UNIVERSIDADE FEDERAL DE SANTA CATARINA

MIDDLEWARE PARA GERÊNCIA DE INFORMAÇÕES CONTEXTUAIS DE DISPOSITIVOS DE COMPUTAÇÃO MÓVEL

HELÔ PETRY

Florianópolis - SC

2004/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

HELÔ PETRY

MIDDLEWARE PARA GERÊNCIA DE INFORMAÇÕES CONTEXTUAIS DE DISPOSITIVOS DE COMPUTAÇÃO MÓVEL

Trabalho de conclusão de curso apresentado como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.

Florianópolis - SC

2004/2

Helô Petry

Middleware para Gerência de Informações Contextuais de Dispositivos de Computação Móvel

Trabalho de conclusão de curso apresentado como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.

Orientador: Prof. Luís Fernando Friedrich

Banca Examinadora:

Prof. Mário Dantas

Prof. José Mazzucco Júnior

AGRADECIMENTOS

Inicialmente, agradeço a meu orientador, professor Luís Fernando Friedrich pelo auxílio acadêmico e pessoal não somente durante o período de realização deste trabalho, mas também durante quase toda minha graduação na figura de tutor do PET.

À banca avaliadora pelas correções, em especial ao professor Mário Dantas pela ajuda, incentivo e comentários sempre pertinentes.

Agradeço também a minha família linda: minha mãe Hellê e minhas irmãs Késia, Mitzy, Grécia e Mônica pelo apoio, paciência, amor e incentivo constantes. Amo muito vocês.

Aos amigos e colegas do grupo PET Computação da UFSC, do qual fui membro desde a segunda fase da graduação, pela ajuda, incentivo e amizade.

A todos professores do INE e da minha vida inteira, pois seus ensinamentos ajudaram a construir minha formação.

A todos amigos com os quais dividi esse caminho de muito estudo e trabalho, pela companhia, amizade e bom humor. E em especial à Érika e à Criss, pela profunda amizade que cultivamos durante esses quatro anos e que tenho certeza que será para a vida toda. Adoro vocês.

A meu namorado, Ricardo, pela paciência para agüentar meu estresse e mau humor, pelas correções cuidadosas, sugestões valiosas e discussões técnicas madrugada adentro.

E especialmente agradeço a Deus por todas Suas bênçãos que possibilitaram a conclusão dessa etapa da minha vida.

RESUMO

A utilização de informações contextuais de sistemas móveis pode melhorar a eficiência dos dispositivos móveis que as utilizam e prover mais conforto ao usuário. Sensibilidade a contexto pode, portanto, criar novas e inúmeras possibilidades para os dispositivos de computação móvel. Mas para alcançar-se esse objetivo, faz-se necessário criar uma plataforma adequada de software a fim de facilitar o desenvolvimento e interoperabilidade de aplicações sensíveis a contexto. Propondo uma solução, o presente trabalho apresenta um middleware orientado a objetos que gerencia informação de localização (a informação contextual mais marcante em sistemas móveis) de dispositivos móveis. Para utilização da informação de localização foram definidos um modelo de representação desse dado e um modelo de representação de espaço. O middleware apresentado utiliza uma arquitetura de cliente-servidor em um ambiente de computação móvel estruturada ou nômade. Conclui-se que um middleware fornece uma infra-estrutura padronizada para troca de informação contextual e favorece o crescimento e evolução de sistemas sensíveis a contexto, que por sua vez podem ser muito úteis para aumentar a satisfação, o conforto e a produtividade do usuário.

Palavras-chaves: computação sensível a contexto, middleware, computação móvel.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE REDUÇÕES

API: Interface para Programação de Aplicações

CLDC: Configuração de Dispositivos Conectados Limitados

CDC: Configuração de Dispositivos Conectados

GPS: Sistema de Posicionamento Global

IEEE: Instituto dos Engenheiros Elétricos e Eletrônicos

J2EE: Java 2 Enterprise Edition

J2ME: Java 2 Micro Edition

J2SE: Java 2 Standard Edition

JDK: Kit de Desenvolvimento Java

JVM: Máquina Virtual Java

LDIS: Serviço de Informação Dependente de Localização

MIDP: Perfil de Dispositivo de Informação Móvel

OLOS: Sistema de Localização Orientado a Objetos

PC: Computador Pessoal

PDA: Assistente Digital Pessoal

RCS: Sistema de Coordenadas por Referência

RMI: Invocação de Método Remoto

UML: Linguagem de Modelagem Unificada

WLAN: Rede Local sem Fio

XML: Linguagem de Marcação Extensível

1 INTRODUÇÃO

A contínua evolução dos dispositivos de computação móvel, como os Assistentes Digitais Pessoais (Personal Digital Assistant **-** PDAs), vem tornando esses equipamentos cada vez mais poderosos, capazes de executar softwares de alto nível e mais sofisticados, incorporando funcionalidades antes somente suportáveis por computadores convencionais. As tecnologias de transmissão de dados sem fio também têm se desenvolvido consideravelmente, tornando possível que dispositivos móveis se conectem a uma rede sem a necessidade de utilização de cabos, fornecendo mais liberdade a seus usuários. Com a união da comunicação sem fio com dispositivos poderosos surgiu a computação móvel. A computação móvel permite que uma numerosa nova gama de aplicações possa ser desenvolvida e possibilita que os dispositivos móveis forneçam a seus usuários muitas novas possibilidades de experiências. E juntamente, surge o desafio de adaptar as soluções já desenvolvidas para a computação tradicional às dificuldades encontradas pela computação móvel.

Assim, com a evolução dos dispositivos de computação móvel, é possível que a utilização desses equipamentos se tornará mais comum que o uso de PCs num futuro próximo. Mas para que isso realmente ocorra, ainda é necessário criar uma plataforma de software adequada. E para melhorar a experiência dos usuários de dispositivos móveis, é possível adicionar funcionalidades que explorem as características específicas desses equipamentos: estar constantemente em movimento e em poder do usuário. Essas características reforçam a relevância do contexto em que o dispositivo se insere. Contexto, resumidamente, é o ambiente em que o dispositivo encontra-se, com todas as

informações que afetam, de alguma maneira, o comportamento e intenções do usuário e das aplicações suportadas. Portanto, informações contextuais podem ser complexas, mas também muito poderosas e úteis para melhor entendimento das intenções, necessidades e hábitos do usuário. Fica fácil perceber que a utilização de informações de contexto possibilita a geração de inúmeras novas classes de aplicação que, entre outras coisas, pode sintonizar o comportamento do dispositivo móvel com o do seu usuário.

As aplicações sensíveis a contexto se adaptam de acordo com a localização do usuário, pessoas próximas, dispositivos acessíveis, serviços disponíveis, servidores, entre outros, assim como as mudanças desses fatores durante o tempo. Um sistema com essas capacidades pode examinar o ambiente computacional e reagir a mudanças do ambiente.

Embora sensibilidade a contexto seja uma característica bastante importante e útil em sistemas móveis, ainda há falta de infra-estruturas genéricas para o desenvolvimento de aplicações sensíveis a contexto [AS98]. Apesar de ter-se feito progresso na área de frameworks e toolkits para sensibilidade a contexto, ainda há a necessidade de um middleware que suporte abstrações de contexto em alto nível [FC04]. Propondo uma solução possível, o presente trabalho apresenta o desenvolvimento de um middleware que busca e gerencia informações contextuais a fim de personalizar o comportamento do dispositivo móvel. Fazendo isso, esse middleware poderá fornecer serviços a aplicações que utilizem essas informações e facilitar o desenvolvimento de aplicações sensíveis a contexto. Dessa forma, essas aplicações precisam se preocupar apenas com sua

lógica de funcionamento interno e deixar a cargo do middleware a disponibilidade das informações contextuais necessárias.

O objetivo central do middleware desenvolvido é coletar e disponibilizar, de forma homogênea, informações contextuais específicas dos dispositivos móveis clientes. Porém, as informações disponibilizadas não são relativas apenas a cada usuário isoladamente, mas de todos usuários cadastrados. Assim, também é possível obter-se um contexto social: quem está onde, quem está por perto, etc. Além disso, a arquitetura utilizada procura diminuir ao máximo o consumo de energia do dispositivo móvel para melhorar a experiência do usuário, que não gosta de precisar recarregar seu aparelho constantemente.

Neste trabalho é apresentada uma proposta de solução para o gerenciamento de informações contextuais de dispositivos móveis. No capítulo 2 apresenta-se o gênero de dispositivo móvel considerado, algumas tecnologias de comunicação sem fio e duas arquiteturas de redes móveis. No capítulo 3 são definidos informações contextuais e sistemas sensíveis a contexto e discutidos suas finalidades, utilidades e desafios. No capítulo 4 é apresentado com mais profundidade o contexto de localização, informação contextual mais marcante nos sistemas móveis, e discutido sobre a necessidade de existência de um modelo que represente essa informação e suas formas de representação e armazenamento, além de três categorias de modelos de localização existentes e os motivos de escolha de um deles para o trabalho. No capítulo 5 são apresentados dois trabalhos relacionados que mostram propostas de solução para gerência de localização de dispositivos móveis através de middleware. No capítulo 6 é apresentada a solução proposta para o gerenciamento de contexto de

localização em sistemas móveis, a arquitetura de computação móvel, as tecnologias e o domínio de aplicação utilizados, além dos serviços disponibilizados pelo middleware desenvolvido. Finalmente, no capítulo 7 são esplanadas as conclusões sobre computação móvel e sensível a contexto e gerenciamento de contexto de localização. Apresenta-se também os possíveis trabalhos futuros que poderão ser realizados. Nos anexos apresenta-se o código fonte da solução proposta.

2 COMPUTAÇÃO MÓVEL

2.1 Equipamentos Portáteis

Neste trabalho, é assumido que os usuários móveis possuem um dispositivo de computação móvel (um PDA, por exemplo) que possui algumas funções avançadas, como navegadores e a possibilidade de executar aplicativos. Esses equipamentos estão em constante evolução incorporando cada vez mais recursos de processamento, memória e conexões sem fio e apresentam uma grande variedade de acessórios, como GPS, chips de conexão bluetooth, câmeras digitais, modems, interfaces de rede sem fio, entre muitos outros [NA03].

Mas é importante lembrar que apesar de sua evolução, os dispositivos portáteis não possuem tanto poder de processamento e capacidade de armazenamento quanto os computadores de mesa. Outra observação relevante é o fato de que esses equipamentos possuem interface mais limitada em função de suas telas e teclados pequenos e isso deve ser levado em consideração na elaboração das interfaces.

Uma característica comum a todos os aparelhos apresentados é a existência de implementações da máquina virtual Java, o que permite a execução de aplicativos escritos na linguagem Java em qualquer um daqueles. Esta característica determinou a escolha de Java como a linguagem de programação para este trabalho. Portanto, assumimos que o usuário possuirá um dispositivo móvel com máquina virtual Java.

2.2 Tecnologias de Comunicação sem Fio

2.2.1 Bluetooth

Bluetooth é uma especificação para comunicação sem fio e de pequeno tamanho para interligar computadores móveis, telefones celulares, outros dispositivos portáteis e fornecer conexão com a Internet. Uma das características desta tecnologia é o alcance apenas em distâncias curtas, da ordem de alguns metros (entre 10 e 30 metros). A taxa de transmissão do Bluetooth é tipicamente da ordem de 1 Mb/s [NA03].

Bluetooth é apoiado principalmente pelas indústrias de telefones celulares e de microcomputadores. Não surpreendentemente, seu mercado principal é o de transferência de dados e voz entre dispositivos de comunicação e microcomputadores. A comunicação é feita por tecnologia de rádio freqüência. Devido a isso, os dispositivos não precisam estar na mesma linha de visão e podem até conectar-se através de paredes ou outros objetos não-metálicos [SP03].

Uma desvantagem do Bluetooth é a baixa velocidade de transmissão. Ela é insuficiente para transmissão de vídeo digital, por exemplo, mas é perfeitamente adequada para transferência de arquivos e aplicações de impressão [SP03].

Finalmente, o principal ponto forte do Bluetooth é sua habilidade de lidar simultaneamente com transmissões de dados e voz. É capaz de suportar um canal assíncrono de dados e até três canais síncronos de voz, ou um canal suportando dados e voz. Essa capacidade combinada com conexão *ad hoc* a

dispositivos e descoberta automática de serviços faz do Bluetooth uma solução superior para dispositivos móveis e aplicações de Internet [SP03].

2.2.2 IEEE 802.11

O padrão IEEE 802.11 é um tipo de tecnologia por rádio freqüência usada para redes locais sem fio (WLANs) e desenvolvido pelo IEEE (Institute of Electrical and Electronic Engineers) [WF04]. IEEE é uma organização internacional que desenvolve padrões para centenas de tecnologias elétricas e eletrônicas.

A largura de banda das redes IEEE 8002.11 decresce de acordo com a distância da estação base e do ambiente onde a rede está instalada, sendo que a melhor velocidade é de 11 Mb/s. À medida que a distância da estação base aumenta, a taxa de transmissão cai gradativamente, até atingir 1Mb/s no limite. A nova especificação IEEE 802.11a fornece velocidades de até 54 Mb/s, porém com alcance menor, cerca de 50 metros [NA03].

2.3 Modelos de Computação Móvel

2.3.1 Computação Nômade

No modelo de computação nômade quando um computador portátil faz uma conexão sem fio ele o faz com um outro computador denominado *estação base*. A estação base está ligada a uma antena, satélite ou outro receptor/transmissor capaz de se comunicar com os computadores móveis. As estações base são conectadas a uma rede fixa com cabos, onde podem se comunicar com outros computadores/servidores [NA03].

O ambiente de computação nômade difere do ambiente de computação fixa de várias formas importantes. Primeiro, como já citado, o usuário móvel conecta-se à rede via uma conexão sem fio. Então, esses usuários geralmente terão acesso a larguras de banda mais baixas e experimentarão taxas de erros mais altas que os usuários que utilizam conexão via cabos. Segundo, a conexão sem fio pode ser visto como não confiável em termos de disponibilidade. Devido à mobilidade, os usuários podem freqüentemente não estar dentro de uma área coberta pela rede, por vontade própria ou por causa de limitação na cobertura de rede. Terceiro, usuários nômades não estarão mais imóveis, e mudarão seu ponto de acesso à rede repetidas vezes. A mobilidade dos usuários apresenta desafios para o roteamento e oportunidades para novas aplicações sensíveis à localização. Finalmente, dispositivos móveis terão poder de processamento e bateria limitados comparados com computadores de mesa [PS96].

Destacamos que neste trabalho, bem como na maioria dos trabalhos analisados, o modelo de computação móvel utilizado é o da computação nômade.

2.3.2 Redes *Ad Hoc*

Nas redes *ad hoc*, os dispositivos móveis são capazes de se comunicar uns aos outros de maneira direta. Ao contrário da computação nômade, não existem computadores fixos (estações base) para conexão dos equipamentos móveis. Os nós podem se mover arbitrariamente, mudando a topologia da rede freqüentemente. Isso cria grandes desafios para o gerenciamento e transmissão de dados na rede [NA03]. Redes *ad hoc* são principalmente indicadas para situações nas quais não se pode, ou não faz sentido, instalar uma rede fixa.

Se duas estações estão dentro da área de alcance das ondas de rádio, elas têm um canal de comunicação direto entre elas. Quando duas estações que desejam comunicar-se não estiverem na mesma área de alcance, a rota entre os dois computadores pode ser formada por um ou mais dispositivos móveis intermediários na rede. Ao contrário, na rede infra-estruturada, mesmo que dois dispositivos estejam lado a lado, toda comunicação deve passar pela central na rede fixa [MA05].

Entre as vantagens das redes *ad hoc* pode-se citar mobilidade, facilidade de instalação, uma vez que não necessita de estações base, tolerância a falhas, já que a rede pode se reconfigurar automaticamente. Por outro lado, não há nenhuma informação a respeito da localização dos usuários, a taxa de erros é mais alta e a largura de banda global do sistema é menor [NA03].

3 INFORMAÇÕES DE CONTEXTO

Recentemente PDAs e outros dispositivos móveis têm se tornado mais baratos e suficientemente poderosos para executar softwares de alto nível. Claramente a ubiqüidade desses dispositivos aumentará e muitos algoritmos, aplicações e idéias de computação distribuída que anteriormente só eram suportáveis por computadores e redes convencionais serão úteis em redes sem fio de PDAs. Mas embora o progresso de hardware seja significativo e com preços acessíveis, ainda há falta de software que atenda a potencialidade desses dispositivos [HJ03].

Aplicações para dispositivos móveis apresentam problemas desafiadores para os programadores. Esses dispositivos enfrentam perda temporária de conexão de rede quando se movem, descobrem outros hosts de maneira ad-hoc; é provável que tenham recursos escassos, como pouca bateria, menor poder de processamento e memória pequena; precisam reagir a mudanças freqüentes e não anunciadas do ambiente, como alta variabilidade da banda de rede, nova localização, etc. Portanto, sistemas móveis precisam detectar e se adaptar a mudanças drásticas no ambiente. O contexto de um dispositivo pode ajudar muito na realização dessa tarefa [CE01].

O ambiente em que um sistema computacional se localiza pode tornar claro seu significado ou intenções e necessidades do usuário que o sistema computacional está suportando. Esse ambiente em que um sistema computacional se localiza chama-se contexto [LS+02]. Contexto é ainda o conjunto de dados externos que projetistas decidiram que pode influenciar o comportamento de uma aplicação correspondente e que pode ser usado para caracterizar a situação de

uma entidade ([HU02], [LS+02]). Uma entidade é uma pessoa, lugar ou objeto que é considerado relevante para a interação entre o usuário e a aplicação, incluindo os próprios usuário e aplicação em si [LS+02]. São características de um sistema sensível a contexto segundo [MR03]:

- ciência de seu próprio estado e dos sistemas relacionados;
- ciência das intenções, tarefas e sentimentos dos usuários;
- capacidade de adaptar automaticamente seu comportamento em mudanças de contexto.

Sensibilidade a contexto melhora aplicações existentes e até habilita novas classes de aplicações em computação pervasiva. Essas aplicações podem auxiliar o usuário a navegar em territórios desconhecidos, encontrar restaurantes próximos, receber mensagens da forma mais útil e menos intrusiva possível, entre outros. O maior benefício da sensibilidade a contexto é possibilitar que as aplicações se adaptem para melhor prover as necessidades do usuário e suas tarefas, exigindo menos do usuário e aumentando sua satisfação e produtividade [LS+02].

Em sistemas móveis, como em outras áreas de interação homem-computador, não é suficiente focar na interface específica do dispositivo. O dispositivo opera em um contexto mais amplo. Esse contexto inclui a rede e a infra-estrutura computacional, o domínio de aplicação, e o ambiente físico [DR+00].

Considerando-se o projeto de sistemas móveis, deseja-se focar particularmente na situação em que dispositivos móveis agem diferentemente e oferecem diferentes possibilidades de interação dependendo do contexto

particular em que o sistema está sendo usado. Apresentamos a classificação de contexto feita por [DR+00].

3.1 Contexto de Infra-estrutura

A interação oferecida por aplicações móveis não é dependente apenas das características particulares do dispositivo móvel utilizado, mas é um produto do dispositivo e da infra-estrutura de suporte utilizada para entender a aplicação. Em sistemas móveis é provável que a natureza da infra-estrutura mude à medida que a aplicação é usada, assim como o tipo de serviço disponível pode mudar dramaticamente. Essa variabilidade na infra-estrutura pode afetar muito a interação e é essencial que os estilos de interação e as interfaces também reflitam o estado da infra-estrutura. Em essência, as interfaces de usuário para aplicações móveis precisam ser projetadas para lidar com o nível de incerteza que inevitavelmente é introduzido em qualquer sistema que utilize comunicação sem fio.

3.2 Contexto de Sistema

Aplicações móveis mais avançadas são distribuídas por natureza. Em vez de a funcionalidade residir apenas em um único dispositivo, é espalhada pelo sistema como um todo. Isso significa que é necessário considerar as propriedades interativas do sistema em termos da natureza distribuída da aplicação.

Outro aspecto do contexto de sistema é a extensão em que o dispositivo é ciente dos dispositivos em sua vizinhança e, de acordo com isso, a extensão em que uma aplicação é ciente das outras aplicações. Isso é importante, parte porque esses dispositivos podem contrariamente afetar um ao outro enquanto competem

por recursos, mas mais importante, porque combinações de dispositivos podem oferecer serviços mais avançados para o usuário.

3.3 Contexto de Domínio

Assim como a infra-estrutura e o sistema, a semântica do domínio de aplicação precisa ser considerada pelas aplicações móveis distribuídas. A natureza de aplicações multimídia avançadas é tal que seu projeto precisa identificar explicitamente a natureza do trabalho a ser suportado e as viabilidades desse trabalho. Fazendo isso, desenvolvedores precisam considerar a relação entre os dispositivos e seus usuários e como isso pode ser usado para determinar a natureza das interfaces apresentadas. No caso de aplicações móveis, as considerações normais de projeto são amplificadas pela necessidade de considerar as facilidades de interação limitadas dos dispositivos móveis.

Outro aspecto do domínio é o nível de confiança e ciência mútua entre os participantes em interações colaborativas. Isso é particularmente importante se os dispositivos são utilizados para identificar usuários e potencialmente tornar disponível para outros informação sobre suas localizações e o que estão fazendo. Neste caso, algum gerenciamento de privacidade é essencial.

3.4 Contexto Físico

Finalmente, sistemas computacionais móveis são aptos a ter ciência do que os cerca fisicamente. Freqüentemente porque eles são embutidos em um dispositivo específico para uma aplicação, como telefone celular ou carro. Nessas situações, o sistema computacional é móvel por ser parte de um artefato móvel maior. Esse contexto pode e faz efeito na interface de aplicação. Esses sistemas

podem ainda conhecer o contexto ambiental, como a velocidade do carro. Algumas dessas informações sensoriais podem ser usadas simplesmente para enviar uma informação diretamente para o usuário, mas algumas podem ser usadas para modificar o comportamento da interface ou mesmo do dispositivo.

São exemplos de contexto físico: níveis de ruído, condições de tráfego, temperatura e iluminação.

4 CONTEXTO DE LOCALIZAÇÃO

Embora cada um dos tipos de contexto discutidos seja importante e relevante, optou-se por focar predominantemente no contexto físico e, mais especificamente, no uso da localização porque a mudança dinâmica de localização é uma característica única dos sistemas móveis.

Claramente, a idéia de mobilidade exige um entendimento de localização, e um dos aspectos únicos dos dispositivos móveis é que eles podem ter ciência do local em que estão sendo utilizados. Além disso, essa informação de localização pode ser explorada como meio de entendimento do contexto completo em que o sistema está posto. Essencialmente, a localização torna-se um dispositivo útil a partir do qual deduz-se o contexto global que está influenciando a aplicação móvel [DR+00].

Qualquer noção de localização põe o dispositivo dentro de algum tipo de espaço. O espaço em que o dispositivo se encontra pode conter ainda outros dispositivos e usuários com os quais o dispositivo pode interagir. Um dispositivo envolvido em um sistema móvel pode ser considerado como tendo uma localização no espaço, tendo um efeito no espaço e sendo sujeito de influência em eventos do espaço [DR+00].

Usuários de dispositivos de computação móveis podem estar em diferentes locais e podem desejar diferentes comportamentos de seus dispositivos em função da sua localização. É desejável que dispositivos móveis sejam aptos a lidar com desconexão temporária. É possível que o dispositivo móvel e a estação base do usuário interpretem a informação de localização do usuário e talvez antecipem

quando uma desconexão ocorrerá – devido a uma falta de cobertura de rede, por exemplo [HJ03].

É possível fazer experimentos com dispositivos existentes usando aplicações adaptadas e usando os vários meios de comunicação disponíveis para comunicações sem fio. Acredita-se que um middleware de infra-estrutura robusto é a melhor abordagem para habilitar aplicações móveis com informação contextual móvel [HJ03].

Podemos simplesmente imaginar que no futuro todos dispositivos móveis terão capacidade de posicionamento global e poderão saber com muita precisão sua localização. O modelo classicamente utilizado pelos engenheiros de telecomunicações é composto por um conjunto de células hexagonais numeradas. Mas esse modelo não é interessante porque, na prática, o mundo de interesse do usuário móvel é separado por zonas nomeadas pelo usuário e possuem significado para ele. O ideal seria dispor de um modelo em que o usuário pode montar direta ou indiretamente um mapa pessoal das zonas que interessam para ele [HJ03].

As propriedades ou atributos orientados a localização relacionam-se com estar dentro ou fora de uma determinada zona, ou são associados com transições de ou para uma ou mais zonas [HJ03]. De acordo com a combinação dessas propriedades o usuário pode comportar-se de acordo com um perfil e seria ideal que o seu PDA se adaptasse a esse perfil.

Aplicações em computação ubíqua dependem de informação sobre o ambiente. A modelagem do ambiente pode ser feita implicitamente, por exemplo, a aplicação contém a informação do modelo, ou explicitamente através de um

serviço que armazena o modelo de informação e permite que aplicações pesquisem o modelo de dados [BB02]. Nesse trabalho utilizaremos o modelo explícito. O desenvolvimento de um modelo de localização detalhado é uma tarefa custosa com respeito à estrutura inicial e manutenção, então acredita-se que diferentes aplicações deveriam conseguir compartilhar o mesmo modelo de informação [BB02]. Isso aumentaria a interoperabilidade entre aplicações e tornaria possível novas classes de aplicação [BB02].

4.1 Classificação dos Modelos Baseada no Ambiente de Aplicação

[BB02] classifica os modelos de localização de acordo com os ambientes de aplicação. Eles são distintos dependendo do modelo de armazenamento de dados e cooperação e distribuição da aplicação.

4.1.1 Baseado em Infra-estrutura

Armazenando o modelo de dados em um repositório globalmente acessível permite que aplicações distribuídas interajam com seus ambientes de acordo com o modelo e suas localizações. Nesses sistemas, nodos móveis requerem acesso à infra-estrutura para obter o modelo de dados armazenado. Informação de localização dos nodos deve ser informada para a infra-estrutura, assim, os mesmos podem recuperar informações dependentes de localização. Como resultado, nodos móveis devem obter sua posição geográfica via GPS ou de alguma outra forma. Isso requer nodos móveis com certa capacidade de

comunicação, poder de processamento e entradas de sensor para determinação de localização.

Essa classe de aplicação é caracterizada pelo modelo de informação logicamente centralizado. Aplicações distribuídas são baseadas no modelo de informações persistentes. Se os dados são armazenados localmente, o é feito principalmente por razões de performance.

4.1.2 *Ad hoc*

Redes móveis *ad hoc* são constituídas por nodos móveis e não possuem, a princípio, qualquer topologia conhecida, o que resulta em configurações altamente dinâmicas. A interação dos dispositivos que constituem os nodos da rede é dependente do ambiente. Primeiro, comunicação de longo alcance é muito cara e consome muita energia. Contar com dados acessíveis localmente ajuda otimizar o consumo de energia. Segundo, aplicações nesses dispositivos são tipicamente relacionadas com o usuário e suas vontades. Então, a descoberta de serviços nas proximidades do usuário pode guiá-lo através de um ambiente inteligente e prover inúmeras informações.

Armazenamento do modelo de informação sobre o mundo em uma rede móvel *ad hoc* é diferente de uma abordagem baseada em infra-estrutura. Redes móveis *ad hoc* não permitem o acesso a serviços de todas posições na rede devido ao particionamento da rede. Como resultado, aplicações dependentes de dados do modelo devem armazená-los localmente e, por conseguinte, precisam gerenciar as restrições impostas pelos recursos limitados dos dispositivos. Diferentes aplicações em diferentes nodos podem trabalhar com estados potencialmente inconsistentes do mesmo modelo. Portanto, faz-se necessário

haver um controle de consistência do modelo nos dispositivos móveis. Uma abordagem possível é de disseminação da informação baseada em difusão para atualizar as informações de localização.

4.1.3 Isolado

Até agora discutimos aplicações que compartilham o mesmo modelo em um armazenamento de dados centralizado ou distribuído. Um tipo diferente de aplicação não compartilha o modelo, mas depende apenas de seu modelo local. Não ocorrem inconsistências – pelo menos do ponto de vista da aplicação – uma vez que decisões baseadas no modelo são baseadas nos dados armazenados localmente. Exemplos são do domínio de robótica, onde os nodos são principalmente autônomos. O modelo contém dados sobre o ambiente, por exemplo, um mapa de fábrica e transições permitidas dentro do ambiente.

4.1.4 Base Comum

As abordagens mencionadas anteriormente descrevem diferentes modelos de aplicação dependente de dados de modelo. Informação de localização é crucial para a maioria dos cenários de aplicação que computação ubíqua visa. As abordagens infra-estruturada e *ad hoc* irão coexistir em muitos cenários. Uma rede móvel *ad hoc* pode ser utilizada para propagar informação que tem vida curta demais para armazenamento centralizado ou interessante apenas em uma pequena área. A infra-estrutura pode fornecer aos nodos da rede *ad hoc* os dados do modelo em áreas distintas, dessa forma, dar aos nodos móveis dados do modelo precisos para a área à volta deles. Então, é desejável buscar-se uma integração.

4.2 Classificação dos Modelos Baseada na Técnica de Representação

4.2.1 Modelo Geométrico

Uma localização é especificada como uma coordenada n-dimensional (tipicamente n = 2 ou 3) – por exemplo, o par latitude-longitude ou tripla latitude-longitude-altitude retornado por um Sistema de Posicionamento Global (GPS) – ou um conjunto de coordenadas definindo a forma geométrica das fronteiras de uma área [LX+02]. Esse modelo é baseado em um (modelo simples) ou mais (modelo unificado) sistemas de coordenadas (reference coordinate system – RCS) [DO01]. O sistema pode computar formas geométricas regulares a partir de representações concisas (por exemplo, um círculo pode ser representado pelo centro e raio) [LX+02].

A maior vantagem do modelo geométrico é sua compatibilidade entre sistemas heterogêneos, uma vez que os mesmos precisam saber apenas qual o RCS utilizado para trocarem informações [DO01]. Além de possibilitar a obtenção de distâncias de maneira bastante simples (pelo cálculo de distâncias euclidianas, por exemplo). Entretanto, pode ser bastante custoso em termos de volume de dados envolvidos e da necessidade de mapear a representação geométrica para um nível semântico adequado para as aplicações [LX+02].

4.2.2 Modelo Hierárquico

Também chamado de Simbólico ou Descritivo. Modelos de localização hierárquica decompõem o ambiente físico em diferentes níveis de precisão e normalmente apresentam uma representação de localização auto-descritiva

[JS02]. Entidades lógicas do mundo real descrevem o espaço de localização. Entidades podem ser prédios, ruas, cidades, ou elementos definidos pelo sistema como células sem fio, e são identificáveis unicamente por um sistema de nomeação hierárquico. Um exemplo típico é o de endereço postal, como Brasil, Santa Catarina, Florianópolis, Universidade Federal de Santa Catarina, Centro Tecnológico, 2º andar, sala 201. O modelo hierárquico tipicamente tem granularidade de localização menor que o modelo geométrico porque enfatiza a representação dos relacionamentos entre entidades lógicas em vez de coordenadas precisas [LX+02]. Em um nível semântico, o modelo hierárquico é mais adequado para aplicações do tipo Serviço de Informação Dependente de Localização (Location-Dependent Information Services – LDIS). Ainda, por ser discreta e bem estruturada, informação de localização hierárquica é mais fácil de gerenciar. Entretanto, é difícil converter localizações entre sistemas heterogêneos [LX+02].

Os modelos de localização geométrico e hierárquico têm diferentes *overheads* na identificação de localizações e representam informação de localização com diferentes níveis de precisão. O modelo de localização adequado depende da aplicação.

Um LDIS freqüentemente necessita da informação de localização expressa em ambos modelos. Por um lado, o sistema precisa mapear uma localização geométrica para um modelo hierárquico antes de usá-la para buscar objetos expressos no modelo simbólico. Por exemplo, um LDIS mapeia uma localização geométrica obtida por um GPS de um shopping, então o cliente pode procurar todos os restaurantes do shopping. Por outro lado, muitas operações como

encontrar a distância entre dois objetos (por exemplo, entre um shopping e uma estação de trem) deve ser calculada em coordenadas geográficas [LX+02].

4.2.3 Modelo Híbrido

Segundo [JS02], da perspectiva de aplicações sensíveis a contexto, nenhum dos dois modelos é completamente satisfatório sozinho. As duas classes de modelos têm vantagens e desvantagens com respeito às necessidades de aplicações sensíveis a contexto [JS02]. O modelo hierárquico é bom para representações implícitas de relacionamentos espaciais, como *possessão*, *proximidade*, além de ser legível por humanos. A maior desvantagem do modelo hierárquico é a falta de precisão de posição e o fato de ser difícil ou ineficiente calcular distâncias. As falhas do modelo de localização hierárquico expressam os benefícios do modelo de localização geométrico. Com atributos geométricos embutidos, modelos de localização geométricos são bem adaptados para especificar localização precisamente e para calcular distâncias exatas. Entretanto, o modelo de localização geométrico esconde relacionamentos hierárquicos. Então, ele precisa de detalhes adicionais para deduzir relacionamentos espaciais.

[JS02] propõe um modelo de localização híbrido para aplicações sensíveis a contexto que combina os benefícios dos dois lados. O ponto de partida do modelo híbrido é o modelo de localização hierárquico: vê-se o mundo como uma hierarquia de espaços em que cada nível adiante refina e subdivide os espaços do nível anterior. O modelo geométrico entra permitindo que cada espaço na hierarquia defina um sistema de coordenadas que pode ser utilizado para definir pontos ou áreas dentro daquele espaço. Diferentes espaços podem usar diferentes sistemas de coordenadas. As coordenadas permitem que sejam

definidos pontos ou áreas para o qual não há nome no sistema de nomes hierárquico. Por exemplo, podemos identificar a localização de uma câmera em uma sala concatenando o nome hierárquico da sala com as coordenadas da câmera na sala.

5 ESTUDOS DE CASOS

Neste capítulo são apresentados dois trabalhos relacionados encontrados na literatura que propõem middlewares para computação sensível a contexto. Considerou-se importante apresentar com mais detalhes alguns trabalhos interessantes para situar o leitor no estado da arte em sensibilidade a contexto móvel. Na seção 5.1 apresenta-se um middleware para aplicações móveis sensíveis a contexto e que gerencia Preferências de Usuário. Na seção 5.2 apresenta-se um middleware orientado a objetos para aplicações sensíveis a localização, que utiliza a arquitetura cliente-servidor e permite independência da fonte de informação de localização.

5.1 Hawick e James

Foi desenvolvido um middleware para aplicações móveis sensíveis a contexto que utiliza informação de contexto de localização e preferências de usuário. O trabalho completo é descrito em [HJ03].

O aspecto que define usuários de computadores móveis é o de que eles podem estar em diferentes locais e desejar diferentes comportamentos de seus dispositivos móveis dependendo da sua localização.

Uma aplicação chave dessas informações contextuais é um gerenciador de mensagens instantâneas, que em si pode ser parte do middleware, organizando mensagens de eventos entre aplicações ou entre usuários móveis e estações base. O usuário pode ajustar várias políticas sobre quando ele deseja ou não deseja receber mensagens em função de sua localização. Gerenciar informações contextuais não é trivial. Com o crescimento da complexidade das aplicações,

torna-se difícil organizar dados de preferências. Esse problema torna-se ainda mais complexo quando mobilidade/localização e tempo são agregados a preferências estáticas.

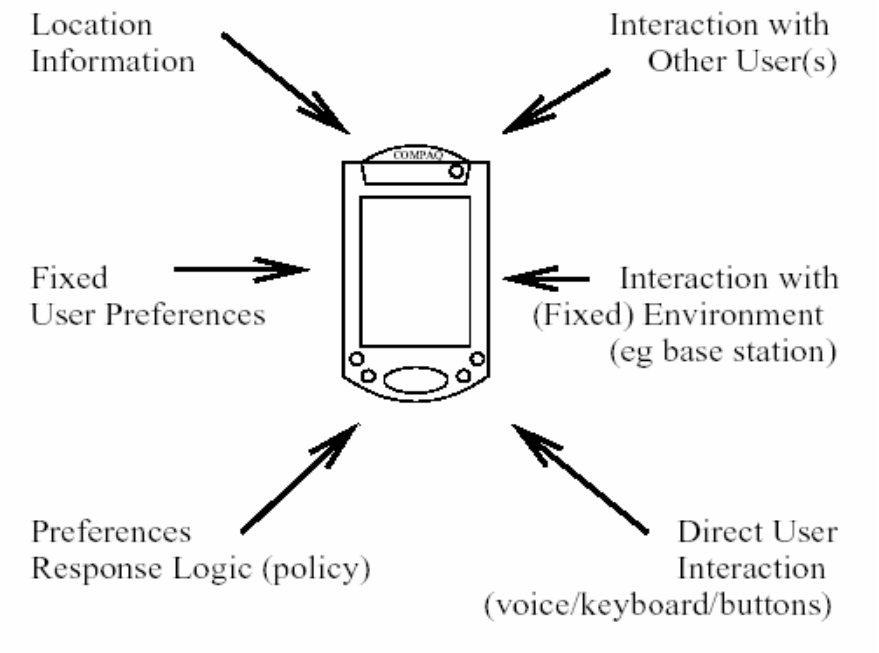

Figura 1: PDA de usuário móvel em contexto [HJ03]

Em máquinas convencionais os usuários conectados à Internet têm sido bombardeados com informações não solicitadas, de Spams a pop-ups, informações essas que não são particularmente relevantes no momento em que são recebidas. Além disso, quando estão trabalhando com dispositivos móveis, que freqüentemente são mais restritos em alguns aspectos que computadores de mesa (como tamanho de tela, largura de banda, tempo de vida da bateria), usuários podem querer restringir as informações que recebem.

O objetivo principal desse sistema é criar um sistema de software que pode ser usado para reduzir a quantidade de informação irrelevante que é vista por um usuário móvel. Primeiramente, define-se Contextualmente Ciente: o sistema é

ciente da localização do usuário, a hora corrente e a informação que o usuário deseja receber.

O sistema é projetado para reagir a eventos assim que eles são fornecidos pelas aplicações. Quando um evento ocorre, como um e-mail sendo recebido, um arquivo sendo criado, ou simplesmente uma mensagem sendo adicionada a uma fila, o Gerenciador de Contexto consulta os outros componentes (módulos preferências ativas, gerenciador de localização, calendário) e decide se o usuário ou sua aplicação devem ser avisados do evento.

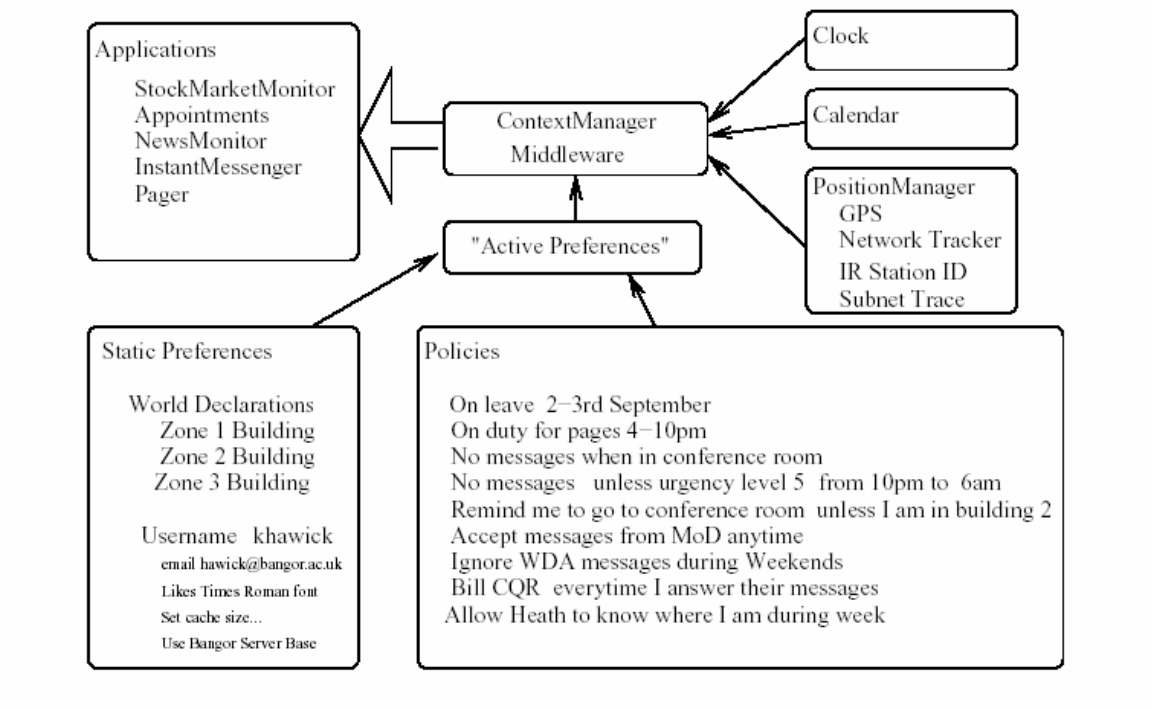

Figura 2 Arquitetura utilizando preferências de usuário [HJ03].

A figura 2 mostra a arquitetura usada para combinar preferências estáticas de usuário com políticas e informação de sistema como posição e tempo para obter preferências ativas que podem alimentar aplicações para determinar curso de ação ou disparar eventos. Middleware pode agrupar essas informações e suportar serviços para aplicações móveis [HJ03].

Preferências ativas são informações personalizadas pelo usuário que são expressas com dependências espacial e temporal [HJ03]. Um exemplo de preferência convencional seria “use essa como minha fonte preferida”. Uma preferência ativa incorpora condições, então um exemplo é “desligue o som quando estou nesses lugares”. Mas não é trivial organizar os metadados necessários para permitir que os usuários entrem com as informações de preferências ativas. Torna-se importante organizar informações de preferências e fornecer ao usuário ferramentas para colocar políticas de preferências ativas e buscar sua própria base de dados de preferências para verificar as conseqüências.

O componente mais crítico do sistema é o módulo de Preferências Ativas. Foram feitas experiências com uma representação baseada em XML de uma linguagem simples e geral acoplada com uma máquina Prolog baseada em Java. Os resultados iniciais foram promissores, mas considera-se ainda a necessidade de nomear usuários individuais, aplicações, componentes de software e zonas.

O sistema mantém as preferências de usuário como uma árvore XML. O uso de tal estrutura de dado também permite a publicação de políticas de aplicação ou de sistema que devem ser incorporadas nas tomadas de decisão. Outros metadados como: quais formatos as aplicações podem entender, são armazenados como informação de preferência de sistema.

Estão sendo desenvolvidas ainda ferramentas que permitem ao usuário manipular suas preferências de maneira geral e para verificar seus conjuntos de regras salvas contra cenários chamados “o que – se” (what - if). Esta ferramenta

permitirá conferir a consistência para garantir que as preferências do usuário não são redundantes ou contraditórias.

5.2 Järvensivu, Pitkänen e Mikkonen

Foi desenvolvido um middleware orientado a objetos especialmente projetado para aplicações móveis sensíveis a localização. O trabalho completo é apresentado em [JP04].

Aplicações baseadas em localização tipicamente consistem de componentes cliente e componentes servidor. Dependendo da tecnologia de posicionamento, dados de localização prontos para usar podem ser produzidos por terminais (como em GPS) ou por componentes de rede (como em posições computacionais em um componente de rede baseado na estrutura celular de uma rede móvel). Além do mais, aplicações podem incluir vários componentes servidor e terminal que colaboram para executar suas funções.

O domínio baseado em localização é heterogêneo, com muitos dispositivos terminais diferentes, sistemas operacionais, plataformas e tecnologias de posicionamento. Como desenvolvedores querem produzir aplicações que rodam em várias combinações de plataforma para habilitar aplicações amplamente adaptáveis ao mercado, APIs e produtos já começaram a emergir para responder à necessidade de uma camada de abstração acima das plataformas específicas de posicionamento. De forma similar, portabilidade de software para clientes e servidores genéricos é intencionada, por exemplo, pelo uso da linguagem de programação. De qualquer maneira, há lugar para mais uma camada de abstração sobre esses níveis primitivos.

Aplicações baseadas em localização são inerentemente sistemas distribuídos. Sistemas distribuídos modernos normalmente são construídos sobre algum tipo de middleware, fazendo a distribuição tão transparente ao programador de aplicação quanto possível. Para atender os requisitos levantados em ambos domínios: servidor e terminal, Järvensivu, Pitkänen e Mikkonen implementaram um protótipo experimental chamado OLOS (Object-Oriented LOcation System – Sistema de Localização Orientado a Objetos).

OLOS é construído com Java e RMI, e integra o conceito de localização com uma visão orientada a objetos do mundo. Localização transforma-se em um atributo de uma classe de objetos. Além disso, OLOS fornece facilidades úteis para aplicações que rodam em um ambiente altamente assimétrico incluindo pequenos terminais equipados com conectividade de banda estreita e servidores de alta capacidade com capacidade de rede de banda larga, o que é indicado como um dos desafios fundamentais da computação móvel. OLOS integra-se com diferentes plataformas de posicionamento e APIs pelo fornecimento de uma interface de escrita de adaptadores de sistema de posicionamento.

OLOS distingue entre dois tipos de objetos cientes de localização. Objetos do primeiro tipo são ligados a entidades móveis como terminais de usuário. A localização de um objeto desse tipo muda de acordo com os movimentos da entidade física. Um perfil de serviço sensível a localização é um exemplo desse tipo de objeto. Objetos cientes de localização do segundo tipo podem posicionar-se e mover-se livremente no espaço geográfico, e a localização de um objeto desse tipo é puramente uma noção virtual. Um sinal virtual colocado perto de um local de interesse é um exemplo desse tipo de objeto. Além disso, OLOS fornece

um mecanismo para clientes tanto em terminal móvel quanto em servidor para criar, descobrir e usar objetos cientes de localização. Sem considerar onde um objeto é criado, seu código atual é executado em um servidor registrado para receber objetos da classe em questão.

5.2.1 Arquitetura do OLOS

No modelo de distribuição OLOS, as solicitações de processamento no lado do cliente, por exemplo terminais móveis, são reduzidas para beirar o mínimo. Em essência, apenas uma pequena implementação de interface local roda no dispositivo portátil para encapsular chamadas a métodos remotos para serviços remotos. A abordagem típica de *factory* (fábrica) é utilizada para criar objetos remotos.

Em uma configuração típica, as fábricas de objetos OLOS e implementações do lado do servidor rodam em um servidor de aplicação, com o qual os terminais podem se comunicar via interface RMI Java. Alem desses, vários outros componentes são necessários. Um componente chave na arquitetura OLOS é *OLOS Registry* (*Registro OLOS*), que associa objetos a dados de localização e age como uma interface para procurar objetos e fábricas. Registro OLOS é utilizado transparentemente pelo programador de aplicação através de classes base e classes locais e fábricas geradas automaticamente.

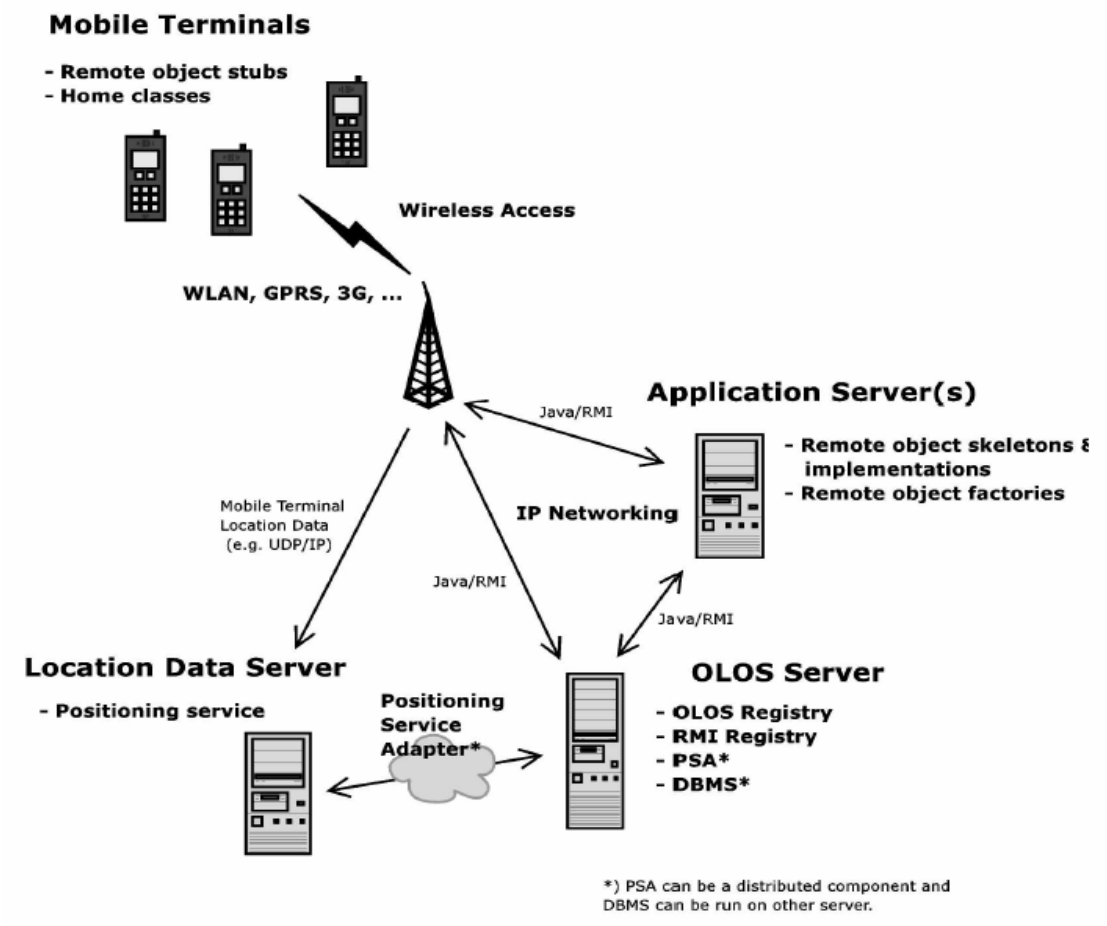

Figura 3: Típico ambiente OLOS [JP04]

Registro OLOS deve ser conectado a um serviço de posicionamento. Um serviço de posicionamento pode ser qualquer sistema adequado que possa ser usado para obter dados de localização de um terminal móvel, por exemplo, software de posicionamento em uma WLAN, uma interface GPS ou um serviço de posicionamento de telefone móvel 3G. A comunicação com o serviço de posicionamento é encapsulado em um componente denominado Adaptador de Serviço de Posicionamento (Positioning Service Adapter - PSA). O serviço de posicionamento tipicamente é um sistema distribuído em si, e dependendo do tipo de interface que o mesmo fornece, pelo menos partes do PSA correspondente são freqüentemente executadas na mesma máquina que o Registro OLOS.

A versão atual do Registro OLOS depende de um banco de dados objeto-relacional. Um sistema de gerenciamento de banco de dados é utilizado via JDBC e pode rodar em qualquer servidor inclusive na mesma máquina física que o Registro OLOS.

O Registro OLOS precisa rodar em um servidor acessível pelos outros componentes. Java/RMI oferece um serviço simples de nomeação chamado Registro RMI. Esse serviço de nomeação tem uma limitação importante: obriga que as implementações dos objetos remotos do servidor tenham que rodar na mesma máquina física que o servidor de nomeação. Essa limitação foi implementada principalmente por razões de segurança nos primeiros JDKs e ainda existe. Entretanto, Registro RMI é utilizado nessa versão inicial de OLOS (um serviço mais avançado de nomeação pode ser adotado mais tarde). Na arquitetura OLOS, o propósito do serviço de nomeação é fazer com que o Registro OLOS seja facilmente acessível para os clientes. Programas no lado do cliente e fábricas de objetos são fornecidos com um descritor de finalidade que é usado para acessar o registro transparentemente para os programadores de aplicação. Descritores de finalidade no OLOS são arquivos de texto legíveis por humanos processados e gerados com propriedades de objetos Java.

Objetos OLOS são objetos remotos cientes de localização de dois subtipos exclusivos: *físico* e *virtual*. Objetos físicos OLOS encapsulam uma política segundo a qual a localização de um objeto coincide com a localização do dispositivo móvel que criou o objeto. Instâncias de objeto virtual OLOS representam entidades cujas localizações podem ser ajustadas programaticamente, ou seja, não possuem contraparte no mundo físico.

Do ponto de vista do programador de aplicação, OLOS é uma extensão relativamente pequena de Java/RMI para implementar objetos sensíveis a localização. Objetos OLOS são objetos Java/RMI distribuídos com funcionalidade adicional para adquirir sensibilidade a localização. Além disso, uma classe local é gerada para cada classe objeto OLOS para prover serviços básicos de ciclo de vida de objeto e buscas baseadas em localização.

Para criar uma nova aplicação baseada em OLOS, o programador de aplicação deriva classes sensíveis a localização necessárias a partir de classes abstratas base fornecidas. Isso é feito primeiro derivando uma interface remota ou de PhysicalOLOSObject ou de VirtualOLOSObject, de acordo com o tipo de objeto necessário. A declaração da interface criada é compilada usando o Compilador de Interface OLOS, uma ferramenta cômoda para geração de código, que gera classes internas e fábricas e uma fábrica de descritor de finalidade. Após a compilação bem sucedida da interface, uma implementação remota da interface precisa ser fornecida derivando uma classe base abstrata: PhysicalOLOSRemoteObject ou VirtualOLOSRemoteObject.

A obtenção de um objeto OLOS é feita via uma classe local particular pelo pedido de criação de um novo objeto ou via busca de um objeto adequado. Buscas sempre se aplicam a *percepções* recentes, não desatualizadas de objetos sensíveis a contexto a não ser que outro intervalo de tempo seja especificado. Uma percepção consiste de coordenadas e um selo de tempo. Possíveis critérios de busca incluem coordenadas com raio de procura assim como nomes de locais fixos. Métodos de busca retornam simplesmente um *iterator* remoto, que pode ser buscado para os próximos *n* objetos resultantes.

Uma das características chave do OLOS é a independência do serviço de posicionamento. Adaptadores de Serviço de Posicionamento (PSA) têm um papel chave para alcançar isso.

Um programador de aplicação pode implementar um PSA adequado para um serviço de posicionamento desenvolvido fornecendo uma classe que implementa a interface LocationServiceController. Registro OLOS instancia a classe implementação usando o carregador de classe padrão. Middleware OLOS fornece uma implementação da interface OLOSLocationListener e registra a mesma (possivelmente mais que uma instância) para o PSA. Todas as percepções de dispositivos registrados rastreados são relatados ao middleware através dessa interface.

6 AMBIENTE GERENTE DE CONTEXTO MÓVEL PROPOSTO

Neste trabalho é apresentado um middleware que gerencia informações de contexto em sistemas móveis com o objetivo de fornecer uma camada de abstração aos desenvolvedores de aplicações sensíveis a contexto para dispositivos móveis.

6.1 Decisões de Projeto

O middleware desenvolvido colhe determinadas informações contextuais a fim de fornecer informações personalizadas às aplicações móveis sensíveis ao contexto (desenvolvidas para o dispositivo móvel), permitindo a estas aplicações adequar-se às necessidades do usuário.

Definiu-se que as informações contextuais utilizadas são localização do usuário e data/hora. Com a combinação dessas fontes pode-se fornecer informações bastante ricas e úteis para melhorar a experiência e satisfação do usuário com seu dispositivo móvel.

A fim de melhorar a eficiência do middleware e seu melhor ajuste às aplicações que o utilizarem, foi definido que o domínio de aplicação abordado é o de ambientes internos (*indoor*). Dessa maneira pode-se assumir algumas premissas no desenvolvimento do middleware, como por exemplo, de que GPS não será uma fonte válida de localização, uma vez que GPS não funciona bem em ambientes *indoor* porque a força do sinal é muito baixa para penetrar na maioria dos prédios. Além disso, reflexões podem fazer com que a leitura não seja confiável.

Verificou-se na literatura ([CK00]) que alguns projetos importantes de pesquisa em ambientes *indoor* (Olivetti Active Badge System [WH+92], Xerox ParcTab [WS+93], projeto Cyberguide [AA+97], Personal Shopping Assistant [AC94], entre outros) constroem seus sistemas próprios de obtenção de localização baseados em infra-vermelho ou rádio freqüência, ou ambos. Mas optou-se por não limitar o tipo de fonte de localização utilizada a fim de não restringir a classe de dispositivo cliente e, assim, fornecer total portabilidade ao middleware.

Apesar da contínua evolução dos dispositivos de computação móvel, esses aparelhos apresentam uma característica invariável: são alimentados por bateria com tempo de vida limitado. E por melhor que seja o equipamento, o usuário não ficará satisfeito se precisar recarregar o seu dispositivo móvel constantemente. Portanto, os softwares desenvolvidos para esses aparelhos precisam se preocupar com economia de processamento do dispositivo móvel para ampliar o tempo de vida da sua bateria. A abordagem que utilizaremos para contornar essa dificuldade é a de distribuir o middleware entre estruturas cliente e servidor. No dispositivo móvel cliente permanece apenas uma pequena interface que encapsula chamadas a métodos remotos para serviços remotos. O servidor mantém os objetos remotos da aplicação que efetivamente processam toda a lógica interna de funcionamento. Dessa forma, o processamento é transferido do aparelho móvel para o servidor. Java/RMI possibilita a implementação desta solução.

A obtenção da informação de localização é feita pelo Gerenciador de Localização, um componente que encapsula os detalhes da interação com as

fontes de localização para coletar e distribuir a informação de onde o usuário se encontra. A fim de não restringir o middleware a qualquer fonte de localização predefinida, esse módulo disponibiliza uma interface para adaptador de serviço de localização, que deve ser implementado para cada tipo de serviço disponível.

No lado do cliente, o middleware possibilita a realização de algumas tarefas: registrar e descadastrar o cliente no sistema, informar a localização do cliente, solicitar a localização de um usuário específico, solicitar ser informado dos usuários próximos e fabricar objetos remotos. Em virtude da distribuição da carga de processamento para o servidor, permanecem no cliente apenas as referências aos objetos remotos localizados no servidor. Assim, a chamada de métodos é feita no cliente, mas processada no servidor.

No servidor é mantido um serviço de registro dos objetos localizáveis (usuários móveis e serviços disponíveis) a partir de uma informação de identificação fornecida pelo mesmo. Depois do registro, o middleware conhece os objetos e suas localizações. O registro pode ainda manter alguma informação semântica sobre o objeto, como nome do usuário ou serviço. O middleware pode manter registro de Pontos de Referência como serviços e locais específicos.

O servidor também possui fábricas de objetos que são utilizadas pelas aplicações do cliente para instanciar suas classes. Uma fábrica cria objetos locais e retorna a referência remota para o cliente.

O servidor é mantido atualizado da informação de localização dos objetos localizáveis fornecida pelos gerenciadores de posição pelo recebimento de evento de atualização da localização. Ainda atualiza o histórico das últimas 10 localizações de cada objeto. A partir desse histórico, pode-se incluir

funcionalidades que verificam tendências de comportamento como localizações prováveis em função da hora ou do dia da semana.

As informações dinâmicas que o middleware disponibiliza para a camada de aplicação foram definidas como as seguintes:

- localização física do usuário
- hora atual
- últimas 10 localizações dos usuários
- serviços disponíveis próximos
- usuários próximos
- localização de um usuário específico (um cliente pode buscar a localização de outro)

Definimos ainda que o middleware será orientado a eventos: reage a eventos e gera eventos para a camada de aplicação. Por exemplo, no momento da atualização da informação de localização, o dispositivo de localização gera um evento no middleware, que por sua vez pode gerar outros eventos dependendo do ocorrido. O cliente pode também fazer requisições que gerarão eventos internos.

Utiliza-se o modelo de localização híbrido, que inclui informação de localização tanto simbólica quanto em coordenadas, pois o mesmo cobre as deficiências de ambos modelos geométrico e hierárquico e é apropriado para representar ambientes *indoor*. Localizações *indoor* consistem, por exemplo, de prédios, andares e salas. O middleware vê localização de maneira hierárquica, como em estrutura de árvore em que cada nodo possui um dado hierárquico e um dado em coordenadas. Essa estrutura é uma fonte de dados de possíveis localizações e considerada disponível ao middleware e seus clientes. A fim de

obter-se a informação de localização no modelo híbrido a partir apenas do dado geométrico, mantém-se um registro com a relação das fontes de localização geométrica (sensores, pontos de acesso Wi-Fi, etc) e suas respectivas localizações no modelo hierárquico.

Escolhemos implementar o middleware em Java por vários motivos: ser suportada em praticamente todos dispositivos móveis, fornecer portabilidade, uma vez que a linguagem é compilada em bytecode para ser executado em qualquer máquina virtual Java, além de facilitar o uso de chamadas a procedimentos remotos através de RMI.

Em virtude da indisponibilidade de dispositivos reais, utilizaremos softwares de simulação para simular a rede sem fio e o próprio dispositivo móvel.

6.2 J2ME

J2ME é dirigido para dispositivos com potência limitada. A utilização de J2ME possibilita que esses dispositivos adquiram novos aplicativos além dos configurados durante o processo de produção [MU01].

J2ME é uma plataforma para dispositivos pequenos cuja intenção é, eventualmente, substituir os vários produtos baseados em JDK com uma solução mais unificada baseada em Java 2 [TO02]. Diferente dos mundos de desktops e servidores mirados por J2SE e J2EE, o mundo de microdispositivos inclui uma gama de dispositivos com capacidades tão diferentes que não é possível criar um único produto de software para satisfazer todos eles. Em lugar de ser uma única entidade, então, J2ME é uma coleção de especificações que define um conjunto de plataformas, cada uma adaptada para um subconjunto da coleção total de dispositivos de consumidor que cai dentro do seu escopo [TO02]. O subconjunto

do ambiente de programação Java completo para um dispositivo particular é definido por um ou mais *Perfis*, que estendem as capacidades básicas de uma *Configuração*. A Configuração e Perfil(is) que são apropriados para um dispositivo dependem da natureza do seu hardware e do mercado alvo [TO02].

6.2.1 Configurações

Uma Configuração é uma especificação que define uma plataforma Java para uma larga gama de dispositivos [TO02]. Uma Configuração é fortemente amarrada a uma Máquina Virtual Java (JVM). Na verdade, uma Configuração define as características da linguagem Java e as bibliotecas Java centrais da JVM para aquela Configuração particular [MU01].

O conjunto de características desses dispositivos alvo geralmente considerado é do tipo [TO02]:

- tipo e quantidade de memória disponível
- tipo e velocidade do processador
- tipo de conexão de rede disponível para o dispositivo

A seguir as características típicas dos dispositivos dentro das duas Configurações definidas atualmente [MU01].

Connected Device Configuration (CDC)

- 512 kilobytes (mínimo) de memória para rodar Java
- 256 kilobytes (mínimo) de memória de alocação em tempo de execução
- conectividade de rede, possivelmente persistência e largura de banda alta.

Connected, Limited Device Configuration (CLDC)

- 128 kilobytes de memória para rodar Java
- 32 kilobytes de memória de alocação em tempo de execução

- interface de usuário restrita
- pouca energia, tipicamente suportado por bateria

6.2.2 Perfis

Um Perfil é uma extensão de uma Configuração. Fornece as bibliotecas para um desenvolvedor escrever aplicações para um tipo particular de dispositivo [MU01]. Por exemplo, o Mobile Information Device Profile (MIDP) define APIs para componentes de interface de usuário, entrada e manipulação de eventos, armazenamento persistente, rede e timers, levando em consideração as limitações de memória e tela de dispositivos móveis [MU01].

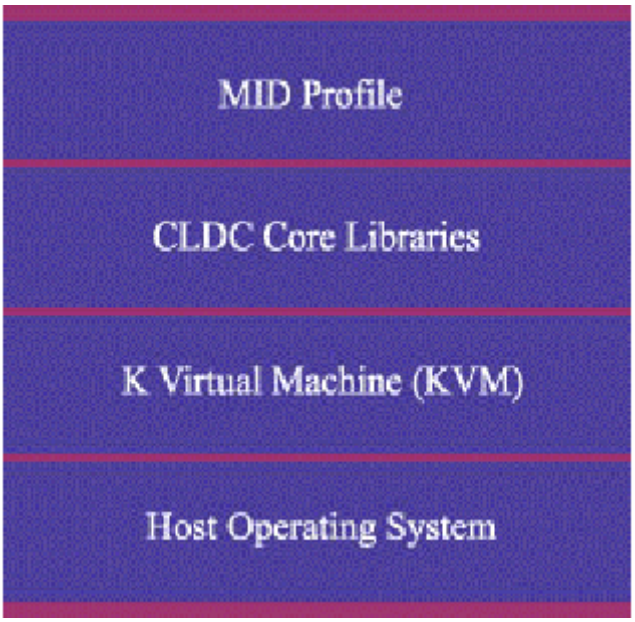

Figura 4: Arquitetura do Perfil MID [MU01]

6.2.3 J2ME Wireless Toolkit

J2ME Wireless Toolkit é um software ferramenta livre fornecido pela Sun para simplificar o ciclo de desenvolvimento J2ME. Essa ferramenta oferece uma

GUI básica para gerência de projeto, pré-visualização e empacotamento de programas J2ME [MU01].

6.3 Modelagem

A seguir apresentamos alguns diagramas em UML para ilustrar arquitetura e parte do funcionamento internos do middleware.

A figura 5 ilustra a distribuição das funcionalidades entre cliente e servidor. Cabe ao cliente obter a informação de localização e cabe ao servidor registrar os usuários remotos e suas localizações.

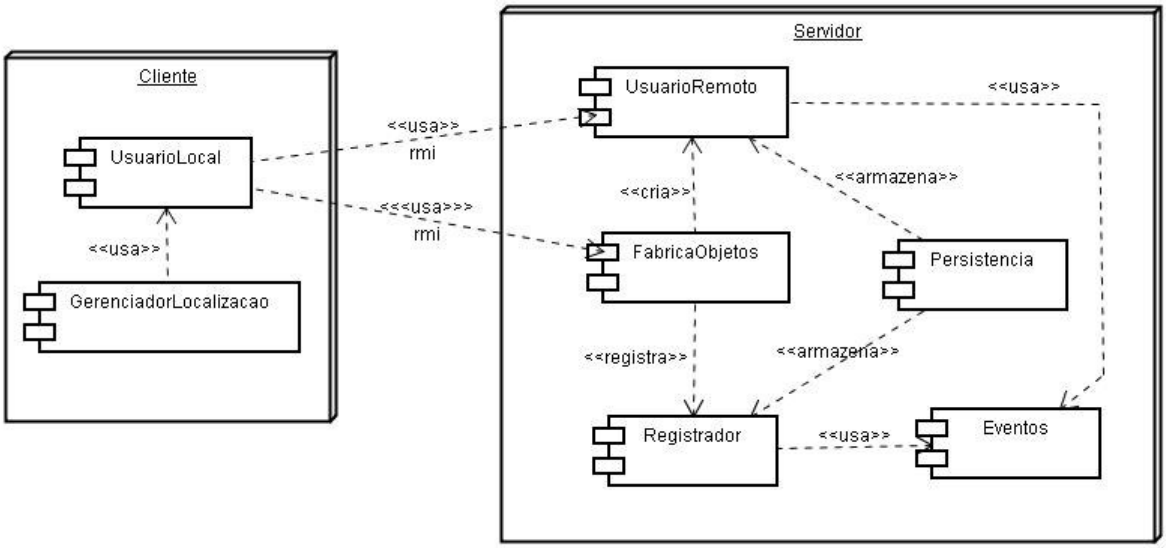

Figura 5: Diagrama de Instalação

A figura 6 mostra o diagrama de classes completo do middleware.

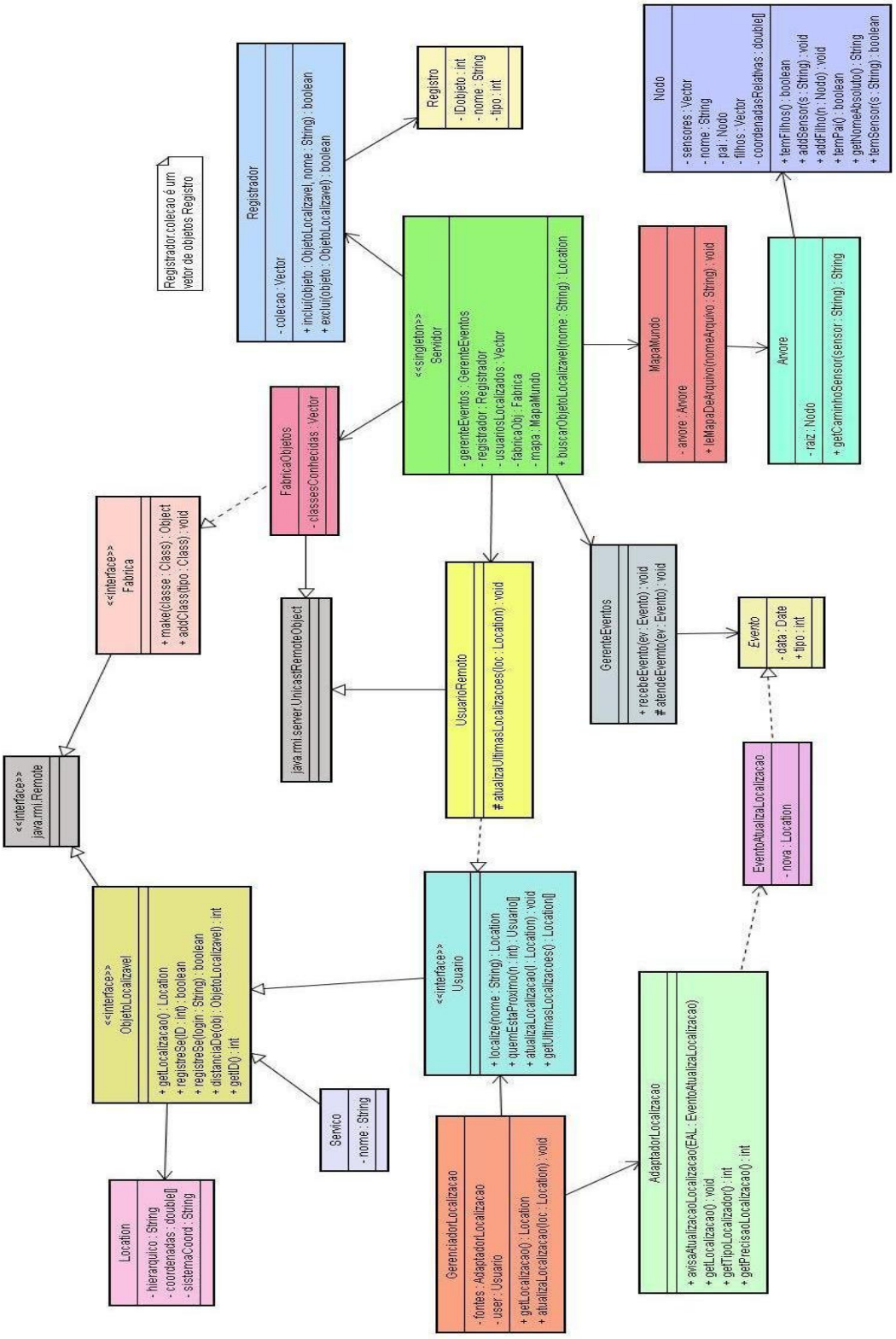

Figura 6: Diagrama de Classes Completo

A figura 7 mostra em detalhes as classes que ficam hospedadas no cliente. O AdaptadorLocalizacao encapsula os detalhes de interação com a fonte de informação de localização e é responsável por verificar a mudança na localização do usuário e gerar um evento para o middleware. Usuario é uma interface de objeto remoto RMI e uma especialização de ObjetoLocalizavel e possui vários métodos para obtenção da localização própria e de outros usuários. Location representa a informação de localização no modelo híbrido: o atributo hierarquico é um string que contém o endereço hierárquico (o nome de cada nível é separado por "\\"); o atributo coordenadas é um array de coordenadas. GerenciadorLocalizacao obtém o objeto Usuario via RMI e repassa a ele sua localização cada vez que o AdaptadorLocalizacao atualiza essa informação.

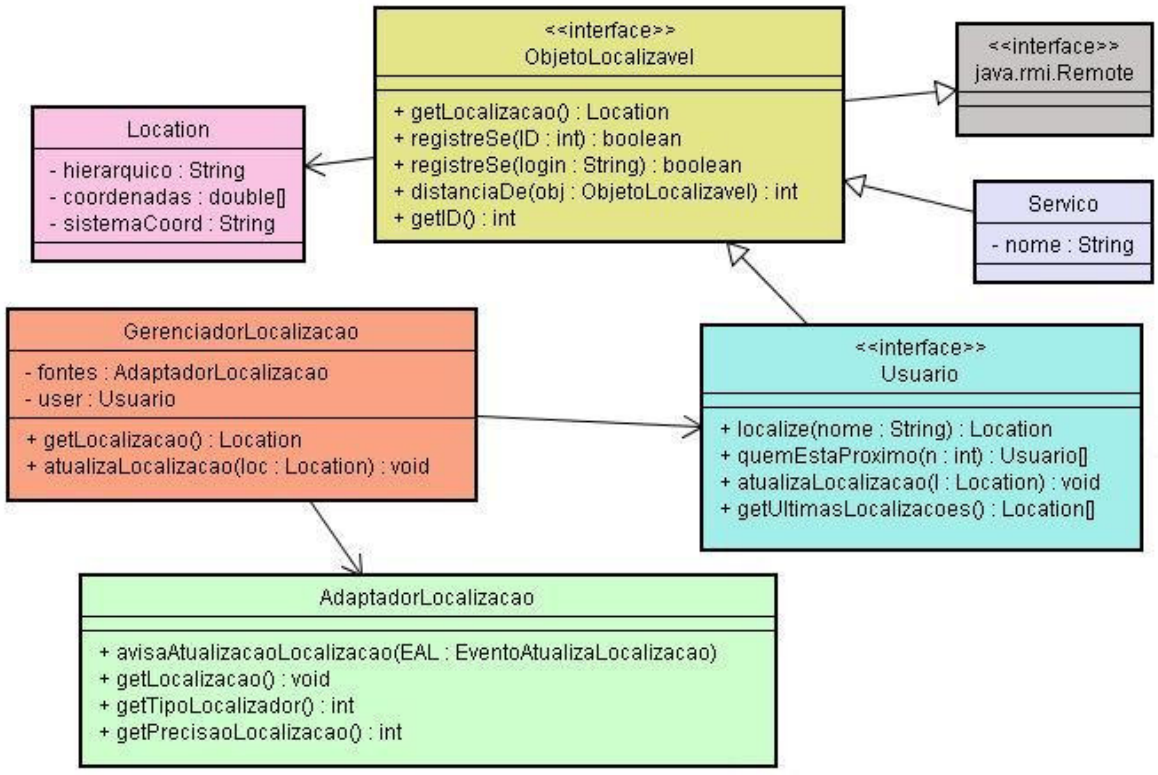

Figura 7: Classes hospedadas no cliente

A figura 8 ilustra em mais detalhe as principais classes do servidor. A classe UsuarioRemoto é a implementação da interface Usuario presente no cliente e suas instâncias são distribuídas via RMI pelo servidor. A interface Fabrica pode ser utilizada por aplicações cliente para enviar suas classes para serem instanciadas pelo objeto remoto FabricaObjetos, localizado no servidor. Uma vez instanciados, os objetos são disponibilizados às suas aplicações via RMI. Portanto, as aplicações clientes podem cadastrar apenas classes que estendam a classe java.rmi.server.UnicastRemoteObject, uma vez que seus objetos serão obtidos através de um serviço RMI.

O objeto Servidor é o coração do middleware. Nele ficam registrados os usuários cadastrados e suas respectivas localizações, além de disponibilizar as instâncias dessa classe (UsuarioRemoto) via RMI. O Servidor mantém também um Registrador que cadastra os usuários e lhes fornece um identificador único. Depois do registro, o Servidor conhece a localização do usuário e pode fornecer essa informação se for solicitado. O servidor possui ainda um GerenteEventos que o avisa quando determinado evento ocorre (inicialmente apenas mudança na informação de localização de algum usuário cadastrado). O servidor também processa os pedidos de informação de contexto social, aquelas que dizem respeito a um grupo de usuários, uma vez que um usuário isoladamente não teria condições de obter esse tipo de dado. Esses pedidos são do gênero: “que usuários estão próximos de mim?” ou “onde o usuário João está?”. O servidor possui ainda um Mapa de Mundo, explicado com mais detalhes a seguir.

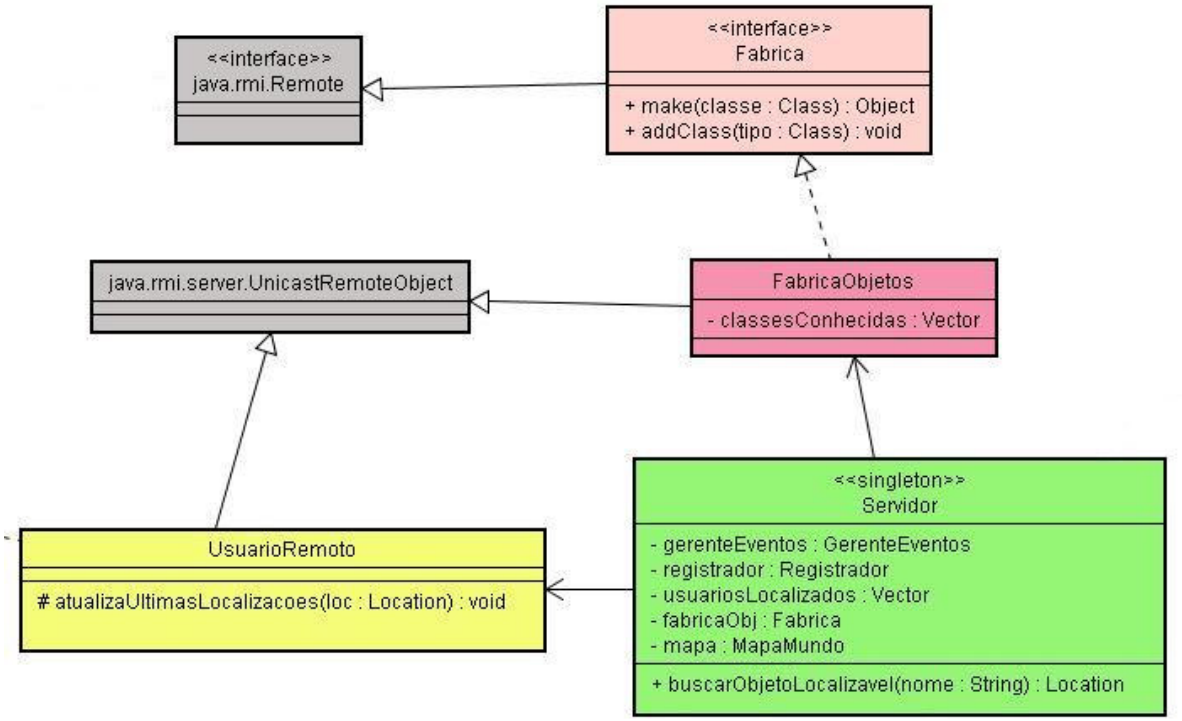

Figura 8: Principais classes do servidor

A figura 9 mostra as classes utilizadas para representar o espaço ao qual o sistema se destina. Uma vez que o modelo de localização utilizado foi o híbrido em que a base é hierárquica, o mapa de mundo foi representado por uma estrutura de árvore em que cada nodo representa um espaço hierárquico que refina o espaço superior. O atributo coordenadasRelativas define em coordenadas onde, em relação ao espaço superior hierarquicamente, se localiza um nodo dentro do seu nodo pai (exemplo: coordenadas de onde fica uma sala em um andar). O atributo sensores enumera as fontes de localização colocadas no espaço que o nodo representa. Assim, quando um dispositivo móvel informar a identificação de um sensor ou de um ponto de acesso a rede sem fio, essa identificação é mapeada para um objeto nodo e obtém-se a localização do usuário no modelo híbrido de localização.

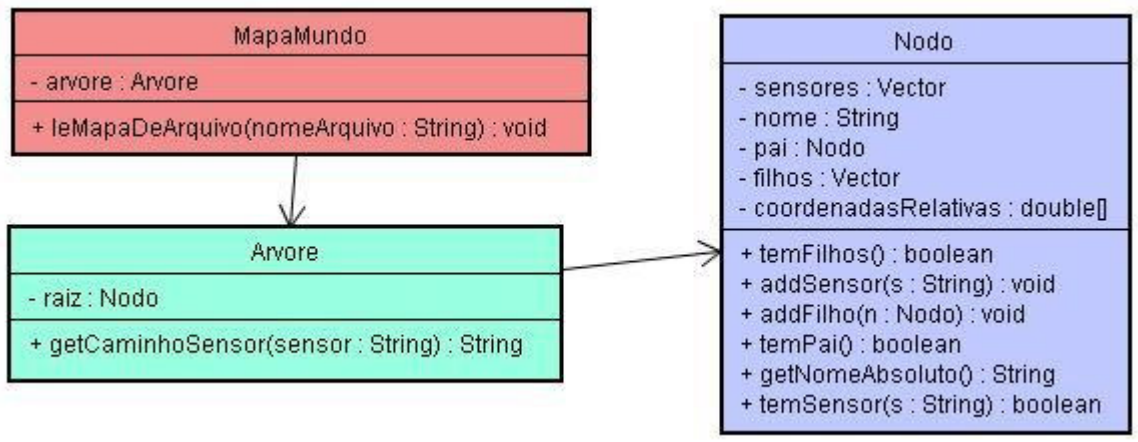

Figura 9: Classes para representação de mundo

A figura 10 mostra o diagrama de seqüência da atualização da localização de um usuário. A mudança dessa informação é percebida pelo adaptadorLocalizacao que lança um EventoAtualizaLocalizacao, recebido pelo GerenciadorLocalizacao, que repassa para o Usuário, resultando na atualização da informação de localização pelo UsuarioRemoto e notificação do GerenteEventos. Esse último, por sua vez, faz com que o Servidor atualize os dados do Usuario que lançou o evento.

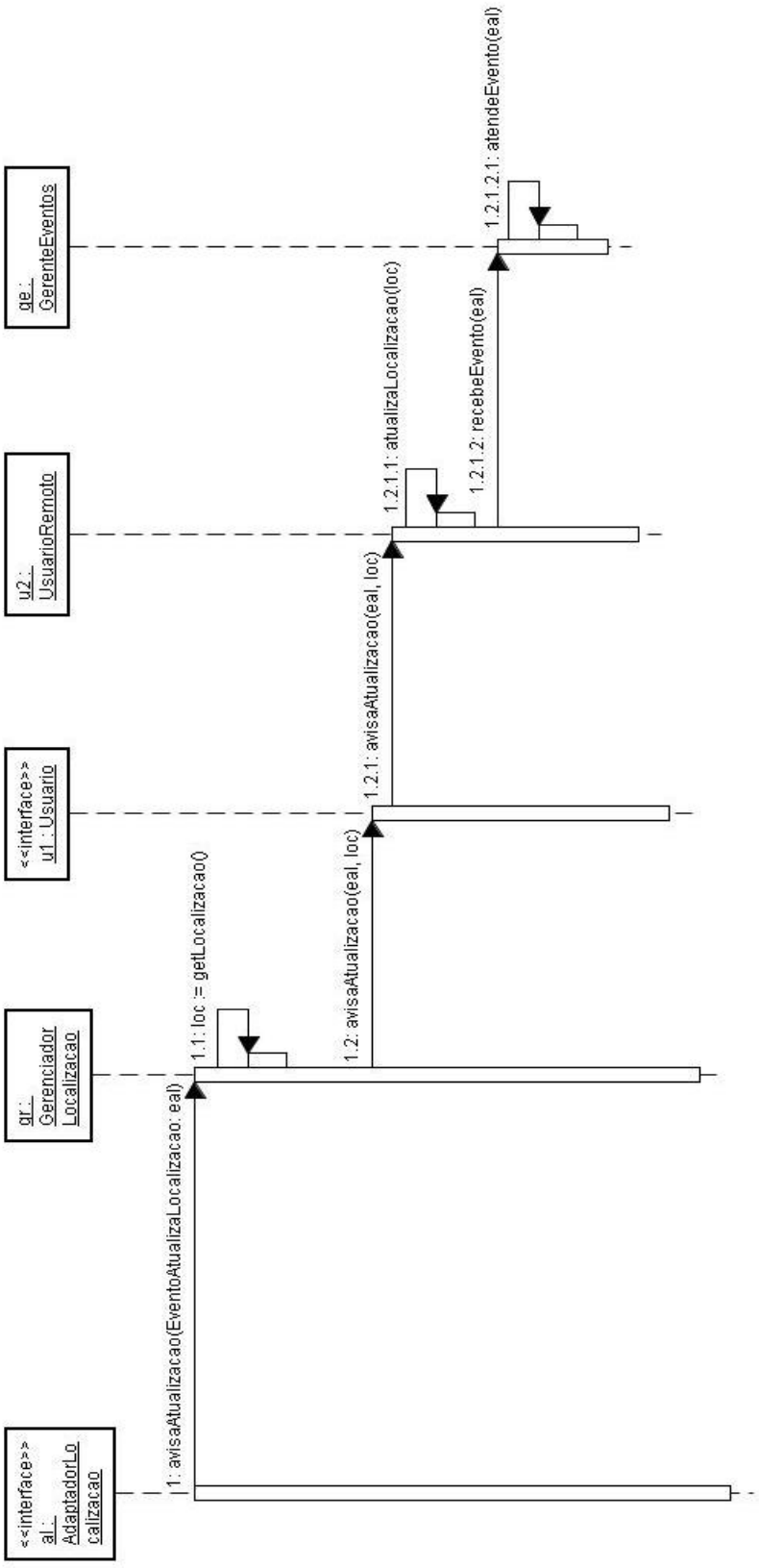

Figura 10: Diagrama de Seqüência

7 CONCLUSÕES

O mercado de computação móvel está em pleno crescimento, favorecido pela evolução das tecnologias de transmissão de dados e dos dispositivos móveis disponíveis ao usuário. Essas tecnologias também estão tornando-se progressivamente mais acessíveis aos usuários comuns e começaram a popularizar-se. Acredita-se inclusive que o número de dispositivos de computação móvel em circulação ultrapassará em breve o numero de computadores de mesa. Mas, apesar dessa evolução tecnológica, os usuários ainda sentem falta de soluções que os ajudem a melhorar sua experiência com seus dispositivos, aumentando seu conforto, satisfação e produtividade.

A utilização de informações de contexto sobre o dispositivo móvel e seu usuário possibilita sintonizar o comportamento do dispositivo móvel com o seu usuário, prever ações do usuário e preparar-se para suportá-las, fornecer mais conforto ao usuário, economizar recursos internos do aparelho, detectar e reagir a mudanças do ambiente, entre outras funcionalidades possíveis. Mas até então, o desenvolvimento de aplicações sensíveis a contexto era uma tarefa bastante custosa para os programadores, que precisavam implementar suas próprias soluções para a obtenção e gerenciamento das informações contextuais necessárias.

Para resolver esse problema, foi apresentado nesse trabalho um middleware que busca e gerencia informações temporais e de localização dos usuários de dispositivos de computação móvel. Acredita-se que a abordagem de utilização de um middleware distribuído seja a melhor solução para fornecer uma infra-estrutura padronizada para troca de informação contextual. Dessa forma,

disponibiliza-se uma plataforma de software que facilita o desenvolvimento de aplicações sensíveis a contexto para dispositivos móveis, uma vez que o programador da aplicação não precisa se preocupar com a obtenção das informações contextuais e apenas considera que as mesmas são disponibilizadas de forma padronizada pelo middleware. Assim, o middleware fornece mais uma camada de abstração para o desenvolvedor. Outra vantagem obtida pelo middleware apresentado é economia de recursos internos do dispositivo móvel. Isso é obtido através de uma arquitetura de servidor de informação em que o processamento dos dados obtidos pelo aparelho é realizado no servidor. Dessa maneira, a utilização de poder de processamento do aparelho móvel é diminuída e, portanto, a bateria do mesmo é poupada. Para acentuar essa economia, o middleware disponibiliza ainda um serviço de fábrica remota de objetos das aplicações clientes, de forma a possibilitar a distribuição também das aplicações que rodam sobre o middleware.

A arquitetura de computação móvel utilizada foi a de computação nômade ou estruturada, uma vez que o modelo do middleware é de cliente e servidor, além dessa arquitetura facilitar a obtenção da informação de localização. A utilização da arquitetura de computação nômade também favoreceu a escolha da abordagem de manutenção do modelo de representação de espaço como um serviço disponibilizado pelo servidor. Portanto, o modelo do ambiente em que um usuário pode localizar-se é obtido explicitamente através de um serviço. Outro resultado da utilização dessa arquitetura foi a restrição do domínio de aplicação para ambientes internos (*indoor*).

A principal informação contextual utilizada pelo middleware é a localização dos usuários. Mas para facilitar a interoperabilidade de aplicações sensíveis a localização, o modelo de informação de localização utilizado deve ser definido. O modelo utilizado é o modelo híbrido, que mistura o modelo geométrico e o modelo hierárquico ou simbólico a fim de combinar os pontos fortes de cada abordagem: cálculo de distâncias exatas proveniente da primeira abordagem e obtenção de relacionamentos espaciais proveniente da segunda abordagem.

Outra característica do middleware apresentado é a encapsulamento dos detalhes de interação com a fonte específica de localização, através de uma interface de adaptador de serviço de localização. Essa interface precisa ser implementada para cada tipo de fonte de localização disponível no dispositivo. Assim, o middleware não possui nenhuma restrição ao tipo de fonte de informação de localização a fim de não restringir sua aplicabilidade a uma determinada classe de dispositivos móveis.

Os serviços disponibilizados pelo middleware são de localização do usuário, últimas localizações do usuário, localização de outro usuário conhecido, usuários próximos, fábrica remota de objetos clientes e modelo de representação do espaço conhecido (mapa de mundo).

Com o desenvolvimento do middleware apresentado propõe-se uma plataforma de software para facilitar a criação de aplicações sensíveis a contexto que possibilitem melhorar a experiência do usuário com seu dispositivo móvel. Buscou-se ainda a utilização de modelos de representação de localização e espaço suficientemente genéricos para abranger qualquer aplicação sensível a localização em ambientes *indoor*.

Planeja-se aperfeiçoar o middleware desenvolvido acrescentando persistência dos dados em um banco de dados temporal, políticas de segurança que exijam autenticação dos usuários e garantam privacidade e confiabilidade das informações disponibilizadas. Pretende-se também incorporar Preferências de Usuário para permitir maior personalização do comportamento do dispositivo móvel de acordo com os desejos de seu usuário. Além de utilizar mais tipos de informação contextual, como contexto computacional (nível de energia do dispositivo, largura de banda disponível, etc) para auxiliar na identificação do contexto geral em que um dispositivo se encontra.

Apesar de utilizar poucos tipos de informações de contexto, verificou-se que o gerenciamento de contexto é uma tarefa complexa. Sentiu-se ainda a necessidade de padronização do modelo de representação de contexto para possibilitar a interoperabilidade de sistemas sensíveis a contexto. Mas apesar dos desafios, vislumbra-se que a computação sensível a contexto será uma ferramenta poderosa para auxiliar os usuários na realização de suas tarefas cotidianas e possivelmente acelerará o desenvolvimento de soluções para a computação ubíqua.

REFERÊNCIAS BIBLIOGRÁFICAS

[AA+97] ABOWD, G. D., *et al*. **Cyberguide: A Mobile Context-Aware Tour Guide**. ACM Wireless Networks, 3:421-433, 1997.

[AC94] ASTHANA, A., CRAVATTS, M., KRZYZANOWSKI, P. **An Indoor Wireless System for Personalized Shopping Assistance**. In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, p.69-74, Santa Cruz, California, 1994. IEEE Computer Society Press.

[AS98] SALBE, D., ABOWD, G, D. **The Design and Use of a Generic Context Server.** 1998. In Proceedings of the 1998 Workshop on Perceptual User Interfaces (PUI'98), p.63-66.

[BB02] BAUER, M., BECKER, C., ROTHERMEL, K. **Location Models from the Perspective of Context-Aware Applications and Mobile *Ad Hoc* Networks.** Personal and Ubiquitous Computing, Londres, v.6, n.5-6, p.322-328, 2002.

[BB04] BERFIELD, A., BEAVER, J., CHRYSANTHIS, P. K. **Profile and Context Filtering of Streaming Data for a Mobile Personal Assistant.** 2004. ACM Symposium on Applied Computing (SAC2004), p.1615-1616.

[CE01] CAPRA, L., EMMERICH, W., MASCOLO, C. **Reflective Middleware Solutions for Context-Aware Applications**. 2001. In: International Conference on Metalevel Architectures and Separation of Crosscutting Concerns, 3., 2001.

[CK00] CHEN, G., KOTZ, D. **A Survey of Context-Aware Mobile Computing Research**. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[DO01] DOMNITCHEVA, S. **Location Modeling: State of the Art and Challenges.** In Proceedings of the Workshop on Location Modeling for Ubiquitous Computing, p.13-19, Atlanta, Georgia, United States, 2001.

[DR+00] DIX, A. *et al.*; **Exploiting Space and Location as a Design Framework for Interactive Mobile Systems.** ACM Transactions on Computer-Human Interaction (TOCHI), 7(3), p. 285-321, 2000.

[FC04] FAHY, P., CLARKE, S. **CASS - Middleware for Mobile Context-Aware Applications.** In MobiSys Conference on Mobile Systems, Applications, and Services, 2., 2004, Boston, EUA.

[HJ03] HAWICK, K. A., JAMES, H. A. **Middleware for Context Sensitive Mobile Applications.** 2003. In Proc. Workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing, Adelaide, Australia. Conferences in Research and Practice in Information Technology, 21. Johnson, C., Montague, P. and Steketee, C., Eds., ACS. 133-141.

[HU02] HUANG, Q. **Supporting Context-Aware Computing in *Ad Hoc* Mobile Environments.** Technical Report WUCS-02-36, Washington University, Department of Computer Science and Engineering, St. Louis, Missouri. 2002.

[JP04] JÄRVENSIVU, R., PITKÄNEN, R., MIKKONEN, T. **Object-Oriented Middleware for Location Aware Systems.** In Proc. 19th ACM Symposium on Applied Computing (SAC2004), p.1184-1190, 2004.

[JS02] JIANG, C., STEENKISTE, P. **A Hybrid Location Model with a Computable Location Identifier for Ubiquitous Computing.** In Proc. Ubicomp 2002, number 2498 in Lecture Notes in Computer Sience, p.246ff, Gothenburg, Sweden, 2002.

[LS+02] LEI, H.; *et al.*; **The Design and Applications of a Context Service.** ACM SIGMOBILE Mobile Computing and Communications Review, New York, v.6, p.45-55, 2002.

[MA05] MANICA, H. **Gerência de Cache Semântico para Sistemas Dependentes de Localização.** Exame de qualificação a ser submetido à Universidade Federal de Santa Catarina para a obtenção do grau de Doutor em Ciência da Computação. Florianópolis, 2005.

[MR03] MEYER, R., RAKOTONIRAYNY, A. **A Survey of Research on**

Context-Aware Homes. 2003. In Proc. Workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing, Adelaide, Australia. Conferences in Research and Practice in Information Technology, 21. Johnson, C., Montague, P. and Steketee, C., Eds., ACS. 159-168.

[MU01] MUCHOW, J. W. **Core J2ME Technology & MIDP.** Prentice Hall, 2001. p. 02-26.

[NA03] NASSU, E. A. **Consultas sobre "aqui" em Sistemas de Bancos de Dados em Ambientes de Computação Nômade.** 2003. 122f. Tese (Doutorado em Ciências da Computação) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2003.

[PS96] LA PORTA, T., SABNANI, K. K., GITLIN, R. **Challenges for Nomadic Computing: Mobility Management and Wireless Communications.** ACM Journal of Nomadic Computing, Vol.1 No.1, 1996.

[SP03] SPAKER, R. **Bluetooth Basics.** Disponível em <http://www.embedded.com/internet/0007/0007ia1.htm> Acesso em: 30 de agosto de 2004.

[TO02] TOPLEY, K. **J2ME in a Nutshell.** O'Reilly & Associates, 2002. p.09-15.

[WF04] WI-FI ALLIANCE. **Glossary of Terms.** Disponível em < http://www.wi-fi.org/OpenSection/glossary.asp?TID=2> Acesso em: 31 de agosto de 2004.

[WH+92] WANT, R.; *et al*. **The Active Badge Location System**. ACM Transactions on Information Systems, vol. 10, pp. 91--102, 1992.

[WS+93] WANT, R.; *et al*. **The ParcTab Mobile Computing System**. In Proceedings of the Fourth Workshop on Workstation Operating Systems, p. 34-39, 1993.

ANEXOS

Anexo 1: Artigo

Middleware para Gerência de Informações Contextuais de Dispositivos de Computação Móvel

Helô Petry
helô@inf.ufsc.br

Abstract: Context awareness may provide more comfort to user and enchance efficiency of mobile devices. But to achieve this goal, it's necessary to create an adequate software infrastructure in order to ease development and interoperability of context sensitive applications. To propose a solution, this paper presents an object-oriented middleware that manages mobile devices location information (the most important contextual information in mobile systems). In order to use location information, we defined one model to represent this data and one model to represent space. The middleware presented uses a server-client in a nomadic or structured mobile environment.

Resumo: Sensibilidade a contexto em sistemas móveis pode prover mais conforto ao usuário e melhorar a eficiência dos dispositivos móveis. Mas para alcançar-se esse objetivo, faz-se necessário criar uma infra-estrutura adequada de software a fim de facilitar o desenvolvimento e interoperabilidade de aplicações sensíveis a contexto. Propondo uma solução, o presente trabalho apresenta um middleware orientado a objetos que gerencia informação de localização (a informação contextual mais marcante em sistemas móveis) de dispositivos móveis. Para utilização da informação de localização foram definidos um modelo de representação desse dado e um modelo de representação de espaço. O middleware apresentado utiliza uma arquitetura de cliente-servidor em um ambiente de computação móvel estruturada ou nômade.

Palavras-chave: Computação sensível a contexto, sensibilidade a contexto, middleware, computação móvel.

1 Introdução

A contínua evolução dos dispositivos de computação móvel, como os Assistentes Digitais Pessoais (Personal Digital Assistant - PDAs), vem tornando esses equipamentos cada vez mais poderosos, capazes de executar softwares de alto nível e mais sofisticados, incorporando funcionalidades antes somente suportáveis por computadores convencionais. As tecnologias de transmissão de dados sem fio também têm se desenvolvido consideravelmente e a junção dessas duas evoluções, dos dispositivos móveis e da comunicação sem fio, surgiu computação móvel trazendo consigo uma numerosa nova gama de aplicações que podem ser desenvolvidas e possibilitando que os dispositivos móveis forneçam a seus usuários muitas novas possibilidades de experiências.

Esses dispositivos móveis estão se popularizando e possivelmente se tornarão mais utilizados que os PCs. Mas para que isso realmente ocorra, ainda é necessário criar uma plataforma de software adequada. E para melhorar a experiência dos usuários de dispositivos móveis, é possível adicionar funcionalidades que explorem as características

específicas desses equipamentos: estar constantemente em movimento e em poder do usuário. Essas características reforçam a relevância do contexto em que o dispositivo se insere. Contexto, resumidamente, é o ambiente em que o dispositivo encontra-se, com todas as informações que afetam, de alguma maneira, o comportamento e intenções do usuário e das aplicações suportadas. A utilização de informações de contexto possibilita a geração de inúmeras novas classes de aplicação que, entre outras coisas, pode sintonizar o comportamento do dispositivo móvel com o do seu usuário. As aplicações sensíveis a contexto se adaptam às condições do usuário, examinam mudanças do ambiente computacional e reagem a essas mudanças.

Embora sensibilidade a contexto seja uma característica bastante importante e útil em sistemas móveis, ainda há falta de infra-estruturas genéricas para o desenvolvimento de aplicações sensíveis a contexto [AS98]. Sente-se a necessidade de um middleware que suporte abstrações de contexto em alto nível [FC04]. Propondo uma solução possível, o presente trabalho apresenta o desenvolvimento de um middleware que busca e gerencia informações contextuais a fim de personalizar o comportamento do dispositivo móvel. Fazendo isso, esse middleware poderá fornecer serviços a aplicações que utilizem essas informações e facilitar o desenvolvimento de aplicações sensíveis a contexto.

O objetivo central do middleware desenvolvido é coletar e disponibilizar, de forma homogênea, informações contextuais específicas dos dispositivos móveis clientes. Porém, as informações disponibilizadas não são relativas apenas a cada usuário isoladamente, mas de todos usuários cadastrados. Assim, também é possível obter-se um contexto social: quem está onde, quem está por perto, etc. Além disso, a arquitetura utilizada procura diminuir ao máximo o consumo de energia do dispositivo móvel para melhorar a experiência do usuário, que não gosta de precisar recarregar seu aparelho constantemente.

Neste trabalho é apresentada uma proposta de solução para o gerenciamento de informações contextuais de dispositivos móveis.

2 Contexto

Aplicações para dispositivos móveis apresentam problemas desafiadores para os programadores. Esses dispositivos enfrentam perda temporária de conexão de rede quando se movem, descobrem outros *hosts* de maneira *ad-hoc*; é provável que tenham recursos escassos, como pouca bateria, menor poder de processamento e memória pequena; precisam reagir a mudanças freqüentes e não anunciadas do ambiente, como alta variabilidade da banda de rede, nova localização, etc. Portanto, sistemas móveis precisam detectar e se adaptar a mudanças drásticas no ambiente. O contexto de um dispositivo pode ajudar muito na realização dessa tarefa [CE01].

O ambiente em que um sistema computacional se localiza pode tornar claro seu significado ou intenções e necessidades do usuário que o sistema computacional está suportando. Esse ambiente em que um sistema computacional se localiza chama-se contexto [LS+02]. Contexto é ainda o conjunto de dados externos que projetistas decidiram que pode influenciar o comportamento de uma aplicação correspondente e que pode ser usado para caracterizar a situação de uma entidade [HU02, LS+02]. Uma entidade é uma pessoa, lugar ou objeto que é considerado relevante para a interação entre o usuário e a aplicação, incluindo os próprios usuário e aplicação em si [LS+02].

Sensibilidade a contexto melhora aplicações existentes e até habilita novas classes de aplicações em computação pervasiva. Essas aplicações podem auxiliar o usuário a navegar em territórios desconhecidos, encontrar restaurantes próximos, receber mensagens da forma mais útil e menos intrusiva possível, entre outros. O maior benefício da sensibilidade a contexto é possibilitar que as aplicações se adaptem para melhor prover as necessidades do usuário e suas tarefas, exigindo menos do usuário e aumentando sua satisfação e produtividade [LS+02].

Uma possível classificação das informações de contexto, segundo [SAW94, CK00], é a seguinte:

1. Contexto Computacional: inclui informações referentes ao dispositivo como, por exemplo, configuração do hardware do dispositivo, nível de energia, largura de banda atual.
2. Contexto de Usuário: são informações que dizem respeito ao usuário como, por exemplo, humor, preferências, biometria, entre outras.
3. Contexto Físico: informações obtidas a partir do ambiente físico em que o dispositivo se encontra: temperatura, quantidade de luz, localização.
4. Contexto Temporal: são informações do gênero ano, dia, minuto.

Embora as várias informações contextuais citadas acima sejam relevantes, a mais importante nos sistemas móveis é a localização do usuário, uma vez que mobilidade é uma característica única desses sistemas. Além disso, a informação de localização pode ser explorada como meio de entendimento do contexto completo em que o sistema está posto. Essencialmente, a localização torna-se um dispositivo útil a partir do qual deduz-se o contexto global que está influenciando a aplicação móvel [DR+00]. Usuários de computadores móveis podem estar em diferentes locais e podem desejar diferentes comportamentos de seus dispositivos móveis em função da sua localização.

O desenvolvimento de um modelo de localização detalhado é uma tarefa custosa com respeito à estrutura inicial e manutenção, então acredita-se que diferentes aplicações deveriam conseguir compartilhar o mesmo modelo de informação [BB02]. Isso aumentaria a interoperabilidade entre aplicações e tornaria possível novas classes de aplicação [BB02].

A informação de localização pode ser classificada de acordo com a técnica de representação da mesma. Essa classificação é apresentada a seguir.

2.1 Modelo de Localização Geométrico

Uma localização é especificada como uma coordenada n-dimensional (tipicamente n = 2 ou 3) - por exemplo, o par latitude-longitude ou tripla latitude-longitude-altitude retornado por um Sistema de Posicionamento Global (GPS) - ou um conjunto de coordenadas definindo a forma geométrica das fronteiras de uma área [LX+02]. Esse modelo é baseado em um (modelo simples) ou mais (modelo unificado) sistemas de coordenadas (reference coordinate system - RCS) [DO01].

As vantagens desse modelo é sua compatibilidade entre sistemas heterogêneos e a facilidade para obtenção de distâncias (pelo cálculo de distâncias euclidianas, por exemplo). Entretanto, pode ser bastante custoso em termos de volume de dados envolvidos e da necessidade de mapear a representação geométrica para um nível semântico adequado para as aplicações [LX+02].

2.2 Modelo de Localização Hierárquico ou Simbólico

Modelos de localização hierárquica decompõem o ambiente físico em diferentes níveis de precisão e normalmente apresentam uma representação de localização auto-descritiva [JS02]. Entidades lógicas do mundo real descrevem o espaço de localização. Entidades podem ser prédios, ruas, cidades, ou elementos definidos pelo sistema como células sem fio, e são identificáveis unicamente por um sistema de nomeação hierárquico. Um exemplo típico é o de endereço postal, como: Brasil, Santa Catarina, Florianópolis, Universidade Federal de Santa Catarina, Centro Tecnológico, 2º andar, sala 201.

O modelo hierárquico tipicamente tem granularidade de localização menor que o modelo geométrico porque enfatiza a representação dos relacionamentos entre entidades lógicas em vez de coordenadas precisas [LX+02]. Por ser discreta e bem estruturada, informação de localização hierárquica é mais fácil de gerenciar. Entretanto, é difícil converter localizações entre sistemas heterogêneos [LX+02].

2.3 Modelo de Localização Híbrido

Segundo [JS02], da perspectiva de aplicações sensíveis a contexto, nenhum dos dois modelos anteriores é completamente satisfatório sozinho. As duas classes de modelos têm vantagens e desvantagens com respeito às necessidades de aplicações sensíveis a contexto [JS02]. O modelo hierárquico é bom para representações implícitas de relacionamentos espaciais, como possessão, proximidade, além de ser legível por humanos. A maior desvantagem do modelo hierárquico é a falta de precisão de posição e o fato de ser difícil ou ineficiente calcular distâncias. As falhas do modelo de localização hierárquico expressam os benefícios do modelo de localização geométrico. Com atributos geométricos embutidos, modelos de localização geométricos são bem adaptados para especificar localização precisamente e para calcular distâncias exatas. Entretanto, o modelo de localização geométrico esconde relacionamentos hierárquicos. Então, ele precisa de detalhes adicionais para deduzir relacionamentos espaciais.

[JS02] propõe um modelo de localização híbrido para aplicações sensíveis a contexto que combina os benefícios dos dois lados. O ponto de partida do modelo híbrido é o modelo de localização hierárquico: vê-se o mundo como uma hierarquia de espaços em que cada nível adiante refina e subdivide os espaços do nível anterior. O modelo geométrico entra permitindo que cada espaço na hierarquia defina um sistema de coordenadas que pode ser utilizado para definir pontos ou áreas dentro daquele espaço. Diferentes espaços podem usar diferentes sistemas de coordenadas.

3 Trabalhos Relacionados

Nesta seção apresenta-se dois trabalhos interessantes que propõem middlewares para gerenciar sensibilidade a contexto em computação móvel.

3.1 Hawick e James

Foi desenvolvido um middleware gerenciador de contexto utilizando informação de contexto de localização e preferências de usuário. O trabalho completo é descrito em [HJ03].

Usuários podem desejar diferentes comportamentos para seus dispositivos em função da sua localização. O usuário pode ajustar várias políticas sobre quando ele deseja ou não deseja receber mensagens (de eventos entre aplicações ou entre usuários móveis e estações base) em função de sua localização. Considerando ainda que usuários têm sido bombardeados com toda sorte de informações não solicitadas, objetiva-se com esse trabalho criar um sistema de software que pode ser usado para reduzir a quantidade de informação irrelevante que é vista por um usuário móvel.

O sistema é projetado para reagir a eventos assim que eles são fornecidos pelas aplicações. Quando um evento ocorre, como um e-mail sendo recebido, um arquivo sendo criado, ou simplesmente uma mensagem sendo adicionada a uma fila, o Gerenciador de Contexto consulta os outros componentes (módulos preferências ativas, gerenciador de localização, calendário) e decide se o usuário ou sua aplicação devem ser avisados do evento. O sistema é ciente da localização do usuário, a hora corrente e a informação que o usuário deseja receber.

Preferências ativas são informações personalizadas pelo usuário que são expressas com dependências espacial e temporal [HJ03]. Um exemplo de preferência convencional seria "use essa como minha fonte preferida". Uma preferência ativa incorpora condições, então um exemplo é "desligue o som quando estou nesses lugares". O componente mais crítico do sistema é o módulo de Preferências Ativas. Foram feitas experiências com uma representação baseada em XML de uma linguagem simples e geral acoplada com uma máquina Prolog baseada em Java. Os resultados iniciais foram

promissores, mas considera-se ainda a necessidade de nomear usuários individuais, aplicações, componentes de software e zonas. O sistema mantém as preferências de usuário como uma árvore XML. O uso de tal estrutura de dado também permite a publicação de políticas de aplicação ou de sistema que devem ser incorporadas nas tomadas de decisão.

3.2 Järvensivu, Pitkänen e Mikkonen

Foi desenvolvido um middleware orientado a objetos especialmente projetado para aplicações móveis sensíveis a localização. O trabalho completo é apresentado em [JP04].

Aplicações baseadas em localização são inerentemente sistemas distribuídos, tipicamente constituídas de componentes cliente e componentes servidor. Sistemas distribuídos modernos normalmente são construídos sobre algum tipo de middleware, fazendo a distribuição tão transparente ao programador de aplicação quanto possível. Para atender os requisitos levantados em ambos domínios: servidor e terminal, Järvensivu, Pitkänen e Mikkonen implementaram um protótipo experimental chamado OLOS (Object-Oriented LOcation System - Sistema de Localização Orientado a Objetos).

OLOS é construído com Java e RMI, e integra o conceito de localização com uma visão orientada a objetos do mundo. Localização transforma-se em um atributo de uma classe de objetos. OLOS integra-se com diferentes plataformas de posicionamento e APIs pelo fornecimento de uma interface de escrita de adaptadores de sistema de posicionamento.

3.2.1 Arquitetura do OLOS

No modelo de distribuição OLOS, as solicitações de processamento no lado do cliente são reduzidas para beirar o mínimo. Em essência, apenas uma pequena implementação de interface local roda no dispositivo portátil para encapsular chamadas a métodos remotos para serviços remotos. A abordagem típica de *factory* (fábrica) é utilizada para criar objetos remotos. Assim, as fábricas de objetos OLOS e implementações do lado do servidor rodam em um servidor de aplicação, com o qual os terminais podem se comunicar via interface RMI Java.

Um componente chave na arquitetura OLOS é OLOS *Registry* (Registro OLOS), que associa objetos a dados de localização e age como uma interface para procurar objetos e fábricas. Registro OLOS deve ser conectado a um serviço de posicionamento. Um serviço de posicionamento pode ser qualquer sistema adequado para obter dados de localização de um terminal móvel. A comunicação com o serviço de posicionamento é encapsulado em um componente denominado Adaptador de Serviço de Posicionamento (Positioning Service Adapter - PSA). Um programador de aplicação pode implementar um PSA adequado para um serviço de posicionamento desenvolvido fornecendo uma classe que implementa uma interface denominada LocationServiceController.

Do ponto de vista do programador de aplicação, OLOS é uma extensão relativamente pequena de Java/RMI para implementar objetos sensíveis a localização. Objetos OLOS são objetos Java/RMI distribuídos com funcionalidade adicional para adquirir sensibilidade a localização. Além disso, uma classe local é gerada para cada classe objeto OLOS para prover serviços básicos de ciclo de vida de objeto e buscas baseadas em localização.

4 Ambiente Gerente de Contexto Proposto

Neste trabalho é apresentado um middleware que gerencia informações de contexto em sistemas móveis com o objetivo de fornecer uma camada de abstração aos desenvolvedores de aplicações sensíveis a contexto para dispositivos móveis.

O middleware desenvolvido colhe determinadas informações contextuais a fim de fornecer informações personalizadas para aplicações móveis sensíveis ao contexto (desenvolvidas para o dispositivo móvel), permitindo a estas aplicações adequar-se às necessidades do usuário.

São características do ambiente desenvolvido:

- As informações contextuais utilizadas são de localização do usuário e data/hora.
- O domínio de aplicação abordado é o de ambientes internos (*indoor*).
- O modelo de localização utilizado é o híbrido, que inclui informação de localização tanto simbólica quanto em coordenadas.
- Não é feita nenhuma limitação ao tipo de fonte de localização utilizada a fim de não restringir a classe de dispositivo cliente e, assim, fornecer total portabilidade ao middleware.
- Utiliza-se o ambiente de computação nômade (ou estruturada), em que cada dispositivo móvel comunica-se exclusivamente com uma estação base.
- Em virtude do ambiente estruturado, torna-se natural a utilização de uma arquitetura cliente-servidor, em que os serviços de informação são disponibilizados por um computador com maior poder de processamento e mais largura de banda.
- Economia de recursos internos do dispositivo móvel, principalmente processamento e, conseqüentemente, bateria. Consegue-se isso através da distribuição do middleware entre estruturas cliente e servidor: no dispositivo móvel cliente permanece apenas uma pequena interface que encapsula chamadas a métodos remotos para serviços remotos; o servidor mantém os objetos remotos da aplicação que efetivamente processam toda a lógica interna de funcionamento. Java/RMI possibilita a implementação desta solução.
- Possibilita a distribuição inclusive das aplicações que façam uso dele através de uma fábrica de objetos remota e genérica.

A obtenção da informação de localização é feita por um componente que encapsula os detalhes da interação com as fontes de localização para coletar e distribuir a informação de onde o usuário se encontra. A fim de não restringir o middleware a qualquer fonte de localização predefinida, esse módulo disponibiliza uma interface para adaptador de serviço de localização, que deve ser implementado para cada tipo de serviço disponível.

No lado do cliente, o middleware possibilita a realização de algumas tarefas: registrar e descadastrar o cliente no sistema, informar a localização do cliente, solicitar a localização de um usuário específico, solicitar ser informado dos usuários próximos e fabricar objetos remotos. No servidor é mantido um serviço de registro dos objetos localizáveis (usuários móveis e serviços disponíveis) a partir de uma informação de identificação fornecida pelo mesmo. Depois do registro, o middleware conhece os objetos e suas localizações. O middleware pode manter registro de Pontos de Referência como serviços e locais específicos. O servidor também possui fábricas de objetos que são utilizadas pelas aplicações do cliente para instanciar suas classes. Uma fábrica cria objetos locais e retorna a referência remota para o cliente.

As informações dinâmicas que o middleware disponibiliza para a camada de aplicação foram definidas como as seguintes:

- localização física do usuário;
- hora atual;
- últimas 10 localizações dos usuários;
- serviços disponíveis próximos;
- usuários próximos;
- localização de um usuário específico (um cliente pode buscar a localização de outro);

Utiliza-se o modelo de localização híbrido, pois o mesmo cobre as deficiências de ambos modelos geométrico e hierárquico e é apropriado para representar ambientes *indoor*. Localizações *indoor* consistem, por exemplo, de prédios, andares e salas. O middleware vê localização de maneira hierárquica, como em estrutura de árvore em que cada nodo possui um dado hierárquico e um dado em coordenadas. Essa estrutura é uma fonte de dados de possíveis localizações e considerada disponível ao middleware e seus clientes. A fim de obter-se a informação de localização no modelo híbrido a partir apenas do dado geométrico, mantém-se um registro com a relação das fontes de localização geométrica (sensores, pontos de acesso Wi-Fi, etc) e suas respectivas localizações no modelo hierárquico.

As principais classes hospedadas no cliente são o AdaptadorLocalizacao, a interface Usuario e o GerenciadorLocalizacao. O AdaptadorLocalizacao encapsula os detalhes de interação com a fonte de informação de localização e é responsável por verificar a mudança na localização do usuário e gerar um evento para o middleware. Usuario é uma interface de objeto remoto RMI e possui vários métodos para obtenção da localização própria e de outros usuários. GerenciadorLocalizacao obtém o objeto Usuario via RMI e repassa a ele sua localização cada vez que o AdaptadorLocalizacao atualiza essa informação.

As principais classes hospedadas no servidor são o UsuarioRemoto, a FabricaObjetos, o Registrador o GerenteEventos e o Mapa de Mundo. UsuarioRemoto é a implementação da interface Usuario presente no cliente e suas instâncias são distribuídas via RMI pelo servidor. A interface Fabrica pode ser utilizada por aplicações cliente para enviar suas classes para serem instanciadas pelo objeto remoto FabricaObjetos, localizado no servidor. O Registrador é responsável por cadastrar os usuários e lhes fornecer um identificador único. Depois do registro, o Servidor conhece a localização do usuário e pode fornecer essa informação se for solicitado. GerenteEventos avisa o Servidor quando determinado evento ocorre (inicialmente apenas mudança na informação de localização de algum usuário cadastrado). Mapa de Mundo representa o espaço ao qual o sistema se destina. Uma vez que o modelo de localização utilizado foi o híbrido em que a base é hierárquica, o mapa de mundo foi representado por uma estrutura de árvore em que cada nodo representa um espaço hierárquico que refina o espaço superior.

5 Conclusões

Apesar da evolução tecnológica das tecnologias de transmissão de dados e dos dispositivos móveis disponíveis no mercado, os usuários ainda sentem falta de soluções que os ajudem a melhorar sua experiência com seus dispositivos, aumentando seu conforto, satisfação e produtividade.

A utilização de informações de contexto sobre o dispositivo móvel e seu usuário possibilita sintonizar o comportamento do dispositivo móvel com o seu usuário, prever ações do usuário e preparar-se para suportá-las, fornecer mais conforto ao usuário, economizar recursos internos do aparelho, detectar e reagir a mudanças do ambiente, entre outras funcionalidades possíveis. Mas até então, o desenvolvimento de aplicações sensíveis a contexto era uma tarefa bastante custosa para os programadores, que precisavam implementar suas próprias soluções para a obtenção e gerenciamento das informações contextuais necessárias.

Para resolver esse problema, foi apresentado nesse trabalho um middleware que busca e gerencia informações temporais e de localização dos usuários de dispositivos de computação móvel. Dessa forma, disponibiliza-se uma infra-estrutura padronizada para troca de informação contextual que facilita o desenvolvimento de aplicações sensíveis a contexto para dispositivos móveis. Outra vantagem obtida pelo middleware apresentado é economia de recursos internos do dispositivo móvel. Isso é obtido através de uma

arquitetura de servidor de informação em que o processamento dos dados obtidos pelo aparelho é realizado no servidor.

Com o desenvolvimento do middleware apresentado propõe-se uma plataforma de software para facilitar a criação de aplicações sensíveis a contexto que possibilitem melhorar a experiência do usuário com seu dispositivo móvel. Buscou-se ainda a utilização de modelos de representação de localização e espaço suficientemente genéricos para abranger qualquer aplicação sensível a localização em ambientes *indoor*.

Planeja-se aperfeiçoar o middleware desenvolvido acrescentando persistência dos dados em um banco de dados temporal, políticas de segurança que exijam autenticação dos usuários e garantam privacidade e confiabilidade das informações disponibilizadas. Pretende-se também incorporar Preferências de Usuário para permitir maior personalização do comportamento do dispositivo móvel de acordo com os desejos de seu usuário. Além de utilizar mais tipos de informação contextual, como contexto computacional (nível de energia do dispositivo, largura de banda disponível, etc) para auxiliar na identificação do contexto geral em que um dispositivo se encontra.

Apesar de utilizar poucos tipos de informações de contexto, verificou-se que o gerenciamento de contexto é uma tarefa complexa. Sentiu-se ainda a necessidade de padronização do modelo de representação de contexto para possibilitar a interoperabilidade de sistemas sensíveis a contexto. Mas apesar dos desafios, vislumbra-se que a computação sensível a contexto será uma ferramenta poderosa para auxiliar os usuários na realização de suas tarefas cotidianas e possivelmente acelerará o desenvolvimento de soluções para a computação ubíqua.

Referências

[AS98] SALBE, D., ABOWD, G, D. **The Design and Use of a Generic Context Server.** 1998. In Proceedings of the 1998 Workshop on Perceptual User Interfaces (PUI'98), p.63-66.

[BB02] BAUER, M., BECKER, C., ROTHERMEL, K. **Location Models from the Perspective of Context-Aware Applications and Mobile *Ad Hoc* Networks.** Personal and Ubiquitous Computing, Londres, v.6, n.5-6, p.322-328, 2002.

[CE01] CAPRA, L., EMMERICH, W., MASCOLO, C. **Reflective Middleware Solutions for Context-Aware Applications**. 2001. In: International Conference on Metalevel Architectures and Separation of Crosscutting Concerns, 3., 2001.

[CK00] CHEN, G., KOTZ, D. **A Survey of Context-Aware Mobile Computing Research**. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[DO01] DOMNITCHEVA, S. **Location Modeling: State of the Art and Challenges.** In Proceedings of the Workshop on Location Modeling for Ubiquitous Computing, p.13-19, Atlanta, Georgia, United States, 2001.

[DR+00] DIX, A. *et al.*; **Exploiting Space and Location as a Design Framework for Interactive Mobile Systems.** ACM Transactions on Computer-Human Interaction (TOCHI), 7(3), p. 285-321, 2000.

[FC04] FAHY, P., CLARKE, S. **CASS - Middleware for Mobile Context-Aware Applications.** In MobiSys Conference on Mobile Systems, Applications, and Services, 2., 2004, Boston, EUA.

[HJ03] HAWICK, K. A., JAMES, H. A. **Middleware for Context Sensitive Mobile Applications.** 2003. In Proc. Workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing, Adelaide, Australia. Conferences in Research and Practice in Information Technology, 21. Johnson, C., Montague, P. and Steketee, C., Eds., ACS. 133-141.

[HU02] HUANG, Q. **Supporting Context-Aware Computing in *Ad Hoc* Mobile Environments.** Technical Report WUCS-02-36, Washington University, Department of Computer Science and Engineering, St. Louis, Missouri. 2002.

[JP04] JÄRVENSIVU, R., PITKÄNEN, R., MIKKONEN, T. **Object-Oriented Middleware for Location Aware Systems.** In Proc. 19th ACM Symposium on Applied Computing (SAC2004), p.1184-1190, 2004.
[JS02] JIANG, C., STEENKISTE, P. **A Hybrid Location Model with a Computable Location Identifier for Ubiquitous Computing.** In Proc. Ubicomp 2002, number 2498 in Lecture Notes in Computer Sience, p.246ff, Gothenburg, Sweden, 2002.
[LS+02] LEI, H.; *et al.*; **The Design and Applications of a Context Service.** ACM SIGMOBILE Mobile Computing and Communications Review, New York, v.6, p.45-55, 2002.
[SAW94] SCHILIT, B. N., ADAMS, A., WANT, R. **Context-aware computing applications.** In Proceedings Workshop on Mobile Computing Systems and Applications. IEEE, December 1994.

Anexo 2: Código fonte

```
package cliente;
public class Location {
	protected String hierarquico;
	protected double[] coordenadas;
	protected String sistemaCoordenadas;

	public Location(String caminho){
		hierarquico = caminho;
		coordenadas = new double[2];
		coordenadas[0] = 0;
		coordenadas[1] = 0;
	}
	public Location(String caminho, double[] c, String sc){
		hierarquico = caminho;
		coordenadas = c;
		sistemaCoordenadas = sc;
	}
	public double[] getCoordenadas() {
		return coordenadas;
	}
	public String getHierarquico() {
		return hierarquico;
	}
	public String getSistemaCoordenadas() {
		return sistemaCoordenadas;
	}
	public void setCoordenadas(double[] coordenadas) {
		this.coordenadas = coordenadas;
	}
	public void setHierarquico(String hierarquico) {
		this.hierarquico = hierarquico;
	}
	public void setSistemaCoordenadas(String sistemaCoordenadas) {
		this.sistemaCoordenadas = sistemaCoordenadas;
	}
}

package cliente;
import java.rmi.RemoteException;
public interface ObjetoLocalizavel extends java.rmi.Remote{

	public int getID() throws RemoteException;
	public Location getLocalizacao() throws RemoteException ;
	public boolean regristreSe(String login) throws RemoteException;
	public int distanciaDe(ObjetoLocalizavel ob) throws RemoteException;
	public int getTipo() throws RemoteException;
}

package cliente;
import java.rmi.RemoteException;
public interface Usuario extends ObjetoLocalizavel{
	public Location localize(String nome) throws RemoteException;
	public Usuario[] quemEstaProximo(int nivelProximidade) throws RemoteException;
	public Usuario[] quemEstaProximo(float raio) throws RemoteException;
	public void atualizaLocalizacao(Location l) throws RemoteException;
	public Location[] getUltimasLocalizacoes() throws RemoteException;
```

```
	public void setID(int id) throws RemoteException;
}

package cliente;
public class AdaptadorLocalizacao {
// deve ser extendida para cd tipo de fonte de localizaçao, sobrescrevendo o metodo
// getLocalizacao
	public static final int TIPO_HIERARQUICO = 1;
	public static final int TIPO_GEOMETRICO = 2;
	protected int tipo;
	protected double precisao;
	protected GerenciadorLocalizacao gerenciador;

	public AdaptadorLocalizacao(int t, GerenciadorLocalizacao g){
		tipo = t;
		gerenciador  = g;
	}
	public void avisaAtualizacaoLocalizacao(EventoAtualizaLocalizacao eal){
		gerenciador.setAtualizacao(eal);
	}
	public Location getLocalizacao(){
		return null;
	}
	public int getTipoLocalizador(){
		return tipo;
	}
	public double getPrecisaoLocalizador(){
		return precisao;
	}
}

package cliente;
import java.util.Date;
public abstract class Evento {
	public static final int ATUALIZA_LOCALIZACAO = 1;
	protected int tipo;
	protected Date data;

	public Date getData() {
		return data;
	}
	public int getTipo() {
		return tipo;
	}
	public Evento() {
		super();
	}
}

package cliente;
public class EventoAtualizaLocalizacao extends Evento {
	protected Location nova;
	protected Usuario movel;

	public Usuario getUsuario() {
		return movel;
	}
	public Location getLocation() {
```

```
		return nova;
	}
	public EventoAtualizaLocalizacao() {
		super();
		tipo = ATUALIZA_LOCALIZACAO;
	}
	public EventoAtualizaLocalizacao(Location n, Usuario m){
		nova = n;
		movel = m;
		tipo = ATUALIZA_LOCALIZACAO;
	}
}

package cliente;
import java.util.Vector;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
public class GerenciadorLocalizacao {
	Usuario user;
	Vector fontesLocalizacao;	// vetor de adapatadorLocalizacao

	public GerenciadorLocalizacao() {
		super();
		fontesLocalizacao = new Vector();
		try {
			user = (Usuario) Naming.lookup("rmi://localhost/Usuario_Remoto");
		} catch (MalformedURLException e) {
			e.printStackTrace();
		} catch (RemoteException e) {
			e.printStackTrace();
		} catch (NotBoundException e) {
			e.printStackTrace();
		}
	}
	public boolean registraUsuario(String login){
		boolean r = false;
		try {
			r = user.regristreSe(login);
		} catch (RemoteException e) {
			e.printStackTrace();
		}
		return r;
	}
	protected void addAdaptador(AdaptadorLocalizacao adaptador) {
		fontesLocalizacao.addElement(adaptador);
	}
	public void novoAdaptador(int tipo){
		AdaptadorLocalizacao ad = new AdaptadorLocalizacao(tipo, this);
		addAdaptador(ad);
	}
	public Location getLocalizacao() throws RemoteException{
		return user.getLocalizacao();
	}
	public void atualizaLocalizacao(Location l){
		try {
			user.atualizaLocalizacao(l);
```

```
                } catch (RemoteException e) {
                        e.printStackTrace();
                }
        }
        public void setAtualizacao(EventoAtualizaLocalizacao eal){
                atualizaLocalizacao(eal.getLocation());
        }
}

**********************************************************************************************
package servidor;
import java.util.Vector;
import cliente.Location;
import cliente.ObjetoLocalizavel;
import cliente.Usuario;
import java.rmi.Naming;
import java.rmi.RemoteException;

public class Servidor {
        protected Registrador registrador;
        protected Vector objetosLocalizados;    //armazena UsuarioRemoto localizados
        protected GerenteEventos gerEventos;
        private static Servidor instancia;
        protected MapaMundo mapa;

        private Servidor() {                    // Singleton
                objetosLocalizados = new Vector();
                registrador = new Registrador();
                gerEventos = new GerenteEventos(this);
                String arquivo = new String("E:\\Helô\\TCC\\mapaMundo.txt");
                mapa = new MapaMundo(arquivo);
        }

        public static synchronized Servidor getServidor(){
                if (instancia == null) {
                        instancia = new Servidor();
                }
                try {
                        UsuarioRemoto user = new UsuarioRemoto(instancia);
                        Naming.rebind("rmi://localhost:1099/Usuario_Remoto", user);
                        FabricaObjetos fo = new FabricaObjetos();
                        Naming.rebind("rmi://localhost:1099/Fabrica_Objetos_Remota", fo);
                } catch (Exception e) {
                        e.printStackTrace();
                }
                 return instancia;
        }
        public void novoUsuarioRemoto(){
                try {
                        UsuarioRemoto user = new UsuarioRemoto(instancia);
                        Naming.rebind("rmi://localhost:1099/Usuario_Remoto", user);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
        public Vector getUsuariosProximos(Usuario usuario, int nivel) throws
                                                                RemoteException{
                Vector result = new Vector();
```

```
		try{
		Location localizacaoAlvo = usuario.getLocalizacao();
		for(int i = 0; i<objetosLocalizados.size(); i++){
			Usuario user = (Usuario)objetosLocalizados.elementAt(i);
			if(user.getID() != usuario.getID()){
				Location locationTeste = user.getLocalizacao();
				int[] testeNivel = new int[2];
				testeNivel = testaNivelProximidade(localizacaoAlvo,
									locationTeste);
				if (testeNivel[1] == 1){ // 2 usuarios com a mesma localizacao
					result.add(user);
				}else{
					if(!(testeNivel[0]>nivel)){
						result.add(user);
					}
				}
			}
		}
		} catch (RemoteException e) {
			throw(e);
		}
		return result;
	}
	protected int[] testaNivelProximidade(Location l1, Location l2){
		String nodo1, nodo2;
		String h1 = l1.getHierarquico();
		Vector c1 = divideEmNomesDosNodos(h1);
		String h2 = l2.getHierarquico();
		Vector c2 = divideEmNomesDosNodos(h2);
		int tamanho1 = h1.length();
		int tamanho2 = h2.length();
		int nivelEquivalencia = 0;
		int caminhosIguais = 0;		// default: diferentes
		if (tamanho1<tamanho2){
			for(int i=0; i<tamanho1; i++){
				nodo1 = (String)c1.elementAt(i);
				nodo2 = (String)c2.elementAt(i);
				if(nodo1.equals(nodo2)){
					nivelEquivalencia++;
				}
				else{
					break;
				}
			}
		}else{	// tamanho2 <= tamanho1
			for(int i=0;i<tamanho2; i++){
				nodo1 = (String)c1.elementAt(i);
				nodo2 = (String)c2.elementAt(i);
				if(nodo1.equals(nodo2)){
					nivelEquivalencia++;
				}
				else{
					break;
				}
			}
			if((tamanho1==tamanho2)&&(nivelEquivalencia==tamanho1)){
				caminhosIguais = 1;
			}
```

```
                }
                int[] result = new int[2];
                result[0] = tamanho1 - nivelEquivalencia;
                result[1] = caminhosIguais;
                return result;
        }
        protected Vector divideEmNomesDosNodos(String local){
                Vector result = new Vector();
                int i=0,
                        inicio = 0,
                        fim,
                        tamanho = local.length();
                String caminho = new String(local);
                while(i<tamanho){
                        fim = caminho.indexOf("\\");
                        result.add(new String(caminho.substring(inicio, fim)));
                        caminho = caminho.substring(fim+1,tamanho);
                        i = fim+1;
                        inicio = fim+1;
                }
                return result;
        }
        public Vector getUsuariosProximosViaRaio(Usuario usuario, float raio) throws
                                                                        RemoteException{
                Vector result = new Vector();
                Location localizacaoAlvo;
                try {
                        localizacaoAlvo = usuario.getLocalizacao();

                        for(int i = 0; i<objetosLocalizados.size(); i++){
                                Usuario user = (Usuario)objetosLocalizados.elementAt(i);
                                if(user.getID() != usuario.getID()){
                                        Location locationTeste = user.getLocalizacao();
                                        double dist =
                                mapa.getDistanciaAbsoluta(localizacaoAlvo.getHierarquico(),
                                                                locationTeste.getHierarquico());
                                        if (dist == 0.0){   // usuarios estao no mesmo local
                                                dist =
distanciaEuclidiana(localizacaoAlvo.getCoordenadas(), locationTeste.getCoordenadas());
                                        }
                                        if(!(raio>dist)){
                                                result.add(user);
                                        }
                                }
                        }
                } catch (RemoteException e) {
                        throw(e);
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return result;
        }
        protected double distanciaEuclidiana(double[] p1, double[] p2){
                double v1 = (p1[0]-p2[0])*(p1[0]-p2[0]);
                double v2 = (p1[1]-p2[1])*(p1[1]-p2[1]);
                return Math.sqrt(v1+v2);
        }
```

```
	public boolean registraUsuario(Usuario user, String login) throws RemoteException{
		boolean registrou;
		try {
			user.setID(registrador.getNovoID()); // usuario recebe id unico do servidor
			registrou = registrador.inclui(user, login);
			if (registrou){
				objetosLocalizados.addElement(user);
			}
		} catch (RemoteException e) {
			throw(e);
		}
		return registrou;
	}

	public Location buscarObjetoLocalizavel(String nome) throws RemoteException{
		Location result = null;
		int id = registrador.find(nome);
		try {
			for (int i=0; i<objetosLocalizados.size();i++){
				ObjetoLocalizavel ob =
					(ObjetoLocalizavel)objetosLocalizados.elementAt(i);
				if (ob.getID()==id){
					return ob.getLocalizacao();
				}
			}
		} catch (RemoteException e) {
			throw(e);
		}
		return result;
	}

	public void atualizaLocalizados(Usuario us, Location nova){
		int indice = -1;
		try {
			indice = this.indice(us);
		} catch (RemoteException e) {
			e.printStackTrace();
		}
		if (indice != -1) {
((UsuarioRemoto)objetosLocalizados.elementAt(indice)).atualizaLocalizacao(nova);
		}

	protected int indice (ObjetoLocalizavel ob) throws RemoteException{
		try {
			for (int i=0; i<objetosLocalizados.size();i++){
				if (((ObjetoLocalizavel)objetosLocalizados.elementAt(i)).getID() ==
				    ob.getID())
					return i;
			}
		} catch (RemoteException e) {
			throw(e);
		}
		return -1;
	}
public double distanciaEntreObjetos(ObjetoLocalizavel ob1, ObjetoLocalizavel ob2){
		try {
			Location l1 = ob1.getLocalizacao(),
				 l2 = ob2.getLocalizacao();
```

```
                        return mapa.getDistanciaAbsoluta(l1.getHierarquico(),
                                                                l2.getHierarquico());
                } catch (RemoteException e) {
                        e.printStackTrace();
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return 0;
        }
}

package servidor;
import java.rmi.RemoteException;
import java.util.Vector;
import cliente.Location;
import cliente.ObjetoLocalizavel;
import cliente.Usuario;
public class UsuarioRemoto extends java.rmi.server.UnicastRemoteObject implements Usuario{
        protected Location localizacao;
        protected Location[] ultimasLocalizacoes;
        protected Servidor serv;
        protected int ID;
        protected int circular;
        protected String login;
        protected static final int TAMANHO_LISTA = 10;
        public static final int TIPO_USUARIO = 1;

        public UsuarioRemoto(Servidor s) throws RemoteException{
                super();
                ID = -1;
                serv = s;
                ultimasLocalizacoes = new Location[TAMANHO_LISTA];
                circular = 0;
        }
        public int getTipo(){
                return TIPO_USUARIO;
        }
        public int getID(){
                return ID;
        }
        public Location localize(String nome) throws RemoteException {
                try {
                        return serv.buscarObjetoLocalizavel(nome);
                } catch (RemoteException e) {
                        throw(e);
                }
        }
        public Usuario[] quemEstaProximo(int nivelProximidade) throws RemoteException
        {       Vector result = new Vector();
                try {
                        result = serv.getUsuariosProximos(this, nivelProximidade);
                } catch (RemoteException e) {
                        throw(e);
                }
                return (Usuario[])result.toArray();
        }
        public Usuario[] quemEstaProximo(float raio) throws RemoteException{
                Vector result = new Vector();
```

```
                try {
                        result = serv.getUsuariosProximosViaRaio(this, raio);
                } catch (RemoteException e) {
                        throw(e);
                }
                return (Usuario[])result.toArray();
        }
        public void atualizaLocalizacao(Location l) {
                ultimasLocalizacoes[circular] = localizacao;
                circular++;
                circular = circular%TAMANHO_LISTA;
                localizacao = l;
        }
        public Location[] getUltimasLocalizacoes() {
                return ultimasLocalizacoes;
        }
        public Location getLocalizacao() {
                return localizacao;
        }
        public boolean regristreSe(String login) throws RemoteException {
                boolean registrou  = false;
                try {
                        registrou = serv.registraUsuario(this, login);
                } catch (RemoteException e) {
                        throw(e);
                }
                if (registrou){
                        this.login = login;
                }
                return registrou;
        }
        public int distanciaDe(ObjetoLocalizavel ob) {
                return (int) serv.distanciaEntreObjetos(this, ob);
        }
        public void setID(int id) {
                ID = id;
        }
}

import java.rmi.RemoteException;
import java.util.Vector;
import cliente.ObjetoLocalizavel;
public class Registrador {
        protected Vector colecao;
        int cont;
        public Registrador(){
                super();
                colecao = new Vector();
                cont = 0;
        }
        public boolean inclui(ObjetoLocalizavel ob, String nome){
                Registro r;
                try {
                        r = new Registro(ob.getID(), nome, ob.getTipo());
                        if (dadoNovo(ob.getID())){
                                if (find(nome)==-1){  // nao há usuario cadastrado com esse
                                                      // login
                                        colecao.addElement(r);
```

```
                                        cont++;
                                        return true;
                                }
                        }
                } catch (RemoteException e) {
                        e.printStackTrace();
                }
                return false;
        }
        public int getNovoID(){
                while(!dadoNovo(cont)){
                        cont++;
                }
                return cont;
        }
        public boolean exclui(ObjetoLocalizavel ob){
                try{
                        int posicao = find(ob.getID(), ob.getTipo());
                        if (posicao != -1){
                                colecao.removeElementAt(posicao);
                                return true;
                        }
                } catch (RemoteException e) {
                        e.printStackTrace();
                }
                return false;
        }

        protected boolean dadoNovo(int id){
                for (int i=0; i<colecao.size(); i++){
                        Registro r= (Registro)colecao.elementAt(i);
                        if (r.getIDObjeto() == id){
                                        return false;      // jah existe registro desses dados
                        }
                }
                return true;
        }
        protected int find(int id, int tipo){
                for (int i=0; i<colecao.size(); i++){
                        Registro r= (Registro)colecao.elementAt(i);
                        if (r.getIDObjeto() == id){
                                if (r.getTipo() == tipo){
                                        return i;
                                }
                        }
                }
                return -1;          // objeto nao encontrado
        }
        protected int find(String nome){
                for (int i=0; i<colecao.size(); i++){
                        Registro r = (Registro)colecao.elementAt(i);
// atencao: naum eh garantida a unicidade do atributo Nome entre os objetos registrados
                        if (nome.equals(r.getNome())){
                                return r.getIDObjeto();
                        }
                }
                return -1;          // objeto nao encontrado
        }
```

```
}

public class Registro {
	protected int IDObjeto;
	protected String nome;
	protected int tipo;

	public Registro(int objeto, String nome, int tipo) {
		super();
		IDObjeto = objeto;
		this.nome = nome;
		this.tipo = tipo;
	}
	public int getIDObjeto() {
		return IDObjeto;
	}
	public String getNome() {
		return nome;
	}
	public int getTipo() {
		return tipo;
	}
}

package servidor;
import cliente.Evento;
import cliente.EventoAtualizaLocalizacao;
import cliente.Location;
import cliente.Usuario;
public class GerenteEventos {
	protected Servidor serv;

	public GerenteEventos(Servidor s) {
		super();
		serv = s;
	}
	public void recebeEvento(Evento ev){
		atendeEvento(ev);
	}
	protected void atendeEvento(Evento ev){
		int tipo = ev.getTipo();
		switch (tipo){
			case Evento.ATUALIZA_LOCALIZACAO: {
				Usuario u = ((EventoAtualizaLocalizacao)ev).getUsuario();
				Location loc = ((EventoAtualizaLocalizacao)ev).getLocation();
				serv.atualizaLocalizados(u, loc);
			}
			default: System.out.println("Evento recebido desconhecido");
		}
	}
}

package servidor;
public interface Fabrica extends java.rmi.Remote{
	public void make(Class classe) throws Exception;
	public void addClasse(Class classe) throws Exception;

}
```

```
package servidor;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.util.Vector;
public class FabricaObjetos extends java.rmi.server.UnicastRemoteObject implements Fabrica {
	protected Vector classesConhecidas;
	public void addClasse(Class classe) throws Exception{
		boolean repetida = false;
		for (int i=0; i<classesConhecidas.size(); i++){
			Class temp = (Class)classesConhecidas.elementAt(i);
			if ((classe.getName()).equals(temp.getName())){
				repetida = true;
				break;
			}
		}
	   if (!repetida)
		classesConhecidas.addElement(classe);
	}
	public void make(Class classe) throws Exception  {
		String nome = classe.getName();
		System.out.println(nome);
		Class temp = null;
		Object ob = null;
		for (int i=0; i<classesConhecidas.size(); i++){
			temp = (Class)classesConhecidas.elementAt(i);
			if ((classe.getName()).equals(temp.getName())){
				break;
			}
		}
		try{
			ob = temp.newInstance();
			System.out.println("criada instancia da classe "+nome);
			String nomeServiçoRmi = "rmi://localhost:1099/" +
                                                ob.getClass().getName();
			Naming.rebind(nomeServiçoRmi, (java.rmi.Remote)ob);
			} catch (RemoteException e) {
				throw(e);
			} catch (MalformedURLException e) {
				throw(e);
			}
			catch (Exception ex){
				throw new Exception("Classe nao pode criar instancia!
                                        Classe nao conhecida: "+classe.getName());
}
	}
	public FabricaObjetos() throws java.rmi.RemoteException {
		super();
		classesConhecidas = new Vector();
	}
}

import java.io.*;
import java.util.Vector;
public class MapaMundo {
	protected Arvore arvore;
	public MapaMundo(String nomeArquivo) {
		super();
```

```
		arvore = new Arvore();
		try {
			leMapaDeArquivo(nomeArquivo);
		} catch (Exception e) {
			e.printStackTrace();
		}
}
public String getLocalDeSensor(String identificacaoSensor) throws Exception{
		try{
			String local = arvore.getCaminhoSensor(identificacaoSensor);
			return local;
		}catch (Exception e){
			throw e;
		}
}
protected void leMapaDeArquivo(String nomeArquivo) throws Exception{
		Vector nodos  = new Vector();
		String nome = null,
			nodoPai = null;
		Vector sensores = new Vector();
		double[] coordRela = new double[3];
		int contCoord = 0;
		Nodo atual = null;
		 try {
	    BufferedReader in = new BufferedReader(new FileReader(nomeArquivo));
	    String str;
	    while ((str = in.readLine()) != null) {
		str = str.trim();
		if((str.equals("<arvore>"))){
		}else{
		 if(str.equals("<nodo>")){
			atual = new Nodo();
		 }else{
			if(str.startsWith("<nome>")){
				int inicio = str.indexOf(">")+1;
				int fim  = str.indexOf("</nome>");
				atual.setNome(str.substring(inicio, fim));
			}else{
				if (str.startsWith("<nodoPai>")){
					int inicio = str.indexOf(">")+1;
					int fim  = str.indexOf("</nodoPai>");
					nodoPai = new String(str.substring(inicio, fim));
				}else{
					if(str.startsWith("<sensor>")){
						int inicio = str.indexOf(">")+1;
					int fim  = str.indexOf("</sensor>");
					String sensor = str.substring(inicio, fim);
					sensores.add(sensor);
					}else{
						if (str.startsWith("<coordRela>")){
							int inicio = str.indexOf(">")+1;
					int fim  = str.indexOf("</coordRela>");
					String alvo = str.substring(inicio, fim);
					alvo = alvo.trim();
					int c = Integer.parseInt(alvo);
					coordRela[contCoord] = c;
							contCoord++;
					}else{
```

```
                                                    if (str.equals("</nodo>")){
                                                            atual.setSensores(sensores);
                                                    atual.setCoordenadasRelativas(coordRela);
                                                            if (nodoPai.equals("")){
                                                                    nodos.addElement(atual);
                                                            }else{
                                                            for(int i=0; i<nodos.size();i++){
                                                                    Nodo teste =
                                                                    (Nodo)nodos.elementAt(i);
                                                    if(teste.getNome().equals(nodoPai)){
                                                            atual.setPai(teste);
                                                            teste.addFilho(atual);
                                                            nodos.addElement(atual);
                                                                    break;
                                                                            }
                                                                    }
                                                            }
                                                            sensores.clear();
                                                            contCoord = 0;
                                                    }else{

                                                            if(str.equals("</arvore>")){
                                                    arvore = new
                                                    Arvore((Nodo)nodos.firstElement());
                                                            }else
                                                                    throw new
                                            Exception("documento de mapa mal formado");
                                                            }
                                                    }
                                            }
                                    }
                            }
                        }

                    }
                    in.close();
                } catch (Exception e) {
                        throw e;
                }
        }
        public double getDistanciaAbsoluta(String l1, String l2) throws Exception{
                try {
                        return arvore.getDistanciaAbsoluta(l1, l2);
                } catch (Exception e) {
                        throw e;
                }
        }
}

package servidor;
import java.util.Vector;
public class Arvore {
        protected Nodo raiz;
        public Arvore() {
                super();
        }
        public Arvore(Nodo n){
```

```
		raiz = n;
	}
	public String getCaminhoSensor(String sensor) throws Exception{
		boolean achou = false;
		Nodo atual = raiz;
		Nodo certo = null;
		Vector visitar = new Vector();
		while (!achou){
			if (atual.temSensor(sensor)){
				achou = true;
				certo = atual;
			}
			else{
				Vector filhos = atual.getFilhos();
				for(int i=0; i<filhos.size(); i++){
					Object novo = filhos.elementAt(i);
					visitar.addElement(novo);
				}
				if(visitar.size() == 0){
					throw new Exception("Sensor ("+sensor+") nao
							encontrado no espaço conhecido");
				}
				atual = (Nodo)visitar.elementAt(0);
				visitar.removeElementAt(0);
			}
		}
		if (certo == null){
			throw new Exception("Sensor nao encontrado no espaço
					conhecido");
		}
		return certo.getNomeAbsoluto();
	}
	public double getDistanciaAbsoluta(String l1, String l2) throws Exception{
		try {
			Nodo n1 = getNodo(l1);
			Nodo n2 = getNodo(l2);
			double[] ca1 = n1.getCoordenadasAbsolutas();
			double[] ca2 = n2.getCoordenadasAbsolutas();
			return distanciaEuclidiana(ca1, ca2);
		} catch (Exception e) {
			throw e;
		}
	}
	protected double distanciaEuclidiana(double[] p1, double[] p2){
		double v1 = (p1[0]-p2[0])*(p1[0]-p2[0]);
		double v2 = (p1[1]-p2[1])*(p1[1]-p2[1]);
		return Math.sqrt(v1+v2);
	}
	protected Nodo getNodo(String nome) throws Exception{
		Vector nomes = new Vector();
		int i=0, nivel = 0;
		String nome2 = new String(nome);
		while(i<nome2.length()){
			int posicao = nome2.indexOf("\\");
			nomes.add(nome2.substring(0, posicao));
			nome2 = new String(nome2.substring(posicao+1, nome2.length()));
		}
		try{
```

```
                        if(!((String)(nomes.elementAt(0))).equals(raiz.getNome())){
                                throw new Exception("Nodo nao localizado pelo nome");
                        }
                }catch(Exception e){
                        throw new Exception("Endereco de nodo mal formado");
                }
                Nodo atual = raiz;
                nivel++;
                while(nivel<nomes.size()){
                        Vector filhos = atual.getFilhos();
                        boolean achou = false;
                        String n1 = (String)nomes.elementAt(nivel);
                        for(i=0; i<filhos.size();i++){
                                Nodo temp = (Nodo)filhos.elementAt(i);
                                String n2 = temp.getNome();
                                if(n2.equals(n1)){
                                        achou = true;
                                        atual = temp;
                                        break;
                                }
                        }
                        if(!achou){
                                throw new Exception("Nodo nao localizado pelo nome");
                        }//else
                        nivel++;
                }
                return atual;
        }
}

package servidor;
import java.util.Vector;
public class Nodo {
        protected Vector sensores;
        protected String nome;
        protected Nodo pai;
        protected Vector filhos;
        protected double[] coordenadasRelativas;

        public Nodo(String n, Nodo p) {
                super();
                nome = n;
                pai = p;
                filhos = new Vector();
                sensores = new Vector();
                coordenadasRelativas = new double[3];
        }
        public Nodo(String n){
                nome = n;
                filhos = new Vector();
                sensores = new Vector();
                coordenadasRelativas = new double[3];
        }
        public Nodo(){
                filhos = new Vector();
                sensores = new Vector();
                coordenadasRelativas = new double[3];
        }
```

```
public boolean temFilhos(){
	return (filhos.size() > 0);
}
public void addSensor(String s){
	sensores.addElement(s);
}
public void addFilho(Nodo n){
	filhos.addElement(n);
}
public boolean temPai(){
	return (pai != null);
}
public String getNomeAbsoluto(){
	String result = new String(nome);
	String barras = new String("\\");
	Nodo nodo = this;
	while (nodo.temPai()){
		result = barras.concat(result);
		nodo = nodo.getPai();
		String name = nodo.getNome();
		result = name.concat(result);
	}
	return result;
}
public double[] getCoordenadasAbsolutas(){
	Nodo temp = this;
	if(!temPai()){
		return coordenadasRelativas;
	}//else
	double[] result = coordenadasRelativas;
	while(temp.temPai()){
		temp = temp.getPai();
		double[] cr = temp.getCoordenadasRelativas();
		for(int i=0; i<result.length; i++){
			result[i] = cr[i]+result[i];
		}
	}
	return result;
}
public boolean temSensor(String s){
	boolean result = false;
	for(int i=0; i<sensores.size(); i++){
		String sensor = (String)sensores.elementAt(i);
		if (sensor.equals(s)){
			return true;
		}
	}
	return result;
}
public Vector getFilhos() {
	return filhos;
}
public String getNome() {
	return nome;
}
public Nodo getPai() {
	return pai;
}
```

```
	public double[] getCoordenadasRelativas() {
		return coordenadasRelativas;
	}
	public void setCoordenadasRelativas(double[] coordenadasRelativas) {
		this.coordenadasRelativas = coordenadasRelativas;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public void setPai(Nodo pai) {
		this.pai = pai;
	}
	public void setSensores(Vector s) {
		for(int i=0; i<s.size();i++){
			sensores.add(s.elementAt(i));
		}
	}
}
****************************************
```

GeraStubs.bat:

```
start E:\j2sdk1.4.2_03\bin\rmiregistry
E:\j2sdk1.4.2_03\bin\rmic servidor.UsuarioRemoto
E:\j2sdk1.4.2_03\bin\rmic servidor.FabricaObjetos

****************************************
```

Exemplo de arquivo válido de Mapa de Mundo:

```
<arvore>
<nodo>
	<nome> UFSC </nome>
	<nodoPai></nodoPai>
	<sensor> S1 </sensor>
	<sensor> S2 </sensor>
	<coordRela> 0.0 </coordRela>
	<coordRela> 0.0 </coordRela>
	<coordRela> 0.0 </coordRela>
</nodo>
<nodo>
	<nome> Reitoria </nome>
	<nodoPai> UFSC </nodoPai>
	<sensor> S3 </sensor>
	<sensor> S4 </sensor>
	<coordRela> 50.0 </coordRela>
	<coordRela> 50.0 </coordRela>
	<coordRela> 0.0 </coordRela>
</nodo>
<nodo>
	<nome> Gabinete Reitor </nome>
	<nodoPai> Reitoria </nodoPai>
	<sensor> S5 </sensor>
	<coordRela> 20.0 </coordRela>
	<coordRela> 10.0 </coordRela>
	<coordRela> 5.0 </coordRela>
</nodo>
</arvore>
```