Usando Aspectos e Composição Dinâmica para prover Adaptação Ciente ao Contexto em Sistemas Ubíquos

Isanio Lopes Araujo Santos[1], Flavia C. Delicato[1], Paulo F. Pires[1], Thais Batista[1], Ana Liz Souto Oliveira[1], [*]Luci Pirmez[2]

[1]DIMAp – Universidade Federal do Rio Grande do Norte - RN,Brasil

[2]NCE/IM – Universidade Federal do Rio de Janeiro - RJ, Brasil

isanio@natalnet.br;{flavia.delicato,paulo.pires,thais}@dimap.ufrn.br;
analiz@lcc.ufrn.br; luci@nce.ufrj.br

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, nós propomos 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. A orientação a aspectos aliada a composição dinâmica de software oferecem suporte para adaptação ciente ao contexto, guiada por políticas previamente definidas.*

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, we propose PACCA, a framework for developing and executing adaptive context aware applications. PACCA employs aspect-oriented techniques to modularize the adaptive behavior and to keep apart the application logic from this behavior. It exploits the synergy between aspect-orientation and dynamic composition to achieve context-aware adaptation, guided by predefined policies.*

1. Introdução

Aplicações que operam em cenários de Computação Ubíqua [Weiser 1991] caracterizam-se por constantes mudanças em seu estado de execução, causadas, dentre outros fatores, pela mobilidade e diferentes preferências de seus usuários, pelas características de seus dispositivos, e pela variabilidade dos recursos disponíveis (p. ex., redes e serviços). Esse caráter dinâmico do ambiente de execução demanda da aplicação a capacidade de adaptar-se em função desses fatores. Portanto, 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 2006]; e (ii) capacidade de adaptar-se para atender as mudanças no ambiente, selecionando e incorporando novos comportamentos em tempo de execução, conhecida como *adaptação dinâmica* [Canal 2006]. Aplicações adaptativas cientes de contexto monitoram seu ambiente de execução e exploram a

[*] Parcialmente Financiado pelo CNPq

natureza contextual provocada pelas mudanças nesse ambiente, com o intuito de fornecer serviços adaptáveis e centrados no usuário.

O dinamismo do ambiente de execução faz com que aplicações ubíquas possuam requisitos e níveis de complexidade diferentes das aplicações que executam em ambientes tradicionais, requerendo abordagens distintas de desenvolvimento. Adicionalmente, é necessária a existência de serviços e ambientes que ofereçam suporte para tais aplicações explorarem e reagirem a mudanças de contexto dentro de seu domínio dinâmico e heterogêneo. É importante ressaltar que as adaptações necessárias às aplicações ubíquas podem ocorrer desde o nível do sistema até o nível do usuário. Há atualmente várias propostas de infra-estruturas [Mukhija e Glinz 2005, Rocha 2006] para suporte ao desenvolvimento e execução de aplicações adaptativas, em diferentes níveis. Porém, elas sofrem de limitações, principalmente com respeito ao suporte para manipular os diversos atributos contextuais, os quais podem interferir de modo global no funcionamento da aplicação. Ou seja, as abordagens existentes carecem de mecanismos para gerenciar a interação entre a aplicação e o conjunto completo de atributos que podem disparar a adaptação, ao nível de sistema.

Um fator complicador no desenvolvimento de aplicações adaptativas sensíveis ao contexto é o fato de que o código de adaptação, que implementa o comportamento adaptativo, encontrar-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 um comportamento transversal (*crosscuting concern*) [Kiczales 2007]. O comportamento transversal gera impactos negativos em diversos atributos desejáveis do sistema de software construído, como, por exemplo, sua modularização, potencial de reuso e manutenção. O desenvolvimento orientado a aspectos (*Aspect Oriented Software Development* - AOSD [AOSD 2008]) fornece um paradigma para modularização dos conceitos transversais que pode ser usado para desacoplar o comportamento adaptativo ciente de contexto da lógica funcional das aplicações. Nesse paradigma, comportamentos transversais são encapsulados em elementos denominados **aspectos** que são combinados com a lógica de negócio (componentes base) através de mecanismos de composição. Portanto, componentes que implementam funcionalidades representando a lógica das aplicações podem ser separados dos componentes que implementam adaptação e manipulação de contexto. Dessa forma, todo o código que implementa a adaptação pode ser encapsulado em um conjunto de aspectos potencialmente reutilizáveis, e as aplicações geradas podem ser melhor modularizadas, facilitando sua manutenção, extensão e contribuindo para o reuso.

Este trabalho visa apresentar o PACCA (*Projeto de Aplicações Cientes ao Contexto e Adaptativas*), um arcabouço que fornece uma infra-estrutura para apoio a construção e execução de aplicações adaptativas cientes de contexto em ambientes de Computação Ubíqua. A modularização dos aspectos de adaptação é feita no PACCA através da definição de um aspecto abstrato de adaptação, ao qual é associado um conjunto de aspectos concretos previamente identificados como sendo genéricos para vários tipos de aplicações ubíquas. Além disso, para lidar com o caráter dinâmico do ambiente ubíquo e com as características limitadas dos dispositivos, usa-se a técnica de composição dinâmica de software [Le Sommer, Guidec, e Roussain 2006] para a construção e adaptação das aplicações. As funcionalidades relativas à ciência de contexto são providas por um *middleware* de provisão de contexto, que realiza funções para aquisição, processamento e armazenamento de informações contextuais. Há inúmeros sistemas de *middleware* de provisão de contexto disponíveis [Le Sommer, Guidec, e Roussain 2006, Sacramento 2004]. No presente trabalho usamos o MOCA [Sacramento 2004], um *middleware* que possui os serviços necessários ao funcionamento do PACCA, de fácil utilização e cuja implementação encontra-se disponível, além de ser adotado na construção de aplicações ubíquas. O enfoque deste artigo é apresentar os serviços providos

pelo PACCA bem como seu funcionamento e a abordagem de desenvolvimento proposta. O artigo está estruturado como segue. A Seção 2 descreve a arquitetura e a abordagem de desenvolvimento. A Seção 3 apresenta um estudo de caso para ilustrar a proposta. A Seção 4 mostra como construir aplicações usando o PACCA. A Seção 5 apresenta a avaliação da proposta. A Seção 6 descreve trabalhos relacionados e a Seção 7 contém as conclusões.

2. PACCA

No PACCA, tanto os serviços de adaptação e ciência de contexto quanto as aplicações a serem executadas consistem em conjuntos de módulos de software (classes) interligados. As estratégias de adaptação baseiam-se na especificação do 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 pelo *middleware* de provisão de contexto. Embora a versão atual do PACCA use o MOCA como *middleware* de contexto, outros sistemas de *middleware* podem se usados uma vez que a arquitetura do PACCA usa uma abordagem modular com independência entre os componentes. As próximas seções descrevem o MOCA e, a seguir, a arquitetura e o funcionamento do PACCA.

2.1 MOCA

A manipulação de informação contextual na versão atual do PACCA é feita pelo MOCA [Sacramento 2004]. O MOCA fornece meios de coletar, armazenar e processar dados relativos ao contexto de dispositivos móveis e consiste essencialmente em um conjunto de API's e serviços capazes de suportar a execução de aplicações sensíveis ao contexto. 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. O Monitor é um componente armazenado e executado em um dispositivo móvel, responsável por repassar periodicamente informações de contexto para o CIS. 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 IEEE 802.11 ao alcance de um dispositivo e as correspondentes intensidades de sinais recebidas [Rocha, Casanova e Endler 2007]. Atualmente, as informações de contexto manipuladas pelo MOCA são descritas em um arquivo XML [XML 2008] usando um modelo *par-valor*. O LIS (*Location Inference Service*) é o serviço do MOCA responsável pela inferência da localização lógica aproximada, a qual é realizada com base na comparação do sinal padrão de rádio freqüência corrente do dispositivo com sinais medidos previamente e definidos como pontos de referência. Os serviços do MOCA utilizados pelo PACCA são o *CIS* e o *LIS*. Além desses serviços, o PACCA utiliza o componente Monitor e o Modelo de Contexto do MOCA.

2.2 Arquitetura do PACCA

O diagrama de componentes da Figura 1 representa a arquitetura do PACCA e sua integração com os componentes do MOCA. Os componentes do PACCA são responsáveis pela adaptação ciente ao contexto. O *Gerente de Contexto* armazena as informações de contexto corrente, as quais são constantemente atualizadas pelas notificações recebidas do CIS, 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*. Na atual implementação do MOCA, para possibilitar que o CIS notifique mudanças de contexto associadas a um dispositivo, é necessária a subscrição do endereço MAC do dispositivo junto ao CIS. Por tratar-se de um requisito específico da plataforma de manipulação de contexto, optou-se por realizar tal subscrição no *Módulo de Adaptação* do PACCA, deixando-a transparente para a aplicação.

Outra forma de notificar mudanças de contexto é a subscrição, junto com o endereço MAC, de tópicos específicos, como por exemplo, "notificar quando a quantidade de memória disponível atingir determinado nível". Na versão atual é usada apenas a primeira opção, onde o CIS notifica qualquer mudança de contexto que afete o dispositivo com MAC subscrito.

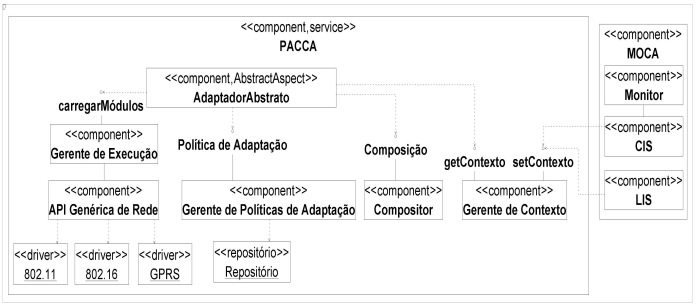


Figura 1 - Componentes do PACCA e sua integração com o MOCA

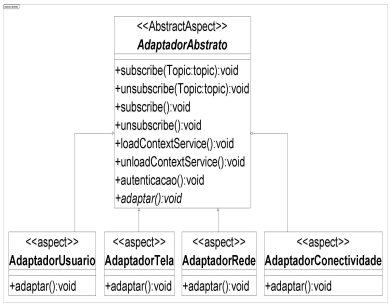


Figura 2 – Aspectos abstrato e concretos

O *Repositório de Políticas* armazena as políticas de adaptação, as quais são definidas de forma declarativa utilizando XML e definem o comportamento adaptativo a ser adotado nas diferentes situações de mudança de contexto. 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* (tal análise inclui um parser XML, ou seja, é feita a conversão das políticas definidas sob a forma declarativa para a forma imperativa/procedural), 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 que módulos devem ser dinamicamente carregados. O *Compositor* é 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 relacionem-se e funcionem corretamente, ou seja, validar a composição. Porém, na atual fase do trabalho não foi tratada a 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 irá buscá-las através de um serviço de nomes (no trabalho está sendo utilizado o JNDI [JNDI 2008]), podendo a princípio localizar-se tanto local quanto remotamente. Finalmente, 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 o código que é realmente necessário em cada contexto de execução.

O *Módulo de Adaptação* é o componente que responde pelo processo de adaptação dinâmica gerenciando todos os demais componentes do PACCA. Esse módulo foi desenvolvido como um aspecto abstrato (*AdaptadorAbstrato*, ver Figura 2) no qual estão definidas operações comuns às aplicações adaptativas cientes de contexto, como: (i) carga e descarga do serviço de provisão de contexto, (ii) subscrição do endereço MAC do dispositivo ou de tópicos de interesse da aplicação, (iii) consulta ao contexto corrente e (iv) consulta as políticas de execução. 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, a funcionalidade padrão de carga e descarga de provisão de contexto utiliza o CIS do MOCA. Tal comportamento pode ser redefinido através da implementação de um aspecto concreto.

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). Porém, independente do comportamento específico, tal método sempre obtém o contexto corrente do *Gerente de Contexto*, repassa o contexto atual para o *Gerente de Políticas de Adaptação* e recebe uma especificação de módulos, a qual é enviada ao *Gerente de Execução*, para que a política possa efetivamente ser aplicada, através da composição dinâmica da aplicação (Figura 3).

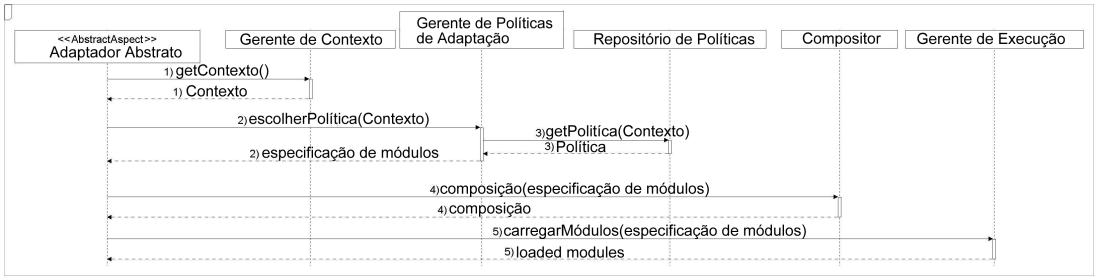


Figura 3. Diagrama de Seqüência parcial do PACCA

Com relação aos aspectos concretos, para a sua definição foi identificado um conjunto de *interesses adaptativos* (referidos na literatura como *adaptive concerns* [Dantas e Borba 2003]), 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 négocio, eles deverão ser aplicados e, por outro lado, quando necessário, podem facilmente estender o modelo proposto, bastando para isso incluir novos aspectos concretos representando o comportamento adaptativo adicional. Na Figura 2 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, (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.

O PACCA provê ainda uma API genérica de acesso à infra-estrutura física de rede. Como em ambientes de computação ubíqua tipicamente há várias redes sem fio com tecnologias distintas coexistindo, essa interface fornece ao desenvolvedor um mecanismo de abstração que apresenta uma visão homogênea dessas redes diferentes e heterogêneas.

2.3 Abordagem de Desenvolvimento

A construção de aplicações adaptativas cientes ao contexto no PACCA requer que o desenvolvedor siga um conjunto de passos. Tais passos caracterizam a abordagem de desenvolvimento de aplicações adaptativas e são adaptações da abordagem tradicionalmente adotada em AOSD, pois incorporam tarefas inerentes ao desenvolvimento de aplicações adaptativas cientes ao contexto. As tarefas são: (i) implementar os componentes de interface gráfica e da lógica de negócio da aplicação usando uma abordagem modular, ou seja, separando as funcionalidades da aplicação em módulos de software que devem ser compostos para originar uma aplicação completa; (ii) definir as políticas de adaptação, as quais descrevem o comportamento adotado pela aplicação em cada contexto, através de regras que determinam quais componentes devem ser carregados em cada possível contexto; (iii) definir o conjunto de eventos contextuais de interesse da aplicação e os componentes de negócio que serão afetados pelos mesmos, bem como os pontos no código dos componentes a serem interceptados (*join points*); (iv) encapsular todo o comportamento adaptativo (definido nos passos ii e iii) em um ou mais aspectos, ou seja, definir os *pointcuts* (conjunto de pontos onde serão injetados comportamento adaptativo) e os *advices* (código que contém o comportamento adaptativo a ser injetado na aplicação nos pontos determinados). Portanto, cada aspecto a ser definido consiste de um conjunto de *advices* e *pointcuts,* onde estão os comportamentos adaptativos e os pontos onde serão injetados esses comportamentos.

O aspecto concreto a ser definido pelo desenvolvedor da aplicação será uma concretização do aspecto abstrato. Como visto, o PACCA já considera um conjunto *default* de interesses adaptativos (como contexto do usuário, do dispositivo e mobilidade), os quais representam um subconjunto dos interesses adaptativos comuns a aplicações ubíquas conforme descrito em [Loughran 2006]. Assim, o PACCA fornece os aspectos concretos relativos aos respectivos interesses adaptativos e o desenvolvedor precisa apenas redefinir os respectivos métodos *adaptar()* de cada aspecto concreto de acordo com as necessidades específicas da aplicação. Ou seja, 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 da aplicação. 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.

3. Estudo de caso: Aplicação para ambiente de medicina ubíqua

Para ilustrar o uso do PACCA e da abordagem de desenvolvimento a ser adotada 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. O cenário requer que as informações relativas aos pacientes sejam concentradas em um servidor e acessadas a partir de diferentes locais (hospital, consultório e residência do paciente). Os dispositivos para visualizar tais informações, bem como o formato dos dados apresentados (textual ou gráfico, criptografado ou não) podem variar, dependendo do usuário e sua localização. Todas essas variações representam o comportamento adaptativo da aplicação, e são possíveis devido ao uso do PACCA.

A aplicação desenvolvida para automatizar o cenário descrito disponibiliza serviços de visualização e prescrição de exames, medicamentos e tratamento, cadastro de diagnósticos, consultas ao histórico do paciente e gerência do tratamento do paciente. Possibilita ainda 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

provido pelo PACCA com a aplicação, é possível usar as informações médicas disponíveis de forma seletiva, a qualquer instante, lugar, e em qualquer dispositivo habilitado.

A aplicação pode ser acessada por profissionais, médicos ou enfermeiros, e pelo paciente, 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 notificar a administração de medicamentos e procedimentos, como p.ex., medir pressão arterial, realizar exames e coletar material fisiológico. Esse perfil não possui permissão de acessar as informações pessoais do paciente, nem de prescrever medicamentos. 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 (histórico, resultado de exames, medicamentos em uso). O perfil *Paciente* é usado pelos pacientes e seus acompanhantes para consultar o sistema a qualquer momento, a fim de obter informações sobre exames solicitados e seus resultados, bem como medicamentos e procedimentos prescritos. O perfil *Administrador do sistema* é encarregado de realizar o cadastro dos médicos, enfermeiros e pacientes aptos a utilizar a aplicação. 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 importância do emprego do PACCA no cenário dessa aplicação médica, um caso comum passível de ocorrer aplica-se aos hospitais que oferecem atendimento *home-care* a pacientes. Eles podem utilizar o PACCA e a aplicação para possibilitar ao médico assistido pelo sistema a dispor, em seu dispositivo portátil, de informações relativas ao paciente a ser atendido. 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 perfil correspondente; (ii) *Tipo e Estado do Dispositivo de Acesso*: usado pela aplicação 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); essa adaptação também inclui verificar o nível de energia atual do dispositivo, e se estiver abaixo de um limiar deve ser exibida para o usuário uma interface baseada em texto, em vez de uma interface gráfica; (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. Quando o estado é *desconectado*, a aplicação utiliza a Base de Dados Local para armazenar informações, repassando-as posteriormente para a Base de Dados Central com o restabelecimento do sinal de rede; (iv) *Localização*: representada por uma *região simbólica* [Sacramento 2004] que reproduz o local do hospital. A localização permite dividir o ambiente de execução em externo e interno; um ambiente interno representa o acesso à aplicação de um ponto dentro do hospital, e é considerado "seguro" na abordagem apresentada. Nesse caso não há 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 (carregando o componente 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 podem mudar ***enquanto*** a aplicação está em execução, e isso é tratado pela abordagem de adaptação baseada em composição dinâmica provida pelo PACCA.

4. Construção da aplicação usando o PACCA

O primeiro passo para implementar uma aplicação usando o PACCA consiste em modelar e implementar os componentes de interface gráfica e da lógica de negócio da aplicação usando uma abordagem modular de desenvolvimento de *software*. Na implementação do protótipo do estudo de caso, foi utilizada a linguagem Java (plataforma JSE versão 5.0) para implementar os componentes da aplicação. A implementação das funcionalidades do PACCA que usam AOP foi feita em duas versões: uma usando o arcabouço orientado a aspectos JBoss AOP [JBoss AOP 2008] e a outra usando AspectJ [AspectJ 2006]. O desenvolvimento de uma versão em AspectJ foi necessário para realizar as medidas de avaliação de desempenho, porém apenas o JBoss AOP provê a capacidade de *weaving* dinâmico necessária para o pleno funcionamento da presente proposta.

Os componentes da aplicação são: *GUI, Módulos Funcionais, Módulo de Armazenamento, Módulo de Criptografia, e Bases de Dados Central e Local*. A *GUI* (Graphic User Interface) 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. Os *Módulos Funcionais* representam a lógica da aplicação (realizar *login*, prescrever/consultar medicamento, etc.).

O *Módulo de Armazenamento* representa o componente da aplicação encarregado da persistência das informações nas *Bases de Dados Central* ou *Local*. 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. O contexto ativador da criptografia é representado por regiões simbólicas de ambientes externos ao hospital. Cabe ressaltar que a definição de ambientes "seguros" foi uma abstração criada no escopo do trabalho, não sendo consideradas questões como sincronização, necessárias para o uso de criptografia. 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 Central*.

O próximo passo no desenvolvimento de uma aplicação consiste na definição das políticas de adaptação através de regras que determinam quais componentes (módulos funcionais) devem ser carregados em cada contexto. Na versão atual não são tratadas questões quanto a interdependência dos componentes a serem carregados, nem quanto a validação das composições geradas. Exemplos de políticas de adaptação para o estudo de caso são representados através de pseudocódigo da Figura 4. No exemplo a política especifica que será carregada a interface *TelaAdminNotebook* quando o perfil do usuário logado for *admin* e o dispositivo corrente em uso pelo usuário for *Notebook*. Essas políticas são definidas de forma declarativa e são armazenadas no *Repositório de Políticas.*

```
Se perfil admin e dispositivo notebook então
    carregar TelaAdminNotebook;
Se ambiente externo e rede conectada então
    carregar BancoCentral e Módulo de Criptografia;
```

Figura 4. Pseudocódigo de política de adaptação

```
<politica nome="AdaptarTela">
   <contexto>
     <Perfil>admin</Perfil>
     <Dispositivo>Notebook</Dispositivo>
   </context>
   <decision>
     <carregar>gui.TelaAdmin</carregar>
   </decisao>
 </politica>
```

Figura 5. Política de Adaptação definida declarativamente

Os próximos passos consistem: (*i*) na definição dos eventos contextuais de interesse da aplicação e dos pontos no código dos componentes a serem interceptados pelo código adaptativo (*join points*); e (*ii*) no encapsulamento do comportamento adaptativo (definido nos passos anteriores) em um ou mais aspectos. Esses passos envolvem o uso de AOP e são descritos nas próximas seções.

4.1 Uso de AOP no PACCA

Como descrito na Seção 3, os eventos de contexto tratados no estudo de caso são: perfis do usuário; tipo e estado da rede; localização e tipo e estado dos dispositivos. Tais interesses são modelados como aspectos, visto que modificam o contexto, provocando algum tipo de adaptação nas aplicações ubíquas, tal como referido em [Loughran 2006]. Dessa forma, o uso da AOP visa prover maior modularidade e separação de interesses, pois nas abordagens tradicionais vários módulos da aplicação invocariam métodos para adaptar a aplicação devido a mudanças no contexto, caracterizando o problema de espalhamento de código. A versão atual considera somente redes sem fio 802.11 (*Wi-Fi*). No estudo de caso considerou-se que todos os Módulos Funcionais da aplicação são afetados pelo contexto, ou seja, devem ser definidos pontos de junção para cada um deles. A Figura 6 ilustra um ponto de atuação representando a execução do método *loginButtonMouseClicked*. 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 recompilar o código.

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

Figura 6. Pointcut AdaptarTela que define a interceptação do método de login

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 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 há políticas para situações específicas. O conjunto de regras para cada interesse adaptativo será especificado através do método *setOperacao()*, chamado a partir do método *adaptar()*. Tal solução torna a busca pela política de adaptação mais eficiente, visto que serão consultadas apenas um subconjunto de políticas contidas no *Repositório de Políticas de Adaptação*. Em seguida, a política retornada é enviada ao *Gerente de Execução* para o carregamento dos módulos requisitados. O código apresentado na Figura 7 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 6.

```
<bind pointcut ="AdaptarTela">
```

```
    <advice name="adaptar" aspect= "adaptacao.AdaptadorTela"/>
</bind>
```

Figura 7. Advice associado ao pointcut AdaptarTela

A Figura 8 apresenta o código do método que implementa o *advice* a*daptar* para o caso da adaptação da Tela. 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 8 está associado ao *pointcut adaptarTela* através da declaração exibida na Figura 7.

```
public Object adaptar(Invocation invocation) {
try{                        return invocation.invokeNext();   }
catch (Throwable e) {       e.printStackTrace();
                            return null; }
 finally {
      GerenteContexto.setOperacao("adaptarTela");
      Contexto contexto = GerenteContexto.getContexto();
     ArrayList<String> politica = GerentePoliticas.escolherPolitica(contexto);
     GerenteExecucao.executarPolitica(politica);}}
```

Figura 8. Código do método que implementa o *advice* adaptar (para adaptarTela)

Como o comportamento adaptativo depende do contexto, no aspecto também é tratada a manipulação de contexto provida pelo CIS. 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* corresponde ao método da aplicação responsável pelo *login* do usuário. O *advice* definido para esse ponto realiza o *login* e, em seguida, recupera da Base de Dados 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 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 as políticas de adaptação definidas para este contexto.

O processo de composição ou *weaving* é realizado de forma transparente pelo arcabouço JBoss AOP através de seu *plugin para* o Eclipse [Eclipse 2007] utilizado para implementar o PACCA em sua versão orientada a aspectos.

Para possibilitar o comportamento adaptativo em tempo de execução, o *Gerente de Execução* encarrega-se da carga dinâmica dos componentes necessários. Para tal, ele usa os mecanismos disponíveis na linguagem Java para carga e descarga dinâmica de classes e realiza as composições de módulos de acordo com o contexto. Em relação à descarga dinâmica são realizadas chamadas explícitas ao *Garbage Collector* de Java para retirar da memória os componentes (classes) não necessários ao contexto. O *Gerente de Execução* também utiliza o princípio da reflexão [Capra 2002] para inspecionar os componentes e ter acesso aos construtores e os seus respectivos parâmetros, obtendo em tempo de execução o construtor adequado para instanciar o componente. Neste trabalho não tratamos composição no nível de interface. Dessa forma, consideramos todos os módulos compatíveis.

Na Figura 9 tem-se um exemplo de possíveis composições dos módulos em execução de acordo com seis diferentes situações contextuais. Na ***situação 1,*** é identificado o perfil

Administrador, implicando que será carregada a *GUI de Administração* para realizar, por exemplo, uma operação de cadastro de Médico. Para isso necessitará que o *Gerente de Execução* carregue o *Módulo de Armazenamento* 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 representado pela ***situação 3***, podendo ser carregada a *GUI Médico Notebook* ou a *GUI Médico Celular*. Para realizar, p, ex., uma operação de cadastro de consulta, é necessário que seja carregado o *Módulo de Armazenamento*. A partir de então, tem-se três novas situações. A ***situação 4*** é 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 as repassará para a *Base Central*. Na ***situação 5*** há a presença de sinal de rede e o ambiente é interno. Nesse caso o *Gerente de Execução* carrega o módulo de conexão com a *Base de Dados Central*. Na ***situação 6*** existe o sinal de rede, porém o ambiente é externo, fazendo com que seja necessário carregar adicionalmente o *Módulo de Criptografia*. Deve-se ressaltar que em qualquer das situações quando houver uma queda no nível da bateria, a *TEXT-INTERFACE* será automaticamente carregada.

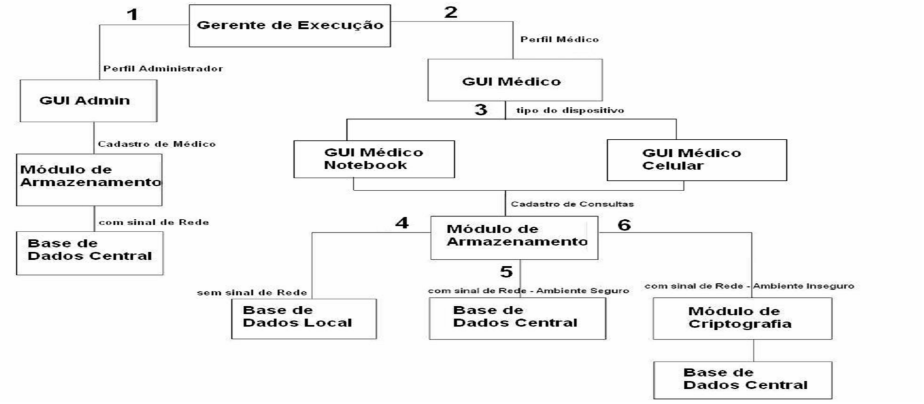


Figura 9. Composição dos Módulos em Execução

5. Trabalhos Relacionados

Em [Mukhija e Glinz 2005] é apresentado o *framework* CASA (*Contract-based Adaptative Software Architecture*), que fornece 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. O foco desse trabalho é nas estratégias de recomposição e substituição de componentes, enquanto no PACCA o foco é o arcabouço de suporte a adaptação ciente de contexto usando, para isso, orientação a aspectos.

[Cámara 2007] apresentou uma proposta de adaptação em tempo de execução para sistemas sensíveis ao contexto. O uso da POA nessa proposta possibilita 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 adaptação de um conjunto de componentes em tempo de execução. O PACCA também utiliza orientação a aspectos, porém, como diferencial, faz o uso de um aspecto abstrato e aspectos concretos, cujo objetivo é permitir que possam ser utilizados interesses adaptativos pré-definidos e incorporados novos interesses adaptativos.

Em [Preuveneers 2006] foi proposto um *framework,* baseado em componentes e construído utilizando o padrão arquitetural em camadas, capaz de realizar adaptações sensíveis ao contexto para um sistema de computação ubíqua. Nesse *framework*, as informações de contexto são modeladas através de uma ontologia de contexto. O *framework* possui estratégias de adaptação dinâmica baseadas na carga dinâmica de componentes visando economia de memória. O principal diferencial em relação ao presente trabalho é que no PACCA adota-se a orientação a aspectos com o intuito de separar o código da aplicação do código da adaptação, proporcionando código mais legível, permitindo a inserção ou modificação de estratégias de adaptação de maneira mais flexível e simples. Além disso, o PACCA pode utilizar diferentes sistemas de *middleware* de provisão de contexto, os quais devem fornecer os serviços necessários à captura e notificação sobre mudanças das informações de contexto.

6. Avaliação

Um dos objetivos do PACCA é aumentar a modularidade das aplicações adaptativas cientes de contexto, obtendo assim, as vantagens decorrentes da modularização, ou seja, produzindo aplicações mais fáceis de compreender, manter, gerenciar e evoluir. Um 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 na computação ubíqua, caracterizados por dispositivos com recursos computacionais limitados.

O processo de avaliação consistiu em comparar duas versões da aplicação descrita no estudo de caso: uma Orientada a Objetos (OO) e sem composição dinâmica e outra Orientada a Aspectos (OA) e com composição dinâmica. Na versão OO, o único componente do PACCA implementado foi o *Gerente de Contexto.* O código relativo ao comportamento adaptativo foi embutido no código da aplicação, encontrando-se, portanto espalhado pelos seus módulos funcionais. A implementação da versão OO teve 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 usou-se AspectJ, pois como no JBoss AOP não há distinção entre classes e aspectos, o uso das ferramentas de medição OA disponíveis é dificultado.

6.1 Métricas de Modularização

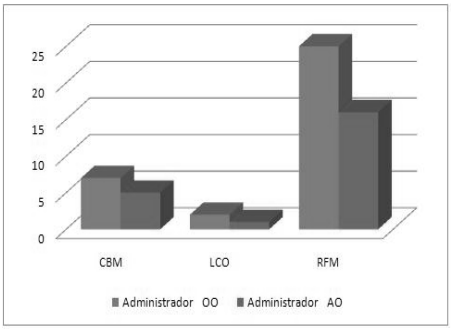
A construção de código modular reduz a complexidade das aplicações e possibilita o desenvolvimento dos módulos de forma independente, com cada módulo concentrando e tratando um requisito específico. No presente trabalho, os principais indicadores usados para identificar as melhorias na modularização 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, já 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 como um todo. 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 como um todo 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

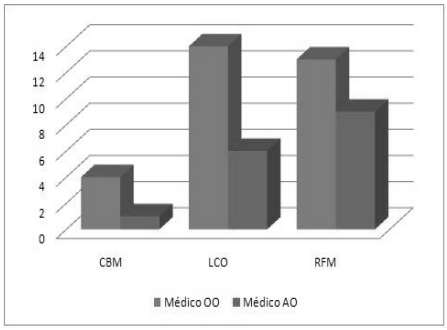
com múltiplos interesses, o esforço para entender esse módulo é grande. No presente trabalho, as métricas LOC (*Lines of Code*) e RFM (*Response for a Module)* foram utilizadas para avaliar a compreensibilidade. A mantenabilidade é uma medida do grau de modificabilidade do código base e é negativamente afetada pelas dependências entre os módulos. No trabalho, foram usadas as métricas de acoplamento e coesão CBM (*Coupling Between Modules*) e CMC (*Coupling on Method Call*), 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].

6.2 Resultados

A Figura 10 apresenta os resultados da avaliação. As medições foram realizadas nos módulos do *Administrador* e do *Médico*, porém poderiam ser escolhidos quaisquer outros módulos que contivessem comportamento adaptativo transversal.



Módulo *Administrador*



Módulo *Médico*

Figura 10. Métricas de comparação entre as versões AO e OO

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 na Figura 10 percebe-se que, com a retirada do comportamento adaptativo do código base, há uma redução de cerca de 30% no acoplamento identificado na versão OO em relação à versão AO, para o módulo *Administrador*, e uma redução de 75% para o módulo *Médico*. Tal melhoria deveu-se ao fato de, na abordagem AO, a adaptação passar ser encapsulada nos aspectos. Observa-se também que a melhoria em relação ao acoplamento é maior quanto maior o número de interesses adaptativos manipulados. Como os módulos que implementam as funcionalidades do *Médico* estão sujeitos a um número maior de adaptações do que os módulos do *Administrador*, o resultado dessa métrica na versão AO é significativamente melhor nesses módulos em comparação com os de *Administrador.*

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 dos outros e isso implica em 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 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. Na Figura 10 pode ser vista uma redução de 50% na falta de coesão proporcionada pela abordagem orientada a aspectos para o módulo *Administrador* e de 57,2% para o módulo *Médico*.

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 um mensagem recebida por um dado módulo. Essa métrica em sistemas AO 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 36% para o módulo *Administrador* e 30,8% para o módulo *Médico*.

Com relação à métrica de número de linhas de código (LOC), obteve-se o valor de 5.178 para a versão OO e de 3.880 para a AO. Portanto, verifica-se que a retirada do código de adaptação espalhado pela lógica da aplicação e a sua concentração no aspecto reduz significativamente o esforço de programação, pelo reuso do código contido no aspecto. 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 1 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 tais 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 AOSD permite a produção de aplicações com um uma maior manutenibilidade, devido ao código de adaptação estar concentrado em uma porção menor de código.

Tabela 1. Tamanho em disco e número de classes do código base e do código OA

Implementação	Tamanho do Código em Disco	Classes
Código Base	298 Kbytes	27
Código Orientado a Aspectos	83 Kbytes	5 + 3 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 GUI 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 6 da Figura 9, foi escolhido pela necessidade de carregamento de vários módulos ao mesmo tempo, exigindo um consumo grande de memória. A Tabela 2 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 2. Consumo de Memória das Versões OO e com Composição dinâmica

	Situação (a)	Situação (b)
PACCA – Versão Monolítica	1.50 MB	1.69 MB
PACCA – Versão com composição Dinâmica	1.25 MB	1.41 MB

Conforme visto na Tabela 2, 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 3 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 que é mais vantajoso carregar e descarregar os módulos de forma dinâmica do que mantê-los em memória, já que dessa forma consegue-se diminuir a quantidade de memória no dispositivo sem comprometer o desempenho.

Tabela 3. Tempo de Carga dos módulos

Módulo	Tempo de carga (ms)
Tela de Administração	364 ms
Tela do Médico	350 ms
Módulo de Armazenamento	20 ms
Criação da conexão com a Base de Dados	219 ms
Módulo de Criptografia	32 ms

7. Conclusões

Esse artigo 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 ao contexto. A abordagem proposta busca aumentar o grau de modularidade e flexibilidade na construção de aplicações ubíquas. A principal contribuição reside no uso conjunto das técnicas de AOSD 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 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.

Referências

AOP Metrics (2008). Disponível em: http://aopmetrics.tigris.org/. Acesso em: abr 2008.

AspectJ (2006). Disponível em: http://www.eclipse.org/aspectj. Acesso em: abr 2006.

AOSD (2008). Aspect-oriented Software Development Web Site. http://aosd.net. Acesso em: abr 2008.

Cámara, J., Salaun, G. and Canal, C. (2007) On Run-time Behavioural Adaptation in Context-Aware Systems. 1st Workshop on Model-driven Software Adaptation at ECOOP 2007.

Canal, C., Murillo, J.M. and Poizat, P. (2006) Software Adaptation. L'Objet, 12(1), 2006. Special Issue on WCAT'04.

Capra, L., et al., (2002) Exploiting Reflection in Mobile Computing Middleware. ACM SIGMOBILE.

Dantas, A. and Borba, P. (2003) Developing Adaptive J2ME Applications Using AspectJ. Journal of Universal Computer Science, vol. 9, no. 8.

Eclipse (2007). Disponível em: <http://www.eclipse.org/> Acesso em: 02 nov. 2007.

Hoh, S., Tan, J.S. and Hartley, M. (2006) Context-aware systems - a primer for user-centred services. BT Technology Journal, v. 24 (2), april 2006, pp. 186 – 194, Kluwer Academic Pubs, MA, USA.

JNDI (2008). Java Naming and Directory Interface (JNDI). Disponível em: http://java.sun.com/products/jndi/ Acesso em: 05 abr 2008.

JBoss AOP (2008). Disponível em: http://labs.jboss.com/jbossaop Acesso em: 08 mar 2008.

Kiczales, G. et al. (1997) Aspect-Oriented Programming. In Proc. of ECOOP. Finlândia, 1997.

Le Sommer, N., Guidec, F. and Roussain, H. (2006), A Context-Aware Middleware Platform for Autonomous Application Services in Dynamic Wireless Networks. 1st International Conference on Integrated Internet Ad hoc and Sensor Networks, May 2006, France.

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

Mukhija, A. and Glinz, M., (2005) Runtime Adaptation of Applications through Dynamic Recomposition of Components. 18th International Conference on Architecture of Computing Systems (ARCS 2005), Innsbruck, Austria, March 2005, pp. 124-138.

Preuveneers, D. et al. (2006), Context-aware adaptation for component-based pervasive computing systems. In Procs. of Pervasive 2006, Dublin.

Rocha, R.C.A, Endler (2006), M. Supporting Context-Aware Applications: Scenarios, Models and Architecture. XXIV SBRC, Curitiba, Brasil, 2006.

Rocha, R.C.A., Casanova, M. A. and Endler, M. (2007) Promoting efficiency and separation of concerns through a hybrid model based on ontologies for context-aware computing. 4th IEEE Workshop on Context Modeling and Reasoning (CoMoRea), New York-USA.

Sacramento, V. et al., (2004) MoCA: A Middleware for Developing Collaborative Applications for Mobile Users. ACM/IFIP/USENIX Middleware Conference, Toronto, 2004.

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

XML (2008) W3C eXtensible Markup Language**.** Disponível em: <http://www.w3.org/XML/> Acesso em: fev 2008.