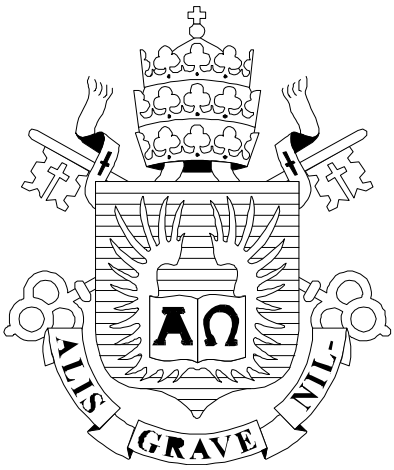

PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 37/06

Experiências no Desenvolvimento de uma Arquitetura de Middleware para Ciência de Contexto

Markus Endler, Vagner Sacramento, Hana Rubinsztejn,
Fernando Ney do Nascimento, Ricardo C.A. da Rocha,
José Viterbo Filho, Gustavo Luiz B. Baptista

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 37/06 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena December, 2006

Experiências no Desenvolvimento de uma Arquitetura de Middleware para Ciência de Contexto

Markus Endler, Vagner Sacramento[1], Hana Rubinsztejn, Fernando Ney do Nascimento, Ricardo C. A. da Rocha, José Viterbo Filho, Gustavo Luiz B. Baptista

[1] Instituto de Informática
Universidade Federal de Goiás
CEP 74690-815
Goiânia – GO – Brasil

endler@inf.puc-rio.br, vagner@inf.ufg.br, hana@lac.inf.puc-rio.br
{ney, rcarocha, viterbo, gustavo}@lac.inf.puc-rio.br

Abstract. MoCA is a middleware for developing and deploying context-aware collaborative applications for mobile users. It comprises a service for collecting, storing and distributing context data acquired from mobile devices, a service for inferring the location of such devices, and a framework for developing proxies, among others. This article discusses the experience obtained from developing this architecture and some context-aware applications that were built upon these services.

Keywords: Context-Awareness, middleware, mobile computing

Resumo. MoCA é uma arquitetura que oferece recursos para o desenvolvimento e execução de aplicações colaborativas sensíveis ao contexto que envolvem usuários móveis. Esses recursos incluem um serviço para a coleta, armazenamento e distribuição de informações de contexto, um serviço de inferência de localização de dispositivos móveis, e um framework para desenvolvimento de proxies, entre outros. Este artigo apresenta a experiência adquirida no desenvolvimento desta arquitetura e de aplicações sensíveis ao contexto que contam com estes serviços como base.

Palavras-chave: Ciência do contexto, middleware, computação móvel

In charge for publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

1 Introdução

O uso de dispositivos portáteis e a grande de mobilidade dos usuários nas redes sem fio têm aumentado o interesse por aplicações sensíveis ao contexto. Os desenvolvedores destas aplicações geralmente exploram aspectos/propriedades do ambiente operacional (i.e., contexto computacional, pessoal e físico) para oferecer serviços customizados ao usuário, mudar o comportamento da aplicação de acordo com contexto inferido (e.g., localização corrente do usuário) ou dar início a alguma adaptação para lidar com as limitações ou variações do ambiente computacional.

Entretanto, a diversidade de informações de contexto que podem ser exploradas e a abundância de tecnologias de sensoriamento tornam cada vez mais complexo o desenvolvimento e a implantação de sistemas sensíveis ao contexto [1]. Sendo assim, é desejável desacoplar a lógica da aplicação do *software/hardware* responsáveis pelo sensoriamento/coleta de contexto por dois motivos principais: i) para reduzir a complexidade no desenvolvimento requerida na interação com os sensores para realizar a coleta; e ii) para evitar a sobrecarga no processamento da informação coletada.

Com o intuito de oferecer uma infra-estrutura que auxiliasse o desenvolvimento de tais aplicações, projetamos e implementamos uma arquitetura de provisão de contexto chamada **M**obile **C**ollaboration **A**rchitecture - MoCA [2]. Esta arquitetura é baseada em serviços que realizam a coleta, o processamento e a divulgação do contexto computacional dos dispositivos móveis, da rede sem fio IEEE 802.11 e da localização do usuário. Esses serviços têm sido utilizados como base para o desenvolvimento de outros middlewares e várias aplicações sensíveis ao contexto.

Apesar de existirem várias outras infra-estruturas de provisão de contexto [3] , que inferem e divulgam a localização do usuário de redes sem fio IEEE 802.11, nós decidimos projetar e implementar uma infra-estrutura equivalente por dois motivos principais: o interesse do grupo em adquirir experiência no desenvolvimento de serviços de provisão de contexto, e a especificidade da maioria das arquiteturas propostas, que atendem a objetivos específicos dos projetos nos quais foram desenvolvidas.

Passados quase três anos desde o início do projeto, julgamos ter adquirido uma certa experiência no desenvolvimento deste middleware que valha a pena ser relatada e discutida. Em especial, acreditamos ser interessante compartilhar com o leitor diversas considerações de projeto das diferentes componentes da MoCA que eventualmente poderão ser úteis para futuros desenvolvedores de sistemas desta natureza.

A próxima Seção apresenta uma visão geral da arquitetura MoCA, seus principais serviços e interfaces. As Seções 3, 4, 5, e 6 descrevem em mais detalhes quatro componentes centrais da MoCA, algumas decisões de projeto e outras abordagens. A Seção 7 descreve aplicações e outros middlewares desenvolvidos usando a MoCA. Na Seção 8 apresentamos uma discussão sobre os principais lições aprendidas ao longo do desenvolvimento da MoCA. Por fim, a Seção 9 apresenta a conclusão e trabalhos futuros.

2 Visão Geral

A arquitetura MoCA foi desenvolvida com a finalidade de oferecer suporte ao desenvolvimento e execução de aplicações distribuídas sensíveis ao contexto, particularmente aquelas que envolvem dispositivos móveis (handhelds e notebooks) interconectados através de redes

locais sem fio infra-estruturadas baseadas do padrão IEEE 802.11. Os serviços disponibilizados pela MoCA oferecem meios para coletar, armazenar e processar informações de contexto obtidas dos dispositivos móveis e da rede sem fio. Além disso, é fornecido um conjunto de API's para o desenvolvimento de aplicações, que interagem com esses serviços como consumidores de informações de contexto.

2.1 Arquitetura

Entre os serviços que fazem parte da MoCA, destacam-se o Monitor, o serviço de informação de contexto (*Context Information Service* - CIS), e o serviço de inferência de localização (*Location Inference Service* - LIS). O Monitor é executado em cada dispositivo móvel para coletar dados brutos de contexto, tais como percentual de uso da CPU, memória disponível, nível de energia da bateria, etc.

Essas informações são enviadas periodicamente para o CIS (*Context Information Service*), que armazena e processa os dados recebidos, tornando-os disponíveis para as aplicações interessadas. As aplicações podem consultar estas informações de forma síncrona ou assíncrona. Na forma síncrona, aplicações podem solicitar informações atualizadas sobre o contexto de um determinado dispositivo. Através da forma assíncrona, aplicações podem registrar interesse em notificações sobre a ocorrência de estados de contexto específicos, descritos por expressões lógicas e que envolvem diversas variáveis de contexto relacionadas a um dispositivo.

O LIS é responsável por inferir a localização aproximada de um dispositivo móvel comparando o padrão corrente de sinais de radio-freqüência observados pelo dispositivo (de todos os pontos de acesso 802.11 dentro do raio de cobertura) com o padrão de sinais medidos em pontos de referência pré-definidos. O serviço infere a localização em termos de regiões simbólicas atômicas, ou seja, áreas geográficas com uma semânica bem definida (por exemplo, salas, corredores, halls, regiões de um campus, etc.), que sejam relevantes para as aplicações sensíveis à localização. Uma aplicação cliente pode consultar quais são as regiões simbólicas mapeadas pelo serviço, em que região simbólica se localiza um dado dispositivo ou quais dispositivos se encontram em uma determinada região simbólica. Ela pode ainda receber notificações sobre a mudança de região de um dispositivo específico ou a entrada/saída de um dispositivo em uma dada região simbólica.

Outros serviços complementares da MoCA incluem o SRM (*Symbolic Region Manager*), que permite definir áreas simbólicas compostas (por exemplo, andares, prédios, etc.) a partir das regiões simbólicas atômicas definidas pelo LIS, descrevendo assim uma hierarquia de regiões simbólicas aninhadas; o DS (*Discovery Service*), um serviço de descoberta; e o CS (*Configuration Service*), um serviço de configuração que armazena endereços dos serviços básicos da MoCA, e CoPS (*Context Privacy Service*) [4]. Através deste último serviço (que é opcional), o usuário de uma aplicação sensível ao contexto ou localização pode definir e gerenciar suas políticas de privacidade em relação às suas informações de contexto, ou seja, controlar como, quando e para estas devem ser fornecidas. Antes de atender a qualquer solicitação de informações de contexto, os serviços CIS e LIS consultam o CoPS para decidir se o acesso àquelas informações deve ser concedido ou negado. As principais funcionalidades oferecidas pelo CoPS são controle de acesso baseado em grupos, políticas de acesso pessimista, otimista e interativa, para o controle de acesso, regras de privacidade hierárquica e análise da especificidade de regras.

São disponibilizadas também duas interfaces genéricas para comunicação: o Cm, para

comunicação síncrona de objetos Java usando sockets e TCP ou UDP, e a ECI (*Event based Communication Interface*), para subscrição e notificação baseada em tópicos.

Além disso, visando facilitar o desenvolvimento de aplicações móveis com adaptação de conteúdo sensível ao contexto, a MoCA fornece o ProxyFramework, um framework para a implementação de proxies de aplicação responsáveis por estas adaptações. Neste framework são fornecidos mecanismos para gerência de contexto, para criação de adaptadores e para a configuração da ativação dos mesmos. O ProxyFramework será apresentado em mais detalhes na Seção 6.

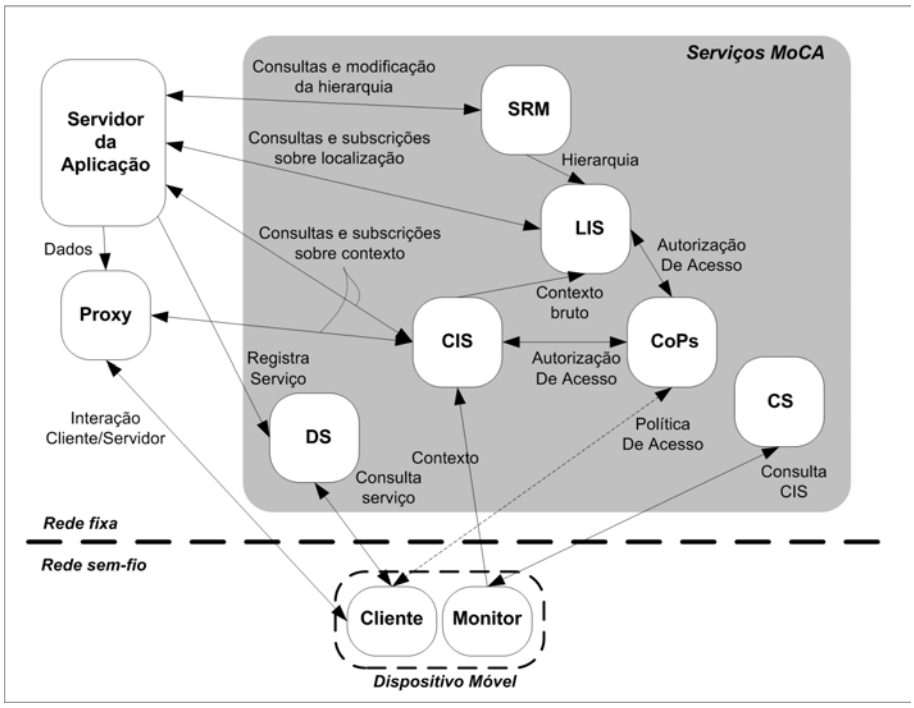

Figura 1: A arquitetura típica de uma aplicação cliente/servidor MoCA

Na sua forma mais geral, uma aplicação sensível ao contexto baseada na MoCA é composta por um servidor da aplicação e um, ou mais, proxies, executado na rede fixa, e clientes da aplicação, executando em dispositivos móveis. O servidor da aplicação e/ou os proxies geralmente são os clientes dos serviços MoCA, ou seja, os consumidores de informações de contexto. Os serviços da MoCA e a arquitetura típica de uma aplicação usuária da MoCA são mostrados na Figura 1.

2.2 Personalidades MoCA

Para permitir que os seriços de contexto da MoCA possam também ser utilizados por aplicações implementadas em outras plataformas de computação distribuída diferentes de Java, são oferecidos também três "Personalidades MoCA", isto é, interfaces para sistemas multi-agentes, Serviços Web (Web Services) e CORBA.

O MoCA/MAX (MoCA/Multi-Agent eXtension) permite que aplicações multi-agentes desenvolvidas com o framework JADE possam obter informações de contexto através de

troca de mensagens ACL [1] assíncronas com agentes de contexto, que por sua vez acessam diretamente os serviços MoCA. Já o MoCA/WS (MoCA para Web Services) oferece interfaces SOAP/HTTP similares às das APIs MoCA para aplicações Java. Como SOAP apenas oferece invocações síncronas, foram oferecidas duas formas de emulação da comunicação assíncrona da MoCA, uma usando temporizadores para a coleta de todos os eventos ocorridos em um intervalo de tempo, e a outra através do uso de *tickets* como descritores de uma assinatura, e que armazenam as notificações MoCA até que sejam consultados. Finalmente, o MoCA/ORB é constituído por proxies independentes para o CIS e o LIS, que permitem a clientes CORBA acessar estes serviços MoCA de forma síncrona e assíncrona. Para tal, criou-se IDLs correspondentes que definem a assinatura das chamadas, os *listeners* e alguns tipos básicos, como `DeviceContext`. A Figura 2 ilustra as formas de acesso à MoCA direto e através das três personalidades.

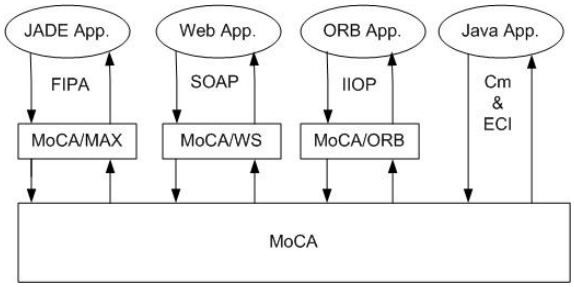

Figura 2: As três diferentes personalidades MoCA

2.3 Discussão

No projeto da MoCA, definimos uma clara separação de tarefas entre os serviços, e os implementamos como processos independentes e comunicantes. Por um lado, essa decisão de projeto favoreceu a flexibilidade no uso dos serviços, pois o desenvolvedor pode selecionar somente aqueles de que realmente necessita, por exemplo, pode-se dispensar o LIS ou o DS. Além disso, essa abordagem permite que novos serviços possam ser incorporados ao middleware de forma modular, o que é especialmente vantajoso em projetos de caráter evolutivo e desenvolvido por várias pessoas. Por outro lado, essa decisão acarretou um forte acoplamento entre os serviços de provisão de contexto, que durante a execução precisam interagir frequentemente uns com os outros para desempenhar suas funções.

3 Monitor XP/CE/Simulador

O Monitor coleta e envia periodicamente para o CIS as seguintes informações que descrevem o contexto do dispositivo:

- Estado dos recursos locais, como taxa de uso da CPU, memória RAM disponível, e nível de bateria.

[1] Agent Communication Language.

- Estado da conectividade sem fio, como intensidade do sinal e endereço MAC de todos os pontos de acesso IEEE 802.11 detectados pelo dispositivo, além de endereços de rede do dispositivo, com qual AP está conectado, etc.

- Identificador do tipo do dispositivo, formado pelo nome do fabricante e o modelo, (e.g. 'HP.hx2400').

O Monitor também oferece uma interface genérica de comunicação local, a partir da qual as aplicações executando no dispositivo cliente podem consultar o contexto computacional e da rede. Para isso, o Monitor atende às requisições enviando como resposta a última informação de contexto coletada na forma de um *stream* de bytes estruturada no formato "chave/valor", e.g. *FreeMemory=5000K, AvailableEnergyLevel=20%*.

Até o momento, temos uma implementação completa do Monitor para Windows XP e Windows CE, e um protótipo para Linux. A maior dificuldade de implementação do(s) Monitor (es) foi o desenvolvimento do módulo responsável por obter as informações da rede sem fio. Para tanto, adaptamos e usamos o *driver* que implementa a arquitetura de comunicação com dispositivos de rede definida pela Microsoft, conhecida por *Network Driver Interface Specification* (NDIS) [5]. Através deste, o Monitor dispara e coleta os resultados dos *scans*[2] realizados na rede sem fio usando interfaces da *NDIS*, que estão padronizados e são usados por *drivers* de rede de todos os fabricantes de placas IEEE 802.11.

Através da NDIS, a implementação do Monitor tornou-se independente do fabricante da placa de rede sem fio, pois toda a comunicação entre o Monitor e o driver da placa de rede ocorre via a *NDIS Wrapper*. As demais informações como CPU e nível de energia, são obtidas utilizando APIs específicas definidas pelo sistema operacional. No Linux, a lista de APs (com as respectivas intensidades do sinal de RF) é obtida através das bibliotecas disponibilizadas pela *Wireless Tools for Linux* em [6] e, as demais informações são coletadas a partir dos arquivos contidos no */proc* na estrutura de diretórios do Linux.

De uma forma geral, o serviço de sensoriamento é um componente chave para o desenvolvimento de qualquer arquitetura de provisão de contexto. No entanto, a implementação deste serviço geralmente demanda um grande esforço de programação por causa da especificidade e complexidade do acesso às informações sobre o estado dos recursos (CPU, bateria, interface de rede) em cada tipo de dispositivo (iPAQs, Palms e Laptops) e sistema operacional. Como exemplo da complexidade envolvida, vale destacar que, às vezes, a implementação da coleta de um contexto específico (e.g., utilização da CPU ou intensidade de sinal dos APs 802.11) para dispositivos de uma mesma plataforma e de um mesmo fabricante pode variar de acordo com a série do produto, ou versão do sistema operacional.

A MoCA oferece ainda o Monitor Simulator, um emulador do Monitor que permite realizar testes com os serviços MoCA mesmo na ausência de um dispositivo móvel com interface IEEE 802.11, além de permitir a simulação de cenários com muitos dispositivos móveis, para testes de escalabilidade.

[2]Varredura na rede sem fio para obter a intensidade do sinal dos APs que estão no raio de cobertura do dispositivo.

4 Serviço de Informação de Contexto

O *Context Information Service* (CIS) é um serviço que processa, armazena e dissemina informações de contexto para aplicações interessadas. Aplicações ou serviços interessados em informações de contexto podem requisitá-las por meio de requisições síncronas ou notificações assíncronas, utilizando o paradigma *publish/subscribe*.Através das notificações, as aplicações podem manter-se a par das mudanças do estado corrente dos recursos do dispositivo móvel (e.g., quantidade de memória livre, nível de energia, etc.) ou da rede sem fio (e.g., variação da qualidade do sinal de rádio, troca do AP corrente) que representem eventos importantes para disparar uma adaptação. Por exemplo, uma aplicação interessada em realizar uma adaptação quando o nível de bateria (`EnergyLevel` de um dispositivo estiver abaixo de 20% e este estiver conectado à rede (`Online`=true), registraria o interesse usando a seguinte expressão, onde `Subject` se refere ao endereço MAC do dispositivo:

```
(Subject="02:DA:20:3D:A1:2B",
 Properties={Device.LocalResouces.EnergyLevel < 20% AND
             Device.Connectivity.Online = True})
```

A aplicação recebe então uma notificação sempre que a condição informada na expressão for satisfeita.

A aplicação também pode ser notificada quando a condição especificada através da expressão voltar a ser falsa, evento este conhecido também como *negação da expressão*. Essa funcionalidade é essencial para as aplicações que implementam algum tipo de adaptação baseada em contexto. Por exemplo, após ser notificada que um determinado estado de alguma variável de contexto do dispositivo está abaixo do limiar aceitável (e.g., nível de energia abaixo de 10%), a aplicação pode desempenhar alguma função, que possivelmente demanda certos recursos para lidar com a situação corrente, como, por exemplo, ativar uma política de *caching*, compactar ou adaptar parte dos dados enviados para o cliente da aplicação no dispositivo em questão. Por isto, a aplicação também deve ser notificada quando o referido estado do nível de energia voltar a ficar acima do limiar aceitável. Essa funcionalidade permite que a aplicação pare de executar a adaptação. desativando, por exemplo, o *caching* ou a compactação.

Uma notificação contém o valor corrente das variáveis de contexto do dispositivo. Atualmente, o CIS provê as informações de contexto indicadas na Tabela 1. As informações estáticas do perfil do dispositivo são obtidas a partir de descrições CC/PP UAProf, mantidas em um repositório de perfis, obtidas a partir do tipo do dispositivo informado pelo monitor.

4.1 Discussão e comparação com outras abordagens

Um dos objetivos do da MoCA foi o de possibilitar o acesso ao contexto (de qualquer dispositivo) também a partir de dispositivos móveis, com poucos recursos computacionais. Isso motivou o projeto do CIS com um modelo de contexto e interfaces com o cliente mais simples e leves. Middlewares como CoBrA [7] permitem inferências complexas por meio de ontologias, mas dificultam o uso efetivo em dispositivos ou aplicações móveis. O desacoplamento necessário em ambientes móveis é conferido por meio de notificações baseadas em eventos, assim como nos middlewares ActiveCampus [8], Steam [9] e YCab [10], com a

Tabela 1: Informações de contexto de dispositivo fornecidas pelo CIS

Categoria	Sub-categoria	Variáveis
Dispositivo	Tipo do dispositivo	Fabricante e modelo do dispositivo
	Recursos locais	Uso da CPU, memória livre, nível de energia
	Meta-atributos	Periodicidade de publicação de dados de contexto, timestamp
	Conectividade	Endereço IP, endereço MAC, taxa de transmissão, em roaming, dispositivo online, conectividade com pontos de acesso, ...
Perfil do Dispositivo	Hardware	Tipo do dispositivo, dimensões da tela, tipo de CPU, número de cores suportadas, ...
	Software	Sistema operacional, Conjunto de caracteres e linguagens permitidos, tipo de conteúdo aceito, ...

diferença que esses dois últimos foram desenvolvidos para uma colaboração em redes ad hoc, enquanto o CIS leva em conta o uso de uma rede infra-estrutura para prover acesso remoto a contexto em serviços centralizados. Por este motivo, CIS foi projetado para atender tipicamente a um domínio administrativo.

Uma das limitações do CIS é a ausência de um repositório para abstrações de adaptações como as *situações* e *preferências* do middleware PACE [11]. Essa limitação nos levou a uma dificuldade em trabalhar com adaptações mais complexas, sem comprometer os requisitos iniciais do serviço.

5 Serviço de Inferência de Localização

O *Location Inference Service* (LIS) é responsável por inferir e disponibilizar a localização simbólica de dispositivos móveis em áreas cobertas por redes sem fio IEEE 802.11. A inferência é realizada através da comparação - cujo resultado é uma estimativa da "distância" - entre o vetor de RSSI corrente do dispositivo (onde cada elemento do vetor representa o sinal de um ponto de acesso 802.11) e os vetores RSSI previamente coletados em "pontos de referência" nas áreas de interesse, e armazenados em uma base de dados do LIS.

O LIS é direcionado para aplicações que necessitam conhecer a posição de um dispositivo em termos de regiões simbólicas (ao invés de coordenadas) e onde tais regiões simbólicas são áreas geográficas internas ou externas, não menores do $2 - 4m^2$, por exemplo, salas, partes de salas maiores, setores de uma rua, etc. A literatura sobre aplicações baseadas em localização revela que a granularidade da localização que o LIS é capaz de prover é suficiente para uma grande classe de aplicações sensíveis à localização.

O LIS foi projetado como um serviço ao qual aplicações, executando em dispositivos móveis ou não, possam realizar consultas ou serem notificadas sobre a localização de qualquer dispositivo móvel sendo monitorado. Aplicações podem interagir com o LIS através de requisições síncronas ou assíncronas. Na forma síncrona aplicações podem, por exemplo, consultar qual é a região simbólica corrente de um dispositivo, ou então, quais são os dispositivos localizados em uma determinada região. Na comunicação assíncrona, uma aplicação, pode registrar o interêsse em receber notificações de eventos relacionados a um dispositivo (e.g. a aplicação é notificada toda vez que o dispositivo muda de região) ou a uma região (e.g. a aplicação é notificada a toda vez que um dispositivo entra ou sai da

região em questão).

Escolhemos como estratégia realizar o processamento da inferência de forma centralizada, em um servidor da rede fixa, ao invés de fazê-lo no cliente móvel. O principal motivo para esta decisão foi a constatação de que a inferência de localização baseada em padrões RSSI requer um processamento intensivo, tornando-se inviável em dispositivos com poucos recursos e reserva finita de energia. Além disso, a inferência de localização unicamente no dispositivo móvel inviabilizaria consultas globais do tipo: *quais são todos os dispositivos presentes em uma área simbólica?*, que são possíveis no LIS.

A arquitetura do LIS foi projetada buscando flexibilidade e eficiência. Com respeito à flexibilidade, o fraco acoplamento entre o LIS e o serviço de contexto da MoCA (CIS) facilita a integração do LIS com qualquer outro serviço de provisão de contexto que disponibilize dados RSSI de todos os APs 802.11 detectados por um dispositivo móvel. A arquitetura e os componentes do LIS foram projetados também para oferecer diversos pontos de flexibilização e customização que, por exemplo, facilitam a implementação de diferentes algoritmos de inferência. Entre outros benefícios, estas características permitiram que nós experimentássemos e avaliássemos mais de um método do inferência. Com respeito à eficiência, o objetivo central foi otimizar o tempo de resposta dos clientes do LIS.

5.1 Discussão e comparação com outras abordagens

Uma experiência obtida com a utilização do LIS foi constatar que sua implantação envolve um certo trabalho, i.e. a necessidade de realizar um mapeamento prévio da região de interesse para possibilitar a inferência (*RF Fingerprinting*), e que isto é um fator que pode dificultar uma adoção mais ampla de um serviço deste tipo. Para minimizar este problema, existem algumas alternativas tais como: (*a*) Incorporar o serviço a sistemas tipo PlaceLab [12], que utilizam apenas a informação sobre o ponto de acesso 802.11 sendo acessado pelo dispositivo para estimar a sua localização, mesmo que isto acarrete uma perda de precisão; (*b*) Automatizar o máximo possível a fase de mapeamento, usando equações analíticas para a geração automática de pontos de referência; ou (*ii*) Realizar um mapeamento incremental onde usuários "confiáveis" seriam autorizados a adicionar à base de dados do serviço novos pontos de referência com suas respectivas amostras de vetores RSSI mensurados.

Outro fator limitante para a implantação de qualquer sistema de localização baseado em sinais de RF é o fato de que cada fabricante de NIC (interface de rede) 802.11 define sua própria escala de intensidades RSSI. Uma solução óbvia é transformar estes valores em uma escala uniforme (e.g. 0-100). Porém, esta não é uma tarefa trivial, já que diferentes chipsets adotam diferentes padrões de variação de intensidade de sinal para criar a sua escala. Por enquanto, não tratamos deste problema, e adotamos a abordagem de definir uma tabela de conversão apenas para os fabricantes/chipsets mais utilizados.

Enquanto outros sistemas similares, tais como RADAR [13] e o Ekahau [14] tratam apenas da inferência da localização propriamente dita, o LIS foi projetado como um serviço de localização completo, ou seja, um serviço que realiza não só a inferência, como também disponibiliza interfaces para vários tipos de consulta e notificações assíncronas (e.g. por dispositivo e por região).

A maioria dos sistemas existentes eram ou comerciais ou então não disponibilizavam o código fonte, dificultando assim as adaptações necessárias, fato que foi a principal razão para implementar o serviço a partir do zero. Apesar do esforço dispendido, isto fez com

que ganhássemos bastante experiência no desenvolvimento e otimização de algoritmos de inferência baseados em RSSI, e pudéssemos projetar um serviço flexível, capaz de ser utilizado com diferentes algoritmos e/ou estratégias de localização.

Estes fatos nos levaram a decidir por uma implementação do serviço de localização a partir do zero. Com essa decisão, também foi obtida uma grande experiência com o desenvolvimento de algoritmos para a inferência de localização e a possibilidade de desenvolver um serviço flexível capaz de ser utilizado com diferentes algoritmos e/ou estratégias de localização.

6 ProxyFramework

A MoCA provê um framework (ProxyFramework) para facilitar o desenvolvimento de proxies capazes de adaptação de conteúdo sensível aos contextos de clientes móveis. Este framework visa aumentar o reuso de código, permitindo sua extensão e personalização para criar instâncias de *proxies de aplicação* de acordo com as necessidades específicas da aplicação. O framework é responsável por identificar o contexto corrente dos dispositivos clientes e acionar as adaptações apropriadas, de acordo com regras refinidas pelo desenvolvedor da aplicação. Entre outros, provê mecanismos para bufferização de de mensagens em momentos de desconexão ou fraca conectividade do cliente, e seleção e invocação de adaptadores sobre tipos de mensagens, tanto para comunicação do tipo request-reply, como para comunicação 1-para-muitos do tipo publish/subscribe.

Para tal, o ProxyFramework inclui funções para se registrar (junto ao CIS ou LIS), como interessado em notificações sobre mudanças de contexto relevantes para a aplicação, de cada cliente da aplicação.

Para instanciar um proxy de aplicação a partir do ProxyFramework, desenvolvedores devem fazer duas coisas: Primeiro, devem implementar os adaptadores, i.e. as ações de adaptação sobre as mensagens, de acordo com as necessidades específicas da aplicação; Segundo, devem definir regras em formato XML contendo: as condições (expressões de contexto) e prioridades para ativação das adaptações (incluindo politicas de bufferização), bem como as cadeias/sequências de adaptadores correspondentes.

As componentes que constituem o proxy são: gerência de contexto, gerência de adaptadores, gerência de políticas de adaptação, gerência de desconexões, camada comunicação, conforme ilustrado pela Figura 3.

A gerência de contexto é responsável por coletar e gerenciar o estado dos clientes. Tais estados são definidos pelos contextos de interesse da aplicação (configurados via XML), e a obtenção destes contextos para cada cliente é realizado registrando-se no CIS da Moca para recebimento de notificações de alteração de contexto de cada cliente.

O módulo de gerência de políticas de adaptação carrega do repositório as regras para adaptação (contextos que disparam as adaptações) definidas via XML. O núcleo deste módulo é a máquina de decisão, que é responsável por verificar o estado (contexto) corrente do cliente e indicar quais os adaptadores devem ser usados (cadeia de adaptadores). A máquina de decisão é acionada sempre que há o envio de uma mensagem para o cliente, e é neste momento que se avalia o contexto do cliente e ativa-se ou não os adaptadores, geralmente de conteúdo.

A gerência de adaptadores é responsável pela execução da cadeia de adaptadores selecionados (pela máquina de decisão) e garantir a seqüência correta da mesma. Caso

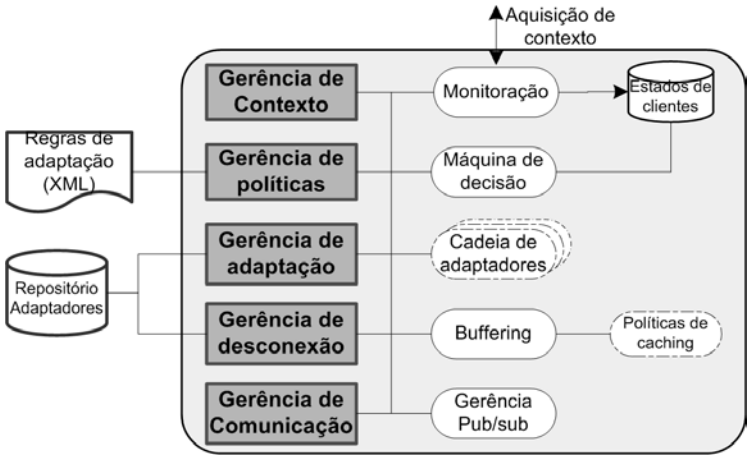

Figura 3: Arquitetura Proxy Framework

o adaptador não esteja disponível, ele faz a carga dinâmica, buscando o adaptador no repositório, e instanciando-o no proxy.

A gerência de desconexão requisita informações de desconexão da gerência do contexto, e pode na ocorrência de desconexões (voluntárias ou não) ativar a bufferização (caching) de mensagens destinadas ao cliente desconectado. A política de caching deve ser implementada pelo desenvolvedor do proxy.

O módulo de comunicação permite a interação com clientes via diferentes protocolos de comunicação. Além disso, provê suporte à comunicação assíncrona (publish/subscribe), permitindo que a adaptação individualiza segundo o contexto de cada cliente.

6.1 Discussão e Comparação com outras abordagens

Várias arquiteturas genéricas de proxies para adaptação foram propostas e implemntadas, tais como [15, 16, 17]. Assim como a maioria destes sistemas, o ProxyFramework permite a criação de adaptações complexas através da combinação de adaptadores em cadeia. Entretanto, apenas o ProxyFramework permite, na especificação das regras de ativação de adaptações (em XML), o uso de expressões booleanas (cf. seção 4), descrevendo um estado de contexto complexo envolvendo diferentes variáveis de contexto dinâmico (memória, energia, localização) e estático (propriedades do dispositivo), e e não apenas de contextos simples, como ocorre nos demais sistemas. Além disso, apenas ProxyFramework fornece um suporte flexível para bufferização de mensagens, que pode ser ativada ou desativada dependendo do estado de qualquer variável de contexto (e.g. qualidade do enlace, nível de energia, desconexão, etc.), e onde a política de gerenciamento do buffer pode ser definida pelo desenvolvedor da aplicação. Por fim, c considerando aspectos de comunicação, o ProxyFramework é o único a oferecer suporte à comunicação assíncrona do tipo publish/subscribe, permitindo adaptações personalizadas.

Comparando esses sistemas, todos permitem adaptação de conteúdo, alguns fornecem suporte a mecanismos de caching, enquanto que apenas Mobiware e TACC provêm suporte a handoff. Entretanto, apenas ProxyFramework provê suporte a desconexão, oferecendo mecanismos de buffering de mensagens sensíveis à conectividade, com políticas programadas pelo desenvolvedor da aplicação. Além disso, considerando aspectos de co-

municação, o ProxyFramework é o único a oferecer suporte à comunicação assíncrona do tipo publish/subscribe, permitindo adaptações personalizadas.

Algumas características do ProxyFramework são encontradas em apenas alguns sistemas. Por exemplo, MARCH [Ardon et al. 2003] e MobiPADS [18] também verificam aspectos do dispositivo (como uso de CPU e memória disponível) além de aspectos do enlace sem fio e de perfis de clientes. MobiPADS também permite uma composição hierárquica de contextos, entretanto apenas de contextos de sensores simples (dispositivo, rede, cliente). Dentre esses sistemas, apenas o ProxyFramework permite a composição de contextos complexos na especificação das políticas de adaptação.

7 Aplicações que utilizam a MoCA

Vários protótipos de aplicações sensíveis a contexto (e de localização) foram desenvolvidos baseados na MoCA. Devido à restrições de espaço, apresentaremos aqui apenas três aplicações. Uma relação completa das aplicações desenvolvidas usando a MoCA pode ser consultada em [19].

Notes in the Air (NITA) [20], é uma aplicação similar ao GeoNotes [21], que permite enviar mensagens de texto (e arquivos em geral) para regiões simbólicas, como se estas fossem quadros de avisos virtuais. Desta forma, clientes que estejam localizados (ou entrem) em uma dessas regiões poderão receber essas mensagens. Além de escolher a região simbólica destino, o remetente de uma mensagem pode definir o nome dos usuários autorizados a recebê-la e quanto tempo ela ficará ativa. Como em salas de bate-papo, usuários podem também indicar se desejam ficar visíveis ou não para outros usuários, escolher os tipos de mensagens que querem receber e se as mensagens devem ser exibidas imediatamente ou armazenadas para leitura posterior. Sendo *NITA* um serviço de recuperação de informação baseado em localização, naturalmente este interage bastante com o SRM e o LIS, tanto para conhecer a estrutura hierárquica das regiões, e para ser notificado das mudanças de localização de qualquer um dos dispositivos sendo monitorados. O fato de que todas as questões relativas à inferência de localização e à notificação de mudanças de contexto (e.g. localização e conectividade) serem tratadas pelos serviços da MoCA (SRM/LIS/CIS) permitiu reduzir significativamente a complexidade do desenvolvimento do NITA, que passou a focar no controle de visibilidade, permissões, grupos de usuários (*buddy list*), armazenamento e apresentação das mensagens.

O *Ubiquitous Guide (uGuide)* é uma aplicação cliente/servidor que permite associar no servidor da aplicação uma URL a cada região simbólica registrada no SRM. O servidor *uGuide*, portanto, atua como cliente do LIS e registra interesse por informações de mudança de região simbólica dos dispositivos executando os clientes `uGuide`. Cada vez que um destes dispositivos entra em uma região simbólica, surge uma pequena aba *popup* acima da barra de tarefas na tela do dispositivo móvel indicando a URL associada à nova região, e permitindo que o usuário abra em a página correspondente a aquela região seu Browser Web preferência.

O *Serviço de Notícias Noites Cariocas (SNNC)* é um serviço push de notícias (tipo RSS), que envia notícias e fotos sobre eventos/shows cariocas. Esta aplicação é estruturada como servidor/proxy/cliente, onde o servidor regularmente publica novos conteúdos (textos/imagens) para todos os clientes móveis cadastrados. Além disso, utiliza o ProxyFramework para adaptar o conteúdo publicado de acordo com as propriedades do dispos-

itivo móvel e o contexto corrente do mesmo. Por exemplo, para handhelds, as imagens são reduzidas para tamanhos compatíveis com o tamanho do seu display (descritos usando perfis UAProf). Além disso, quando o dispositivo apresenta um nível de reserva de energia baixo, ou é detectada sua desconexão, as mensagens RSS passam a ser armazenadas em cache no *proxy SNNC* até que o dispositivo volte a se reconectar. Neste caso, em vez de acessar diretamente o CIS, o desenvolvedor do SNNC precisou apenas definir as regras de adaptação dependentes de contexto no arquivo XML do ProxyFramework.

Além de aplicações sensíveis à localização, a MoCA também está sendo usada como plataforma de provisão de contexto para outros middlewares mais específicos.

ContexTV [22] é uma arquitetura para clientes (receptores) de TV digital interativa. Ela fornece um conjunto de APIs para serviços básicos como interface gráfica e comunicação, um monitor de atividade, e um gerente do ciclo de vida das aplicações de TV digital enviadas aos receptores. Além dos serviços MoCA e da JSR 272, a ContextTV implementa um serviço de preferências, de perfil de usuário e um cache de conteúdo de TV digital.

A MoCA está sendo integrada também ao middleware MAG [23], um ambiente para a execução de agentes móveis em grades. Neste projeto, os serviços de contexto e o ProxyFramework da MoCA são usados para desenvolver um proxy de uma aplicação para análise de imagens de tomografia computadorizada. Este proxy será responsável pela adaptação do conteúdo e formato dos resultados (textos e imagens) retornados pela grade a dispositivos handhelds usados pelos clientes.

8 Lições Aprendidas

Além das diversas decisões de projeto e experiências específicas obtidas, relatadas nas seções anteriores, também aprendemos algumas lições mais gerais durante o desenvolvimento da MoCA, que serão discutidas a seguir.

Bottom-up vs. top-down Por um lado, a opção por desenvolver incicialmente componentes básicas da arquitetura, e resolver problemas concretos relacionados à tecnologia 802.11 e à captura de dados de contexto para diferentes dispositivos e plataformas, certamente teve como benefício não só a aquisição de competência específica e concreta pelo grupo, mas também a motivação em galgar rapidamente um estágio de percepção de funcionamento real e de utilidade do middleware. Por exemplo, não resta dúvida de que o funcionamento correto da inferência de localização usando sinais 802.11, que ainda hoje exerce um certo fascínio, mostrou à equipe da MoCA o amplo leque de aplicações do middleware que estavam desenvolvendo, o que certamente contribuiu para um empenho maior em *fazer tudo funcionar bem*.

Por outro lado, a abordagem *bottom-up* adotada na MoCA, trouxe consigo o problema de que as interfaces das componentes básicas iniciais nem sempre serviriam exatamente às demandas de novos componentes sendo implementados. No entanto, a menos que a funcionalidade de um sistema esteja previamente fixada e esteja precisamente especificada, este mesmo problema pode ocorrer também em projetos tipo *top-down* de sistemas de grande porte. O problema mencionado acima naturalmente pode ser minimizado com um projeto cuidadoso das interfaces inter-componentes. Isto, felizmente, foi o nosso caso, uma

vez que conseguimos estender a MoCA para a atual versão sem modificações substanciais nas interfaces e nas componentes.

Simulação é essencial em computação sensível a contexto Aplicações para redes móveis geralmente demandam a implementação de uma série de tipos e formas de adaptação baseada em contexto, desde mudanças em sua interface gráfica, formato e conteúdo da informação, até ao comportamento da aplicação e dos serviços oferecidos pelo middleware.

A depuração e o teste de aplicações sensíveis ao contexto exigem, portanto, um controle preciso do contexto dinâmico, dos sensores e do ambiente distribuído assumido pela aplicação, o que só pode ser feito por meio de ferramentas de simulação. Neste sentido, o Monitor Simulator trouxe muitos benefícios para o desenvolvimento das aplicações baseadas na MoCA, permitindo a simulação de padrões de mobilidade e conectividade intermitente dos dispositivos. Em particular, a simulação foi muito útil quando: o desenvolvedor não dispunha de um dispositivo móvel real para executar os testes, a mobilidade gerava dificultades para execução dos testes, ou quando o desenvolvedor desejava testar a sua aplicação com vários dispositivos móveis em cenários bem definidos de co-localização e conectividade mútua entre os dispositivos. Uma outra necessidade da simulação é a dificuldade intrínsica de controlar aspectos dinâmicos do ambiente computacional em dispositivos reais, como por exemplo uso da CPU, memória e nível de energia, para realizar testes com aplicações.

Gerenciamento de privacidade é complexo Conforme já discutido, as aplicações sensíveis ao contexto podem ser muito úteis. Entretanto, sem mecanismos de controle de privacidade que permitam aos usuários gerenciar o acesso a suas informações de contexto, essas aplicações podem apresentar riscos à perda de privacidade, que podem ser mais evidentes e impactantes do que os seus benefícios. No entanto, há vários desafios para oferecer controles de privacidade para os usuários dessas aplicações, tais como: atender as demandas das diferentes categorias de usuários (conservadores vs liberais), oferecer flexibilidade na configuração das regras e amenizar a complexidade de gerenciamento da política de privacidade. O gerenciamento de privacidade é complexo porque privacidade é um conceito dependente da cultura e do indivíduo, o que por sua vez impossibilita elencar um conjunto universal de requisitos para todos os usuários. De fato, cada indivíduo é o mais apto a decidir sobre o que pode ou não ser disponibilizado, pois só ele é capaz de avaliar, de acordo com as circunstâncias correntes, o risco *versus* o benefício de divulgar uma informação de contexto.

Para tratar dessas questões, nós incorporamos à MoCA um serviço de privacidade chamado CoPS - **Co**ntext **P**rivacy **S**ervice [4], através do qual os usuários podem definir e gerenciar a sua política de privacidade gradativamente, durante o uso de uma aplicação propriamente dita. No projeto deste serviço adotamos como princípio geral tentar não sobrecarregar os usuários com a configuração das suas preferências de privacidade, e oferecer-lhes uma série de recursos que lhes permitem adotar uma postura conservadora ou liberal em relação a divulgação de suas informações.

Um dos problemas em aberto em relação ao gerenciamento de privacidade é a perda de controle sobre uma informação divulgada. Uma vez que a informação contextual foi divulgada pela rede, o usuário não tem controle sobre para quem ela será repassada, por quanto tempo ela será armazenada em um meio digital, dentre outras questões. Esse

problema, "*imortalidade da informação*", se estende para toda e qualquer informação publicada/divulgada na Internet.

Preocupação com o desenvolvedor da aplicação Desde a concepção e disponibilização da MoCA, houve sempre uma grande preocupação com o usuário (i.e. o desenvolvedor da aplicação), dada a nossa consciência de que o sucesso do middleware dependeria da sua satisfação e percepção do benefício *versus* custo (i.e. curva de aprendizado). Além disso, sabíamos de seu papel fundamental como testadores e indicadores de problemas do software. A nossa experiência mostrou que os três elementos mais relevantes para atrair (e manter) usuários são: (a) manter uma rica documentação na Web com muitos exemplos; (b) disponibilizar APIs simples e intuitivas e (c) prover um suporte rápido e eficiente ao usuário. A conjunção destes três fatores, acreditamos, contribuiu bastante para aumentar a comunidade de usuários da MoCA. Uma outra medida para aumentar a base de usuários da MoCA foi investir no desenvolvimento das Personalidades MoCA, fazendo com que desenvolvedores habituados a outros ambientes de programação pudessem também criar aplicações cientes de contexto e localização.

9 Conclusões e Trabalhos Futuros

Nossos esforços de pesquisa atuais envolvem uma reengenharia da MoCA com o objetivo de promover flexibilidade e ubiquidade no acesso e uso de contexto, levando em conta a heterogeneidade do ambiente computacional. Neste sentido, propusemos uma modificação no modelo de contexto simplificado do MoCA para permitir a modelagem de vários tipos de contexto e o relacionamento entre eles, assim como a descrição de abstrações de contexto necessárias para o seu uso efetivo pelas aplicações, tais como eventos e consultas contextuais. Outro aspecto explorado na atual pesquisa é o desenvolvimento de mecanismos para permitir a evolução dos tipos de contexto em tempo de execução, sem afetar diretamente a execução das aplicações em execução [24]. Por fim, desejamos oferecer acesso ubíquo a informações contextuais e para tal estamos reestruturando o serviço de contexto para permitir gerenciamento distribuído de contexto. Esta ubiquidade exigirá ainda a adoção de mecanismos de descoberta de serviços e contexto mais complexos que os atualmente implementados na arquitetura MoCA.

Agradecimentos

Agradecemos a todos que direta- ou indiretamente, contribuíram para o projeto, implementação ou documentação de alguma componente da MoCA ou aplicação ciente de contexto. Este trabalho foi parcialmente apoiado pelo CNPq, processos 55.2068/02-2 (Projeto ESSMA) e 479824/04-5 (Edital Universal).

Referências

[1] CHEN, G.; KOTZ, D.. **A survey of context-aware mobile computing research**. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.

[2] SACRAMENTO, V.; ENDLER, M.; RUBINSZTEJN, H. K.; LIMA, L. S.; GONCALVES, K.; NASCIMENTO, F. N. ; BUENO, G. A.. **MoCA: A middleware for developing collaborative applications for mobile users**. IEEE Distributed Systems Online, 5(10):2, October 2004.

[3] BALDAUF, M.; DUSTDAR, S. ; ROSENBERG, F.. **A survey on context-aware systems**. International Journal of Ad Hoc and Ubiquitous Computing, 2004. (forthcoming).

[4] SACRAMENTO, V.; ENDLER, M. ; DO NASCIMENTO, F. N.. **A privacy service for context-aware mobile computing**. In: SECURECOMM '05: PROC. OF THE FIRST IEEE/CREATNET INTERNATIONAL CONFERENCE ON SECURITY AND PRIVACY FOR EMERGING AREAS IN COMMUNICATION NETWORKS, p. 182–193. IEEE Computer Society Press, September 2005.

[5] MICROSOFT. **Network driver interface specification (ndis) - msdn library**, 2005. http://msdn.microsoft.com/library/en-us/randz/protocol/ndis.asp (Last visited: July 2005).

[6] TOURRILHES, J.. **Wireless tools for linux**, 1996. http://hplabs.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html (Last visited: April 2006).

[7] CHEN, H.. **An Intelligent Broker Architecture for Pervasive Context-Aware Systems**. PhD thesis, University of Maryland, Baltimore County, December 2004.

[8] GRISWOLD, W.; BOYER, R.; BROWN, S. ; TRUONG, T.. **A Component Architecture for an Extensible, Highly Integrated Context-Aware Computing Infrastructure**. International Conference on Software Engineering, 2003.

[9] MEIER, R.; CAHILL, V.. **Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications**. Proceedings of the 4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'03), LNCS, 2893:285–296.

[10] BUSZKO, D.; LEE, W.-H. D. ; HELAL, A. S.. **Decentralized ad-hoc groupware api and framework for mobile collaboration**. In: GROUP '01: PROCEEDINGS OF THE 2001 INTERNATIONAL ACM SIGGROUP CONFERENCE ON SUPPORTING GROUP WORK, p. 5–14, New York, NY, USA, 2001. ACM Press.

[11] HENRICKSEN, K.; INDULSKA, J.; MCFADDEN, T. ; BALASUBRAMANIAM, S.. **Middleware for distributed context-aware systems**. Lecture Notes in Computer Science, 3760:846–863, 2005.

[12] LAMARCA, A.; CHAWATHE, Y.; CONSOLVO, S.; HIGHTOWER, J.; SMITH, I.; SCOTT, J.; SOHN, T.; HOWARD, J.; HUGHES, J.; POTTER, F.; TABERT, J.; POWLEDGE, P.; BORRIELLO, G. ; SCHILIT, B.. **Place lab: Device positioning using radio beacons in the wild**. In: PROCEEDINGS OF PERVASIVE 2005, THIRD INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING, Munich, Germany, 2005.

[13] BAHL, P.; PADMANABHAN, V. N.. **RADAR: An in-building RF-based user location and tracking system**. In: INFOCOM (2), p. 775–784, 2000.

[14] EKAHAU, I.. **Ekahau home page**. http://www.ekahau.com/ (Last visited January 2005).

[15] MCKINLEY, P. K.; PADMANABHAN, U. I.; ANCHA, N. ; SADJADI, S. M.. **Composable proxy services to support collaboration on the mobile internet**. IEEE TRANSACTIONS ON COMPUTERS, 52(6):713–726, June 2003.

[16] ARDON, S.; GUNNINGBERG, P.; LANDFELDT, B.; Y. ISMAILOV, M. P. ; SENEVIRATNE, A.. **March: a distributed content adaptation architecture**. International Journal of Communication Systems, Special Issue: Wireless Access to the Global Internet: Mobile Radio Networks and Satellite Systems., 16(1), 2003.

[17] CHUANG, S. N.; CHAN, A. T.; CAO, J. ; CHEUNG, R.. **Actively deployable mobile services for adaptive web access**. IEEE Internet Computing, 08(2):26–33, 2004.

[18] CHAN, A. T.; CHUANG, S.-N.. **Mobipads: A reflective middleware for context-aware mobile computing**. IEEE Transactions on Software Engineering, 29(12):1072–1085, 2003.

[19] MOCA. **MoCA Applications Home Page**, last visited: july 2006. http://www.lac.inf.puc-rio.br/moca/applications.html.

[20] GONÇALVES, K.; RUBINSZTEJN, H.; ENDLER, M.; SANTANA, B. ; BARBOSA, S.. **Um aplicativo para comunicação baseada em localização**. In: VI WORKSHOP DE COMUNICAÇÃO SEM FIO E COMPUTAÇÃO MÓVEL (WCSF 2004), p. 224–231, October 2004.

[21] ESPINOZA, F.; PERSSON, P.; SANDIN, A.; NYSTRÖM, H.; CACCIATORE, E. ; BYLUND, M.. **GeoNotes: Social and navigational aspects of location-based information systems**. LNCS, 2201, 2001.

[22] DA C.A. NETO, F.. **Desenvolvimento de aplicações de tv digital interativa sensíveis a contexto auxiliado por uma infra-estrutura de middleware**. M.sc. thesis, Centro de Informática, UFPE (Brazil), August 2006.

[23] SILVA, F. J. D. S.; LOPES, R. F.; SOUSA, B. B.; VIANA, A. E. B. ; SOUSA, S. A.. **MAG, um middleware de grade baseado em agentes: estado atual e perspectivas futuras**. In: In WCGA 2006: Anais do IV Workshop on Computational Grids and Applications. Simpósio Brasileiro de Redes de Computadores (SBRC 2006), Curitiba, Maio 2006. Sociedade Brasileira de Computação.

[24] DA ROCHA, R. C. A.; ENDLER, M.. **Context management in heterogeneous, evolving ubiquitous environments**. IEEE Distributed Systems Online, 7(4), April 2006. art. no. 0604-o4001.