Utilizando Reconfiguração Dinâmica e Notificação de Contextos para o Desenvolvimento de *Software* Ubíquo

Lincoln S. Rocha[1], Carlos E. Pontual de L. Castro[2], Javam C. Machado, Rossana M. C. Andrade

Grupo de Redes, Engenharia de Software e Sistemas (GREat)
Universidade Federal do Ceará (UFC)
Campus do Pici, Bloco 910, CEP 60455-760, Fortaleza – CE, Brasil
{lincoln,carlospontual}@great.ufc.br,{javam,rossana}@ufc.br

Resumo. *A computação ubíqua tem se mostrado um paradigma computacional de grande abrangência, com aplicabilidades que se estendem de soluções para um cidadão comum até o tratamento de informações complexas em ambientes hospitalares. Entretanto, possíveis variações de recursos, sensibilidade ao contexto e adaptação são desafios a serem vencidos neste ambiente de alta mobilidade e heterogeneidade. O objetivo deste artigo é utilizar mecanismos de reconfiguração dinâmica e notificação de contexto para facilitar o desenvolvimento de aplicações móveis e ubíquas sensíveis ao contexto. Para isso, um middleware adaptativo, denominado AdaptativeRME, é implementado provendo esses mecanismos.*

Abstract. *Ubiquitous computing has become a computational paradigm suitable to a wide range of different applications, for example, they can be solutions to regular citizens or useful to handle complex medical environment data. However, resources variations, context-aware, and adaptation are challenges to be overcome in this high mobility and heterogeneous environment. The main goal of this paper is to use dynamic reconfiguration and context notification mechanisms to help the development of mobile and ubiquitous context-aware applications. In order to apply these mechanisms, an adaptive middleware, called AdaptativeRME, that provides these mechanisms, is implemented.*

1. Introdução

Levando em consideração os ambientes computacionais predominantes, pode-se dividir a história da computação em três gerações distintas: a passada dos *mainframes*, a atual dos computadores pessoais e a futura da computação ubíqua. Idealizada por *Mark Weiser* [Weiser, 1991], a computação ubíqua promete unir características de pervasividade com a alta mobilidade da computação móvel [Araújo, 2003], com o

[1]Bolsista de mestrado (Mestrado e Doutorado em Ciência da Computação/UFC) financiado pela CAPES.
[2]Bolsista de iniciação científica (Departamento de Computação/UFC) financiado pelo PIBIC/CNPq.

intuito de prover aos usuários acesso a informação e processamento a qualquer instante e de qualquer lugar.

Avanços contínuos nas áreas de microeletrônica e telecomunicação têm contribuído significativamente para concretizar as idéias da computação ubíqua [Hartwig e Buchmann, 2007] [KDDI Corporation, 2006]. Não é difícil encontrar dispositivos móveis capazes de capturar e processar informações sobre o ambiente onde executam (e.g., posição geográfica e temperatura ambiente) e ao mesmo tempo habilitados a se conectar a internet através das mais variadas tecnologias de comunicação (e.g., *bluetooth*, IEEE 802.11 e GPRS - *General Packet Radio Service*).

Tais avanços impulsionam novos tipos de aplicações que buscam melhor atender ao usuário levando em consideração o contexto onde este está inserido. Em [Dey, 2001] é definido contexto como sendo "qualquer informação que pode ser usada para caracterizar uma situação de uma entidade". Neste caso, uma entidade pode ser uma pessoa, um lugar ou um objeto que é considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a própria aplicação. Desta forma, a sensibilidade ao contexto é a palavra chave para computação ubíqua [Bardram, 2004] e, sendo assim, aplicações ubíquas devem ser capazes de observar o contexto e modificar seu comportamento em função de variações no mesmo.

Entretanto, além da alta heterogeneidade e dinamicidade de ambientes ubíquos, aspectos como variação na disponibilidade de recursos e serviços devem ser levados em consideração durante o processo de desenvolvimento de *software* para esses ambientes [Endler e Silva, 2001]. Assim, capturar, interpretar cada possível contexto de execução e utilizá-lo para delinear o comportamento do *software* ubíquo são atividades não triviais.

Neste cenário, é necessário que o sistema avalie o contexto de execução e adapte-se em função de suas variações. Tal adaptação está relacionada com a capacidade de reconfiguração dinâmica do sistema em resposta a variações no contexto. Sistemas de *middleware* adaptativos surgem então como uma alternativa que provê uma camada de abstração para facilitar o desenvolvimento de sistemas ubíquos.

Desta forma, um *middleware* adaptativo pode reconfigurar-se dinamicamente com o intuito de garantir o atendimento a determinado requisito não funcional (e.g., qualidade de serviço e segurança) ou ainda garantir que o conteúdo acessado seja adaptado às características de um determinado dispositivo ou usuário. Além disso, um *middleware* adaptativo pode notificar as aplicações sobre eventuais variações no contexto, possibilitando que elas próprias tomem decisões em função destas variações.

Este artigo propõe a incorporação de um Serviço de Invocação Remota de Métodos Sensível ao Contexto (SIRMSC) e um Serviço de Notificação de Contextos (SNC) ao *middleware* RME (*Remote Method Invocation for* J2ME) [Pereira et al., 2006]. O modelo de reconfiguração dinâmica implementado no SIRMSC tem por objetivo garantir que determinados requisitos não funcionais sejam atendidos, dinamicamente, durante uma invocação remota. Já o SNC permite que aplicações possam registrar interesse em determinados contextos de tal forma que sejam notificadas quando os mesmos estiverem ativos. Esta proposta é denominada AdaptiveRME, uma versão adaptativa de RME que oferece suporte ao desenvolvimento de aplicações móveis e ubíquas sensíveis ao contexto.

O restante deste artigo é descrito como segue: a Seção 2 aborda a adaptabilidade de *software* focando nos modelos de adaptação e nas tecnologias utilizadas para fazer adaptação dinâmica; na Seção 3 são discutidos alguns trabalhos relacionados; na Seção 4 é apresentada a arquitetura de AdaptiveRME e detalhes sobre o SIRMSC e o SNC; em seguida, na Seção 5, um estudo de caso é descrito; por fim, na Seção 6 são apresentadas as conclusões e possíveis linhas de pesquisa para trabalhos futuros.

2. Adaptação de *Software*

Na literatura, a adaptabilidade de sistemas de *software* é, em geral, referenciada utilizando como sinônimo as palavras "adaptável" e "adaptativo". Entretanto, no escopo deste trabalho, um *software* adaptável é entendido como sendo um sistema que permite adaptar facilmente uma estrutura completa ou partes específicas devido a mudanças nos requisitos [Tekinerdogan et al., 1997], ao passo que um *software* adaptativo é entendido como um sistema capaz de modificar dinamicamente seu comportamento em função de variações no contexto do ambiente em que executa [Lieberherr, 1995]. Sendo assim, a adaptabilidade pode ser atingida de maneira estática (adaptável) ou dinâmica (adaptativa).

2.1. Modos de Adaptação

Para que se possa empregar a adaptabilidade no desenvolvimento de *software* é necessário conhecer as maneiras de se fazer adaptação. *Carvalho* e *Andrade* utilizam em [Carvalho e Andrade, 2006] alguns modelos de adaptação, destacando em [Carvalho et al., 2005] os seguintes:

- **Adaptação à descrição** é a capacidade de descrever uma aplicação permitindo sua adequação ou transformação em diferentes linguagens ou plataformas de programação. Nesse caso, a aplicação é descrita em uma meta-linguagem e é transformada em uma linguagem de programação (e.g., Java e C++) ou em uma linguagem de marcação de hipertexto (e.g., XHTML e WML);
- **Adaptação ao dispositivo** é a habilidade que uma aplicação possui de adequar seu modo de execução às características do dispositivo. Essas características podem ser estáticas, como número de cores e dimensões da tela, e dinâmicas, como a quantidade de memória de execução disponível;
- **Adaptação ao contexto** é a propriedade de uma aplicação de se adequar a mudanças no contexto em que executa. As mudanças no contexto podem ser decorrentes, por exemplo, das alterações da localização do dispositivo, do interesse do usuário e da largura de banda de comunicação.

Dependendo do modelo de adaptação escolhido, a aplicação pode ser adaptada em tempo de compilação, inicialização ou execução. Se a adaptação em tempo de compilação é considerada, o engenheiro de *software* deve desenvolver uma aplicação generalizada que será disponibilizada na forma de versões adequadas às características do ambiente de execução. Por outro lado, se a adaptação em tempo de inicialização for adotada, o engenheiro de *software* deve desenvolver uma aplicação que seja passível de ajustes por meio de parâmetros (e.g., variáveis de ambiente e arquivos de configuração). Diferente dos dois modelos citados anteriormente, se a adaptação em tempo de execução é adotada, o engenheiro de *software* deve desenvolver apenas uma única aplicação que

seja capaz de identificar o ambiente de execução onde está inserida e adaptar seu comportamento a ele.

É importante ainda salientar que a adaptação não está restrita apenas ao comportamento da aplicação em função do ambiente de execução alvo, mas também à adequação dos dados aos quais ela pode acessar. Esta maneira de fazer adaptação é denominada **adaptação de conteúdo**.

2.2. Adaptação Dinâmica

Segundo *Mckinley* [Mckinley et al., 2004], uma variedade de técnicas permite ao *software* se adaptar ao ambiente de execução. Tais técnicas possibilitam que a estrutura do *software* seja mudada para reparar erros, melhorar o desempenho, aumentar a disponibilidade e a segurança em resposta a ataques. *Mckinley* aponta duas abordagens gerais para fazer adaptação dinâmica no *software*, são elas: adaptação parametrizada e adaptação composicional. A adaptação parametrizada envolve a modificação de variáveis de ambiente que determinam o comportamento do *software*. Já a adaptação composicional possibilita que módulos sejam adicionados ou substituídos, dinamicamente, com o intuito de melhor ajustar o *software* às variações do ambiente em que execução.

É importante observar que a adaptação parametrizada permite apenas ajustar parâmetros ou configurar o *software* para usar uma estratégia diferente, previamente implementada, não possibilitando a incorporação de novas estratégias. Por isso, a adaptação composicional se mostra mais eficiente, pois permite ao *software* se ajustar de acordo com suas novas necessidades. *Weiser* denomina de *calm computing* [Weiser e Brown, 1996] a adaptação sem a ação direta do homem, sendo este um dos principais objetivos no desenvolvimento de *software* ubíquo, foco deste artigo.

De acordo com *Mckinley,* as tecnologias mais utilizadas para fazer adaptação composicional são: separação de interesses (*separation of concerns*), reflexão computacional, desenvolvimento baseado em componentes e sistema de *middleware*.

A separação de interesses preconiza a identificação e o tratamento individualizado dos diferentes interesses envolvidos no desenvolvimento de um *software*. Esta separação possibilita o desenvolvimento em separado de requisitos funcionais (i.e., lógica de negócio da aplicação) dos requisitos transversais (*crosscutting concerns*), por exemplo, consumo de energia e qualidade de serviço. Desta forma, a decomposição do *software* em módulos independentes simplifica o desenvolvimento, facilita a manutenção e promove o reuso.

Já a reflexão computacional, que se refere à habilidade de um programa examinar sua própria estrutura, estado e representação, divide-se em duas atividades: introspecção, que permite ao *software* observar seu próprio comportamento; e interceptação, que permite ao *software* utilizar as informações observadas para modificar seu comportamento. Através da reflexão computacional, o *software* pode observar detalhes de seu fluxo de execução e modificá-lo, contribuindo, assim, para a portabilidade.

Por outro lado, o desenvolvimento baseado em componentes permite a utilização de módulos de *software* que foram desenvolvidos, testados, compostos e disponibilizados por outros engenheiros de *software*. O desenvolvimento baseado em componentes suporta dois tipos de composição: estática e dinâmica [Mckinley et al.,

2004]. Na composição estática, o engenheiro de *software* pode combinar diversos componentes, em tempo de compilação, e gerar uma aplicação. Já na composição dinâmica, o engenheiro de *software* pode adicionar, remover ou reconfigurar componentes dentro de uma aplicação em tempo de execução.

Diferentemente das demais abordagens para fazer composição dinâmica, o *middleware* fornece uma camada de abstração na qual engenheiros de *software* podem inserir comportamento adaptativo utilizando todas as outras tecnologias descritas anteriormente. Sistemas de *middleware* adaptativos podem modificar seu comportamento em resposta a mudanças de requisitos ou variações no ambiente de execução de maneira transparente para as aplicações [Sadjadi e Mckinley, 2003].

Contudo, uma questão importante é definir o momento em que a adaptação dinâmica deve ser feita. Em geral, esta decisão é tomada pelo próprio *software* que encapsula o código responsável pelo monitoramento do contexto e pela adaptação propriamente dita. Entretanto, esta abordagem pode comprometer o reuso, a manutenção e a qualidade geral do *software* construído. Uma abordagem alternativa é delegar a função de monitoramento do contexto para outra entidade, que se encarrega de notificar as aplicações quando for o momento mais apropriado de se adaptar. Neste cenário, o conceito de *middleware* adaptativo, que é utilizado neste artigo, pode prover mecanismos que permitam que as aplicações registrem interesses em determinados contextos para serem notificadas no momento em que estes forem identificados.

3. Trabalhos Relacionados

Esta seção apresenta trabalhos que abordam a reconfiguração dinâmica e o uso de notificação de contextos em ambientes móveis de forma separada. Não foi encontrada na literatura uma solução como a proposta neste artigo que reúne esses dois mecanismos.

3.1. UIC

UIC (*Universally Interoperable Core*) [Román et al., 2001] não é propriamente um *middleware*, mas uma infra-estrutura formada por componentes abstratos inter-relacionados. UIC permite que componentes concretos sejam combinados estática ou dinamicamente para construir um *middleware* específico. Por ser baseado em componentes, UIC permite que diferentes características de *middleware* possam ser configuradas (e.g., protocolo de rede e de transporte, estabelecimento de conexões, semântica da invocação remota e política de prioridades). A reflexividade é largamente utilizada no UIC para possibilitar a construção de sistemas de *middleware* dinamicamente configuráveis. Sua arquitetura destina-se ao ambiente heterogêneo e dinâmico da computação móvel, buscando resolver problemas relacionados com heterogeneidade de dispositivos, dinamicidade do ambiente e limitação de recursos.

Cada instância de *middleware* derivada de UIC é denominada de personalização. Uma personalização pode ser configurada como cliente (i.e., envia pedidos e recebe respostas), servidora (i.e., recebe pedidos e envia respostas) ou *middleware* híbrido. Uma personalização de UIC pode ainda ser classificada como mono-personalizada ou multi-personalizada. Uma plataforma mono-personalizada é capaz de interagir com somente um tipo de *middleware*, ao passo que uma plataforma multi-personalizada é capaz de interagir com vários sistemas de *middleware* diferentes.

3.2. MoCA

MoCA (*Mobile Collaboration Architecture*) [Sacramento et al., 2004] e [Viterbo Filho et al., 2006] é um *middleware* que oferece suporte ao desenvolvimento de aplicações distribuídas sensíveis ao contexto que envolvem dispositivos móveis interconectados através de redes sem fio infra-estruturadas (IEEE 802.11b/g). Os serviços disponibilizados pelo MoCA provêem meios para coletar, armazenar e processar informações de contexto computacional (e.g., o estado dos recursos dos dispositivos e da rede sem fio) obtidas diretamente dos dispositivos móveis. Além disso, MoCA engloba um conjunto de APIs para o desenvolvimento de aplicações que interagem com esses serviços como consumidores de informações de contexto.

MoCA não se preocupa com o atendimento a requisitos não funcionais de sistemas distribuídos (e.g., qualidade de serviço), pois este não é o seu foco. No entanto, ele fornece um *framework*, denominado *ProxyFramework*, o qual possibilita que aplicações usem as informações de contexto processadas pelo MoCA para tomar decisões acerca de eventuais adaptações nas mensagens trafegadas entre o dispositivo móvel e a rede fixa. Contudo, o MoCA não apresenta nenhum mecanismo que permita fazer reconfiguração dinâmica de seus componentes internos.

3.3. RME

O RME (*Remote Method Invocation for* J2ME) [Pereira et al., 2006] é um sistema de *middleware* orientado a objetos, que permite a invocação remota de métodos segundo uma sintaxe semelhante à de Java RMI. RME foi derivado de Arcademis [Pereira et al., 2006], sendo assim uma instância do mesmo. Existem duas versões de RME implementadas. Uma delas destina-se à plataforma J2SE da linguagem Java, permitindo que aplicações distribuídas usufruam das características e serviços providos por esta edição. A outra versão foi desenvolvida para ser utilizada na configuração CLDC da plataforma J2ME de Java, contendo somente as funcionalidades necessárias a aplicações clientes, isto é, fornece às aplicações distribuídas a capacidade de invocar métodos sobre objetos remotos.

Assim como Java RMI, RME apresenta uma arquitetura orientada a serviços [Baresi et al., 2003]. Em RME, fornecedores de serviços são representados pela implementação dos objetos remotos. A agência de localização é representada por um serviço de resolução de nomes, no qual objetos remotos podem registrar-se e aplicações clientes podem fazer consultas em busca de determinados nomes. Por fim, toda aplicação capaz de utilizar o diretório de nomes para obter informações sobre um objeto remoto e solicitar a execução de métodos remotos sobre esse objeto é denominado cliente.

Com o intuito de facilitar a tarefa de reconfiguração, a maior parte dos componentes de RME são criados a partir de fábricas de objetos, seguindo os padrões de projeto *Abstract Factory* e *Factory Method* [Gamma et al., 1994]. O acesso a cada uma das fábricas de objetos que compõem RME acontece via uma estrutura denominada *broker* cuja implementação segue o padrão de projeto *Singleton* [Gamma et al., 1994]. A configuração do *broker* consiste na definição, em tempo de projeto, de quais fábricas farão parte dele, o que, de certa forma, determina que tipo de *middleware* é gerado. Uma vez definida a configuração do *broker*, ela não mais poderá ser modificada durante a execução da aplicação.

É importante observar, que parte dos sistemas de *middleware* apresentados netas seção está focada na heterogeneidade de *software* e *hardware* (e.g., UIC e RME); parte na reconfiguração dinâmica para atendimento a redefinição de requisitos (e.g., UIC e RME); e parte no suporte ao desenvolvimento de aplicações sensíveis ao contexto (MoCA). Contudo nenhum deles se preocupa com o atendimento simultâneo às três questões. Sendo este o foco do *middleware* adaptativo proposto neste trabalho.

4. Reconfiguração Dinâmica e Notificação de Contexto em AdaptiveRME

Este trabalho propõe a incorporação ao RME de um Serviço de Invocação Remota de Métodos Sensível ao Contexto (SIRMSC), o qual utiliza informações contextuais do dispositivo móvel para reconfigurar dinamicamente o *middleware*, e de um Serviço de Notificação de Contexto (SNC), o qual permite que aplicações registrem interesse em determinados contextos para que sejam notificadas quando estes estiverem ativos. Essa versão adaptativa e dinâmica de RME, AdaptiveRME, possibilita o desenvolvimento de aplicações ubíquas sensíveis ao contexto.

Além de prover os serviços SIRMSC e SNC, AdaptiveRME, diferente de RME, permite que dispositivos móveis atuem como fornecedores serviços. Tal característica permite que dispositivos móveis utilizem recursos de outros dispositivos móveis para os mais variados fins. Câmera fotográfica, mp3 *player* e acesso a *internet* são tipos de recursos que podem ser compartilhados entre dispositivos móveis para criação de ambientes colaborativos e integrados.

4.1. Arquitetura de AdaptiveRME

Para manipular o contexto, AdaptiveRME propõe uma arquitetura dividida em três camadas (Figura 1): Camada de Aquisição de Contexto, Camada de Interpretação de Contexto e Camada de Serviços. Cada uma destas camadas é responsável por uma parte do processamento do contexto, desde sua aquisição até o momento em que o *middleware* é reconfigurado e/ou as aplicações são notificadas.

A Camada de Aquisição de Contexto exerce um papel fundamental na arquitetura proposta, ela fornece uma abstração sobre como as informações contextuais são capturadas e processadas. Informações de contexto são informações que dizem respeito a uma entidade de contexto, podendo ser estáticas (e.g., o tamanho e a escala de cores do *display* de um celular) ou dinâmicas (e.g., localização, memória de execução livre e temperatura do ambiente). As informações contextuais são coletadas por componentes de *software* denominados sensores. Componentes sensores funcionam como *wrappers* que convertem as informações coletadas a partir de sistemas de informação, sistemas operacionais e dispositivos de *hardware*, para um formato passível de manipulação pelo *middleware*. Cada sensor é responsável por coletar e converter um tipo de informação de contexto. Sensores podem ser desenvolvidos por terceiros através da interface Sensor da API de AdaptiveRME. Os sensores são implantados em AdaptiveRME através de XML arquivos de configuração.

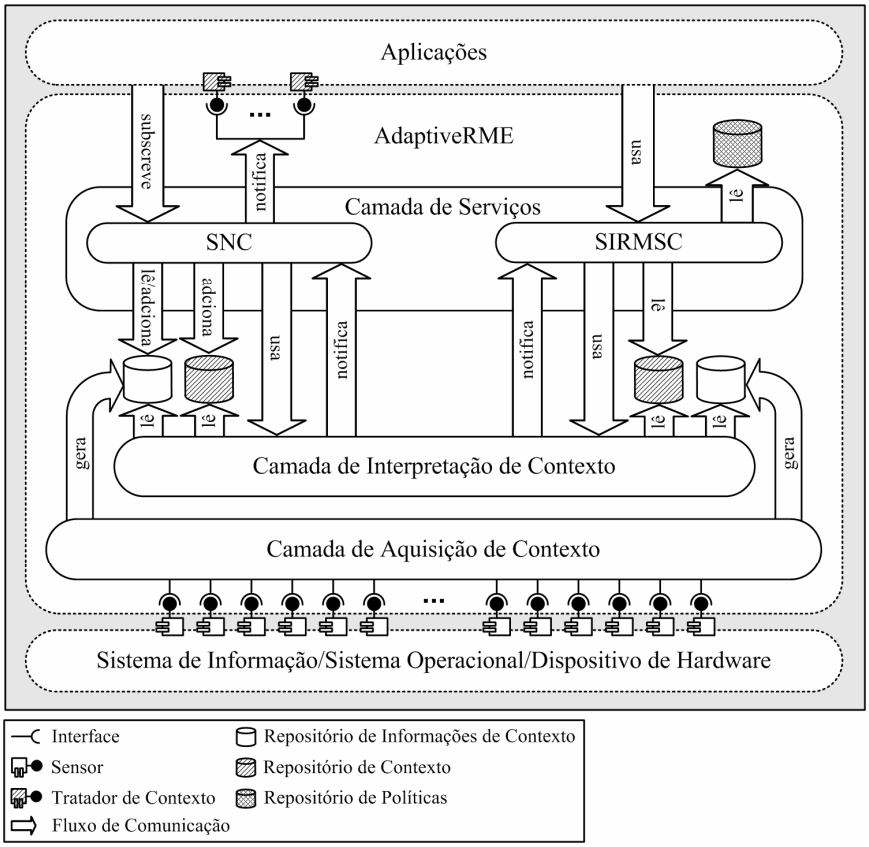


Figura 1: Arquitetura de AdaptiveRME.

Já a Camada de Interpretação de Contexto, é responsável por verificar quais contextos estão ativos num dado momento. Em AdaptiveRME, contexto representa um estado que se deseja observar e pode ser descrito por meio de expressões lógicas que relacionam uma ou mais informações de contexto de uma ou mais entidades de contexto. Um exemplo de expressão lógica que caracteriza um contexto de um dispositivo móvel é: "((SIGNAL_STRENGTH >= 80) && (FREE_MEMORY >= 500))", neste exemplo "SIGNAL_STRENGTH" representa a porcentagem da qualidade do sinal de comunicação de um dispositivo móvel num dado instante e "FREE_MEMORY" representa a quantidade em *bytes* de memória de execução livre no dispositivo. Os símbolos ">=" e "&&" representam, respectivamente, o sinal de maior ou igual e o operador lógico de conjunção. Um contexto é dito ativo se a avaliação da expressão lógica que o representa resultar num valor lógico verdadeiro. Contextos podem ser criados via API de AdaptiveRME ou a partir de arquivos XML de configuração. Contextos criados pelas aplicações utilizando a API são destinados ao SNC. Por outro lado, contextos criados a partir de arquivos de configuração destinam-se ao SAIRM.

Para aferir o contexto, AdaptiveRME faz uso de elementos denominados interpretadores de contexto. Tais elementos avaliam todas as expressões lógicas que representam cada contexto em função das informações de contexto disponíveis. Após avaliar todas as expressões, os interpretadores de contexto fazem a notificação dos contextos ativos para cada um dos serviços da Camada de Serviços.

Por fim, a Camada de Serviços, é responsável por fazer o interfaceamento entre as aplicações e o *middleware* provendo os dois mecanismos essenciais propostos neste artigo e descritos com mais detalhes nas seções seguintes, que são o SIRMSC e o SNC.

4.2. Serviço de Invocação Remota de Métodos Sensível ao Contexto - SIRMSC

O SIRMSC pode ser dividido em três atividades: sincronização, reconfiguração dinâmica e adaptação de conteúdo. A Figura 2 ilustra o fluxo de execução de uma invocação remota sobre o SIRMSC.

Invocações remotas partem da aplicação e são redirecionadas para o *stub* que, por sua vez, aciona o processo de sincronização. Depois de concluída a sincronização, o gerente de configuração reconfigura os componentes do *middleware* permitindo que a invocação remota seja processada normalmente. Dependendo do método invocado, o *skeleton* aciona os adaptadores de conteúdo passando para eles as informações de contexto enviadas pelo dispositivo móvel.

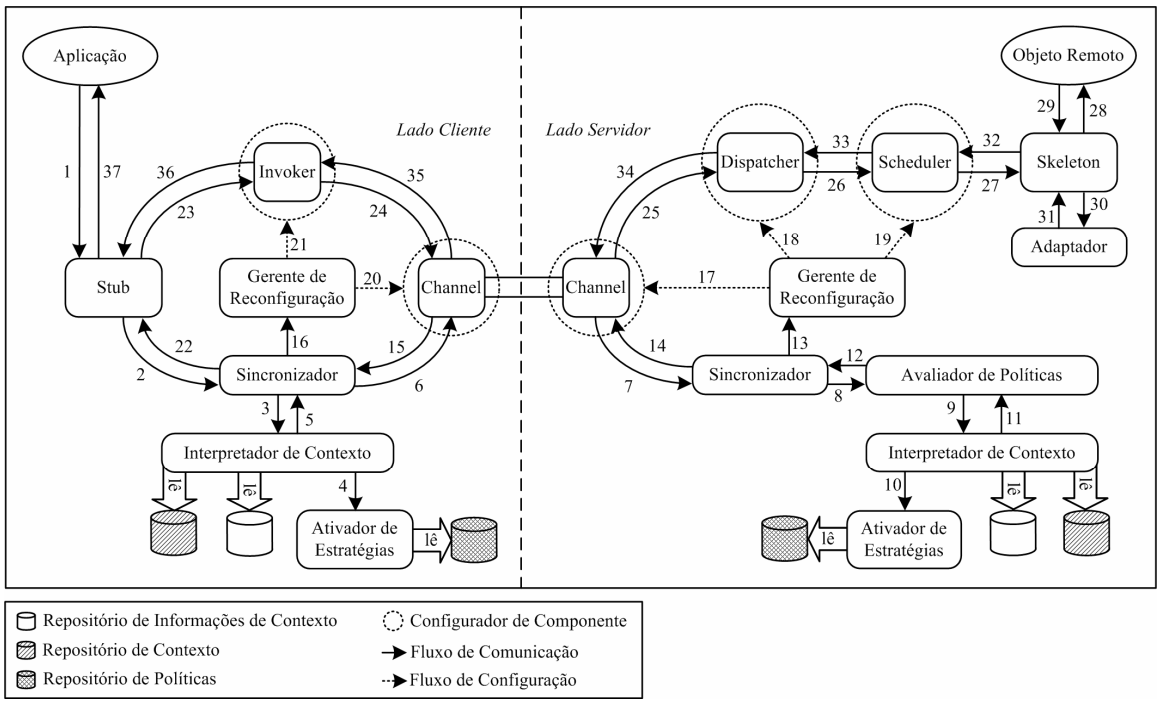


Figura 2: Serviço de Invocação de Métodos Sensível ao Contexto.

Em AdaptiveRME, uma política está relacionada com o atendimento a um requisito não-funcional durante uma invocação remota, ao passo que as estratégias representam a maneira que o requisito não-funcional será atendido pelo *middleware* em função do contexto corrente. Por exemplo, uma política de segurança pode ser implementada por vários algoritmos criptográficos; entretanto, em função do contexto do dispositivo (e.g., memória disponível, nível de bateria e capacidade de processamento), um algoritmo pode ser selecionado em detrimento de outro. Cada política agrupa um conjunto de estratégias que descrevem quais componentes do *middleware* serão reconfigurados. A reconfiguração é realizada por meio de elementos denominados configuradores de componente aplicando o padrão de projeto *Decorator* [Gamma et al., 1995] aos componentes internos do *middleware*.

A Figura 3 ilustra os arquivos utilizados para descrever políticas e contexto. O arquivo *XML Context Descriptor* define como os contextos que serão avaliados durante uma invocação remota devem ser especificados, ao passo que o arquivo *XML Policy Descriptor* define quais políticas e respectivas estratégias serão utilizadas numa invocação remota. As associações entre as estratégias e os contextos são feitas através dos atributos *name* (da *tag context* do arquivo *XML Context Descriptor*) e *context* (da *tag strategy* do arquivo *XML Policy Descriptor*).

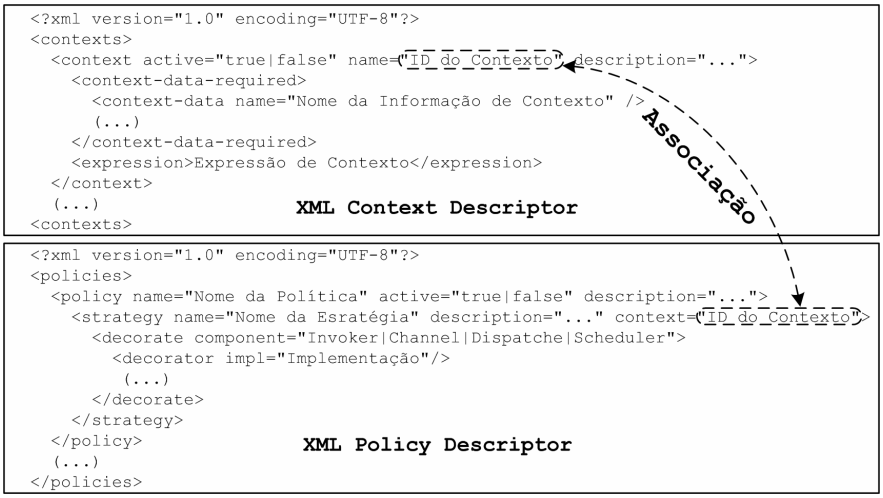


Figura 3: Associação entre contextos e estratégias.

4.2.1. Sincronização

Durante uma invocação remota, todas as políticas descritas no repositório de políticas (do lado cliente) são utilizadas, entretanto, cabe ao processo de sincronização garantir que para cada política utilizada na invocação remota, a mesma estratégia da política seja aplicada tanto no lado cliente quanto no lado servidor.

Acionado pelo *stub*, o sincronizador (cliente) se comunica com o interpretador de contexto que avalia os contextos armazenados no repositório de contexto. Para cada contexto ativo, o interpretador notifica o serviço de invocação que, por sua vez, notifica os ativadores de estratégias. Ativadores de estratégias são responsáveis por selecionar as estratégias que serão negociadas com o servidor em função dos contextos ativos.

Já no lado servidor, o sincronizador se comunica com o avaliador de políticas que decide se as estratégias de reconfiguração requeridas pelo cliente serão utilizadas na comunicação. Para cada estratégia solicitada pelo cliente, pode existir algum contexto associado no lado servidor. Caso exista, o avaliador de políticas utiliza o interpretador de contexto para avaliar cada um dos contextos associados, para só então decidir se a estratégia solicitada pelo cliente será contemplada. Caso a estratégia solicitada pelo cliente não seja contemplada, o próprio avaliador de políticas define qual estratégia será adotada. Contextos avaliados pelo avaliador de políticas podem ser compostos por variáveis de contexto coletadas do ambiente de execução do próprio servidor e/ou por informações de contexto fornecidas pelo cliente durante o processo de sincronização.

4.2.2. Reconfiguração Dinâmica

Após o processo de sincronização, tanto o cliente como o servidor estão conscientes de quais estratégias de reconfiguração deverão ser aplicadas ao *middleware* durante a

invocação. Neste momento, os configuradores de componente são acionados para decorar os componentes seguindo as estratégias de renconfiguração estabelecidas pelo processo de sincronização.

Cada decorador permite que funcionalidades sejam adicionadas e, posteriormente, removidas dinamicamente de um componente. AdaptiveRME permite que os componentes *Invoker*, *Channel*, *Dispatcher* e *Scheduler*, responsáveis, respectivamente, pela invocação (cliente), manipulação do canal de comunicação (cliente/servidor), pelo despacho de requisições (servidor) e pelo escalonamento das requisições recebidas (servidor) sejam decorados dinamicamente. Para isso, cada um destes componentes possui um decorador abstrato (Figura 4).

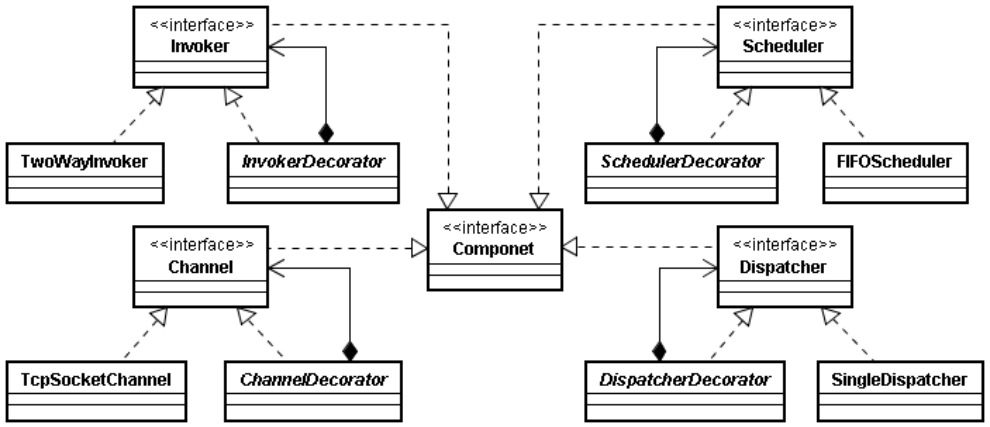


Figura 4: Diagrama de classes dos Decoradores de Componentes.

Cada uma das interfaces (i.e., *Invoker*, *Channel*, *Dispatcher* e *Scheduler*) possui uma implementação concreta e um decorador abstrato. Todas elas estendem a interface Componente que permite que configuradores de componentes ajam sobre suas implementações concretas. A classe *TwoWayInvoker* implementa um invocador que envia requisições e espera o seu retorno de maneira síncrona. A classe *TcpSocketChannel* implementa o canal de comunicação baseado em *sockets* sobre o protocolo TCP/IP. A classe *SingleDispatcher* implementa um *dispatcher monothread*. A classe *FIFOScheduler* implementa o algoritmo FIFO (*First In First Out*) onde a primeira requisição que chega é a primeira a ser atendida.

Em AdaptiveRME, configuradores de componentes funcionam como *Adapters* [Gamma et al., 1994] que encapsulam implementações concretas dos componentes e disponibilizam uma interface que permite reconfigurá-los dinamicamente. Quando um módulo do *middleware* requer uma implementação de algum componente, ele a solicita através do *broker*, que, por sua vez, delega a tarefa à fábrica responsável pela criação do componente requerido. Durante a criação, a fábrica instância uma implementação do componente e a insere no seu respectivo configurador, retornando para o módulo que solicitou a criação, uma referência para o configurador do componente e não para o componente de fato.

A Figura 5 ilustra os configuradores de componentes de AdaptiveRME. Cada configurador realiza duas interfaces, a interface *ComponenteConfigurator* e a interface do componente o qual irá configurar. Desta forma, cada configurador de componente possui a assinatura de métodos responsáveis por reconfigurar o componente e os métodos utilizados para acessar os serviços providos pelo componente. Os métodos

setInternalComponente(Componente) e *getInternalComponente()* são responsáveis, respectivamente, por atribuir e obter uma referência para o componente interno. Já os métodos *decorate(ArcademisVector)* e *undecorate()* são responsáveis, respectivamente, por decorar e fazer o processo inverso no componente interno. Já os métodos que os configuradores implementam das interfaces dos componentes são utilizados apenas para redirecionar requisições para o componente interno.

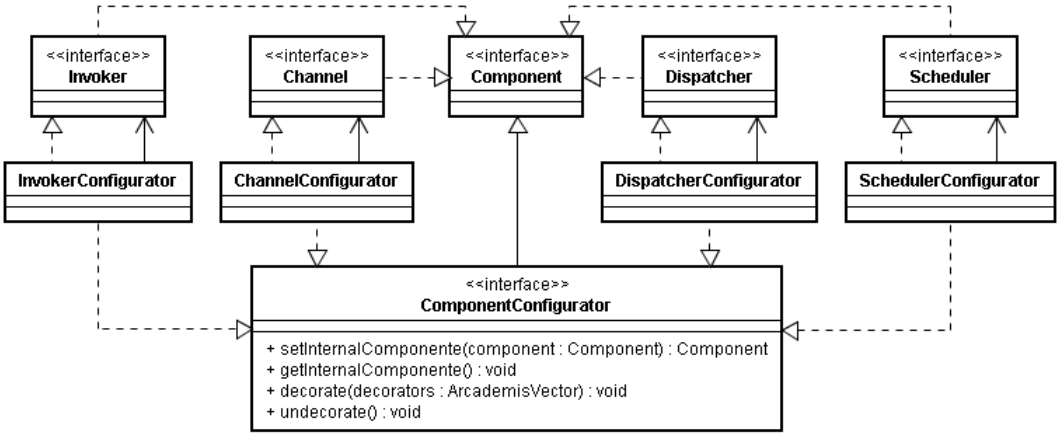


Figura 5: Diagrama de classes do Configurador de Componentes.

4.2.3. Adaptação de Conteúdo

A adaptação de conteúdo em AdaptativeRME é feita por elementos denominados adaptadores. Eles são acionados de dentro do *skeleton* (servidor) e utilizam informações de contexto, enviadas pelo dispositivo, para fazer a adaptação.

Em AdaptiveRME serviços são na verdade métodos de objetos distribuídos, cujo acesso é feito via invocação remota, por isso, cabe ao desenvolvedor alterar o código do *skeleton* do respectivo objeto remoto que provê um determinado serviço, para introduzir adaptadores de conteúdo. Todo *skeleton*, em AdaptiveRME, possui trechos de código associados aos métodos remotos do objeto publicado, nestes trechos devem ser inseridos os adaptadores necessários para adaptar o conteúdo acessado por aquele método.

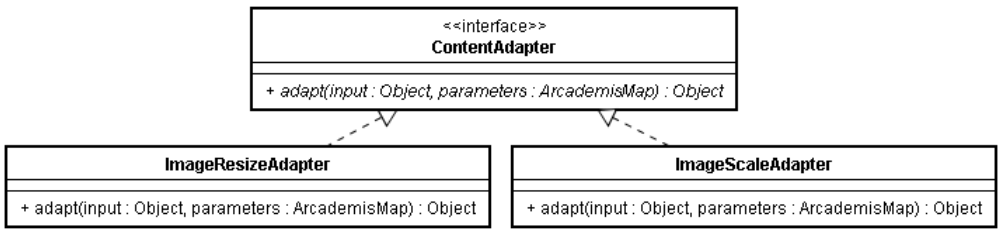


Figura 6: Diagrama de classes do Adaptador de Conteúdo.

Em AdaptiveRME, adaptadores podem ser implementados seguindo a interface *ContentAdapter* (Figura 6). No método *adapt(Object,AcademisMap)*, o argumento input representa o conteúdo que se deseja adaptar e parameters são as informações de contexto necessárias para fazer a adaptação. AdaptiveRME disponibiliza dois adaptadores: *ImageResizeAdapter* e *ImageScaleAdapter*. *ImageResizeAdapter* busca adequar o tamanho (em *bytes*) de uma imagem à capacidade de memória do dispositivo móvel. Enquanto *ImageScaleAdapter* busca re-escalar uma imagem tentando ajustá-la ao tamanho do *display* do dispositivo móvel.

4.3. Serviço de Notificação de Contexto - SNC

O SNC provê uma infra-estrutura para a criação e publicação de serviços contextuais, necessários à construção de aplicações ubíquas. Todo serviço de contexto publicado sobre o SNC é responsável por receber subscrições de aplicações na forma de contexto (i.e., expressões lógicas sobre variáveis que representam informações de contexto) e notificá-las quando estes estiverem ativos. Para cada subscrição, a aplicação deve associar um ou mais tratadores de contexto que serão notificados quando o contexto subscrito estiver ativo. Tratadores de contexto são elementos responsáveis por realizar algum processamento em função de um dado contexto ativo, eles são implementados por terceiros e sua lógica de ação depende dos interesses da aplicação para a qual foram projetados. Tratadores de contexto devem realizar a interface *ContextHandler* da API de AdaptiveRME.

Um serviço de contexto pode ser acessado por dois tipos de dispositivos computacionais: produtores e consumidores.

Os dispositivos produtores são responsáveis por prover as informações de contexto sobre as entidades que o serviço de contexto quer observar. Produtores podem ser dispositivos computacionais quaisquer (e.g., PDAs, computadores pessoais, e eletroeletrônicos em geral), capazes de se comunicar com o serviço de contexto por meio de alguma infra-estrutura de rede. Cada produtor possui um elemento denominado provedor que é responsável por agrupar as informações de contexto, coletadas pelos sensores, e enviá-las para o serviço de contexto associado. Através de arquivos de configuração, o provedor sabe quais informações ele deve enviar para um determinado serviço de contexto.

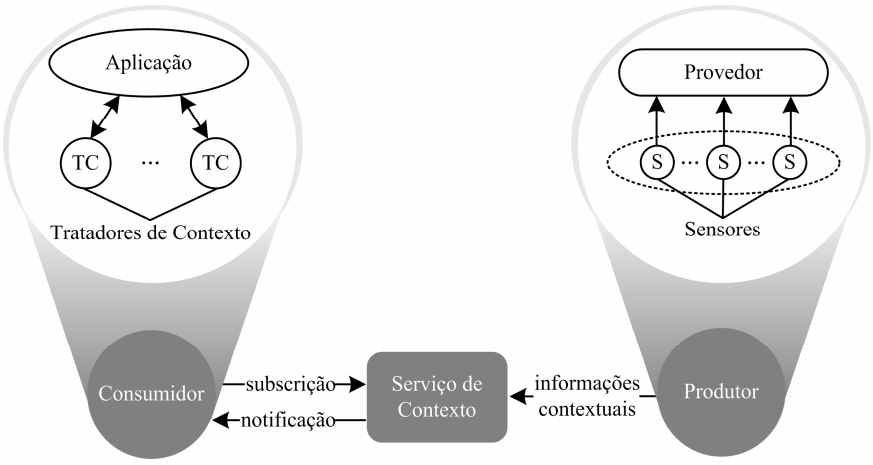


Figura 7: Arquitetura de um Serviço de Contexto.

Consumidores são elementos capazes de fazer subscrições em serviços de contexto. Para que um consumidor utilize um serviço de contexto é necessário que ele saiba, previamente, quais entidades de contexto são monitoradas pelo serviço e quais informações de cada entidade estão disponíveis para consulta, pois como AdaptiveRME não possui um formato padrão para descrever entidades de contexto, não é possível descobrir tais informações dinamicamente, cabendo ao engenheiro de *software* fazer esta checagem em tempo de projeto. A arquitetura típica de um serviço de contexto é ilustrada na Figura 7.

5. Estudo de Caso

Como estudo de caso foi desenvolvida a aplicação UbiPEP, uma versão ubíqua e simplificada do Prontuário Eletrônico do Paciente, a qual ilustra o funcionamento dos dois serviços, SIRMSC e SNC, implementados em AdaptiveRME. UbiPEP foi codificado em Java, sendo a parte cliente em J2ME e a parte servidora em J2SE. O diagrama de classes da Figura 8 apresenta as principais classes da aplicação.

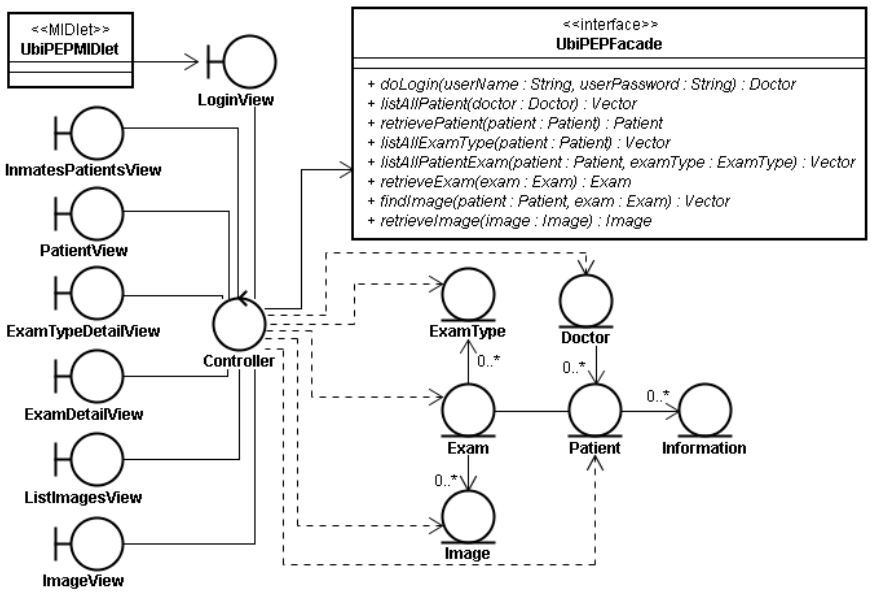
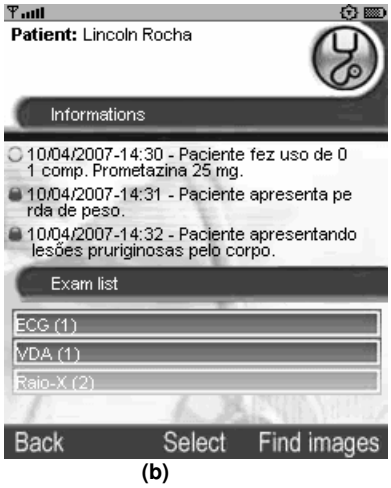


Figura 8: Diagrama de classes do UbiPEP.

O principal objetivo do UbiPEP é permitir que médicos, após se autenticarem (Figura 9-a), possam obter informações sobre pacientes internados através de seus PDAs. Além disso, o UbiPEP permite que o médico veja descrições textuais sobre a evolução dos seus pacientes (Figura 9-b), bem como possibilita o acesso à laudos e imagens (Figura 9-c) de exames realizados. O SIRMSC é utilizado para fazer o acesso às informações dos pacientes e adaptar as imagens dos laudos às capacidades do dispositivo. O objeto remoto acessado pelo UbiPEP através do SIRMSC é uma implementação da interface UbiPEFacade.

(a)



(b)

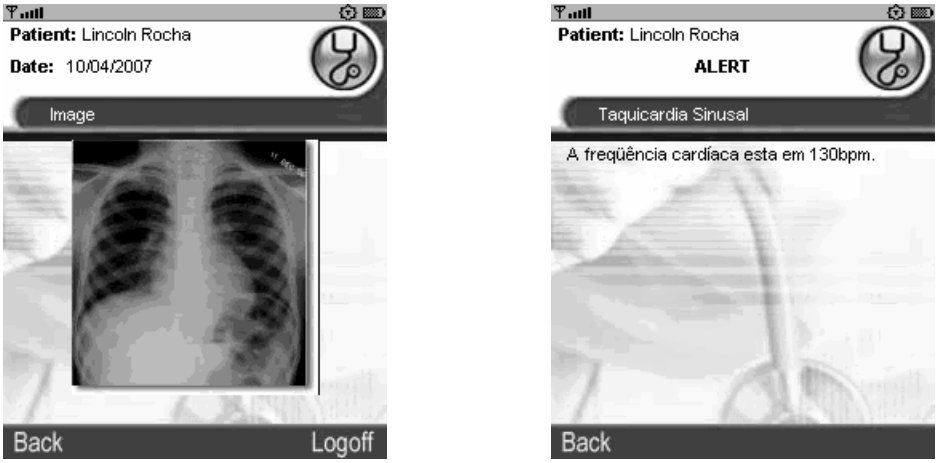


Figura 9: GUI (a) de autenticação, (b) de descrição textual de evolução de pacientes, (c) de exibição de imagens e (d) de exibição de alerta no UbiPEP.

Com o intuito de validar o SNC foi criado um serviço de monitoramento de pacientes cardíacos. Tal serviço monitora a freqüência cardíaca dos pacientes internados, permitindo que aplicações façam subscrições a fim de serem notificadas quando a freqüência cardíaca do paciente variar conforme seus interesses. A classe *Controller* é responsável por fazer as subscrições ao serviço de monitoramento.

```
 1  ArcademisVector inmatesPatients = this.facade.listAllPatient(doctor);
 2  Enumeration e = inmatesPatients.elements();
 3  while (e.hasMoreElements()) {
 4      Patient patient = (Patient) e.nextElement();
 5      if (patient.isInmate()) {
 6        PatientId patientId = new PatientId(patient.getId());
 7        service = new ContextSubscriptionFacade();
 8        service.setLocationService("magneto.great.ufc.br", "HFM");
 9        service.addAlias("E{" + i + "}", patientId);
10        TachycardiaHandler tachycardiaHandler = new TachycardiaHandler();
11        service.addContextHandler("tachycardia", tachycardiaHandler);
12        service.subscribe("Tach" + i, "E{" + i + "}.HEART_FREQUENCY > 100");
13                                    (...)
14      }
15                                    (...)
16  }
```

Figura 10: Fragmento de código da classe Controller.

O fragmento de código da Figura 10 ilustra uma subscrição feita ao serviço de monitoramento. É importante mencionar, que todos os dados manipulados pelo UbiPEP são fictícios, servindo apenas para validar os requisitos funcionais da aplicação. A linha 8 mostra como o endereço da agência de localização (magneto.great.ufc.br) e o nome do serviço de contexto desejado (HFM) são informados. Em seguida, na linha 9 é criado um *alias* para o identificador da entidade de contexto. Na linha 10 é instanciado o tratador de contexto *TachycardiaHandler* que ao ser notificado, faz com que o PDA emita um sinal sonoro e exiba uma tela de alerta informando o nome e a freqüência cardíaca do paciente monitorado (Figura 9-d).

6. Conclusões e Trabalhos Futuros

Este artigo apresentou a utilização de um modelo de reconfiguração dinâmica e um serviço de notificação de contexto no AdaptiveRME, um *middleware* adaptativo que

oferece suporte ao desenvolvimento de aplicações móveis e ubíquas. Além disso, foram apresentados a arquitetura do *middleware* proposto e um estudo de caso que mostra a sua utilização.

Um ponto importante a ser trabalhado em AdaptiveRME é a padronização na descrição de entidades contextuais. Nessa linha de pesquisa, estudos apontam para o uso de ontologias [Chen et al., 2004] [Chen et al., 2005] [Kim et al., 2006] como ferramenta para descrição e representação de entidades contextuais. Outro ponto a ser investigado é o impacto no desempenho causado pelo SIRMSC durante a comunicação. Estudos preliminares mostram que o *overhead* imposto pelo SIRMSC, quando comparado com o serviço de invocação remota de métodos de RME puro, afeta o desempenho de maneira aceitável. Contudo, o desempenho é um fator que deve ser melhor investigado em AdaptiveRME. Finalmente, outro trabalho futuro é o estudo de composição dinâmica de serviços em ambientes ubíquos.

Referências Bibliográficas

Araújo, R. B. (2003). "Computação Ubíqua: Princípios, Tecnologias e Desafios". In: XXI Simpósio Brasileiro de Redes de Computadores, 2003, Natal, p. 45-115.

Bardram, J. E. (2004). "Applications of Context-Aware Computing in Hospital Work – Examples and Design Principles". In: ACM Symposium on Applied Computing, 19. Nova Iorque, NI, EUA: ACM Press, 2004, p. 1574-1579.

Carvalho, W. V.; Fernandes, P.; Teixeira, R.; Andrade, R. M. C. (2005). "Mobile Adapter: Uma abordagem para a construção de Mobile Application Servers adaptativos utilizando as especificações CC/PP e UAProf". In: Seminário Integrado de Software e Hardware, 32., 2005. São Leopoldo, RS, BRA, p. 1914-1929.

Carvalho, W. V. ; Andrade, R. M. C. (2006). "Uma Proposta para a Geração Semi-automática de Aplicações Adaptativas para Dispositivos Móveis". In: Simpósio Brasileiro de Engenharia de Software, 2006. Florianópolis, SC, BRA.

Chen, H.; Perich, F., Finin, T.; Joshi, A. (2004). "SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications". In: Proceedings of International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004.

Chen, H.; Finin, T.; Joshi, A. (2005). "Ontologies for Agents: Theory and Experiences". Whitestein Series in Software Agent Technologies and Autonomic Computing, Birkhäuser Basel, 2005.

Dey, A. K. (2001). "Understanding and Using Context". In: Personal and Ubiquitous Computing, v. 5, n.1, p. 4-7, 2001.

Endler, M.; e Silva, F. S. (2001). "Desenvolvendo Software Adaptável para Computação Móvel". In: Workshop de Comunicação sem Fio e Computação Móvel, 3., 2001. Recife, PE, BRA. p. 93-101.

Gamma, E.; Helm, R.; Johnson, R.; e Vlissides, J. (1995). "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

Hartwig, S.; Buchmann, M. (2007). "Empty Seats Traveling: Next-generation ridesharing and its potential to mitigate traffic and emission problems in the 21st century". Nokia Research Center Bochum, Germany, 2007. Disponível em: <http://research.nokia.com/tr/NRC-TR-2007-003.pdf>.

KDDI Corporation: Ubiquitous Solution Company (2006). Relatório Anual. Disponível em:<http://www.kddi.com/english/corporate/ir/library/annual_report/2006/pdf/kddi_ar2006_e.pdf>. Último acesso em: 10 de Janeiro de 2007.

Kim, Y.; Kim, E.; Kim, J.; Song, E., Ko, I. (2006). "Ontology Based Software Reconfiguration in a Ubiquitous Computing Environment". In: Proceedings of the Sixth IEEE International Conference on Computer and Information Technology (CIT'06), p. 260, v. 00, 2006.

Lieberherr, K. (1995). "Workshop on Adaptable and Adaptive Software". In: Conference on Object Oriented Programming Systems Languages and Applications, Austin, Texas, 1995, p. 149 - 154.

Mckinley, P. K.; Sadjadi, S. M.; Kasten, E. P.; Cheng, B. H. (2004). "Composing Adaptive Software". IEEE Computer Magazine, v. 37, n. 7, p. 56-64, Julho de 2004.

Pereira, F. M. Q.; Valente, M. T. O.; Bigonha, R. S. e Bigonha, M. A. S. (2006). "Arcademis: a Framework for Object-Oriented Communication Middleware Development", ACM Software-Practice & Experience, v 36 n. 5, p. 495 – 512, 2006.

Román, M.; Kon, F. e Campbell, R. (2001). "Reflective Middleware: From Your Desk to Your Hand". 5 ed., IEEE Distributed Systems Online, v. 2, 2001.

Sacramento, V.; Endler, M.; Rubinsztejn, H. K.; Lima, L. S.; Gonçalves, K.; Nascimento, F. N.; e Bueno, G. A. (2004). "MoCA: A middleware for developing collaborative applications for mobile users". In: IEEE Distributed Systems Online.

Sadjadi, S. M.; Mckinley, P. K. (2003). "A survey of Adaptive Middleware". Technical Report MSU-CSE-03-35, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, dec. 2003. Disponível em: <http://35.9.20.31/~sadjadis/Publications/AdaptiveMiddlewareSurvey.pdf>. Último acesso em: 10 de Janeiro de 2006.

Tekinerdogan, B.; (1997) Adaptability in Object-Oriented Software Development. Workshop Report. In: Special Issues in Object-Oriented Programming, M. Muhlhauser (ed.). dpunkt-Verlag, Heidelberg, Germany, p. 7-12.

Viterbo Filho, J.; Sacramento, V.; Rocha, R. C. A.; Markus Endler (2006). "MoCA: Uma Arquitetura para o Desenvolvimento de Aplicações Sensíveis ao Contexto para Dispositivos Móveis". In: Simpósio Brasileiro de Redes de Computadores, Sessão de Ferramentas, 2006, Curitiba.

Weiser, M. (1991). "The Computer for the 21st Century". Scientific American. v. 265, n. 3, p. 94-104, 1991.