LBS com J2ME e a sua API Location

Elias Franco Lopes, Jaime Gomes Nogueira, Thiego George Nacif, Thiene M. Johnson

Centro de Ciências Exatas e Tecnológicas – Universidade da Amazônia (UNAMA)
Av. Alcindo Cacela – Belém – PA – Brazil

{liclopes, snoopirata, thiegonacif}@gmail.com, thienne@ieee.org

Resumo. *O surgimento de uma nova geração de dispositivos móveis, assim como a nanotecnologia e a acessibilidade dos LBS de elevada precisão, aliada à maior facilidade de programação dos dispositivos, abre as portas para uma nova geração de aplicações interativas, incluindo serviços como geocodificação e geocodificação reversa, serviços de planejamento de rotas e serviços de geração e visualização de mapas. Este artigo mostra as tecnologias e serviços utilizados para a implementação de um LBS e é mostrada a aplicação na qual desenvolvemos este serviço e suas tecnologias utilizadas em um contexto de geolocalização, e tendo como objetivo a maior interação sobre o assunto no mundo atual, incentivando assim, a criação e melhoramentos destes serviços.*

1. Introdução

Sistemas ou serviços baseados em localização (Location-Based Services – LBS), podem ser vistos como a convergência de várias novas tecnologias, tais como: sistemas de comunicação móvel, tecnologias de localização, dispositivos móveis (DM) com a Internet, sistemas de Informação Geográfica (SIGs), servidores da aplicação e banco de dados espaciais (SHIODE, 2002). O ponto central desses sistemas está em obter informações para determinar a localização de dispositivos móveis (DM) e, baseado nessa informação, oferecer serviços de acordo com o contexto de utilização do sistema.

Temos como objetivo, através deste artigo, mostrar a API Location da tecnologia J2ME, utilizada para implementação de um sistema LBS com suporte a GPS, para gerar serviços de geolocalização, mostrando seu funcionamento, suas principais classes e especificações para a implementação de uma aplicação básica.

Este artigo divide-se em 5 partes. A seção 1 é a introdução deste trabalho. A seção 2 detalha a arquitetura de um LBS, mostrando em outros itens suas tecnologias. A seção 3 especifica a tecnologia JAVA utilizada para a implementação básica de um LBS, analisando trechos específicos. A seção 4 trata sobre o cenário atual da tecnologia nop Brasil. Por fim, a seção 4 mostra a conclusão deste nosso trabalho.

2. Arquitetura de Sistemas Baseados na Localização

Para desenvolver LBS, deve existir uma estrutura básica de componentes que dêem suporte a determinação da localização (i.e., *software* e *hardware*, e.g., *Position-Determining Equipment*), solicitações de consultas, processamento e gerenciamento da informação

geográfica e contextual e serviços de *gateway* (i.e., *midleware* entre os serviços de processamento e os DM para promover compatibilidade).

2.1 Tecnologias de localização de dispositivos móveis

As tecnologias utilizadas para a localização podem ser classificadas quanto ao local onde as coordenadas de posicionamento são coletadas e calculadas. São essas: soluções baseadas em rede (*network-based*), baseadas no dispositivo (*handset-based*) e soluções híbridas que utilizam tanto a rede quanto o dispositivo para processarem os cálculos.

As *network-based* se caracterizam pela presença de um equipamento de localização colocado nas estações rádio-base (ERB) e também pelo processamento necessário para o LBS, sendo feito na infra-estrutura da própria rede. Por sua vez, as *handset-based* implementam grande parte da tecnologia nos DM e tem como base o uso da tecnologia GPS. Em ambientes fechados deverá ser utilizada uma das técnicas baseadas em sensores.

3. Usando J2ME, Location API e a implementação do LBS

A mais recente edição JAVA é J2ME (JAVA 2 Micro Edition). Esta tecnologia é direcionada ao mercado de pequenos dispositivos. Seu conjunto de especificações tem por objetivo disponibilizar uma JVM (Java Virtual Machine), APIs e ferramentas para equipamentos como: telefones celulares, pagers, handhelds, vídeo-games, sistemas embutidos, etc. Atualmente já existe a KVM ('K' Virtual Machine) que é uma máquina virtual que oferece suporte ao J2ME em micro-controladores de 16 ou 32 bits. Voltada à CLDC (Connected Limited Device Configuration) e CDC (Connected Device Configuration), a KVM possui apenas 40K de código e precisa de poucos kilobytes para a sua execução pois ela não implementa a especificação da JVM e, sim, parte dela.

A julgar pela quantidade de pessoas que utilizam telefones celulares em vários lugares públicos, é certo afirmar que a tecnologia sem fio já tem seu lugar no *mainstream* (SUN MICROSYSTEMS) demostrando como as aplicações LBS logo serão destaques no mundo sem fio, a J2ME é ideal para tornar-se um padrão para o desenvolvimento desse tipo de aplicação, pois fornece um conjunto de classes e funcionalidades voltadas para esse fim (HAIGES 2005). Direcionada para a implementação de LBS, foi desenvolvida a API *Location* (*Java Specification Request* #179 - JSR-179) para J2ME.

Esta seção tem por objetivo mostrar o uso da API *Location* no suporte ao desenvolvimento de sistema LBS. O sistema permite que um DM, equipado com GPS, tenha acesso, via um servidor de bando de dados, à localização de serviços e locais próximos a sua localização atual. A figura 1 exemplifica o funcionamento do sistema.

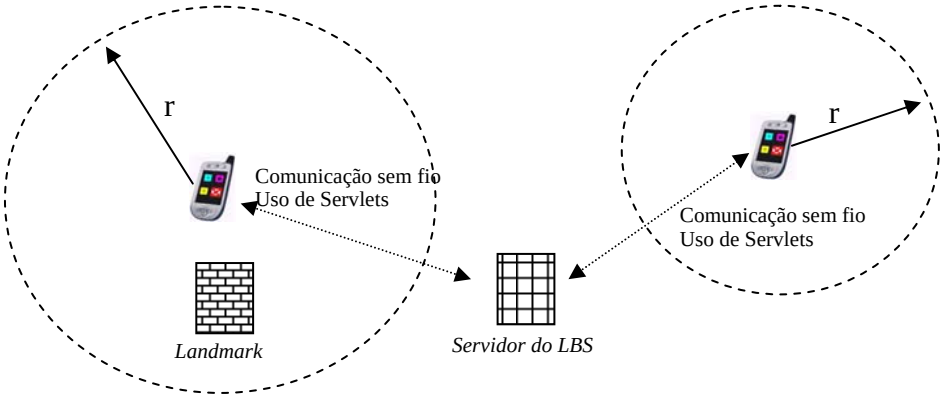

Figura 1: Exemplo do uso do sistema LBS

Na figura, tem-se 2 DM, configurados com diferentes raios de sensibilidade (r) aos *landmarks* (POI previamente cadastrados no sistema). No momento que o raio (configurado no perfil do cliente) do dispositivo alcança o *landkmark*, o sistema avisa ao DM da localização e os serviços disponíveis.

3.1. Especificação da Location API

Essa API fornece uma interface padrão de alto nível para acesso a informações de posição de um DM independente do método de localização utilizado. Esta API é opcional, portanto, não precisa necessariamente estar presente no pacote padrão de APIs encontrados nos dispositivos complacentes com a plataforma J2ME (HAIGES 2005).

Ela contém as classes básicas necessárias para solicitar e obter uma localização geográfica e integra dados genéricos de posição e de orientação com armazenamento persistente de ponto dos objetos de interesse (POI), ou marcos, e conexão via HTTP com *servlets*, que permitem programadores de implementar aplicações e serviços sem fio baseados em localização para dispositivos de recurso limitados como DM (CDC e CLDC) (VENTURELLA 2006),

Essa API permite a implementação com qualquer método comum de localização de forma transparente ao usuário, isto é, sem deixar a amostra os procedimentos e cálculos complexos dos métodos de localização, exibindo apenas os resultados requeridos.

3.2. O Modelo de Objeto do Pacote javax.microedition.location

O pacote *Location* (*javax.microedition.location*) é sub-dividido da seguinte forma: em 6 classes principais e em duas relações de ouvinte de eventos ou classes de interface (*LocationListener* e *ProximityListener*). Das seis classes, duas são classes da exceção (*LocationException* e *LandmarkException*) e outros quatro (*AddressInfo, Criteria, Orientation*, e *QualifiedCoordinates*) são primeiramente objetos do valor.

A classe *LocationProvider* é o ponto inicial para as aplicações usando esta API e representa o objeto de recuperação da posição do DM, usando os método existente de localização (JAVA COMMUNITY 2006). Por via desta, a aplicação cria objetos *Location* (objeto padrão que representa a localização básica da posição), obtendo assim a localização do DM no momento que desejar.

O objeto *Location* contêm objetos *QualifiedCoordinates*, que representam as coordenadas geográficas que estão associados com valores de precisão, hora, velocidade e curso do DM. As classes de exceções ocorrem quando há erros relacionados à integridade dos marcos (*LandmarkException*) ou quando ocorrem erros específicos da API da posição (*LocationException*).

Com o uso do *LocationListener*, a aplicação pode pedir um único objeto de posição ou periodicamente ser atualizada com os objetos novos de posição de acordo com a disponibilidade do provedor de localização (receptor GPS). Já o *ProximityListener*, trabalha com eventos associados a detectar a proximidade (raio de sensibilidade) a algumas coordenadas geográficas (valores de latitude, longitude, altitude) registradas através do *LandmarkStore.*

3.3. Usando as Classes Location e Proximity Listeners

Usando a relação de *LocationListener*, você pode escutar atualizações da sua posição atual em intervalos de tempo pré-determinados, pode expor e processar informação em um contexto a ser alcançado por outros componentes na demanda.Uma vez executada a relação de *LocationListener*, ele pode ser registrado com o *LocationProvider* através do método do setLocationListener, para assim poder obter a posição atual. Usando critérios (disponíveis na classe *Criteria*) para refinar a ação do sistema.

As escolhas que você faz em seus critérios e o projeto da aplicação pode afetar: vida da bateria do DM, latência na aplicação, adequação da precisão (horizontal e vertical), determinação manualmente ou não do método a ser usado para recuperar a posição (e.g. GPS indoor) e o custo (WICK e LYON 2006). Como este é potencialmente um processo longo, este é executado paralelamente em uma *thread* para deixar a interface livre para o processo de execução de outras funções.

```
public class LocationListenerMIDlet extends MIDlet implements CommandListener, LocationListener(){
Criteria cr= new Criteria();
cr.setHorizontalAccuracy(100);
cr.setVerticalAccuracy(100);
cr.setPreferredPowerConsumption(Criteria.POWER_USAGE_LOW);
thread tGetLocation = new thread(){
        public void run(){
            try{
                LocationProvider lp=LocationProvider.getInstance(cr);
                Location loc=lp.getLocation(60);    }
}
locationProvider.setLocationListener(this, 20, 10, 10);
```

A relação do ouvinte contém dois métodos: o *locationUpdated* e *providerStateChanged*. O *LocationProvider* provoca o método *locationUpdated* sempre que a posição mudar. Já o método *providerStateChanged* é informado quando o estado do *LocationProvider* muda entre seus três valores possíveis: AVAILABLE, TEMPORARILY_UNAVAILABLE, e OUT_OF_SERVICE (PARSONS 2005). No exemplo abaixo, somente locationUpdated é executado, chamando o método que atualiza o formulário da tela.

```
public void locationUpdated(LocationProvider provider, Location location){
    this.locationInfo(); }
```

Ao contrário do *LocationListener*, que pode ter somente um registro, o *LocationProvider* permite registrar *ProximityListeners* múltiplo. O *ProximityListener* adiciona a proximidade do *landmark*. Assim ao começo da execução, são registradas três posições de modo que você possa aguardar até escutar eventos de proximidade para três posições diferentes. Cada registro requer o ouvinte, as coordenadas, e o raio de proximidade destas coordenadas como parâmetros.

```
public class ProximityListenerMIDlet extends MIDlet implements CommandListener, LocationListener,
ProximityListener(){
  try{
        locationProvider = LocationProvider.getInstance(null);
        locationProvider.setLocationListener(this, 20, 10, 10);
        float raio = 100.0F;
        locationProvider.addProximityListener(this, UNAMA-AlcCoordinates, raio);
  }
```

A relação de *ProximityListener* têm dois métodos: o *monitoringStateChanged* e o *proximityEvent*. O método *monitoringStateChanged* informa ouvintes se a notificação da proximidade é atualmente ativa. O método *proximityEvent* fornece duas coordenadas dos parâmetros (PARSONS 2005). Nesta execução, compara-se às coordenadas dadas com as que se aguarda quando há modificação e seleciona-se a posição combinada.

```
public void proximityEvent(Coordinates coordinates, Location location){
        if(coordinates.equals(studyCentreCoordinates)){
        showUNAMA-AlcProximityAlert();}
}
```

3.4. A Coordinates e Outras Informações

Os objetos da posição são agregados imutáveis de objetos de *AddressInfo* e de *QualifiedCoordinates*, e os exemplos de posição são adquiridos da classe *LocationProvider* (parametrizado por um objeto *Criteria*). Uma vez que você tem um objeto de posição, você pode usá-lo encontrar as informações atuais através do objeto *QualifiedCoordinates* que é agregado a ele.

```
location = locationProvider.getLocation(20);
coordinates = location.getQualifiedCoordinates();
```

Uma posição pode também conter um objeto de *AddressInfo*, com detalhes tais como o endereço postal da posição, o número de telefone, o país, o URL, e o gosto. Dependendo da execução e do contexto, entretanto, não pode realmente haver um *AddressInfo* associado com uma posição dada. Por exemplo, se você estiver no meio de um campo, não haverá nenhum dados de *AddressInfo*. Na execução, o *AddressInfo* é sempre nulo.

A direção, a velocidade, e a altura atuais são retornados do objeto de *QualifiedCoordinates* como valores reais, quando a latitude e a longitude são retornadas como *double*. Para ajudar na visualização dos dados, a classe *Coordinates* inclui um método de converção de um objeto de coordenadas em uma String (palavra) em um dos

dois formatos: a) Graus, Minutos, e frações decimais de minuto (em doublé, 61.51d e String, 61:30:36) ou b) Graus, Minutos, Segundos, e frações decimais de segundo (em doublé, 61.51d e String, 61:30.6). Abaixo, uma implementação onde a informação da posição é exibido na tela sendo usado o formato "b)", usando o campo constante Coordinates.DD_MM_SS. Alternativamente, o formato "a)" pode ser selecionado usando Coordinates.DD_MM.

```
if((qc != null) || (location.isValid())) {
        lat = new StringItem("Latitude: " + Coordinates.convert
        (coordinates.getLatitude(), Coordinates.DD_MM_SS), "");
        long = new StringItem("Longitude: " +  Coordinates.convert (coordinates.getLongitude(),
        Coordinates.DD_MM_SS), "");
        …
}
```

A classe das coordenadas (super classe de *QualifiedCoordinates*) encapsulam métodos geométricos tais como o cálculo do azimuth (ângulo) e da distância entre posições. A classe *Orientation* é completamente separada do resto do modelo do objeto, não tendo nenhum relacionamento de associação ou de dependência com quaisquer outras classes. Isto é possível porque nem todos os dispositivos podem suportar a informação de orientação(PARSONS 2005). No mínimo, o dispositivo deve fornece um valor do angulo do compasso aos objetos de orientação da sustentação, com o suporte opcional para valores de passo e de rolo.

Pode-se ajustar para recuperar os dados formatados (NMEA, LIF), a descrição de erro de forma detalhada com uso do getExtraInfo (WICK e LYON 2006), e o método intuitivo utilizado para a localização. O valor retornado é bitwise uma combinação (OU) da tecnologia do método, tipo do método e informação do auxílio.

```
Location.getExtraInfo(String mimetype)
getExtraInfo("application/X-jsr179-locationnmea")
getExtraInfo("text/plain")
metodo = location.getLocationMethod();
```

3.5. Usando as Classes Landmarks e the LandmarkStore

O que distingue particularmente esta API de outras bibliotecas LBS é o uso do armazenamento local nos DM, sendo uma base de dados persistente dos POI (*Points of Interests*: marcos - *landmarks*). Isto desloca a ênfase no cliente móvel nos termos de aplicações posição-clientes, permitindo mapeamento locais das posições já conhecidas. Uma vantagem chave desta é que as aplicações são mais prováveis ter a funcionalidade útil, mesmo onde a conectividade da rede é ruim (PARSONS 2005).

O *LandmarkStore* é uma coleção dos POI, podendo haver muitos *LandmarkStores* em um DM compartilhado por aplicações múltiplas. Os POI podem, opcionalmente, serem armazenados em lojas múltiplas e às categorias múltiplas. A única limitação é que um POI não pode ser adicionado várias vezes à mesma categoria no mesmo *LandmarkStore*. Uma característica da API é que um objeto POI pode ser dados de coordenada e de endereço de uma posição gerada pelo *LocationProvider* (o *LocationProvider* pode incluir um *AddressInfo*

junto com o *QualifiedCoordinates*). Isto significa que os POI podem ser adicionados dinamicamente ao *LandmarkStore*.

A seguir, é usado um *ProximityListener* para indicar alertas ao se aproximar de determinadas coordenadas. A primeira etapa é iniciar uma instância do *memso* (o parâmetro nulo é o padrão). A seguir são criados os objetos POI no início e armazenados ao *LandmarkStore*, adicionado usando um nome de categoria, nome do local, descrição, conjunto das coordenadas, e o *AddressInfo* do local.

```
landmarkStore = LandmarkStore.getInstance(null)
Landmark landmark1 = new Landmark ("UNAMA Alc. Cacela","Um dos campi da UNAMA – Universidade
da Amazônia", UNAMACoordinates, info);
landmarkStore.addCategory("campus");
landmarkStore.addLandmark(landmark1, "campus");
```

Mais tarde é feita a recuperação do POI necessário usando o método *getLandmarks*, passando os nomes da categoria e o POI como parâmetros:

```
Enumeration e = landmarkStore.getLandmarks("campus", " UNAMA Alc. Cacela");
Landmark landmark = (Landmark)e.nextElement();
AddressInfo info = landmark.getAddressInfo();
```

3.6. Comunicação com o Servlet

Uma outra forma de fornecer uma base de dados é o uso de *Servlets* que trabalham com banco de dados. Depois de já obtida a informação da posição atual, nós podemos submetê-la ao *servlet* sobre uma conexão do HTTP padrão (definição de uma URL), no caso usando para solicitação o método Get.

```
public void PesquisarGet() throws IOException {
 …
       thread tConnection = new thread(){
           public void run(){
                hc = (HttpConnection) Connector.open(url);
                hc.setRequestMethod(HttpConnection.GET); }
       };
       try{
             AlertType.INFO.playSound(m_Display);
             tConnection.start();
             tConnection.join();
       … }
}
```

Como podemos ver, o MIDLet faz o trabalho de recuperar a posição atual do DM e de entregá-la ao usuário. Em uma aplicação típica tem-se um *servlet* que aceita os pedidos e manipula essas informações de localização. A parte chave é o método do *doGet()*, que faz o trabalho de analisar as informações repassadas do DM, realizar a consulta via uma seqüência *SQL* e então retornar as informações de resposta.

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
       String latitude = request.getParameter("latitude"), longitude = request.getParameter("longitude"),
localidade = pesquisaLocal(latitude, longitude);
        public String pesquisaLocal(String latitude, String longitude){
                // Obejtos de conexão
                …
                String sql = “Select LOCALIDADE From DADOS_LOCALIDADE Where latitude = " +
                latitude - latRadius + “ BETWEEN ” +    latitude - latRadius  + " And longitude = " +
                longitude – lonRadius + “ BETWEEN ” + longitude + lonRadius + ”;”
                try{
                        Class.forName("…");
                …
                ResultSet rs = stm.executeQuery(sql);
        …        }
}
```

4. Atualidade - A VIVO Sai na Frente

Dentre as operadoras brasileiras a VIVO é a pioneira no uso de serviços LBS, como o serviço VIVO Localiza. O serviço utiliza tanto o recurso handset-based com o A-GPS (Assistted GPS) quanto network-based com a triangulação de ERBs. Normalmente ao se estar outdoor é utilizado o A-GPS e indoor a triangulação.

Via menu de download de DM da operadora você contrata o serviço que dá direito a um determinado número de localizações. Com o SW já descarregado para o DM, ele captura as informações geo-localizacionais do GPS interno do DM e os envia para um servidor que converte as informações para que sejam plotadas em um mapa geo-referenciado. Erro em média de 5 a 50m.

5.Conclusões

Os LBS possuem características que resultam em questões que não são comuns nos sistemas de informação tradicionais, sendo um desafio complexo do ponto de vista tecnológico e comercial. A heterogeneidade e dispersão das fontes de informação e a associação destas as localizações físicas, assim como a condição de mobilidade dos utilizadores. Aspectos que necessitam estar integrados e interagindo uniformemente.

Este artigo mostra o funcionamento de uma nova tecnologia J2ME em um contexto de geo-localização, onde o sistema implementado tem por finalidade, através desta tecnologia e sua API, simular um serviço LBS, cujo qual adquire e retorna valores localizacionais de um usuário qualquer que possua um dispositivo GPS em mãos.

Com a implementação deste sistema, tenta-se chamar as atenções de operadoras telefônicas, usuários de telefonia móvel, provedores de serviços e outros para que haja um incentivo e divulgação ainda maior no contexto de LBS, assim como o interesse de criação de novos sistemas genuinamente brasileiros, para assim possuirmos um desenvolvimento tecnológico em um segmento altamente lucrativo, devido a alta agregação de valor aos serviços onde a localização nas proximidades bem como a sua descrição são os principais componentes.

Referências

Haiges, Syen (2003) “The Location API”, Disponível em: http://java.sys_con.com, Agosto.
Process, Java Community (2006) “Location API for Java™ 2 Micro Edition Version 1.0.1”, Disponível em
http://www.forum.nokia.com/main/resources/technologies/java/documentation/java_jsr. Agosto.
Wick, Ryan e Lyon, Zane (2006) “Practical Applications of JSR 179”, Disponível em: http://developers.sun.com/learning/, Agosto.
Parsons, David (2005) “The Java Location API - Knowing where you are is half the battle”, Disponível em: http://www.ddj.com/dept/java/184406388, Setembro.
Venturella, Adam (2006) “J2ME - Location Based Data”, Disponível em: http://www.google.com/notebook/public/04105482259393985038/BDUYhIgoQtqSQzbsh, Setembro.