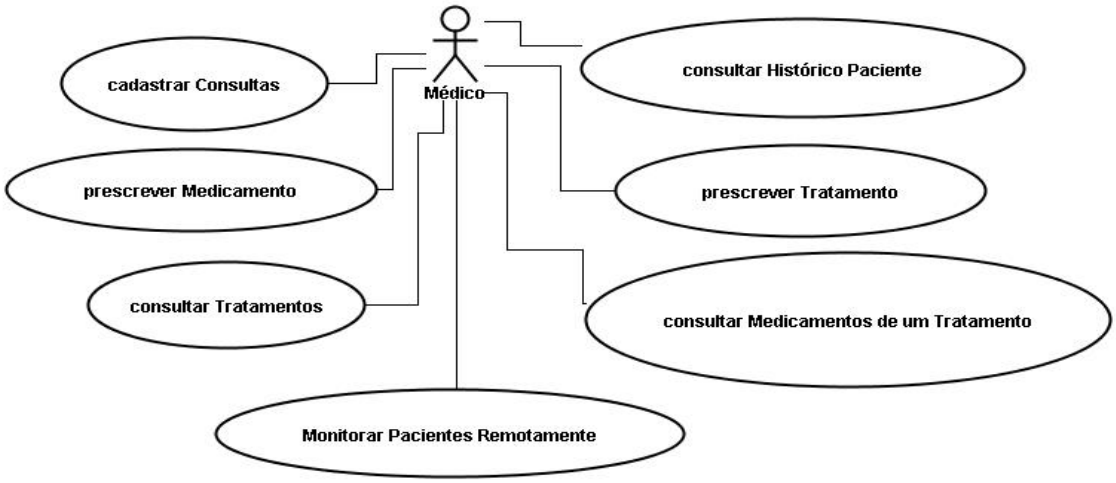

Figura 17 – Casos de Uso de Médico

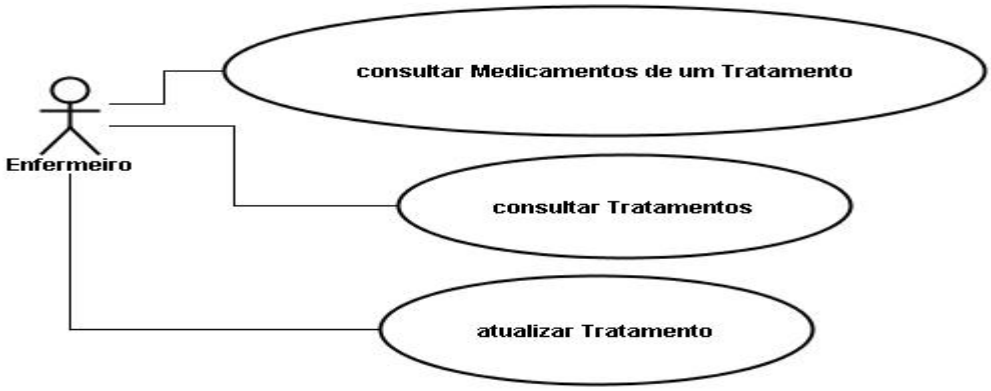

Figura 18 – Casos de Uso de Enfermeiro

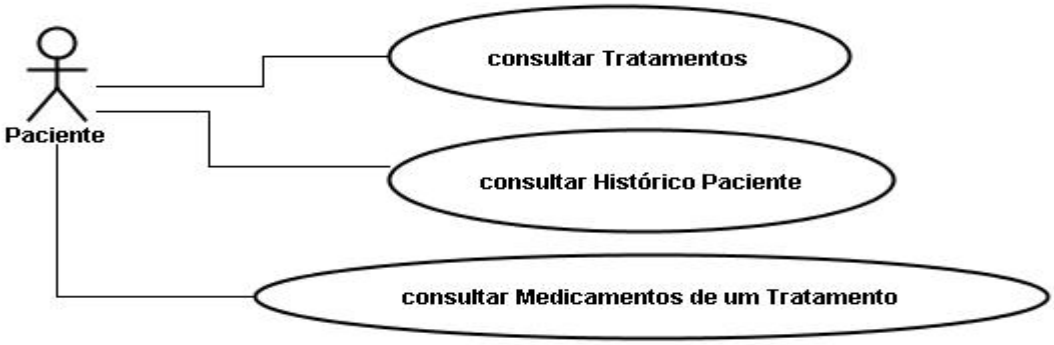

Figura 19 – Casos de Uso de Paciente

Já as principais classes de domínio identificadas para o cenário descrito estão ilustradas nos Diagramas de Classes das Figuras 20, 21, 22, 23 e 24. Nos diagramas são mostrados os relacionamentos entre as classes que compõem a aplicação de forma a permitir uma visão estrutural detalhada. Para a identificação das classes de domínio da aplicação foi usada a técnica Dirigida a Análise Casos de Uso e a Categorização BCE (Boundary,Control,Entity), ou seja, as classes foram identificadas conforme suas responsabilidades no contexto da aplicação. Na categoriazação BCE classes de Entidade (*Entity*) são aquelas que representam as informações a serem persistidas. As classes de Fronteira (*Boundary*) são as responsáveis pela comunicação da aplicação com o mundo exterior (atores) e as classes de Controle (*Control*) são "pontes de comunicação" entre objetos de fronteira e de entidade, sendo responsáveis por controlar a lógica correspondente a cada caso de uso.

A categorização BCE é bastante similar ao padrão de projeto MVC (*Model-View-Controller*) [BUSCHMANN et al., 1996]. No diagrama da Figura 20 é apresentando o modelo de classes da categoria Entidade (*Entity*, segundo a categorização BCE ou *Model*, segundo o padrão MVC). Também é utilizado padrão

DAO (*Data Access Object*) cujo objetivo é separar os detalhes relativos à persistência da lógica da aplicação promovendo isolamento e flexibilidade, permitindo facilmente modificar a tecnologia de persistência, como por exemplo, migrar entre diferentes SGBD's. Os diagramas subseqüentes ilustram as classes das categorias Controle e Fronteira e omitem algumas classes de entidade para melhorar a legibilidade.

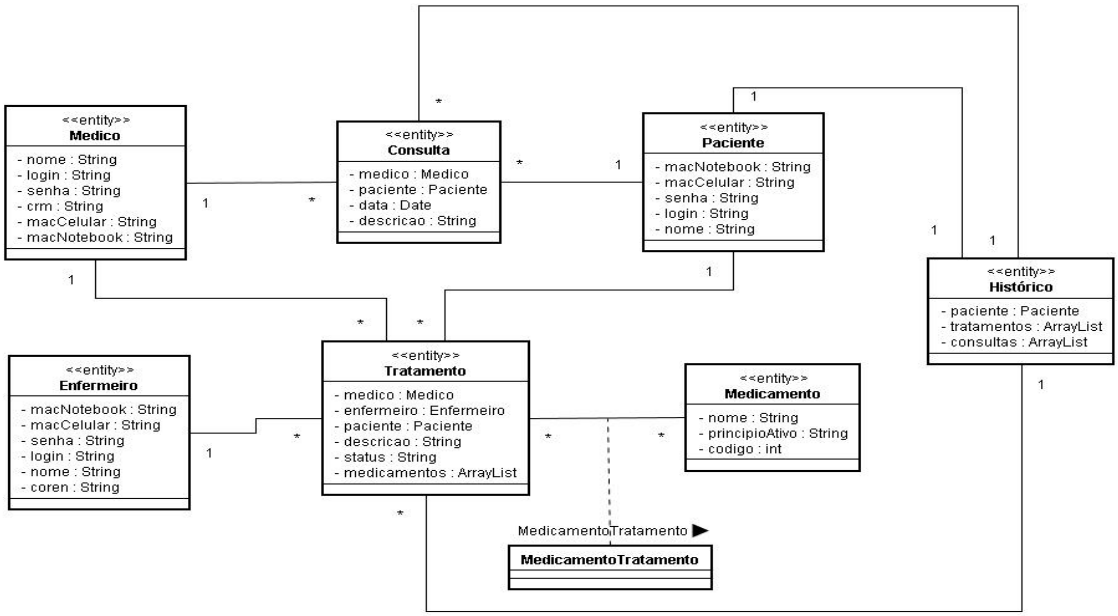

Figura 20 – Modelo de Classes de Entidade

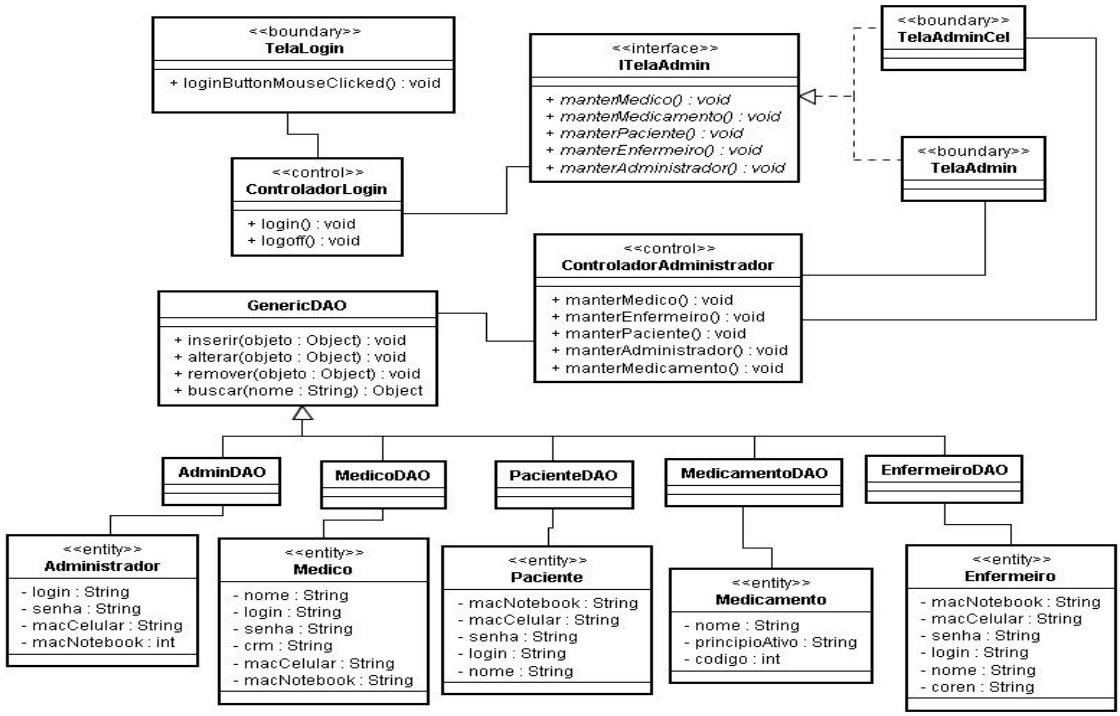

Figura 21 - Diagrama de classes do sistema de informações para medicina ubíqua relacionadas aos casos de uso de administrador

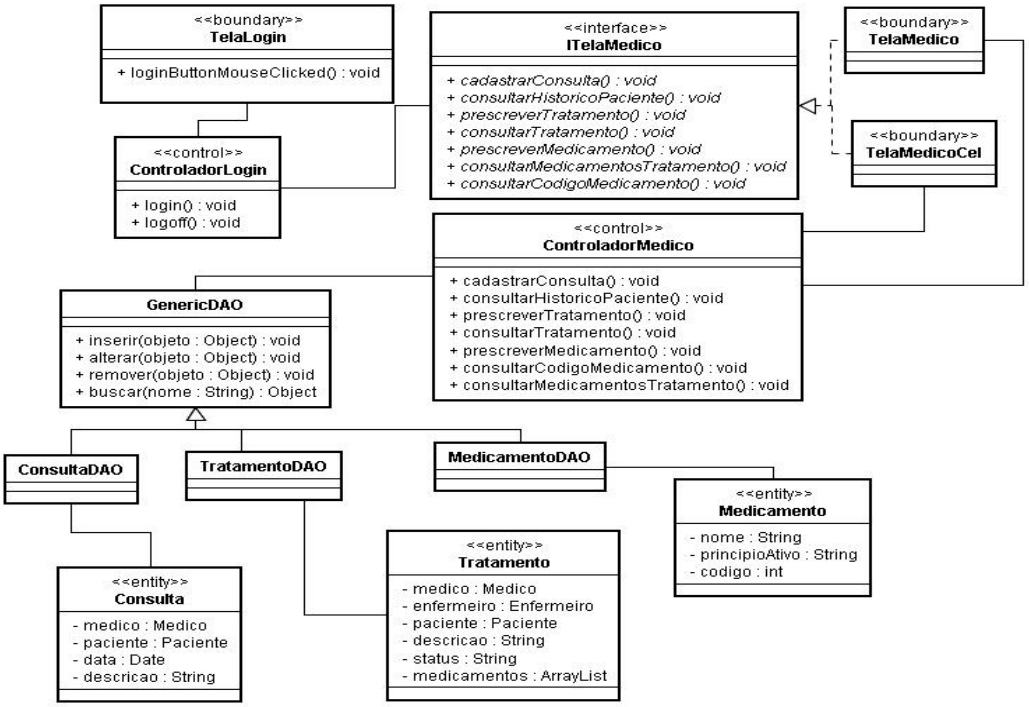

Figura 22 - Diagrama de classes do sistema de informações para medicina ubíqua relacionadas aos casos de uso de Médico

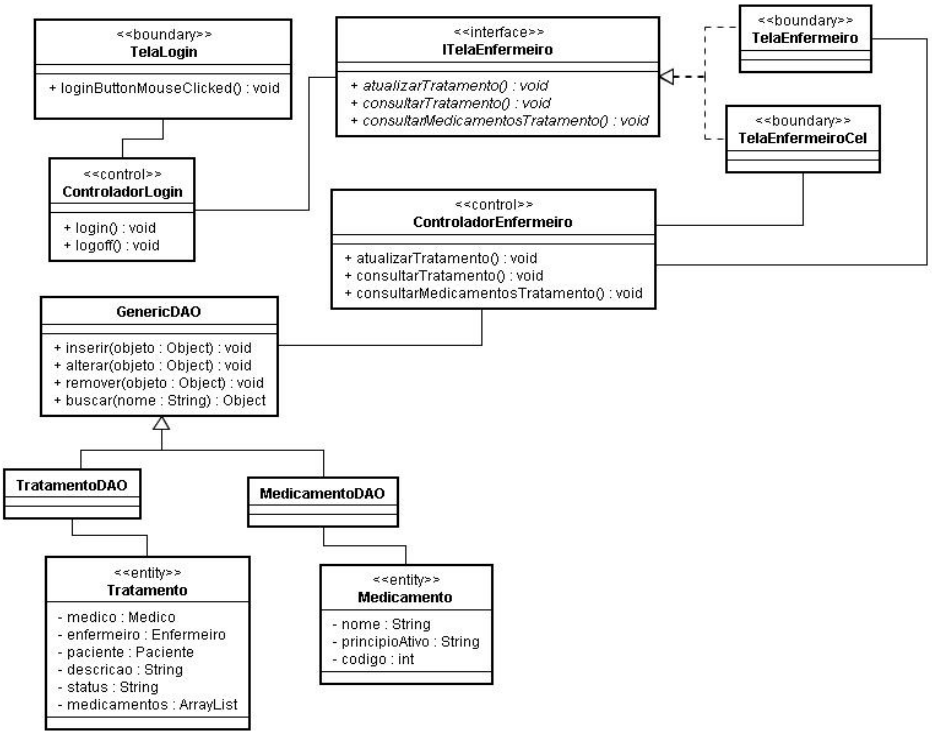

Figura 23 - Diagrama de classes do sistema de informações para medicina ubíqua relacionadas aos casos de uso de Enfermeiro

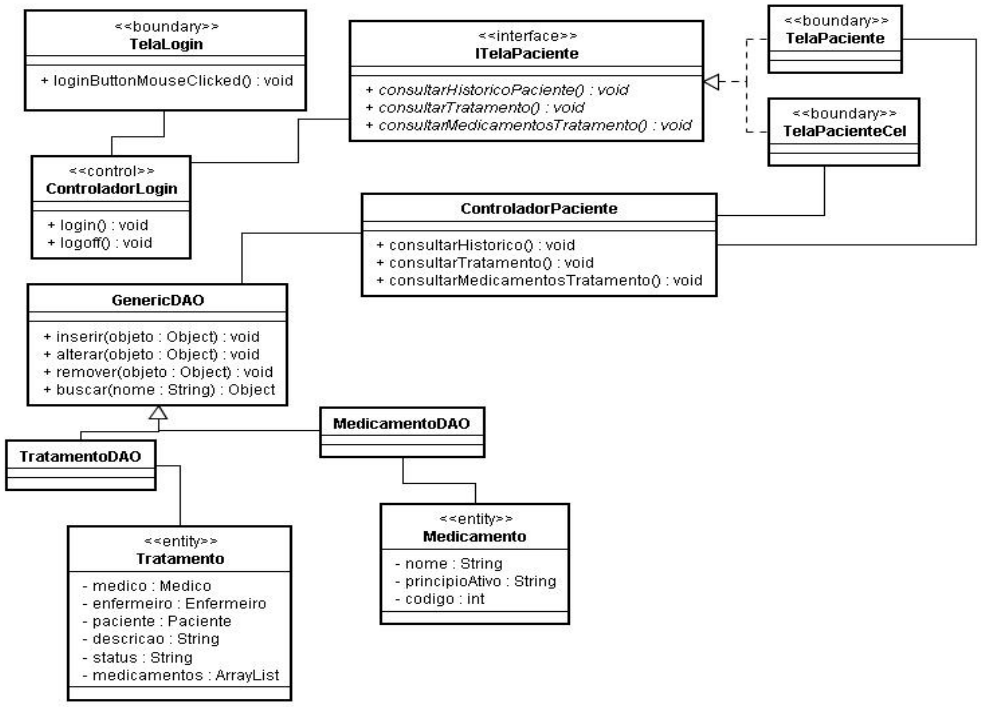

Figura 24 - Diagrama de classes do sistema de informações para medicina ubíqua relacionadas aos casos de uso de Paciente

4.2.2. Identificação de Interesses adaptativos e Definição das Políticas de Adaptação

Os próximos passos do processo proposto consistem em identificar os interesses adaptativos relevantes para a aplicação, bem como as politicas de adaptação, que definem o comportamento adaptativo para cada contexto de execução da aplicação. Como descrito na Seção 4.1, os interesses adaptativos, que irão direcionar a adaptação contextual da aplicação para o estudo de caso são: perfis do usuário; tipo e estado da rede; localização, tipo e estado dos dispositivos e nível de energia do dispositivo.

As políticas de adaptação devem ser definidas através de regras que determinam quais componentes de negócio (módulos funcionais) devem ser usados em cada contexto de execução da aplicação. Na versão atual não são tratadas questões quanto à interdependência dos componentes a serem utilizados, nem quanto a validação das composições geradas. Exemplos de políticas de adaptação para o estudo de caso são representadas através do pseudocódigo da Figura 25:

- *Se perfil admin e dispositivo notebook então*
  usar TelaAdminNotebook;

- *Se ambiente externo e rede conectada então*
  usar Base de Dados Central e Módulo de Criptografia;

Figura 25 – Pseudocódigo de políticas de adaptação

4.2.3. Construção do Modelo de Aspectos

Os próximos passos a serem seguidos de acordo com o processo de desenvolvimento proposto são a construção do Modelo de Aspectos, que é composto pelo conjunto de aspectos a serem utilizados para prover a adaptação da aplicação com base nos interesses adaptativos e a identificação dos componentes de negócio afetados pelos interesses adaptativos, através da construção de uma matriz relacionando os interesses adaptativos com cada componente do Modelo Base (conforme demonstrado na Tabela 1). Com relação ao modelo de aspectos, cabe lembrar que o PACCA provê um conjunto de aspectos concretos *default* para aplicações ubíquas (Figura 13)., portanto já provê um modelo de aspectos *default* para tal domínio de aplicações.

Com relação à identificação dos componentes de negócio afetados pelos aspectos, a Tabela 1 contém a matriz construída para representar tal relação entre os compomentes base e os interesses adaptativos (componentes do modelo de aspectos).

Tabela 1. Matriz de Relacionamento dos interesses adaptativos com os componentes do Módulo Base

	Controlador Login	Controlador Administrador	Controlador Médico	Controlador Enfermeiro	Controlador Paciente	Generic DAO
Perfil de Usuário	√	√	√	√	√	
Nível de Energia	√	√	√	√	√	
Estado da Rede						√
Localização						√
Tipo de Dispositivo	√	√	√	√	√	

Após ilustrar a matriz de relacionamentos, serão apresentandos diagramas que mostram como os aspectos interceptam o Modelo Base. Conforme citado anteriormente, devido à ausência de uma notação padrão, foram adicionados estereótipos às

representações UML (<<Aspect>> para os aspectos e <<AbstractAspect>> para o aspecto abstrato e <<intercepta>> para as relações entre os aspectos e as classes do Modelo Base)

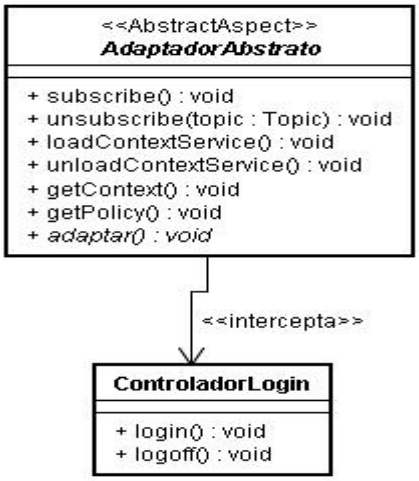

Figura 26 – Classe interceptada pelo *AdaptadorAbstrato*

Na Figura 26, o *AdaptadorAbstrato* faz a interceptação da classe responsável pelo l*ogin* na aplicação, onde antes da efetivação do *login (*método *login)* é realizada a subscrição do endereço *MAC* (*subscribe*) e ao realizar o *logoff* é realizado o cancelamento da subscrição (*unsubscribe*).

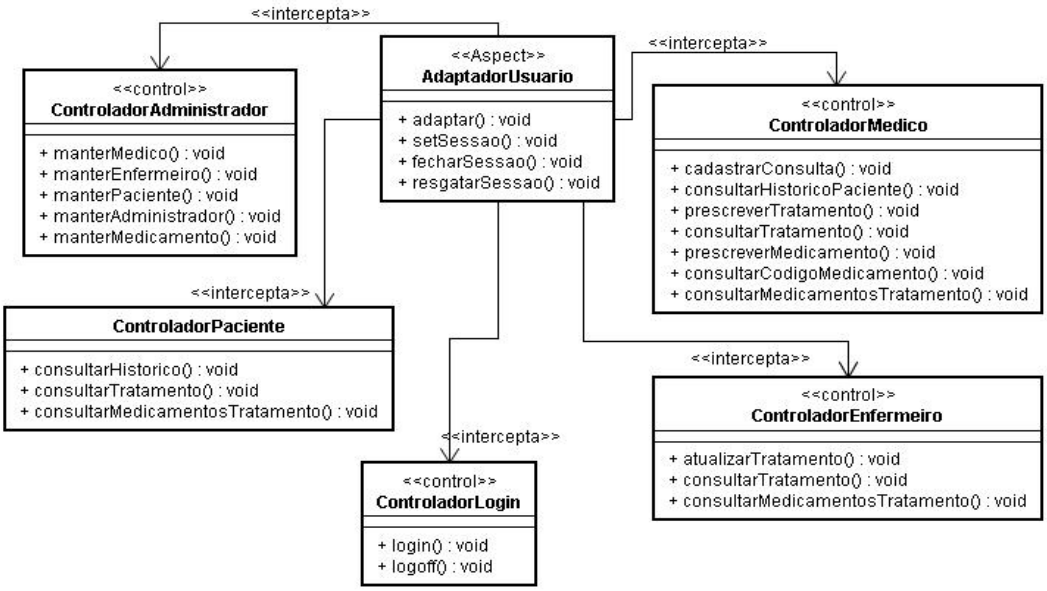

Figura 27 – Classes interceptadas pelo *AdaptadorUsuario*

Na Figura 27, o *AdaptadorUsuario* realiza a interceptação da classe responsável pelo l*ogin* na aplicação para que o usuário seja autenticado e seja iniciada uma nova

sessão ou resgatada uma sessão ainda ativa. Os demais controladores são interceptados devido ao processo de gerenciamento de sessão.

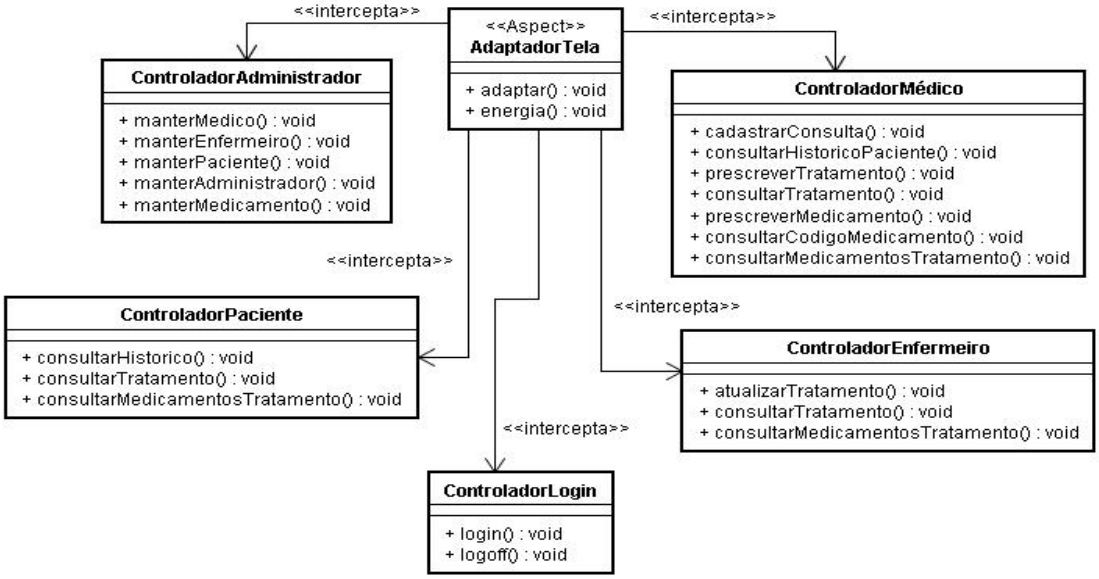

Figura 28 – Classes interceptadas pelo *AdaptadorTela*

Na Figura 28, o *AdaptadorTela* intercepta a classe responsável pelo *login*, pois ao efetuar *login* na aplicação o *AdaptadorTela* se encarrega de adaptar a aplicação de modo que seja carregada a tela adequada ao perfil e ao dispositivo utilizado no *login.* As demais classes de controle (*ControladorAdministrador, ControladorMedico, ControladorPaciente e ControladorEnfermeiro*) são interceptadas pelo *AdaptadorTela* sempre que efetuarem alguma operação, pois assim o *AdaptadorTela* consulta o *Gerente de Contexto* e caso o nível de energia estiver menor que o limite estabelecido a aplicação deve ser adaptada sendo carregada uma interface textual.

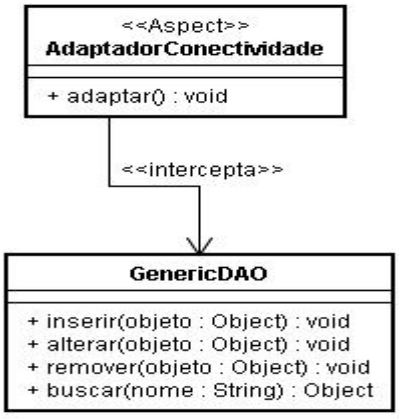

Figura 29 – Classe interceptada pelo *AdaptadorConectividade*

Na Figura 29, o *AdaptadorConectividade* intercepta a classe GenericDAO, Classes DAO são responsáveis pelo processo de persistência das informações. Tal classe é interceptada por ser a responsável pelas interações com as Bases de Dados, logo quando é realizada uma operação para persistir dados, a aplicação se adapta (com a interceptação do *AdaptadorConectividade*) sendo utilizada a Base de Dados adequada ao contexto corrente. O *AdaptadorConectividade* se encarrega de adaptar a aplicação de acordo com a conectividade da rede (*online ou offline*), utilizando a *Base de Dados Local* ou a *Base de Dados Central*, e de acordo com a localização (*ambiente externo*), fazendo uso de criptografia (carga do *Módulo de Criptografia*) no tráfego entre o dispositivo e a *Base de Dados Central.*

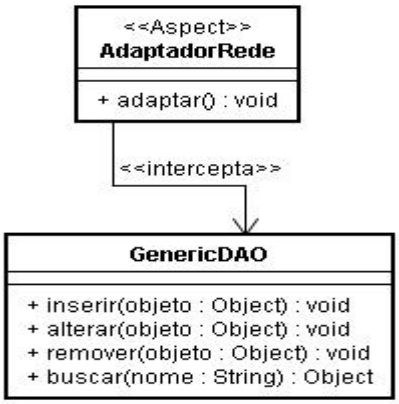

Figura 30 – Classe interceptada pelo *AdaptadorRede*

Na Figura 30, o *AdaptadorRede* intercepta a classe GenericDAO, sendo tal aspecto o responsável por adaptar a aplicação na troca de informações entre a aplicação e a *Base de Dados Central* comprimindo os dados enviados pela rede com base na banda disponível.

Como pode ser notado nos diagramas que ilustram a composição dos aspectos com o modelo base, existem situações onde há vários aspectos interceptando um mesmo módulo funcional da aplicação. Como conseqüência, os métodos contidos nestes módulos são interceptados por mais de um aspecto, podendo se configurar uma situação de conflito entre aspectos. Tal conflito denota a necessidade das interceptações serem ordenadas, ou seja, um aspecto necessita realizar sua interceptação antes de outro, sendo, portanto, imprescindível a utilização de um mecanismo de precedência. Tanto AspectJ quanto JBOSS AOP, apresentam mecanismos de precedência de aspectos.

Em AspectJ, a precedência é declarada através da instrução "**declare precedence: aspecto1, aspecto2;**", dessa forma, o **aspecto1** deve ser analisado antes do

aspecto2. Esta listagem pode possuir quantos aspectos forem necessários para a resolução do conflito. Já no JBOSS AOP, a resolução é realizada através da *tag* **<precedence>** no arquivo de configuração *jboss-aop.xml*, onde também é definida uma listagem de aspectos para serem analisados de acordo com a ordem especificada para a resolução do conflito.

4.3. Construção da Aplicação usando o PACCA – Fases de Projeto, Implementação e Testes

Nesta Seção serão detalhados aspectos relacionados às fases de Projeto, Implementação e Testes, dando sequência ao processo de desenvolvimento definido para a concepção de aplicações adaptativas cientes de contexto.

4.3.1. Definição das Soluções e Tecnologias adotadas e Projeto Arquitetural da aplicação

Como resultado da fase de projeto, ficou definido que para a construção do protótipo do estudo de caso será utilizada a linguagem Java para implementar os componentes da aplicação e AspectJ e JBOSS AO para os componentes orientados a aspectos utilizados. As versões orientadas a objetos e a orientadas a aspectos em AspectJ e JBOSS AOP possuem implementados todos os casos de apresentados na Seção 4.2.1.

Como parte da fase de projeto deve ser especificada a arquitetura da aplicação. Ou seja, devem ser definidos os componentes e suas interfaces, e deve ser especificado ainda onde cada componente será instalado (nós de processamento). O diagrama de implantação da Figura 31 ilustra como os módulos estão dispostos em cada um dos nós (Cliente e Servidor) e a sua respectiva interação.

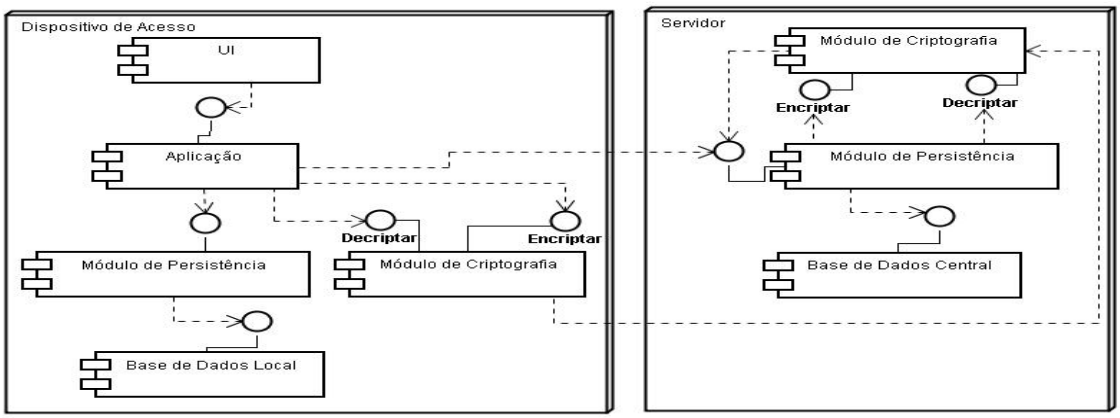

Figura 31 - Diagrama de implantação do sistema de informações para medicina ubíqua

4.3.2. Implementação dos componentes da aplicação

O passo seguinte para implementar o estudo de caso consiste no desenvolvimento dos componentes de interface gráfica e da lógica de negócio da aplicação, previamente modelados, utilizando uma abordagem modular (possivelmente baseada em componentes, conforme apresentado na Figura 31).

Os componentes da aplicação são: *UI (User Interface*, que engloba todas as classes de Fronteira <<Boundary>>), *Módulos Funcionais* (representados pelo componente Aplicação), *Módulo de Persistência, Módulo de Criptografia e Bases de Dados Central e Local*. A *UI* é o componente responsável pela apresentação dos diferentes tipos de interfaces, as quais, para serem exibidas, dependerão do perfil do usuário e do respectivo dispositivo de acesso à aplicação. Adicionalmente, há um módulo de interface textual (*TEXT-INTERFACE*) que pode ser necessário em alguns contextos de execução. As *GUI´s (Graphic User Interface)* e a *TEXT-INTERFACE (*tipos de UI*)* são encarregadas de promover a interação dos usuários com os componentes funcionais da aplicação. Os *Módulos Funcionais* representam a lógica da aplicação (classes responsáveis por realizar *login*, prescrever/consultar medicamento, etc.).

O *Módulo de Persistência* representa o componente da aplicação que é encarregado da persistência das informações nas *Bases de Dados Central* ou *Local*, representado pelas classes DAO (*Data Access Object)*, que implementam o padrão DAO de persistência de dados, cujo objetivo é separar a lógica da aplicação das regras de acesso às Bases de Dados, sendo todas as operações relativas a persistência executadas por classes DAO. Ele também interage com o *Módulo de Criptografia* em situações onde o tráfego necessita de codificação.

Nesses casos, o *Módulo de Criptografia* contido no lado servidor funciona como uma camada acima da Base de Dados realizando os procedimentos de decodificação para o armazenamento das informações na *Base de Dados Central* e codificação para o envio das informações pela rede. Esse módulo foi usado no estudo de caso apenas para exemplificar uma possível funcionalidade que pode ser ativada sob demanda, dependendo do contexto. É importante ressaltar que, para implementar uma solução de segurança completa, outros requisitos se fazem necessário, como por exemplo, ativar e desativar túneis SSH, dentro outros. O contexto ativador da criptografia é representado por regiões simbólicas de ambientes externos ao Hospital. Cabe mencionar que a

definição de ambientes “seguros” foi uma abstração, apenas para fins de ilustrar a carga dinâmica, e não foram consideradas questões como sincronização, necessária para se usar criptografia. Esse módulo pode ser incluído como parte de um serviço de segurança mais abrangente (por exemplo, [PIRMEZ et al., 2008]), o qual além de criptografia, deve garantir a confidencialidade e integridade da informação através de mecanismos adicionais.

A *Base de Dados Central* é o repositório de todas as informações relativas aos usuários da aplicação proposta como estudo de caso. Todavia, também existe a *Base de Dados Local* usada com o objetivo de armazenar informações temporariamente em casos onde, no momento do processo de armazenamento, não haja disponibilidade de sinal de rede entre o dispositivo e a *Base de Dados Central*. Nesses casos, ao ser restabelecido o sinal de rede, as informações são transferidas para a *Base de Dados Central*.

Ainda em relação ao ambiente de implementação, no *Módulo de Criptografia* utilizou-se o algoritmo de criptografia simétrica Triple-DES [COOPERSMITH; JOHNSON; MATYAS, 1996]. O Sistema Gerenciador de Banco de Dados PostegreSQL [POSTGRESQL, 2008] foi utilizado como *Base de Dados Central* e para a *Base de Dados Local* foi utilizada uma estrutura de arquivos.

4.3.3. Especificação das Políticas de Adaptação em XML

O próximo passo no desenvolvimento de uma aplicação consiste na especificação em XML das políticas de adaptação definidas. A Figura 32 ilustra as políticas definidas na Figura 25, agora em XML.

```
<politica name="adaptarTela">
        <contexto>
            <Perfil>admin</Perfil>
            <Dispositivo>Notebook</Dispositivo>
        </contexto>
        <decisao>
           <carregar>ui.TelaAdminNotebook</carregar>
        </decisao>
 </politica>
```

```
<politica name="conectividade">
        <contexto>
                <Online>true</Online>
                <Criptografia>true</Criptografia>
        </contexto>
        <decisao>
                <carregar>banco.BancoRemoto</carregar>
                <carregar>seguranca.ClienteCript</carregar>
        </decisao>
 </politica>
```

Figura 32 – Políticas de adaptação em XML

Essas políticas são armazenadas no *Repositório de Políticas* e lidas pelo *Gerente de Políticas de Adaptação* para gerar a especificação de classes a serem carregadas de acordo com o contexto corrente, sendo, portanto, responsáveis por guiar o processo de adaptação.

4.3.4. Definição dos join points, pointcuts e Encapsulamento do comportamento adaptativo ciente de contexto nos advices

Esta fase consiste da etapa de desenvolvimento na qual é realizada a implementação propriamente dita dos elementos responsáveis pela orientação a aspectos, cujos passos consistem: (i) na especificação dos pontos de junção (join points), e (ii) no encapsulamento do comportamento adaptativo definido em um ou mais aspectos, ou seja, definição dos pointcuts (conjunto de pontos de junção) e dos adendos (advices).

A Figura 33 ilustra a definição de um *pointcut*, em JBOSS AOP, que intercepta a execução do método *login.* Os *pointcuts* são definidos de forma declarativa no arquivo jboss-aop.xml do JBOSS AOP, permitindo que tais *pointcuts* possam ser modificados sem a necessidade de recompilação do código. A definição de *pointcuts* e *advices* em AspectJ pode ser vista na Seção 2.1.1.1.

```
<pointcut name="AdaptarTela" expr = "execution(public * *.*-> login(..))"/>
```

Figura 33 - *Pointcut AdaptarTela* que define a interceptação do método de login utilizando JBOSS AOP

Além dos *pointcuts*, é necessário definir os adendos (*advices)* que implementam o comportamento adaptativo a ser inserido no local determinado pelo *pointcut*. O código contido nos *advices* (representado pelo método *adaptar ()* e suas diversas variações específicas para cada interesse adaptativo) gerencia o processo de adaptação, obtendo o contexto corrente do *Gerente de Contexto*, e repassando-o ao *Gerente de Políticas de Adaptação* para obtenção da política (ou políticas) que será ativada conforme o contexto. Como mencionado anteriormente, um conjunto diferente de regras de adaptação será testado de acordo com o método de negócio interceptado (ou seja, de acordo com o interesse adaptativo), pois existem políticas para situações específicas. O conjunto de regras de interesse para cada interesse adaptativo será especificado através do método *setOperacao()*. Tal solução torna a busca pela política de adaptação mais eficiente, visto que será consultado apenas um subconjunto de todas as políticas contidas no *Repositório de Políticas de Adaptação*. Em seguida, a política retornada é enviada ao Compositor que solicita ao *Gerente de Execução* o carregamento dos módulos requisitados para efetuar a composição. O código apresentado na Figura 34 é definido no arquivo *jboss-aop.xml* e mostra a associação de um *advice* com um *pointcut* indicando que o código de adaptação, contido no método *adaptar* implementado no *AdaptadorTela*, será disparado assim que houver a execução do *pointcut adaptarTela* definido na Figura 33.

```
<bind pointcut =“AdaptarTela”>
        <advice name=“adaptar” aspect= “adaptacao.AdaptadorTela”/>
</bind>
```

Figura 34 - *Advice* associado ao *pointcut* AdaptarTela

A Figura 35 apresenta o código do método que implementa o *advice adaptar*, em JBOSS AOP, no aspecto a*daptadorTela*. Tal código gerencia o processo de adaptação adequado ao contexto e injeta o comportamento adaptativo no ponto da aplicação interceptado, o qual necessita que a tela seja adaptada. O método descrito na Figura 35 está associado ao *pointcut adaptarTela* através da declaração exibida na Figura 34.

```
1   public Object adaptar(Invocation invocation) {
2       try{
3            return invocat ion.invokeNext();
4        }
5        catch (Throwable e) {
6             e.printStackTrace();
7             return null;
8        }
9        finally {
10            GerenteContexto.setOperacao("adaptarTela");
11            Contexto contexto = GerenteContexto.getContexto();
12            ArrayList<String> especificacao= GerentePoliticas.escolherPolitica(contexto);
13           Compositor.composicao(especificacao);
14       }
15  }
```

Figura 35 - Código do método que implementa o *advice* adaptar (para *adaptarTela*) utilizando JBOSS AOP

Como o comportamento adaptativo depende do contexto, nos aspectos também é tratada a manipulação de contexto provida pelo *CIS* do MOCA. Para usar os serviços do MOCA, é preciso inicialmente realizar a subscrição do endereço MAC do dispositivo de acesso junto ao *CIS*. Portanto, um dos *pointcuts* definidos especifica a criação do join point correspondente à interceptação do método da aplicação responsável pelo *login* do usuário. O *advice* definido para esse ponto realiza a autenticação do *login* e em seguida recupera da *Base de Dados Central*, o *MAC* do dispositivo de acesso do usuário, o qual deve estar previamente cadastrado, e envia uma subscrição ao CIS. A subscrição tem o propósito de repassar ao *Gerente de Contexto* qualquer mudança percebida no contexto do dispositivo.

Dessa forma, o código da aplicação fica completamente livre de qualquer manipulação de contexto, incluindo o registro nos serviços de provisão de contexto. Com a subscrição do dispositivo no *CIS*, o monitor do MOCA passa a enviar periodicamente informações sobre o contexto do dispositivo para o *CIS*, que por sua vez notifica o *Gerente de Contexto* para que o mesmo atualize suas informações. Essas informações serão sempre consultadas para a realização de qualquer processo de adaptação. Em todos os *advices* definidos são realizadas consultas ao *Gerente de Contexto* e com o contexto atual são realizadas consultas às políticas de adaptação definidas para este contexto. Portanto, cada aspecto definido consiste de um conjunto de *advices* e *pointcuts*, onde se especificam os comportamentos adaptativos e os pontos onde serão injetados esses comportamentos.

Para possibilitar o comportamento adaptativo em tempo de execução, o *Gerente de Execução* do PACCA encarrega-se da carga dinâmica dos componentes necessários. Para tal, ele utiliza os mecanismos disponíveis na linguagem Java para carregar e

descarregar dinamicamente as classes e realizar as composições de módulos de acordo com o contexto. Em relação à descarga dinâmica, antes de iniciar a composição é feita a invocação do método *descarregarModulos* do *Gerente de Execução* que realiza uma chamada explícita ao *Garbage Collector* de Java para retirar da memória os módulos que não são necessários ao contexto corrente. Neste trabalho não tratamos a composição em nível de interface. Dessa forma, consideramos que as interfaces de todos os módulos que precisam se conectar são compatíveis e qualquer composição especificada é considerada válida.

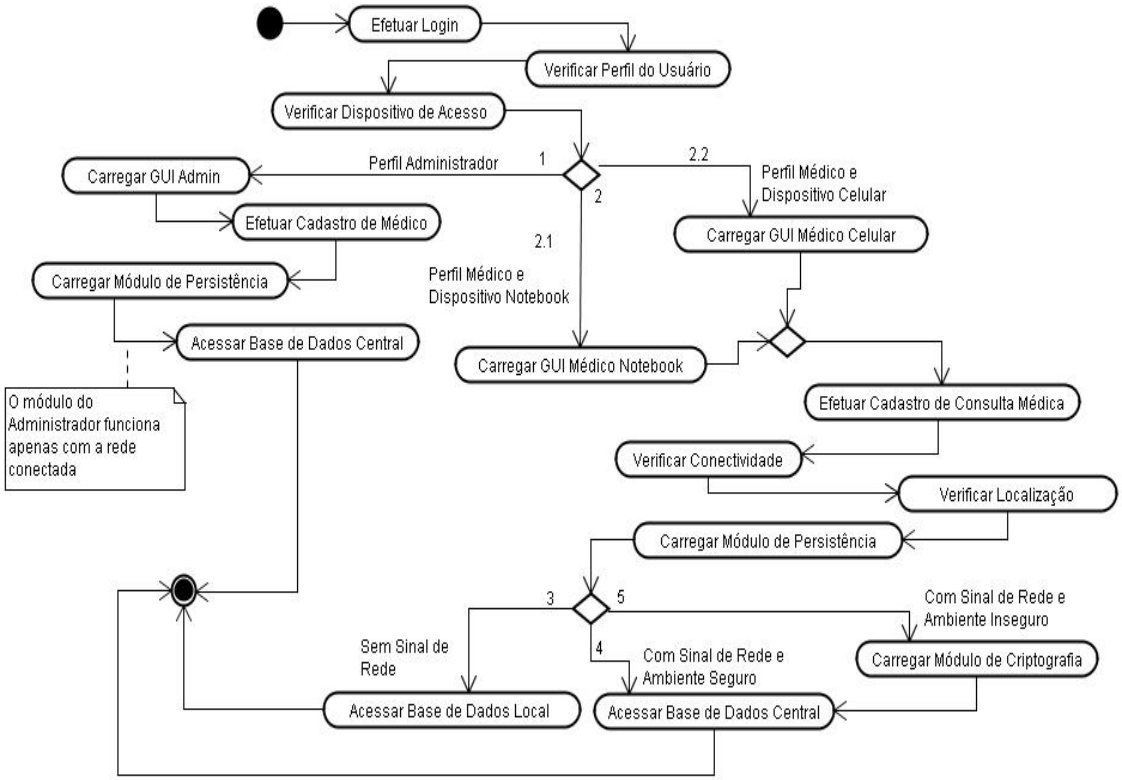

Figura 36 - Composição dos Módulos em Execução

Na Figura 36 tem-se um exemplo de possíveis composições dos módulos em execução de acordo com cinco diferentes situações contextuais, no âmbito do estudo de caso considerado.

Na ***situação 1,*** o perfil é identificado como sendo *Administrador*, implicando que será carregada a *GUI de Administração*. Essa *GUI* para realizar, por exemplo, uma operação de cadastro de Médico, necessitará que o *Gerente de Execução* carregue o *Módulo de Persistência* e em seguida carregue o módulo de conexão com a *Base de Dados Central*, para o armazenamento das informações de cadastro. No estudo de caso

apresentado, as tarefas de administração são realizadas somente a partir de ambiente interno e com sinal de rede.

Na ***situação 2***, identificamos o perfil Médico, o qual faz com que o *Gerente de Execução* carregue a *GUI Médico*. Porém, a *GUI* adequada ao contexto depende do dispositivo de acesso do médico, podendo ser carregada a *GUI Médico Notebook* (**situação 2.1***)* ou a *GUI Médico Celular (***situação 2.2***)*. Para realizar, por exemplo, uma operação de cadastro de consulta, é necessário que seja carregado o *Módulo de Persistência*. A partir de então, têm-se três novas situações com base nas informações de localização e conectividade, sendo a ***situação 3*** caracterizada pela ausência de sinal de rede. Como conseqüência, será carregado o módulo de conexão com *Base de Dados Local*, que armazenará temporariamente essas informações e posteriormente repassará as informações para o *Base de Dados Central*.

Na ***situação 4*** há a presença de sinal de rede e o ambiente é interno. Nesse momento, o *Gerente de Execução* carregará o módulo de conexão com a *Base de Dados Central*. Já na ***situação 5***, existe o sinal de rede, porém o ambiente é externo, fazendo com que seja necessário carregar adicionalmente o *Módulo de Criptografia*.

4.3.5. Verificação e Validação

Por fim, temos a fase de testes onde são realizadas as etapas de verificação e validação (V & V) da aplicação desenvolvida, a qual se destina a mostrar que um sistema está de acordo com suas especificações e que atende às expectativas do cliente.

Os testes se dividem em 5 estágios e evoluem em conjunto com a implementação do sistema, conforme definido em [SOMMERVILLE, 2000]:

- Testes de Unidade: são testados os componentes individuais para garantir que eles operam corretamente, ou seja, cada componente é testado independentemente dos outros módulos.
- Teste de Módulo: Um módulo é uma coleção de componentes dependentes, como uma classe, um tipo abstrato de dados ou algum conjunto mais amplo de procedimentos e funções. Um módulo abrange componentes relacionados, podendo assim ser testado sem outros módulos do sistema.
- Teste de Subsistema: Essa fase envolve testar conjuntos de módulos que foram integrados em subsistemas. O processo de teste de subsistemas deve, portanto, se concentrar na detecção de erros de interfaces de módulos.

- Teste de Sistema: Os subsistemas são integrados para constituírem o sistema. Esse processo dedica-se a encontrar erros que resultem de interações não previstas entre subsistemas e problemas de interfaces de subsistemas. Ele também se ocupa de validar que o sistema cumpra seus requisitos funcionais e não funcionais.
- Teste de Aceitação: Esse é o estágio final do processo de testes, antes que o sistema seja aceito para uso operacional. O sistema é testado com os dados fornecidos pelo cliente, no lugar dos dados de testes simulados. O teste de aceitação pode revelar erros e omissões na definição de requisitos do sistema, porque os dados reais “exercitam” o sistema de modo diferente ao processo realizado nas simulações.

No desenvolvimento do estudo de caso utilizando o PACCA e o processo de desenvolvimento de aplicações cientes de contexto adaptativas proposto nesse trabalho, apenas o teste de aceitação não foi realizado, visto que o PACCA nas versões desenvolvidas caracteriza-se como um protótipo e não está pronto para operar em um ambiente real de Computação Ubíqua.

5. Avaliação

O PACCA possui dois principais objetivos específicos. Um deles é aumentar a modularidade das aplicações adaptativas cientes de contexto, obtendo assim, as vantagens decorrentes da modularização, ou seja, produz aplicações mais fáceis de compreender, manter, gerenciar e evoluir. Outro objetivo é reduzir os requisitos mínimos de memória (*footprint*) das aplicações, mantendo em memória somente o código necessário em cada contexto de execução. Tal benefício é crucial para a computação ubíqua, caracterizada por dispositivos com recursos computacionais limitados.

O processo de avaliação do PACCA consistiu em comparar duas versões da aplicação descrita no estudo de caso. A primeira versão foi construída utilizando puramente o paradigma de Orientação a Objetos (versão OO) e sem composição dinâmica de software. A outra versão foi construída utilizando Orientação a Aspectos (versão OA) e composição dinâmica. Na versão OO, o único componente do PACCA implementado foi o *Gerente de Contexto.* O código relativo ao comportamento adaptativo ciente de contexto foi embutido no código da aplicação, encontrando-se portanto espalhado pelos módulos funcionais da aplicação. A implementação da versão OO foi elaborada com dois intuitos: (i) comprovar a característica transversal do comportamento adaptativo ciente de contexto, justificando o seu encapsulamento como um *Aspecto*; e (ii) servir de base para a comparação quantitativa do código gerado nas duas versões. Na implementação OA para a avaliação usou-se AspectJ, pois como no JBoss AOP não há distinção entre classes e aspectos, o uso das ferramentas de medição OA atualmente é dificultado.

5.1. Métricas de Modularização

A construção de código modular reduz a complexidade das aplicações e possibilita o desenvolvimento dos módulos de forma independente, com cada módulo concentrando e tratando um requisito específico. No presente trabalho, os principais indicadores usados para identificar as melhorias na modularização obtida com a abordagem OA são fatores de qualidade de software bem conhecidos: gerenciabilidade, mantenabilidade e compreensibilidade.

Gerenciabilidade refere-se à facilidade de desenvolver e compor componentes de software. Aplicações que empregam modularização podem ser desenvolvidas mais facilmente, haja vista que cada módulo pode ser implementado de forma independente.

A mantenabilidade indica a facilidade com que modificações e extensões podem ser feitas na aplicação. Modificações feitas em código modular afetam um número menor de módulos e, portanto, há um aumento na flexibilidade da aplicação

A compreensibilidade diz respeito a como os desenvolvedores entendem a aplicação. A separação de interesses em módulos distintos provê aos desenvolvedores uma visão isolada de todas as funcionalidades relacionadas entre si, tornando a aplicação mais fácil de compreender.

Em termos de métricas de avaliação para os atributos de qualidade mencionados, a compreensibilidade de uma aplicação pode ser vista como uma medida do nível de separação de interesses e da complexidade de um módulo. Se um módulo é altamente complexo e lida com múltiplos interesses, o esforço para entender esse módulo é grande. No presente trabalho, as métricas LOC (*Lines of Code*), RFM (*Response for a Module)* e WOM (*Weighted Operations in Module*) foram utilizadas para avaliar a compreensibilidade. A mantenabilidade é uma medida do grau de modificabilidade do código base e é negativamente afetada pelas dependências entre os módulos.

No trabalho, foram usadas as métricas de acoplamento e coesão CBM (*Coupling Between Modules*), CMC (*Coupling on Method Call*) e LCO (*Lack of Cohesion in Operations*), para indentificar a existência de dependências entre os módulos e, portanto, para avaliar a mantenabilidade das aplicações. Finalmente, a gerenciabilidade pode ser avaliada por métricas que identificam a independência de um módulo, habilitando-o a ser desenvolvido e modificado de forma isolada. Portanto, as métricas de acoplamento também são adequadas para avaliar o atributo de gerenciabilidade. A ferramenta utilizada para obter todas as métricas no presente trabalho foi a *AOP Metrics* [AOP Metrics 2008].

5.2. Resultados

As Figuras 37, 38, 39, 40 e 41 apresentam os resultados da avaliação. As medições foram realizadas no módulo responsável pelo *Login* (indicado como *Módulo Login)* e em todos os módulos funcionais relativos aos perfis do *Administrador* (denotado como *Módulo Administrador),* do *Médico (Módulo Médico),* do *Enfermeiro*

(Módulo Enfermeiro) e do *Paciente (Módulo Paciente)*, que são todos os módulos afetados pelo comportamento adaptativo transversal.

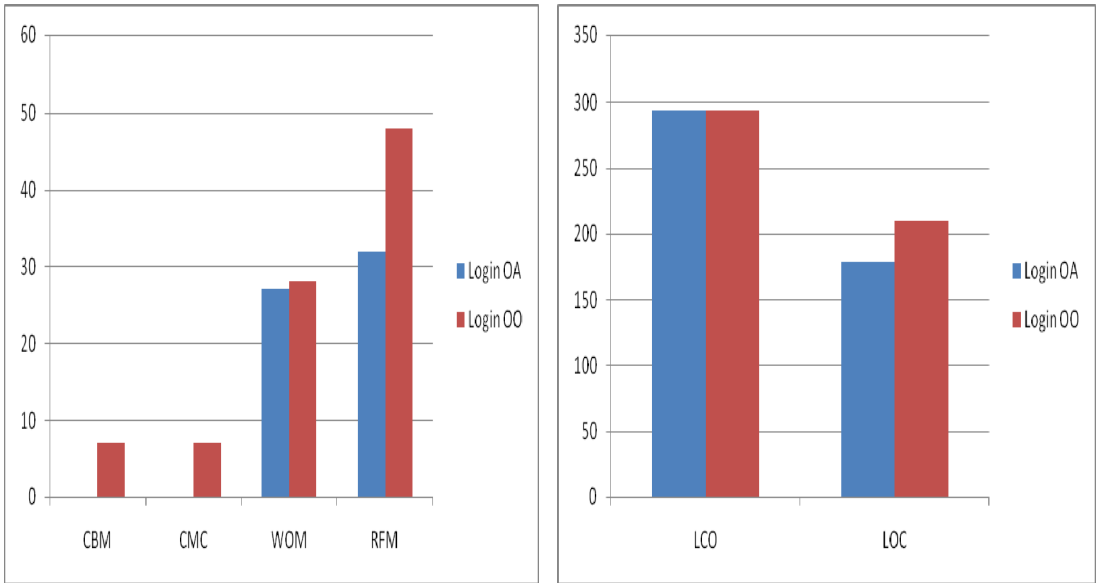

Figura 37 - Métricas de comparação entre as versões OA e OO – Módulo *Login*

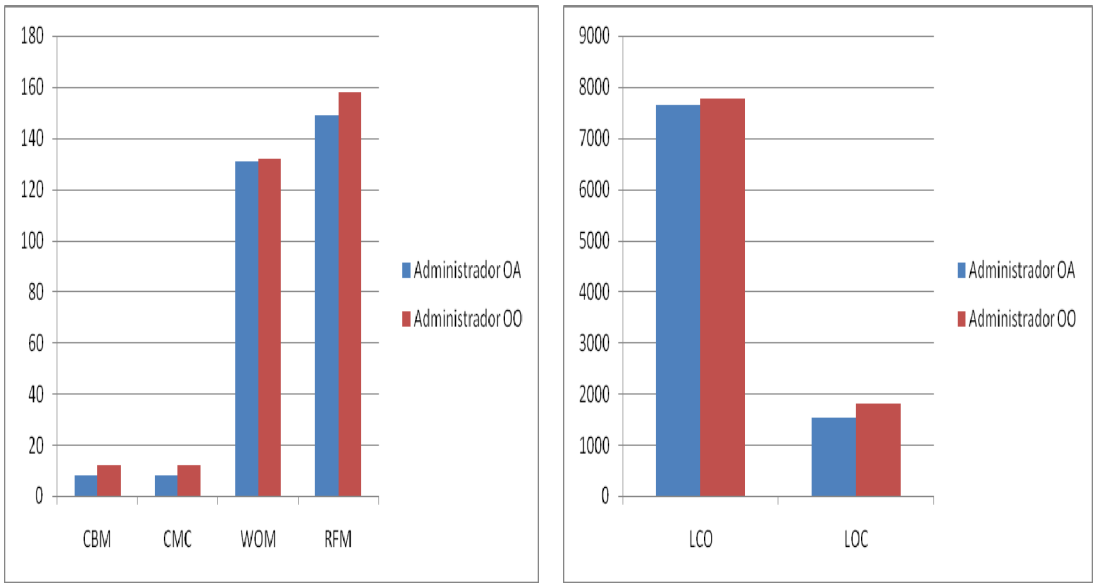

Figura 38 - Métricas de comparação entre as versões OA e OO – Módulo Administrador

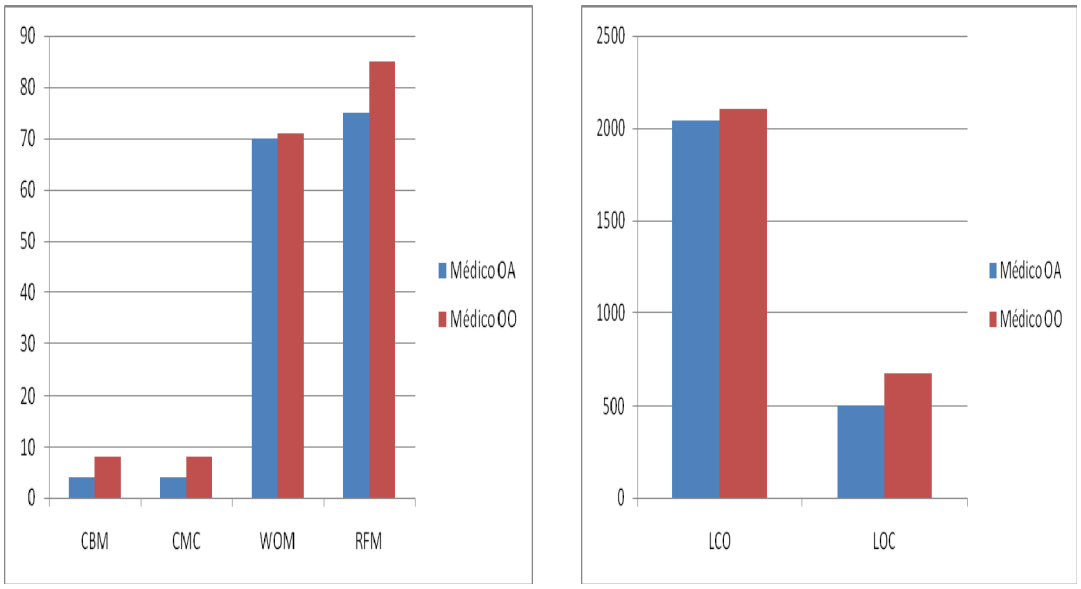

Figura 39 - Métricas de comparação entre as versões OA e OO – Módulo Médico

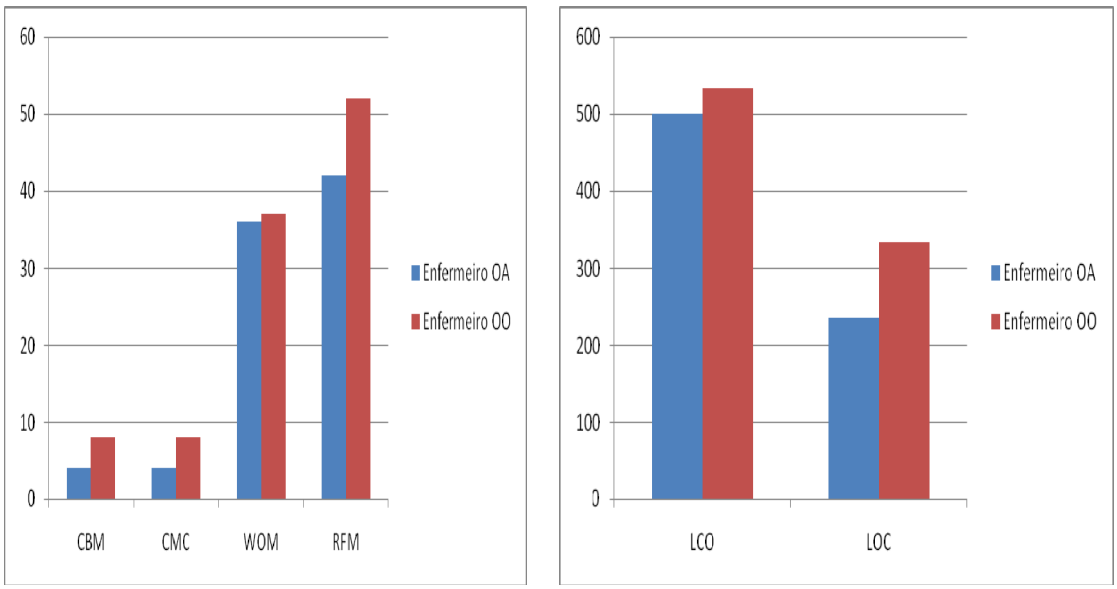

Figura 40 - Métricas de comparação entre as versões OA e OO – Módulo Enfermeiro

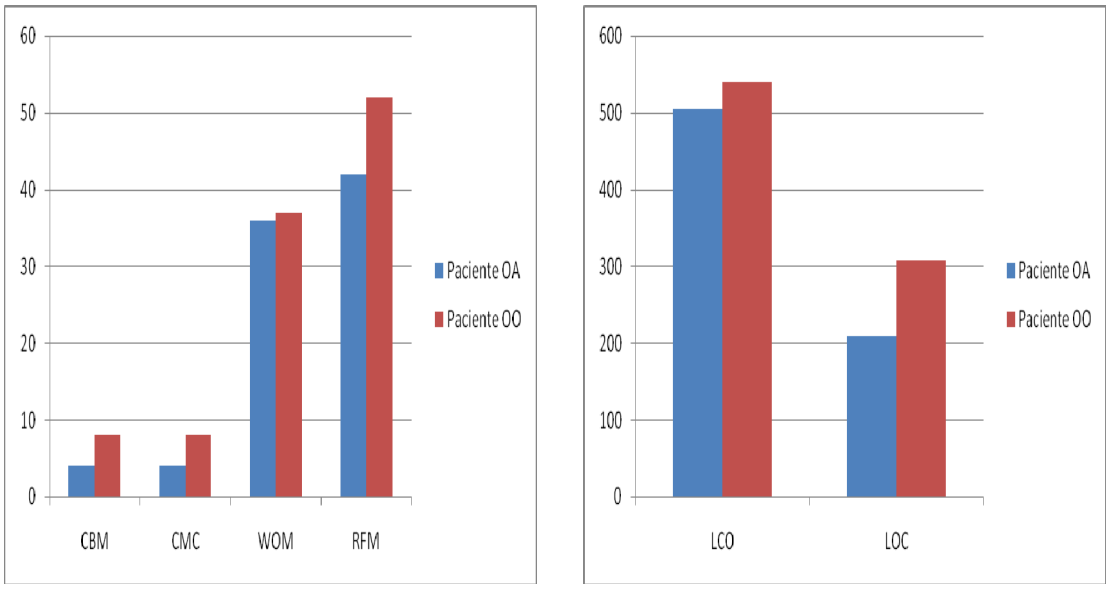

Figura 41 - Métricas de comparação entre as versões OA e OO – Módulo Paciente

A métrica CBM indica o acoplamento entre os módulos, ou seja, a dependência entre os mesmos, e seu valor denota a dependência em relação a um determinado número de módulos. A presença de muitos acoplamentos entre os módulos reduz a possibilidade de reuso e dificulta a modificação da aplicação, tornando mais trabalhoso o processo de evolução do software. Pelos resultados observados nas Figuras 37, 38, 39, 40 e 41 percebe-se que, com a retirada do comportamento adaptativo do código base, há uma redução de cerca de 33% no acoplamento identificado na versão OO em relação à versão OA para o módulo *Administrador*, uma redução de 50% para o módulo *Médico*, uma redução de 50% para o Módulo *Enfermeiro* e uma redução de 50% para o Módulo *Paciente*. Já para o módulo *Login*, a redução foi de 100%, pois com a abordagem orientada a aspectos, esse módulo não realiza mais chamadas a outros módulos. Todo o processo de adaptação inicial da aplicação fica a cargo dos aspectos.

A melhoria mencionada com o uso da abordagem OA foi resultado do processo de adaptação passar a ser encapsulado nos aspectos. Observa-se também que a vantagem em relação ao acoplamento é maior quanto maior o número de interesses adaptativos manipulados. Os módulos que implementam as funcionalidades do *Médico*, do *Paciente* e do *Enfermeiro* estão acoplados a um número menor de outros módulos. A redução em termos percentuais nesses módulos apresenta um impacto maior em comparação ao módulo *Administrador*, uma vez que em termos absolutos, a redução é a mesma devido ao processo de adaptação do PACCA retirar o acoplamento dos mesmos

módulos, pois todos os módulos analisados manipulam o mesmo número de interesses adaptativos.

A métrica CMC indica o número de módulos ou interfaces declarando métodos que são possivelmente chamados por um dado módulo. O uso de uma grande quantidade de métodos de vários módulos diferentes indica que a funcionalidade de um dado módulo não pode ser facilmente isolada das outras e isso implica um alto acoplamento.

Os resultados produzidos para a métrica CMC foram idênticos aos de CBM, pois no presente trabalho, a dependência entre os módulos existe apenas em termos de métodos, não havendo dependência em termos de campos (atributos). A dependência em termos de atributos é dada pela métrica CFA (*Coupling on Field Access*). Em sistemas OO, o valor de CFA é normalmente 0 (zero) devido ao encapsulamento. Por outro lado, em sistemas AO os aspectos, através de seu comportamento intrusivo, podem depender de um determinado campo para realizar uma operação. No projeto do PACCA, porém, não foi inserido esse tipo de dependência.

A métrica RFM refere-se à resposta de um módulo, e indica a possibilidade de comunicação entre os módulos, ou seja, métodos e *advices* geralmente executam em resposta a uma mensagem recebida por um dado módulo. Essa métrica em sistemas OA deve levar em consideração as responsabilidades implícitas que são disparadas quando um *pointcut* intercepta uma operação de um dado módulo. Como foi mostrada pelas avaliações, a abordagem orientada a aspectos produz uma redução da chamada de métodos no código base, sendo tal redução de cerca de 6% para o módulo *Administrador,* 12% para o módulo *Médico,* 20% para os módulos *Enfermeiro* e *Paciente* e 33% para o módulo *Login.*

A métrica LCO representa a falta de coesão entre as operações dos módulos. A coesão é um forte indicador de boa modularidade, visto que em uma abordagem OO envolvendo conceitos transversais, os módulos acabam desempenhando tarefas sobre as quais não têm responsabilidade, devido aos interesses transversais estarem espalhados pelo código, portanto aumentando a falta de coesão. Nas Figuras 37, 38, 39, 40 e 41 pode ser vista uma redução de cerca de 2% na falta de coesão proporcionada pela abordagem orientada a aspectos para o módulo *Administrador*, de cerca de 3% para o módulo *Médico*, de cerca de 7% para os módulos *Enfermeiro* e *Paciente*. Já a retirada do comportamento adaptativo do módulo *Login* não promoveu nenhuma redução na falta de coesão entre as operações do módulo, visto que o processo de adaptação

realizado no módulo de *Login* não reduz o acoplamento com operações da própria classe. Isso ocorre, principalmente, devido ao módulo *Login* na versão orientada a objetos não possuir código adaptativo de gerenciamento de sessão, o qual invoca operações do módulo com o intuito de resgatar ou armazenar uma sessão, código esse contido na versão orientada a objetos dos outros módulos.

Já em relação à métrica WOM a redução em todos os módulos é de 1%. Tal métrica é responsável pela medição do esforço para desenvolvimento e manutenção de um dado módulo e o seu valor denota a quantidade de operações no módulo. Módulos com muitas operações são, em geral, mais específicos da aplicação limitando o potencial de reuso. Os módulos avaliados apresentaram pequena redução nesta métrica por serem todos específicos da aplicação.

Em relação ao número de linhas de código (LOC), verifica-se uma redução de cerca de 16% proporcionada pela abordagem orientada a aspectos para o módulo *Administrador,* de cerca de 26% para o módulo *Médico,* de cerca de 30% para o módulo *Enfermeiro,* de cerca de 32% para o módulo *Paciente* e de cerca de 15% para o módulo *Login*. Dessa forma, verifica-se que a retirada do código de adaptação espalhado pela lógica da aplicação e a sua concentração no aspecto reduz o esforço de programação pelo reuso do código contido no aspecto.

Outra métrica a ser analisada é CAE (*Coupling on Advice Execution*), a qual indica o número de aspectos contendo *advices* disparados pela execução de métodos em um módulo, ou seja, quantos aspectos interceptam tal módulo. Os Módulos *Administrador, Paciente, Enfermeiro e Médico* apresentam como resultado para esta métrica o valor 2 (dois) e o Módulo *Login* o valor 3 (três), conforme ilustrado nas figuras apresentadas na Seção 4.2.3.

Neste momento, analisando não apenas módulos interceptados por aspectos e sim a aplicação em sua totalidade, verifica-se que há uma redução de cerca de 7% no número de linhas de código escritas, sendo reduzidas de 8.456 linhas na versão orientada a objetos para 7.900 linhas na versão orientada a aspectos, reduzindo o esforço de programação.

O tamanho do código produzido foi analisado com o intuito de demonstrar que a POA pode ser usadas para desenvolver aplicações adaptativas que requerem uma sobrecarga baixa em termos de armazenamento em disco e *footprint* de memória. Na Tabela 2 é apresentada uma medição do tamanho do código armazenado em disco e do número de classes do código base e do código OA do PACCA. Com os resultados

aliados a medida de LOC, é possível verificar que os aspectos são compostos por pequenas porções de código que se encarregam da adaptação. Dessa forma, constata-se que a adoção de técnicas de DSOA permite a produção de aplicações com uma maior manutenibilidade, devido ao código de adaptação estar concentrado em uma porção menor de código.

Tabela 2. Tamanho em disco e número de classes do código base e do código OA

Implementação	Tamanho do Código em disco	Classes
Código Base	719 KBytes	28 Classes
Código Orientado a Aspectos	143 Kbytes	7 Classes + 5 Aspectos

Com relação ao consumo de memória, foram realizadas medições em dois momentos: (a) logo após ser efetuado o *login* e ser carregado o componente de UI adequado ao perfil do usuário e (b) ao ser invocado o método *prescrever medicamento* para um contexto de rede *conectada* e ambiente de execução *externo*. Esse segundo momento, que representa a situação 5 da Figura 36, foi escolhido pela necessidade de carregamento de vários módulos ao mesmo tempo, exigindo um consumo grande de memória.

A Tabela 3 apresenta as medições de consumo de memória (em *Megabytes*) para as situações descritas. Para essas medições foram utilizados os métodos *freeMemory* (fornece a memória disponível dentre a alocada pela *Java Virtual Machine* - JVM) e *totalMemory* (fornece o total de memória alocado pela JVM para a aplicação) da classe *Runtime* de Java. Portanto, o consumo de memória é calculado pela subtração de *totalMemory* por *freeMemory*.

Tabela 3. Consumo de Memória das Versões OO e com Composição dinâmica

	Situação (a)	Situação (b)
PACCA – Versão Monolítica	2.16MB	3.08MB
PACCA – Versão com composição Dinâmica	1.90MB	2.77 MB

Conforme visto na Tabela 3, percebe-se que a versão dotada de composição dinâmica de módulos provê uma economia do uso de memória em relação à versão OO monolítica. A fim de avaliar o impacto da composição dinâmica em relação ao tempo de resposta da aplicação, foram ainda conduzidas medições conforme uma métrica de *overhead* de carregamento dos módulos. Para realizar essas medições, foi utilizado o

método *currentTimeMillis*() da classe *System* do Java, que retorna o tempo corrente em milissegundos (ms).

A Tabela 4 apresenta os resultados dessas medições. Com base nos dados dessa tabela, verificamos que o *overhead* de carregamento dos módulos é insignificante para uma aplicação com interação com usuários. Conclui-se assim que é mais vantajoso carregar e descarregar os módulos de forma dinâmica ao invés de mantê-los em memória, haja vista que, dessa forma, consegue-se diminuir a quantidade de memória no dispositivo sem comprometer o desempenho.

Tabela 4. Tempo de carga dos módulos

Módulo	Tempo de carga (ms)
Módulo Administrador	364 ms
Módulo Médico	350 ms
Módulo de Persistência	20 ms
Criação da conexão com a Base de Dados	219 ms
Módulo de Criptografia	32 ms

6. Trabalhos Relacionados

Neste capítulo será realizada uma comparação da abordagem proposta pelo PACCA com outros trabalhos que seguem na linha de pesquisa da Computação Ubíqua, para tanto serão utilizados alguns critérios de avaliação que são os principais pontos abordados pelo PACCA como: a utilização de uma abordagem orientada a aspectos para tratar a adaptação devido ao seu caráter transversal, a utilização de aspectos abstratos permitindo a adição de novos interesses transversais, a definição de um modelo de aspectos *default* para o tratamento de interesses adaptativos comuns às aplicações ubíquas, a definição de um processo de desenvolvimento para aplicações adaptivas cientes de contexto, a utilização de diferentes sistemas de provisão de contexto e a definição de um Gerente de Sessão permitindo a migração entre dispositivos.

Em [MUKHIJA; GLINZ, 2005] foi descrito o *framework* CASA (*Contract-based Adaptative Software Architecture*), capaz de fornecer adaptação dinâmica para aplicações, em resposta às mudanças em seu ambiente da execução. Os mecanismos de adaptação empregados consistem na recomposição ou na substituição dinâmica de componentes da aplicação. A recomposição consiste na mudança entre composições alternativas de uma aplicação em tempo de execução. Por sua vez, a substituição pode ocorrer de duas formas: o componente é substituído depois de terminar sua execução, ou a execução do componente é suspensa e o mesmo é substituído. O CRS (CASA *Runtime System*) é o componente responsável pelas mudanças no ambiente de execução e pela tomada de decisão quanto ao modo de executá-las. No PACCA, diferentemente do CASA, não existe a substituição de componentes; simplesmente um componente é carregado quando necessário e descarregado quando suas funcionalidades já não são mais requeridas, com base no contexto de execução. Isso caracteriza-se por uma simplificação da abordagem utilizada no CASA, visto que o PACCA não trata situações onde seja necessário abortar a execução de um componente para que seja substituído por outro equivalente. O foco do trabalho apresentado em [MUKHIJA; GLINZ, 2005] encontra-se nas estratégias de recomposição e substituição de componentes, enquanto que no presente trabalho, o foco é o arcabouço de suporte a adaptação ciente de contexto, bem como a abordagem utilizando orientação a aspectos e composição dinâmica usada para a adaptação. A abordagem apresentada pelo PACCA faz uso do conceito de aspecto abstrato para permitir o acréscimo de novos interesses adaptativos

não contemplados pelo PACCA, pois o mesmo já possui um conjunto de interesses adaptativos *default* para aplicações ubíquas. Somado a isso, tem-se o uso de diferentes sistemas de provisão para a obtenção das informações relativas ao contexto e da apresentação de um processo de desenvolvimento com o intuito de facilitar a construção de aplicações para ambientes de Computação Ubíqua através da definição de um conjunto etapas a serem seguidas para construir aplicações ubíquas, além disso, o PACCA possui o gerenciamento de sessões permitindo a migração entre dispositivos.

[CÁMARA; SALAUN; CANAL, 2007] apresentou uma proposta de adaptação em tempo de execução para sistemas sensíveis ao contexto. O uso da programação orientada a aspectos no trabalho possibilitou realizar a separação de interesses ligados a adaptação e aplicar a composição de processos, a qual é habilitada para tratar novas informações de contexto. O processo de composição consiste em automatizar a composição e a adaptação de um conjunto de componentes em tempo de execução. Os autores utilizaram LTS (*Labelled Transition Systems*), os quais são autômatos, para expressar o protocolo especificado para os casos de uso descritos no cenário utilizado. De forma semelhante ao PACCA, o trabalho utiliza o conceito de orientação a aspectos, porém a principal diferença está no fato de o PACCA utilizar o *AdaptadorAbstrato* que permite ao desenvolvedor estendê-lo acrescentando novos aspectos para adaptar novos interesses adaptativos e o processo de desenvolvimento, o qual serve de diretriz para que o desenvolvedor possa criar novas aplicações seguindo um passo a passo potencializando o reuso de código, além disso o PACCA ainda oferece um conjunto de aspectos para interesses adaptativos comuns a aplicações ubíquas. O *AdaptadorAbstrato* possui ainda operações requeridas por aplicações ubíquas como carga e descarga de serviços de provisão de contexto e *subscribe* e *unsubscribe* de tópicos para a notificação de mudanças de contexto. Ademais, o processo possui um componente responsável por gerenciar a migração das sessões dos usuários entre os dispositivos.

Em [PREUVENEERS et al., 2006] foi proposto um *framework*, orientado a componentes e construído baseado na divisão em camadas, capaz de realizar adaptações sensíveis ao contexto para um sistema de computação ubíqua. No topo das camadas encontra-se a camada de composição de aplicações e serviços, responsável por agregar os serviços solicitados baseado no contexto. Abaixo dela, há a camada sensível ao contexto, com o propósito de adquirir, armazenar, agregar, examinar e disseminar informações de contexto. Descendo na pilha de camadas, está o Draco, um *middleware* de componentes que realiza o gerenciamento do núcleo principal de componentes. Na

base das camadas, encontram-se os módulos opcionais, responsáveis pelo controle de adaptação, dos recursos de acesso e de monitoramento, estando também sob a responsabilidade do Draco. As informações de contexto são modeladas através de uma ontologia de contexto. Os autores empregaram um mecanismo de inferência para habilitar a camada de contexto a raciocinar sobre o contexto atual e realizar adaptações das aplicações de acordo com as preferências do usuário, as exigências do serviço ou quanto à disponibilidade dos recursos. O Draco possui estratégias de adaptação dinâmica baseadas na carga dinâmica de componentes visando principalmente a economia de memória. O principal diferencial em relação ao presente trabalho é que no PACCA adota-se a técnica de DSOA com o intuito de separar o código da aplicação do código da adaptação, proporcionando uma maior modularidade as aplicações além de permitir a adição de novos interesses adaptativos, além daqueles definidos no Modelo de Aspectos *default*. No PACCA, a inserção ou modificação de estratégias de adaptação pode ser realizada de maneira mais simples através da definição de novos aspectos que estendam o Aspecto Abstrato e da definição de políticas em XML, provendo-se maior flexibilidade ao desenvolvimento de aplicações cientes de contexto adaptativas. Por sua vez, para o desenvolvimento de aplicações ubíquas, o PACCA define um processo de desenvolvimento com o intuito de guiar o desenvolvedor em todas as fases da construção de aplicações ubíquas. O ambiente de computação ubíqua é altamente dinâmico e caracterizado por uma série de dispositivos imersos em tal ambiente, dessa forma o PACCA também apresenta um mecanismo para gerenciar a migração de sessões entre os dispositivos inseridos no ambiente de computação ubíqua.

O RSCM [YAU; KARIM, 2004] (*Reconfigurable Context-Sensitive Middleware*) é um *middleware* adaptativo baseado em objetos para facilitar a comunicação sensível ao contexto em ambientes de computação ubíqua. O RSCM requer um R-ORB, o qual é um ORB que segue o padrão CORBA, e que tem como objetivo prover a comunicação em ambientes distribuídos heterogêneos. Para que a comunicação dos objetos seja possível, no RSCM é definida uma linguagem de definição de interfaces baseada na IDL CORBA chamada *Context-Aware Interface Definition Language* (CA-IDL). A arquitetura do RSCM consiste de dois tipos de componentes: específicos da aplicação e independentes da aplicação, onde componentes específicos da aplicação são implementados em software e componentes independentes da aplicação podem ser desenvolvidos em *software* ou em *hardware* reconfigurável. O foco do PACCA, tal como o RSCM, está em prover uma infra-estrutura para facilitar o

desenvolvimento de aplicações adaptativas cientes ao contexto, porém o RSCM concentra-se em facilitar a interoperabilidade entre essas aplicações com o uso do R-ORB. O PACCA concentra esforços em facilitar a implementação de aplicações cientes ao contexto adaptativas através de orientação aspectos e composição dinâmica de software. No PACCA a adaptação é modelada como um aspecto, e uma clara separação de interesses pode ser observada entre a lógica da aplicação e a lógica responsável pela adaptação. A utilização do *AdaptadorAbstrato* visa prover maior flexibilidade através da definição de novos aspectos além daqueles providos pelo PACCA e sejam de interesse do desenvolvedor. Além disso, a economia de recursos com a carga dinâmica de módulos é possibilitada pela técnica de composição dinâmica. Ainda, no presente trabalho propõe-se um processo para o desenvolvimento de aplicações adaptativas cientes de contexto.

Em [RASHID; KORTUEM, 2004] é proposta uma abordagem de desenvolvimento para aplicações adaptativas em ambientes de computação ubíqua. O argumento principal consiste em que a programação orientada a aspectos não apenas melhora a modularização de aplicações adaptativas como também facilita o desenvolvimento de aplicações para ambientes de computação ubíqua, pois caso sejam necessárias mudanças no código relativo à adaptação este se encontra concentrado no aspecto, evitando a atualização em múltiplos elementos do sistema. O PACCA segue a mesma estratégia, modelando a adaptação como um aspecto e provendo uma infra-estrutura e um processo de desenvolvimento para que o desenvolvimento de aplicações cientes de contexto e adaptativas seja facilitado. Em [RASHID; KORTUEM, 2004] é utilizado AspectJ e o PACCA apresenta duas versões uma utilizando o arcabouço orientado a aspectos JBoss AOP e outra AspectJ. Na versão JBOSS AOP faz-se uso do recurso de *weaving* dinâmico, que permite modificações nos *pointcuts* em tempo de execução, possibilitando que um comportamento adaptativo possa ser inserido em outros pontos da aplicação sem a necessidade de recompilação. A versão AspectJ do PACCA não está utilizando *weaving dinâmico*. Porém o PACCA faz uso do conceito de aspecto abstrato para prover maior flexibilidade e também possui um Modelo de Aspectos *default* que abrange interesses adaptativos comuns a diversas aplicações ubíquas e para melhor utilização dos recursos do PACCA é definido um processo de desenvolvimento para servir de diretriz na concepção de aplicações para ambientes de computação ubíqua.

7. Considerações Finais

Esta dissertação apresentou o PACCA, um arcabouço que, integrado a um *middleware* de provisão de contexto e adotando uma abordagem baseada na separação de interesses, provê uma infra-estrutura para desenvolvimento e execução de aplicações adaptativas cientes de contexto. A abordagem proposta busca aumentar o grau de modularidade e flexibilidade na construção de aplicações ubíquas.

Este capítulo apresenta as principais contribuições do arcabouço proposto, descritas na Seção 7.1 e os trabalhos futuros são apresentados na Seção 7.2.

7.1. Principais Contribuições

As principais contribuições deste trabalho residem no uso conjunto das técnicas de DSOA e composição dinâmica como forma de dar suporte a construção de aplicações adaptativas cientes de contexto, gerando código com alto grau de modularidade e todos os benefícios decorrentes. Resultados obtidos com a implementação de um protótipo do arcabouço proposto demonstraram os ganhos em termos de modularidade e economia de memória. O comportamento adaptativo pôde ser provido sem onerar a aplicação e os seus desenvolvedores.

O uso de orientação a aspectos demonstrou que a retirada do comportamento adaptativo ciente de contexto dos componentes de negócio, além de tornar a aplicação mais flexível e modular com a redução dos níveis de acoplamento e diminuição da falta de coesão, possibilita às aplicações um processo de evolução e manutenção mais simples. Tal benefício advém do fato de que os módulos da aplicação, em vez de conter código adaptativo irão conter apenas a especificação de comportamentos que são realmente de sua responsabilidade. A própria evolução do comportamento adaptativo fica facilitada, já que, o código do processo de adaptação fica concentrado nos aspectos e não espalhado pelos módulos de negócio. Também é importante ressaltar que a definição e utilização de um aspecto abstrato e da provisão de um conjunto de aspectos concretos *default*, específicos para o domínio de aplicações ubíquas, promovem o reuso e tornam o arcabouço proposto mais flexível e extensível com a possibilidade de acréscimo de novos interesses adaptativos (*adaptive concerns*) além daqueles já definidos no PACCA.

Já a Composição dinâmica possibilita às aplicações uma redução no consumo de memória através da carga de módulos sob demanda, conforme comprovado pelos resultados obtidos na avaliação (ver Tabela 3). Tal fato é muito importante considerando o caráter limitado dos recursos disponíveis em dispositivos móveis como celulares e PDA's em relação a *notebooks* e *desktops*, apesar da crescente evolução que proporciona a tais dispositivos um poder computacional cada vez maior. Além disso, a composição dinâmica faz com que uma aplicação possa dinamicamente adaptar a forma de prover um serviço através da adição de novos módulos e sua combinação com os módulos já carregados, tornando seu comportamento mais flexível e adequado ao ambiente de execução típico das aplicações ubíquas, nos quais não é razoável se prever antecipadamente todos os possíveis comportamentos da aplicação.

O PACCA também apresenta como contribuição um processo de desenvolvimento para a concepção de aplicações adaptativas cientes de contexto, o qual foi ilustrado através do estudo de caso idealizado no presente trabalho. O processo de desenvolvimento tem o papel de guiar o desenvolvedor na elaboração de aplicações nesse novo contexto da Computação Ubíqua, usando uma abordagem de separação de interesses, padronizando e simplificando a tarefa de desenvolver este tipo de aplicação.

Outras contribuições relevantes são a definição de um Gerente de Sessão para permitir a migração entre dispositivos, característica essa que é indispensável quando se trata de trabalhar com aplicações ubíquas, além da realização de uma avalição quantitativa com bases em métricas já definidas na literatura.

7.2. Trabalhos Futuros

Como trabalhos futuros apresentam-se algumas perspectivas e metas a atingir, as quais são descritas em mais detalhes nas subseções a seguir.

7.2.1. Versão Java ME

A primeira frente relacionada aos trabalhos futuros consiste no desenvolvimento de uma versão do PACCA utilizando a plataforma para dispositivos móveis Java ME. O desenvolvimento dessa nova versão torna-se necessário, visto que, o objetivo do PACCA está associado a fornecer suporte a aplicações que executam em um ambiente altamente heterogêneo e móvel, como o da Computação Ubíqua. Logo, a plataforma

adequada é a plataforma Java ME, coerente para o desenvolvimento de aplicativos empregados em diversos tipos de dispositivos móveis.

Embora a plataforma Java ME apresente-se como a mais adequada para o desenvolvimento de aplicações destinadas a ambientes de computação ubíqua, ela possui restrições em relação ao *ClassLoader,* não permitindo que sejam redefinidos novos *ClassLoaders*. Tais restrições provêm do fato de que a KVM (*Kilobyte Virtual Machine*), uma versão limitada da JVM, possui um modelo de segurança chamado de *sandbox* tornando bastante restrita a quantidade de recursos disponíveis, entre eles a sobrecarga e redefinição de *ClassLoaders*. [RODRIGUES; RODRIGUES, 2007]

A redefinição de *ClassLoader* permitida no Java SE possibilitaria a criação de novas versões otimizadas do *Gerente de Execução.* Todavia, isso não inviabiliza a nova versão apenas acrescentar novas restrições ao desenvolvimento da nova versão. A plataforma Java ME, apesar das restrições apresentadas, permite a carga dinâmica de classes através do método *forName* da Classe *Class*, ou seja, o mesmo mecanismo utilizado no *Gerente de Execução*.

Em relação à concepção da versão Java ME, é importante ressaltar que a ausência do mecanismo de reflexão nessa plataforma inviabiliza a nova versão, uma vez que as aplicações adaptativas cientes de contexto fazem uso de reflexão para inspecionar a própria aplicação e obter dinamicamente informações sobre o contexto da aplicação, auxiliando no processo de adaptação da aplicação. No entanto, existem pesquisas como a indicada em [LIBORIO; BIGONHA, 2004] visando a introdução de um mecanismo de reflexão na plataforma Java ME, o que tornaria mais eficiente e dinâmico o processo de desenvolvimento de aplicações para ambientes de computação nesta plataforma. Dessa maneira, evita-se a adoção de soluções menos elegantes e abrem-se perspectivas para a implementação da nova versão do PACCA em Java ME.

7.2.2. Adoção de novas plataformas de provisão de contexto

Outra frente de pequisa aponta a linha de habilitar o PACCA com o suporte a diferentes sistemas de *middleware* de provisão de contexto. Essa frente é interessante no sentido em que possibilitará maiores opções ao desenvolvedor e permitirá ao PACCA adaptar-se de acordo com as informações contextuais fornecidas por diferentes tipos sistemas de provisão de contexto, cada um com suas especificidades.

Em adição, poderão ser recepcionadas novas informações de contexto além das providas pelo MOCA, de acordo com o desejo e as necessidades do desenvolvedor da

aplicação ciente de contexto em incorporá-las à sua aplicação. Para alcançar esse objetivo, é necessário estender o *Gerente de Contexto*, capacitando-o a gerenciar as informações adicionais, e também estender o Aspecto Abstrato para novos Adaptadores, em função das novas informações de contexto, de acordo com a capacidade do *middleware* selecionado em fornecê-las, as quais já são permitas pelo PACCA.

Apesar do suporte a outros sistemas de provisão de contexto estar previsto na arquitetura do PACCA, no presente trabalho não foram realizados testes empregando outros sistemas de *middleware* além do MOCA. Todavia, tais testes podem ser realizados sem grandes dificuldades, haja vista que a arquitetura do PACCA, desde sua concepção, foi implementada visando a adição de novos sistemas de provisão de contexto, sem o comprometimento de sua estrutura.

7.2.3. Inclusão de um Modelo de Contexto baseado em Ontologias

Em outra frente de trabalhos futuros, o modelo de contexto adotado será estendido, sendo tanto as informações de contexto (obtidas a partir do *middleware* de contexto) quanto as políticas e estratégias de adaptação, representadas com base em tecnologias e linguagens da área da Web Semântica [BERNERS-LEE; HENDLER; LASSILA, 2001]. Em outras palavras, em uma versão posterior do PACCA, serão utilizadas ontologias para representar a informação contextual a ser manipulada e linguagens da Web Semântica para representar as políticas de adaptação. O objetivo dessa extensão é prover um formato comum e padronizado para o compartilhamento da informação contextual, e usufruir da base formal das tecnologias da Web Semântica para empregar a realização de inferências mais complexas, a partir das informações contextuais.

7.2.4. Construção do Validador de Composições e Melhoria do Gerente de Sessão

Outro ponto a ser pesquisado é a construção de um validador de composições, que deve garantir a consistência e a validade das composições geradas pelo processo de adaptação dinâmica. O validador deve garantir que eventuais interdependências entre módulos sejam respeitadas, fazendo com que a aplicação adapte-se de uma arquitetura consistente para outra também consistente, na literatura esse processo é chamado de "adaptação segura" (*safe adaptation* [ZHANG et al., 2005]). Para garantir o estado de consistência das aplicações adaptativas podem ser utilizadas técnicas de validação e

especificação formal, sendo uma possível abordagem o uso de Redes de Petri [PETRI, 1962].

A adaptação segura, segundo a abordagem definida em [ZHANG et al., 2005] utiliza invariantes para especificar relacionamentos de dependência entre múltiplos componentes. Tais relacionamentos permitem o uso de técnicas de análise para determinar quais componentes são afetados durante uma dada adaptação, e conseqüentemente o conjunto de estados seguros, nos quais adaptações dinâmicas podem ocorrer. Além disso, a abordagem fornece mecanismos de *rollback* no caso de ocorrem erros ou falhas durante o processo de adaptação.

Outra frente de pesquisa será na melhoria do Gerente de Sessão, visto que é um tópico essencial quando se fala de aplicações ubíquas e existem pesquisas que focam apenas no problema de migrações de sessões. As versões futuras terão como foco buscar a resolução de problemas de inconsistências quando se trabalha de modo *offline* em um dispositivo e em seguida se migra para um dispositivo *online*. Além disso, tratar mais detalhes relativos à migração de sessões que foram simplificados na abordagem atual, como por exemplo, armazenar mais informações sobre a sessão, na versão atual são armazenadas apenas login, dispositivo, endereço *MAC* e a última operação realizada pelo usuário.

8. Referências

[ABOWD, 1999] ABOWD, G. D. **Classroom 2000:** An Experiment with the instrumentation of a living educational environment**.** IBM SYSTEMS JOURNAL - PERVASIVE COMPUTING. 10 mar 1999. v. 38, n. 4, p. 508-531. Disponível em: < http://www.research.ibm.com/journal/sj/384/abowd.html>. Acesso em: 16 set 2008.

[AOP METRICS, 2008] Tigris.org. **aopMetrics:** Project Home. Disponível em: <http://aopmetrics.tigris.org/>. Acesso em: 20 abr 2008.

[AOSD, 2008] aosd.net. **Aspect-oriented Software Development Community & Conference**. Disponível em: <http://aosd.net> Acesso em: 12 abr 2008.

[ASPECTJ, 2008] Eclipse. **The AspectJ Project.** Disponível em: <http://www.eclipse.org/aspectj>. Acesso em: 12 abr 2008.

[BANIASSAD; CLARKE, 2004a] BANIASSAD, E.; CLARKE, S. **Finding Aspects in Requirements with Theme/Doc.** In: WORKSHOP ON EARLY ASPECTS. 2004. Lancaster, United Kingdom.

[BANIASSAD; CLARKE, 2004b] BANIASSAD, E.; CLARKE, S. **Theme:** An Approach for aspect-oriented analysis and design. In: 26TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE). 2004. Edimburgo, United Kingdom. Los Alamitos, CA, USA: IEEE Computer Society. p. 158-167.

[BARDRAM, 2005] BARDRAM, J. E. **The Java Context Awareness Framework (JCAF) – A service infrastructure and programming framework for context-aware applications.** In: THE 3RD INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING (PERVASIVE 2005). 2005. Munich, Germany: Springer Berlin. v. 3468, p. 98-115.

[BERNERS-LEE; HENDLER; LASSILA, 2001] BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. **The Semantic Web.** SCIENTIFIC AMERICAN. 17 maio 2001. Disponível em: <http://www.sciam.com/article.cfm?id=the-semantic-eb&print=true> Acesso em: 10 out 2008.

[BULCÃO NETO; TEIXEIRA; PIMENTEL, 2005] BULCÃO NETO, R. F.; TEIXEIRA, C. A. C.; PIMENTEL, M. G. C. **A Semantic Web-based infrastructure for supporting context-aware applications**. In: IFIP - INTERNATIONAL CONFERENCE ON EMBEDDED AND UBIQUITOUS COMPUTING (EUC'05). 2005. Nagasaki, Japan:Springer Berlin. v.3824, p.900-909.

[BUSCHMANN et al., 1996] BUSCHMANN, F., et al. **Pattern-Oriented Software Architecture – A System of Patterns.** 1ª ed. Hoboken, NJ, USA : Wiley, 1996. 476p.

[CÁMARA; SALAUN; CANAL, 2007] CÁMARA, J.; SALAUN, G.; CANAL, C. **On Run-time Behavioural Adaptation in Context-Aware Systems**. In: 1ST WORKSHOP ON MODEL-DRIVEN SOFTWARE ADAPTATION (ECOOP 2007). 2007. Berlin, Germany. p. 26-33.

[CANAL; MURILLO; POIZAT, 2006] CANAL, C.; MURILLO, J.M.; POIZAT, P.; **Software Adaptation**. L'OBJET. Special Issue on Coordination and Adaptation Techniques for Software Entities. 2006. v. 12, n. 1, p. 9-31.

[CANDILLON; VANWORMHOUDT, 2007] CANDILLON, W.; VANWORMHOUDT, G. **Overview and Advices about PHPAspect.** Disponível em: < http://www.phpaspect.org/documentation/ >. Acesso em: 19 set 2008.

[CAPRA et al., 2002] CAPRA, L. et al. **Exploiting Reflection in Mobile Computing Middleware**. In: ACM SIGMOBILE MOBILE COMPUTING AND COMMUNICATIONS REVIEW. 2002. v. 6, n. 4, p. 34-44.

[CHAVEZ, 2004] CHAVEZ, C. **Um enfoque Baeado em Modelos para o Design Orientado a Aspectos.** 2004. 298f. Tese (Doutorado em Informática). Programa de Pós-Graduação em Informática. Pontifícia Universidade Católica do Rio de Janeiro (PUC - Rio), Rio de Janeiro - RJ.

[CHITCHYAN et al., 2005] CHITCHYAN, R., et al. **Survey of Analysis and Design Approaches.** 2005. Lancaster, United Kingdom: Lancaster University. AOSD Europe Report, Deliverable D11. p. 1-259.

[COOPERSMITH; JOHNSON; MATYAS, 1996] COPPERSMITH, D.; JOHNSON, D. B.; MATYAS, S. M. **A Proposed Mode for Triple-DES Encryption**. IBM JOURNAL OF RESEARCH AND DEVELOPMENT. 1996. v. 40, n. 2, p. 253-262.

[CORBA, 2008] OMG. **The OMG's CORBA Web Site**. Disponível em: <http://www.corba.org> Acesso em: 10 mar 2008.

[COSTA; STRZYKALSKI; BERNARD, 2007] COSTA, C. M.; STRZYKALSKI, M. S.; BERNARD, G. **An Aspect Oriented Middleware Architecture for Adaptive Mobile Computing Applications**. In: 31ST ANNUAL INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE (COMPSAC 2007). 2007. Beijing, China: IEEE Computer Society, v.2, p. 81-86.

[DANTAS; BORBA, 2003] DANTAS, A.; BORBA, P. **Developing Adaptive J2ME Applications Using AspectJ**. JOURNAL OF UNIVERSAL COMPUTER SCIENCE. 2003. v. 9, n. 8. p. 935-955. Disponível em: <http://www.jucs.org/jucs_9_8/developing_adaptive_j2me_applications>. Acesso em: jun 2008.

[DANTAS et al., 2004] DANTAS, A., et al. **Using Aspects to Make Adaptive Object-Models Adaptable.** In: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING (ECOOP 2004) – WORKSHOP ON REFLECTION, AOP AND META-DATA FOR SOFTWARE EVOLUTION (RAM-SE). 2004. Oslo, Norway. p. 9-20.

[DEY, 2001] DEY, A. K.; **Understanding and using context**. PERSONAL AND UBIQUITOUS COMPUTING JOURNAL. 2001. London, United Kingdom: Springer-Verlag. v. 5, n. 1, p. 4-7.

[DNS, 2008] INTERNET SYSTEMS CONSORTIUM. **ISC BIND DNS.** Disponível em: < http://www.isc.org/index.pl?/sw/bind/index.php>. Acesso em: 10 out 2008.

[ECLIPSE, 2008] Eclipse. **Eclipse.org**. Disponível em: <http://www.eclipse.org> Acesso em: 15 set 2008.

[FERNANDES; BATISTA, 2004] FERNANDES, F. A.; BATISTA, T. V. **AspectLua:** A Dynamic AOP Approach. JOURNAL OF UNIVERSAL COMPUTER SCIENCE. 2005. v.11 n.3 p. 1177-1197.

[FIGUEIREDO et al., 2005] FIGUEIREDO, E., et al. **Assessing Aspect-Oriented Artifacts:** Towards a Tool-Supported Quantitative Method. In: PROCEEDINGS OF THE 9TH WORKSHOP ON QUANTITATIVE APPROACHES IN OBJECT-ORIENTED SOFTWARE ENGINEERING (QAOOSE) CO-LOCATED WITH ECOOP'05. 2005. Glasgow, United Kingdom. p. 58-69.

[FRITSCH; MUNNELLY; CLARKE, 2006] FRITSCH, S.; MUNELLY, J.; CLARKE, S. **Towards a Domain Specific AOP language for Ubiquitous Computing.** In: PROCEEDINGS OF OPEN AND DYNAMIC ASPECT LANGUAGES WORKSHOP (ODAL 2006). 2006. Bonn, Germany. p. 1-4.

[GOMES et al., 2005] GOMES, A. T. A., et al. **Towards a Ubiquitous Healthcare System for Acute Myocardial Infarction Patients in Brazil.** In: FOURTH IEEE INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING AND COMMUNICATIONS WORKSHOPS (PERCOMW'06). 2006. Pisa, Italy. Washington, DC, USA: IEEE Communications Society. p. 585-589

[GRAY; SALBER, 2001] GRAY, P. D.; SALBER, D. **Modelling and Using Sensed Context Information in the Design of Interactive Applications**. In: 8TH IFIP INTERNATIONAL CONFERENCE ON ENGINEERING FOR HUMAN-COMPUTER INTERACTION. 2001. London, United Kingdom: Springer-Verlag. Lecture Notes In Computer Science (LNCS). v. 2254, p. 317-336.

[GRUNDY, 1999] GRUNDY, J. **Aspect-Oriented Requirements Engineering for Component-based Software Systems.** In: 4th IEEE INTERNATIONAL SYMPOSIUM ON REQUIREMENTS ENGINEERING. 1999. Limerick, Ireland.

[HAGEN; MODSCHING; KRAMER, 2005] HAGEN, K.; MODSCHING, M.; KRAMER, R. **A Location aware mobile tourist guide selecting and interpreting sights and services by context matching.** In: THE SECOND ANNUAL INTERNATIONAL CONFERENCE ON MOBILE AND UBIQUITOUS SYSTEMS: NETWORKING AND SERVICES (MobiQuitous 2005). 2005. San

Diego, CA, USA. p. 293-301.

[HOH; TAN; HARTLEY, 2006] HOH, S.; Tan, J.S.; HARTLEY, M. **Context-aware systems - a primer for user-centred services**. BT TECHNOLOGY JOURNAL. 2006. Hingham, MA, USA: Kluwer Academic Publishers. v. 24, n. 2, p. 186 – 194.

[JBOSS AOP, 2008] JBoss.org. **JBoss AOP**. Disponível em: <http://labs.jboss.com/jbossaop/> Acesso em: 20 mar 2008.

[JBOSS USER GUIDE, 2008] JBoss.org. **JBoss User Guide – The Case of Aspects 1.5.** Disponível em: <http://labs.jboss.com/jbossaop/> Acesso: 20 set 2008.

[JNDI, 2008] Sun. **Java Naming and Directory Interface (JNDI)**. Disponível em: <http://java.sun.com/products/jndi/> Acesso em: abr 2008.

[KANG et al., 2006] KANG, T., et al. **A Context-aware Handoff Management for Seamless Connectivity in Ubiquitous Computing Environment**. In: INTERNATIONAL CONFERENCE ON PERVASIVE SYSTEMS & COMPUTING (PSC'06). 2006. Las Vegas, NV, USA: Hamid R. Arabnia. v. 1 p. 128-134.

[KICZALES et al., 1997] KICZALES, G., et al. **Aspect-Oriented Programming**. In: PROCEEDINGS OF THE EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING (ECOOP). 1997. Helsinki, Finland. Spring-Verlag. Lecture Notes In Computer Science (LNCS). v. 1241. p. 220–242

[KICZALES et al., 2001] KICZALES, G., et al. **An Overview of AspectJ**. In: PROCEEDINGS OF THE 15th EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING (ECOOP). 2001. Budaspest, Hungary. London, United Kingdom: Spring-Verlag. Lecture Notes In Computer Science (LNCS). v. 2072. p. 327–353.

[LDAP, 2008] LDAP.org.br. **LDAP**. Disponível em: <http://www.ldap.org.br/> Acesso em: 20 ago 2008.

[LE SOMMER; GUIDEC; ROUSSAIN, 2006] LE SOMMER, N.; GUIDEC, F.;

ROUSSAIN, H. **A Context-Aware Middleware Platform for Autonomous Application Services in Dynamic Wireless Networks**. In: 1ST INTERNATIONAL CONFERENCE ON INTEGRATED INTERNET AD HOC AND SENSOR NETWORKS. 2006. Nice, France. New York, NY, USA: ACM. v. 138, n. 9. p. 1-8.

[LIBORIO; BIGONHA, 2004] LIBORIO, S. R.; BIGONHA, R. S. **Compilador Reflexivo para Adaptações de Programas a Configurações de Recursos Limitados.** In: VII SIMPÓSIO BRASILEIRO DE LINGUAGENS DE PROGRAMAÇÃO. 2004. Niterói, RJ, Brasil. Anais do VII Simpósio Brasileiro de Linguagens de Programação. Porto Alegre, RS, Brasil: Sociedade Brasileira de Computação. p. 242-256.

[LIMA JR; CALSAVARA, 2008] LIMA JR, L.; CALSAVARA, A. **A Paradigm Shift in the Design of Mobile Applications**. In: ADVANCED INFORMATION NETWORKING AND APPLICATIONS – WORKSHOPS (AINAW 2008). 2008. Okinawa, Japan. v.1, n.1, p. 1631-1635.

[LOUGHRAN et al., 2006] LOUGHRAN, N., et al. **A domain analysis of key concerns – known and new candidates**. 2006. Leuven, Belgium: Katholieke Universiteit Leuven. AOSD Europe Deliverable D43. p. 1-267.

[MUKHIJA; GLINZ, 2005] MUKHIJA, A.; GLINZ, M. **Runtime Adaptation of Applications through Dynamic Recomposition of Components**. In: 18TH INTERNATIONAL CONFERENCE ON ARCHITECTURE OF COMPUTING SYSTEMS (ARCS 2005). 2005. Innsbruck, Austria: Springer Berlin. Lecture Notes In Computer Science (LNCS). v. 3432. p. 124-138.

[MUNNELLY; FRITSCH; CLARKE, 2007] MUNNELLY, J.; FRITSCH, S.; CLARKE, S. **An Aspect-Oriented Approach to the Modularisation of Context**. In: PERVASIVE COMPUTING AND COMMUNICATIONS (PERCOM2007). 2007. New York, NY, USA. p. 114-124.

[PARNAS, 1972] PARNAS, D. L. **On the criteria to be used in decomposing systems into modules.** COMMUNICATION OF THE ACM. 1972. New York, NY, USA: ACM. v. 15, n. 12, p. 1053–1058.

[PETRI, 1962] PETRI, C. A. **Kommunication mit Automaten.** Schriften des IIM Nr. 2. Institut für Instrumentelle Mathematik. Bonn, Deutschland. 1962. Traduzida para o Inglês como: ***Communication with Automata***, Technical Report RADC-TR-65-377, Griffiths Air Force Base, New York, NY, USA: v.1, n.1.

[PIRES et al., 2005] PIRES, R., et al. **Comunicação entre Componentes da Aplicação em Ambiente Pervasivo**. 2005. In: XV SEMINÁRIO REGIONAL DE INFORMÁTICA. Santo Ângelo, RS, Brasil.

[PIRMEZ et al., 2008] PIRMEZ, M., et al. **Prometheus:** Um Serviço de Segurança Adaptativa. 2008. In: VII SIMPÓSIO BRASILEIRO DE SEGURANÇA DA INFORMAÇÃO. Gramado, RS, Brasil. Anais do VII Simpósio Brasileiro de Segurança da Informação. Porto Alegre, RS, Brasil: SBC, 2008.

[POSTGRESQL, 2008] PostgreSQL. **PostgreSQL – The most advanced Open Source database**. Disponível em: <http://www.postgresql.org> Acesso: 21 jan. 2008.

[PREUVENEERS et al., 2006] PREUVENEERS, D., et al. **Context-aware adaptation for component-based pervasive computing systems**. In: 4TH INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING. 2006. Dublin, Ireland: Katholieke Universiteit Leuven. v. 207, p. 125-128.

[RASHID; KORTUEM, 2004] RASHID, A.; KORTUEM, G. **Adaptation as aspect in pervasive computing**. In: Workshop on Building Software for Pervasive Computing at OOPSLA. 2004. Vancouver, Canada.

[RMI, 2008] Sun. **Remote Method Invocation Home**. Disponível em: <http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp> Acesso em: 15 jul 2008.

[RMI-IIOP, 2008] Sun. **RMI-IIOP Programmer´s Guide**. Disponível em: < http://java.sun.com/j2se/1.4.2/docs/guide/rmi-iiop/rmi_iiop_pg.html> Acesso em: 15 jul 2008.

[ROCHA; ENDLER, 2006] ROCHA, R. C. A.; ENDLER, M. **Supporting Context-Aware Applications:** Scenarios, Models and Architecture. In: XXIV SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES (SBRC). 2006. Curitiba, PR, Brasil.

[ROCHA; CASANOVA; ENDLER, 2007] ROCHA, R. C. A.; CASANOVA, M. A.; ENDLER, M.; **Promoting efficiency and separation of concerns through a hybrid model based on ontologies for context-aware computing**. In: PERVASIVE COMPUTING AND COMMUNICATIONS WORKSHOPS. (PERCOM). 2007. New York, NY, USA. p. 9-13.

[RODRIGUES; RODRIGUES, 2007] RODRIGUES, W. C.; RODRIGUES, M. M. **Portando Aplicações da Plataforma Java Standard Edition para a Plataforma Java Micro Edition.** SCIENCE OF COMPUTING – JOURNAL OF COMPUTER SCIENCE. Fortaleza,CE, Brasil: Faculdade Farias Brito (FFB). Jul 2007. v.1 p. 29-42.

[SACRAMENTO et al., 2004] SACRAMENTO, V., et al. **MoCA**: A Middleware for Developing Collaborative Applications for Mobile Users. In: IEEE DISTRIBUTED SYSTEMS ONLINE. 2004. v. 5, n. 10, p. 2.

[SANTOS et al., 2008] SANTOS, I., et al. **Usando Aspectos e Composição Dinâmica para prover Adaptação Ciente ao Contexto em Sistemas Ubíquos.** In: XXII SIMPOSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (SBES), 2008, Campinas, SP, Brasil. Anais do XXII Simposio Brasileiro de Engenharia de Software. Porto Alegre, RS, Brasil: SBC, 2008.

[SANTOS et al., 2009] SANTOS, I., et al. **Using Aspects and Dynamic Composition to provide Context-Aware Adaptation for Mobile Applications.** IN: ACM SYMPOSIUM ON APPLIED COMPUTING (ACM SAC). 2009. Honolulu, HI, USA.

[SALBER; DEY; ABOWD, 1999] SALBER, D.; DEY, A.; ABOWD, G. **The context toolkit**: aiding the development of context-enabled applications. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS (CHI ’99). 1999. Pittsburgh, PA, USA. New York, NY, USA: ACM. p. 434 – 441.

[SATYANARAYANAN, 2001] SATYANARAYANAN, M. **Pervasive Computing: Vision and Challenges.** In: IEEE PERSONAL COMMUNICATIONS. 2001. Washington, DC, USA: IEEE Communications Society. v. 8, n.4, p. 10-17.

[SCHAUERHUBER; RETSCHITZEGGER, 2006] SCHAUERHUBER, S.; RETSCHITZEGGER, W. **Towards a Common Reference Architecture for Aspect-Oriented Modeling**. In: 8TH INT. WORKSHOP ON ASPECT ORIENTED MODELING (AOM). 2006. Bonn, Germany. p. 1-6.

[SCHULTZ et al., 2001] SCHULTZ, et al. **The ISL Meeting Room System.** IN: WORKSHOP ON HANDS-FREE SPEECH COMMUNICATION (HSC). 2001. Kyoto, Japan.

[SILVEIRA, 2001] SILVEIRA, F. F.; **Ferramenta de Apoio ao Teste de Aplicações Java baseada em Reflexão Computacional.** 2001. 96f. Dissertação (Mestrado em Ciência da Computação). Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre – RS.

[SMITH, 1982] SMITH, B. C.; **Reflection and Semantics in a Procedural Language.** 1982. 762p. PHD Thesis. Massachusetts Institute of Technology (MIT).

[SOMMERVILLE, 2000] SOMMERVILLE, I. **Engenharia de Software.** 6ª Edição. Pearson Education. 2000. p. 50-53.

[SPINCZYK; LOHMANN; URBAN, 2005] SPINCZYK, O.; LOHMANN, D.; URBAN; M. **AspectC++:** An AOP Extension for C++. SOFTWARE DEVELOPER´S JOURNAL. Set 2005. p. 68-76.

[SUTTON; ROUVELLOU, 2004] SUTTON, S. M.; ROUVELLOU, I. **Concern Modeling for Aspect-Oriented Software Development.** In: FILMAN, R. E., et al. ASPECT-ORIENTED SOFTWARE DEVELOPMENT. 2004. Boston, MA, USA: Addison-Wesley. p. 479-505.

[WEISER, 1991] WEISER, M. **The Computer for the Twenty-First Century**. SCIENTIFIC AMERICAN. v. 267, n. 3, September 1991 p. 94-104.

[WINCK; GOETTEN JUNIOR, 2006] WINCK, D. V.; GOETTEN JUNIOR, V. **AspectJ:** Programação Orientada a Aspectos com Java. 1. ed. São Paulo, SP, Brasil: Novatec Editora. 2006. 230p.

[XML, 2008] W3C. **W3C eXtensible Markup Language.** Disponível em: <http://www.w3.org/XML/> Acesso em: 10 fev 2008.

[XML SCHEMA, 2008] W3C. **XML Schema.** Disponível em: <http://www.w3.org/XML/Schema> Acesso em: 10 fev 2008.

[YAU; KARIM, 2004] YAU, S. S.; KARIM, F. **An Adaptative Middleware for Context-Sensitive Communications for Real-Time in Ubiquitous Computing Environments**. REAL-TIME SYSTEMS. 2004. Norwell, MA, USA: Kluwer Academic Publishers. v. 26, n. 1, p. 29-61.

[ZHANG et al., 2005] ZHANG, J., et al. **Enabling safe dynamic component-based software adaptation.** In: LEMOS, R.; GACEK, C.; ROMANOVSKY, A. **ARCHITECTING DEPENDABLE SYSTEMS III**. 1ª Ed. Secaucus, NJ, USA: Spring-Verlag New York, Inc. 2005. Lecture Notes In Computer Science (LNCS). v. 3549. p. 194-112.

Apêndice

As políticas de adaptação são definidas em XML, porém todo documento XML precisa ser validado de modo a garantir que seja um arquivo bem formado e siga as regras definidas em um esquema, o qual descreve a estrutura do documento XML. Tal esquema pode ser definido em um arquivo XML Schema [XML SCHEMA, 2008] cujo propósito é definir os blocos de construção, os elementos e os valores permitidos em um determinando documento XML.

A seguir temos o arquivo XML Schema que valida o arquivo de políticas definidas para o estudo de caso utilizado no trabalho. Tais políticas contemplam os interesses adaptativos abordados no estudo de caso, e o arquivo XML Schema define como tais interesses são estruturados dentro arquivo XML de políticas.

```
<?xml version="1.0"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="politicas">
  <xs:complexType>
   <xs:sequence>
    <xs:element maxOccurs="unbounded" name="politica">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="contexto">
        <xs:complexType>
         <xs:sequence>
          <xs:element minOccurs="0" name="Banda">
           <xs:complexType>
            <xs:simpleContent>
             <xs:extension base="xs:unsignedByte">
              <xs:attribute name="limiar" type="xs:string" use="required" />
             </xs:extension>
            </xs:simpleContent>
           </xs:complexType>
          </xs:element>
```

```
            <xs:element minOccurs="0" name="Energia">
              <xs:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:unsignedByte">
                    <xs:attribute name="limiar" type="xs:string" use="required" />
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </xs:element>
            <xs:element minOccurs="0" name="Online" type="xs:boolean" />
            <xs:element minOccurs="0" name="Perfil" type="xs:string" />
            <xs:element minOccurs="0" name="Dispositivo" type="xs:string" />
            <xs:element minOccurs="0" name="Criptografia" type="xs:boolean" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="decisao">
        <xs:complexType>
          <xs:sequence>
            <xs:element maxOccurs="unbounded" name="carregar" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required" />
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo