MONITOR E PREDITOR DE CONECTIVIDADE WIRELESS BASEADA EM LOCALIZAÇÃO GPS

Aluna: Eleonora Cominato Weiner
Orientador: Markus Endler

Introdução

A palavra mobilidade ganha mais importância a cada instante, à medida que tanto usuários comuns e empresas sentem a necessidade de incorporar mais funcionalidades em seus cotidianos para facilitar tarefas, agilizar negócios e até mesmo recursos divertidos.

Smartphones modernos possuem interfaces de rede para mais de uma tecnologia wireless (WiFi, EDGE, 3G e, futuramente, WiMAx), conseguindo mudar automaticamente de uma para outra com o mesmo endereço IP. No entanto, cada tipo de conectividade tem parâmetros de qualidade e custos diferentes, logo é importante que as aplicações consigam saber qual é a conexão wireless sendo usada e se antecipar a uma possível mudança.

Cada local da cidade está coberto por pelo menos uma tecnologia wireless de certa qualidade e, se houver registro das tecnologias wireless por localização e algum tipo de indicação da direção em que o aparelho está se movendo, então a aplicação poderia se antecipar ao novo tipo de conexão wireless que encontrará na nova posição.

Tal aplicação pode tornar-se bastante útil como componente de outras aplicações mais robustas que exijam conexão com a Internet em todo o seu ciclo de vida a serem desenvolvidas no futuro. Deste modo, a aplicação delegar a função, podendo conter apenas a lógica de negócios que seus criadores idealizarem.

Para tal desenvolvimento, foi escolhida a plataforma Android, que vem adquirindo aceitação cada vez maior, tanto de usuários como de programadores, pela sua facilidade de uso, pela grande amplitude de recursos oferecidos e pelo fato de ser um software completamente open-source, parte da Open Handset Alliance. Baseada em um sistema operacional Linux, possui um ambiente de desenvolvimento poderoso, ousado e flexível, uma resposta do Google para ocupar este espaço do mercado móvel.

Objetivo

Implementar um serviço cliente-servidor – que será o monitor – e um aplicativo – que será o preditor – para smartphones com GPS executando a plataforma Android, em que o registro do par (conectividade/coordenadas GPS) seja feito a cada vez que o cliente detectar que houve mudança no status da conexão, no tipo da rede wireless e/ou mudança significativa na qualidade da conectividade.

Implementação

A. Servidor WebService

Um web service é uma das tecnologias mais utilizadas para integrar aplicações. O Android não possui nenhuma API nativa para acessar um web service, mas podemos escolher uma biblioteca leve e compacta e incorporá-la ao projeto. Aqui escolhemos o framework ksoap2[3],

especialmente construída para dispositivos móveis com menos capacidade de processamento. Escolhemos também, para a implementação do web service, o framework Xfire[4].

Em resumo, o servidor é quem receberá as informações de conexão de vários clientes móveis e, tendo uma divisão pré-determinada de uma região maior como uma cidade ou um bairro, fará o cálculo de a qual divisão a informação recebida pertence e registrará no SDM[5].

Eventualmente, todos os tipos de conectividade wireless encontrados em cada região estarão cadastrados, porém, as informações deverão desatualizar-se, isto é, perder a sua validade após um certo tempo, devendo ser novamente coletadas – para tal, publica-se a informação juntamente com a data e hora de coleta e estipula-se a sua validade em três meses.

Para tal, foram criadas uma interface e uma classe que a implementa para este servidor agir como um serviço, chamadas NetworkMonitorServer e NetworkMonitorServerImpl. A interface possui as seguintes funções:

- *public Point findSquare (Location loc)*

Point é uma classe que extende a classe GeoPoint do Android criada para auxiliar na manutenção de coordenadas GPS. Esta função retorna a coordenada que define o quadrado ou divisão da região considerada pelo aplicativo que contém a localização recebida. Segundo a definição utilizada, cada um desses quadrados é caracterizado na base de dados pela sua coordenada GPS central.

- *public boolean publishNetworkInfo (MyNetworkInfo info)*

MyNetworkInfo é uma classe criada para reunir todos os atributos que queremos de informação sobre a rede wireless em único objeto, de forma a facilitar a sua manutenção. Esta função publica o objeto recebido na base de dados e retorna se obteve sucesso (true - 1) ou não (false - 0).

- *public boolean consultLocation (Location loc)*

Esta função consulta se há informações sobre conexões no local onde o usuário se encontra na base de dados e retorna se há (true - 1) ou não (false - 0) conexões disponíveis.

- *public MyNetworkInfo[] getNetworkList (Location loc)*

Esta função retorna uma lista de objetos MyNetworkInfo disponíveis no local recebido como parâmetro, que poderá ser o local onde o usuário se encontra ou o local para onde ele pretende ir.

- *public boolean updateNetworkInfo (MyNetworkInfo info)*

Esta função consulta se o objeto MyNetworkInfo recebido se encontra na base de dados. Se sim, consulta a data e verifica sua validade (não deve ser maior que 90 dias). Se necessário, a atualiza com o objeto recebido. Se o objeto não existir na base dados, a função chama a *publishNetworkInfo*. Ela retorna se foi bem sucedida (true - 1) ou não (false - 0).

Estas funções são implementadas na classe NetworkMonitorServerImpl. Como o servidor é um serviço, este não implementa nenhuma activity ou tela e, portanto, está em atividade apenas em segundo plano no smartphone.

B. Cliente

O cliente é implementado como um serviço Android que periodicamente obtém os parâmetros status da conexão, coordenadas GPS e qualidade do sinal da conectividade corrente através de listeners. Para este último, apenas no caso de uma variação grande (melhora ou queda de pelo menos 30% da qualidade), é que o aplicativo lerá as coordenadas GPS daquele momento e registrará o novo par (conectividade/ coordenadas GPS) na base de dados.

Os valores de qualidade da conexão (Boa/Média/Ruim) serão definidos especificamente para cada tecnologia wireless, por exemplo, em termos de nível de sinal para WiFi (convertido a

partir do sinal RSSI) e intensidade de sinal em dBm para redes de celular como 3G, EDGE e CDMA, por exemplo.

A classe NetworkMonitorClient encapsula todo o acesso ao servidor. Para fazer uma requisição SOAP é preciso definir qual método será consultado e adicionar os argumentos ou parâmetros necessários ao objeto SOAP, cujos nomes (método e variáveis) precisam ter o mesmo nome e, por isso, precisamos de um método cliente para cada método do servidor que quisermos utilizar.

Ela possui os seguintes métodos:

- *public Point findSquare (Location loc)*
- *public boolean publishNetworkInfo (MyNetworkInfo info)*
- *public boolean consultLocation (Location loc)*
- *public MyNetworkInfo[] getNetworkList (Location loc)*
- *public boolean updateNetworkInfo (MyNetworkInfo info)*
- *public void listenerMethod ()*

Esta função implementa os listeners da aplicação: monitora o local do aparelho (LISTEN_CELL_LOCATION), o status da conexão (LISTEN_SERVICE_STATE) e a intensidade do sinal (LISTEN_SIGNAL_STRENGTHS) e, para cada evento, coleta as informações e decide o que fazer com elas – se apenas consultar a data em que aquelas conexões forma submetidas por último e atualizá-las se necessário ou publicá-las. Utilizamos um alarme para que este serviço não utilize memória e recursos demais e este é implementado no método a seguir.

- *public void scheduleListenerMethod ()*

Como mencionado acima, esta função é um uso da classe AlarmManager para que este serviço não desperdice os recursos do aparelho. Por enquanto não teremos a opção do usuário escolher este intervalo de tempo, que é definido para a cada 90 segundos.

Como o cliente também é um serviço, este não implementa nenhuma activity ou tela e, portanto, está em atividade apenas em segundo plano no smartphone.

C. Aplicativo Preditor

O preditor, por sua vez, é um aplicativo através do qual o usuário poderá obter quais são as conectividades disponíveis na região em que se encontra (através de uma integração com o GoogleMaps), bem como nas regiões imediatamente adjacentes à essa região. Este aplicativo obterá essas informações do servidor como resposta a uma consulta enviada via HTTP por meio das funções disponibilizadas pela classe NetworkMonitorClient. Posteriormente, esta consulta poderá conter até mesmo um perfil do usuário.

A integração com o GoogleMaps se dará mostrando, nos arredores da localização do usuário, esta área dividida em quadrados de tamanho fixo como na figura acima em que teremos uma espécie de véu por cima da imagem colorindo os quadrados de verde – quando houver pelo menos uma conexão de boa qualidade –, de amarelo – quando houver apenas conexões de média qualidade –, de vermelho – quando houver apenas conexões de qualidade ruim – e de cinza, quando não houver dados disponíveis.

As cores por cima do mapa farão proveito de uma funcionalidade do Android chamada alpha, que define a transparência das imagens. Para isso, usaremos uma subclasse de Overlay, uma das APIs do Google: desenhamos vários quadrados e, com a ajuda dos métodos do serviço cliente desta aplicação, determinamos onde há conexões disponíveis e suas qualidades e colorimos os quadrados de acordo.

Os quadrados estão definidos para 10 metros de lado, ou seja, quadrados de 100 metros quadrados. Esta medida foi escolhida, pois, apesar de em 100 m^2 ser possível existir um grande número de redes disponíveis, foi uma boa medida para não prejudicar o armazenamento e a consulta de todos estes dados no servidor SDM, uma vez que este processamento deve ocorrer rapidamente.

Não entraremos em muitos detalhes sobre o desenho dos quadrados na região em que o usuário se encontra por este não ser o foco do trabalho. Além da integração com o Google Maps esta é a parte do projeto que entra em contato com o usuário, com activitys como menus e map activitys onde veremos o mapeamento das redes.

Através de um menu, poderemos, por exemplo, escolher entre ver o mapa com as funcionalidades acima ou ver as conexões disponíveis, tanto no seu aparelho como na base de dados. Ao entrar no mapa e clicar em um determinado quadrado, o usuário será levado para uma tela com as conexões disponíveis para aquele quadrado, mesmo que este não seja o que ele se encontra. Ao escolher ver as conexões disponíveis, o usuário poderá ainda escolher entre o local em que está ou algum outro local estipulado por ele.

Caso a aplicação necessite mudar de rede wireless, tal evento será notificado e aparecerá na barra de notificações. Por enquanto, esta funcionalidade se encontra no aplicativo e não no serviço cliente para evitar usuários insatisfeitos, pois ainda não há um menu de configurações disponível onde este poderá ligar ou desligar esta função. Logo, faz mais sentido que isto não fique executando em segundo plano junto com o restante do processamento necessário para que esta aplicação funcione.

D. Outros

Mencionamos as classes criadas para este projeto fora os serviços e o aplicativo e daremos mais detalhes agora.

A classe MyNetworkInfo inclui dados como a data de criação do objeto, o local a partir do qual o aparelho coletou as informações, o nome da rede, o tipo da rede (WiFi ou móvel), intensidade do sinal em dBm. O nome da rede para a rede móvel é o nome da operadora e o nome do provedor do serviço e o subtipo da rede são informações somente para redes móveis e devem possuir valores inválidos para as redes WiFi. Por sua vez, o endereço IP e o Access Point são informações somente para redes WiFi e devem possuir valores inválidos para as redes móveis.

Além dos dados, possui um construtor que coleta estes dados da conexão atual disponível, todos os métodos get para uso das demais classes e um método que converte o sinal RSSI de uma rede WiFi para um número de faixas estabelecido chamando o método calculateSignalLevel da classe WifiManager do Android.

Já a classe Point inclui três construtores, para diferentes parâmetros recebidos. Um nos retorna um ponto baseado na latitude e longitude do objeto Location nativo do Android e os outros dois nos retornam um ponto com coordenadas em graus – um com mais precisão, o outro com menos. É utilizada para facilitar as consultas e o entendimento das coordenadas.

As classes relativas ao desenho em cima do mapa não são de grande importância, como já mencionado, uma vez que a idéia para esta integração com o Google Maps surgiu no decorrer do projeto, apenas como um meio para tornar o aplicativo mais visual.

Conclusão

A implementação deste projeto permitiu o aprendizado da linguagem Java bem como do Android, plataforma de smartphones que está ganhando um espaço enorme no mercado. Foi

também possível a familiarização com um ambiente de programação mais próximo ao profissional, fazendo parte de discussões em fóruns e no próprio laboratório onde parte do projeto foi desenvolvido.

O trabalho descrito pode ser reutilizado através de algumas adaptações para ser empregado em aplicações mais complexas, como o componente que possibilitará o reconhecimento da conexão utilizada e o movimento do aparelho e se antecipar a uma necessária troca para que o aparelho continue conectado à rede. Ainda seria possível expandi-lo, com um menu de configurações e novas funcionalidades, que ficarão para o futuro, expandindo seu uso e integrando possíveis novas versões do componente.

A utilização do aplicativo é simples, podendo-se facilmente navegar entre suas funções e também permitindo que ele seja útil a futuras aplicações desenvolvidas em meio ao projeto Mobilis, do qual este fez parte.

Referências

1 – Deitel, H. M. e Deitel, P. J. **Java™ Como Programar.** 6. ed. São Paulo: Pearson Prentice Hall, 2005. 1110p.

2 – Lecheta Ricardo R. **Google Android: aprenda a criar aplicações para dispositivos movies com o Android SDK.** 2. ed. São Paulo: Novatec Editora, 2010. 608p.

3 – JavaDoc: http://ksoap2.sourceforge.net/doc/api/
Download: http://ksoap2.sourceforge.net/

4 – Página oficial: http://xfire.codehaus.org/

5 – Página oficial: http://www.lac.inf.puc-rio.br/mobilis/index.php/SDM