ATLÂNTICA
UNIVERSITY HIGHER INSTITUTION

FonMan

Autenticador Automático para Redes Wi-fi Fon

Rui Salvaterra, n.º 20151989

Orientadora: Professora Filipa Taborda

Dissertação de Projecto Final para a obtenção do grau de Licenciado em Sistemas e Tecnologias de Informação

Departamento de Ciências e Tecnologias de Informação e Comunicação

Novembro de 2018

Índice

Introdução

O acesso ubíquo à Internet, a partir de dispositivos móveis está dependente da conectividade via rede GSM (2/3/4G) e, sempre que disponível, da conectividade através de redes wi-fi. Estas últimas permitem, potencialmente, uma mais vantajosa relação entre a energia consumida pelo dispositivo e a velocidade da conexão estabelecida — dada a maior eficiência energética da ligação wi-fi[1] ainda que GSM apresente, em modo 4G, resultados competitivos[2] — reduzindo ainda os custos de comunicação, na medida em que a conectividade via GSM está sujeita, na maioria das situações, a limites de tráfego e, consequentemente, a custos acrescidos, caso tais limites sejam excedidos.

Têm vindo a surgir no mercado fornecedores de acesso à Internet através de wi-fi. A Fon[3], um dos operadores com maior expressão a nível mundial, estabelece parcerias com fornecedores de acesso locais, em diversos países, com o objectivo de tornar cada *gateway* residencial num ponto de acesso à Internet para os clientes dos seus parceiros. Deste modo, qualquer cliente de um fornecedor de acesso parceiro da Fon terá acesso à Internet, via wi-fi, em qualquer parte do mundo, desde que se encontre conectado a um ponto de acesso pertencente a um outro parceiro da Fon.

Objectivo

A ligação à Internet através de uma rede Fon envolve o estabelecimento de uma ligação wi-fi a um ponto de acesso cujo ESSID é conhecido (a sua maioria começado por «FON_», havendo algumas excepções), seguido da autenticação numa página *web* (portal cativo). Nesta, o utilizador preenche um formulário com as suas credenciais (nome de utilizador e palavra-chave) que, após submetido e validado, lhe autoriza o acesso à Internet. Ainda que as ligações aos pontos de acessos Fon não sejam encriptadas, o risco de intercepção das credenciais por entidades maliciosas é reduzido, na medida em que os pedidos de autenticação são sempre efectuados sobre HTTPS.

Alternativamente, a autenticação pode ser efectuada através de um protocolo próprio, com base numa versão adaptada/simplificada do *draft* WISPr 1.0[4]. Existem aplicações oficiais (NOS wi-fi, por exemplo) que recorrem a este protocolo para efectuarem a autenticação. Não obstante, o seu comportamento pode ser melhorado e/ou simplificado, quando considerado apenas o seu objectivo primordial (ligação a uma rede Fon e consequente autenticação), evitando o recurso a bibliotecas externas e a funcionalidades

próprias das API da Google, as quais não se encontram presentes num sistema Android puro (AOSP[5]).

Pretende-se, com este projecto, desenvolver uma aplicação para, de forma tão transparente quanto possível, automatizar/optimizar a ligação e autenticação numa rede wi-fi Fon, minimizando o consumo de recursos computacionais escassos (espaço de armazenamento, RAM, tempo de processamento, consumo energético), eliminando dependências externas e suportando o máximo intervalo possível de versões de Android, com foco nas mais antigas e/ou obsoletas.

Metodologia

Para a elaboração deste documento, dadas a quantidade de referências bibliográficas e o facto de serem citadas múltiplas secções das mesmas fontes, foi sentida a necessidade de se recorrer, no texto, a um método de referenciação númerico (ainda que, no respectivo capítulo, a bibliografia se apresente em formato APA), de modo a contextualizar, tornar absolutamente claras e unívocas as referências a cada uma das fontes citadas.

Para a revisão da literatura, tomar-se-ão como referências o *draft* do protocolo WISPr 1.0 e o código fonte da primeira aplicação criada para este fim[6], em 2010, por Joan Fisbein, actual director de engenharia e desenvolvimento da Fon[7]. Esta aplicação consiste em *software* livre, disponível sob a versão 3.0 da licença GNU General Public License[8] (GPLv3).

De seguida, proceder-se-á à análise das funcionalidades presentes nas aplicações existentes, desenvolvidas e/ou adaptadas por fornecedores de acesso parceiros da Fon e, se exequível, com recurso a engenharia inversa (descompilação dos pacotes APK e leitura do código obtido).

Efectuada a análise, será consultada bibliografia técnica específica para esclarecer protocolos, boas práticas, familiarização com a *framework* de Android e ambiente de desenvolvimento (Android Studio), estabelecendo-se o suporte para o desenvolvimento do projecto de engenharia de *software*.

O próximo passo consistirá na especificação dos requisitos e dos restantes artefactos de análise partindo-se, então, para o desenho mais detalhado, em que se tentará documentar os aspectos mais importantes da codificação e arquitectura da solução.

Finalmente, para o desenvolvimento iterativo, poderá ser vantajoso o recurso a ferramentas de gestão de código fonte, de modo a agilizar tanto o processo de modificação

incremental bem como, quando necessário, a reversão para estados de desenvolvimento anteriores.

O principal dispositivo usado no processo de desenvolvimento e depuração será uma *tablet* com a versão 4.0 de Android[9], efectuando-se ainda testes funcionais em dispositivos com as versões 2.1[10] e 4.3[11].

Contexto

Neste capítulo será feita uma introdução de alto nível ao sistema operativo Android, descrevendo-se algumas das suas características e componentes relevantes para a contextualização da engenharia da solução.

Android

Android é um sistema operativo para dispositivos móveis de recursos computacionais limitados (nomeadamente telemóveis e *tablets*, ainda que possa ser instalado em máquinas tradicionais de arquitectura x86[12]), desenvolvido pela Google, com base numa versão modificada do *kernel* Linux e numa biblioteca de C minimalista (Bionic)[13].

Do ponto de vista do programador, Android consiste numa implementação quase completa da API de Java (conjunto de classes Apache Harmony[14]) com extensões adicionais, de modo a suportar o paradigma de desenvolvimento definido pela plataforma. As aplicações são compiladas para *bytecode*[15] e executadas numa máquina virtual (Dalvik) o que as torna, idealmente, independentes da arquitectura computacional subjacente, à semelhança do que acontece com Java.

Na Fig. 1 encontra-se o esquema da arquitectura de Android, desde a camada aplicacional até à do *kernel* (abaixo da qual se encontra o *hardware*, omisso no diagrama), passando pela *framework* de suporte às aplicações, bibliotecas do sistema e componentes de tempo de execução (*runtime*). Os componentes a azul são independentes da arquitectura computacional (consistem em *bytecode* Dalvik a ser executado pela máquina virtual, a amarelo). A verde, encontram-se as bibliotecas pré-compiladas (nativas) de suporte à *framework*, que interagem directamente com o *kernel* (a vermelho).

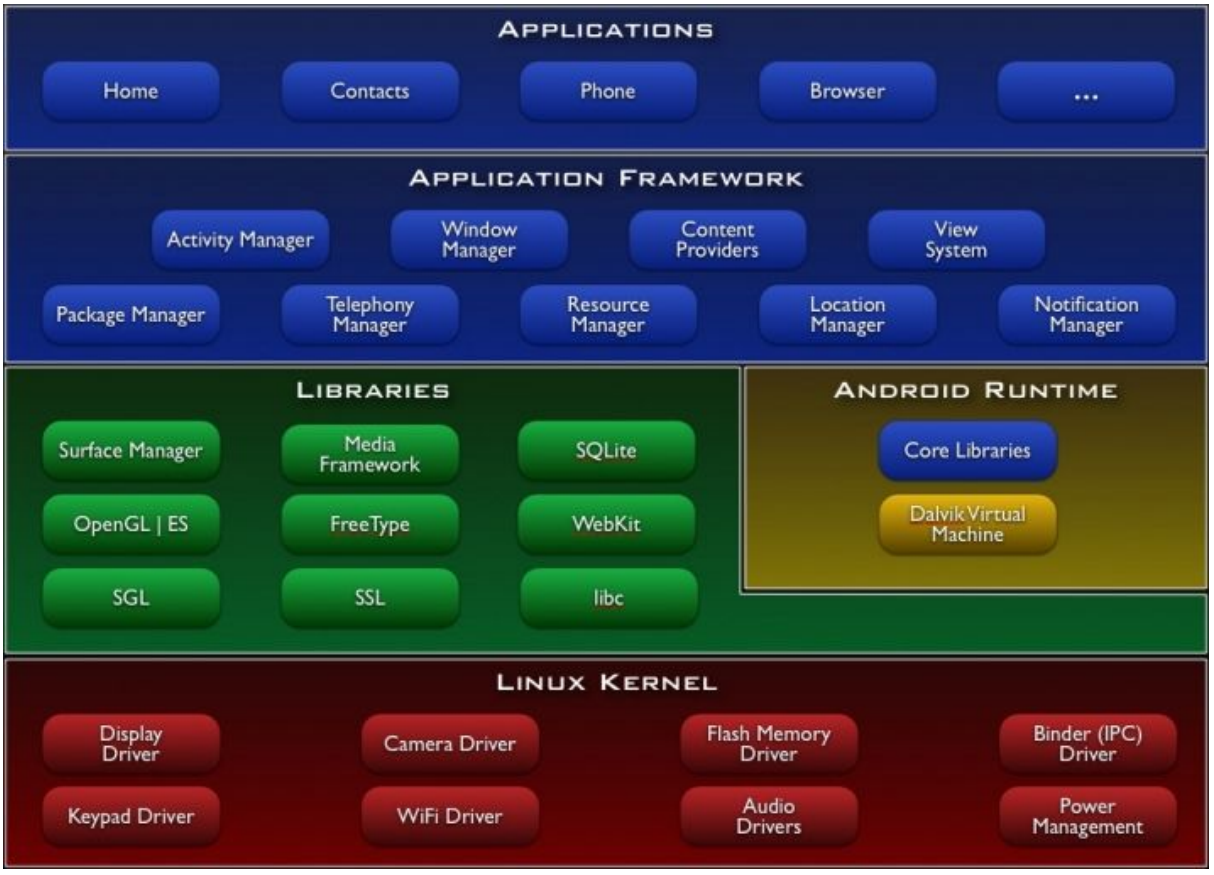

Fig. 1 — Arquitectura do sistema operativo Android[16].

Apesar de assentar sobre o *kernel* Linux, seria incorrecto incluir Android na família de sistemas operativos UNIX, na medida em que não recorre, por concepção, à comunicação entre processos (IPC, *interprocess communication*) de System V[17]. Em alternativa, Linux em Android preenche esta lacuna à custa de dois mecanismos complementares de IPC: binder e ashmem.

Ashmem

Ashmem (*anonymous shared memory)* é um subsistema do *kernel* desenvolvido por Robert Love[18]. Este fornece ao espaço do utilizador uma interface de memória partilhada com base em ficheiros. Funciona como memória anónima, ou seja, resultante da invocação de `mmap()` com `fd = 0`[19], com a excepção de que, se o descritor for partilhado, o mapeamento também o será. A memória partilhada pode ser acedida quer através de `mmap()` quer das operações habituais de entrada/saída de ficheiros.

Adicionalmente, ashmem introduz o conceito de afixação de páginas. As páginas de memória afixadas (todas elas, inicialmente) comportam-se como memória anónima normal. Quando as páginas são desafixadas, passam a estar disponíveis para evicção por iniciativa

do *kernel*, sempre que este sinta essa necessidade (em situações de pressão de memória extrema). Quando o espaço do utilizador tenta reafixar páginas previamente desafixadas, o valor devolvido pela operação indica-lhe se as páginas foram entretanto evencidas.

Estas funcionalidades permitem ao espaço do utilizador implementar *caches* e gerir recursos de forma eficiente, integrada com a gestão de memória do *kernel*.

Binder

Binder é um mecanismo de IPC com base em invocação remota de métodos (RPC, *remote procedure calls*). Como tal, é responsável pela decomposição de dados (objectos) em primitivas capazes de serem interpretadas pelo sistema (*marshalling*), pela transferência de informação entre as fronteiras dos processos envolvidos, pela recomposição da informação no processo remoto (*unmarshalling*) e, por último, pela devolução dos resultados da invocação no processo remoto ao processo de origem[20].

A difusão (*broadcast*) ou envio explícito de um intento[21] ao sistema ou a outra aplicação é, por natureza, uma transacção mediada por Binder (tal como praticamente tudo o que ocorre entre as fronteiras de cada processo, num sistema Android). Está na base de componentes fundamentais da *framework* como, por exemplo, gestor de pacotes, gestor de telefonia, gestor de localização, gestor de notificações, gestor de acessibilidade, gestor de conectividade e gestor de actividades[22].

Dalvik

Dalvik consiste na máquina virtual criada por Dan Bornstein[23], contra a qual é gerado o *bytecode* das aplicações executadas em Android. Em termos de classificação formal, trata-se de uma máquina de registos, por oposição à máquina virtual Java (máquina de pilha). Ainda que o código gerado para uma máquina de pilha seja mais denso do que o de uma máquina de registos, estas últimas compensam largamente esta desvantagem com maiores densidade semântica (complexidade da operação codificada numa instrução) e velocidade de execução[24].

Ciclos de Vida em Android

As infraestruturas fornecidas pelo *kernel* e pela *framework* de Android, cujas finalidades são a máxima conservação de recursos computacionais e energéticos, obrigam a um paradigma específico de desenvolvimento, com ciclos de vida definidos para alguns

dos componentes fundamentais das aplicações criadas sobre esta plataforma. Destacam-se, no contexto deste projecto, as actividades e os serviços.

Actividades

Uma actividade fornece uma janela na qual a aplicação desenha a sua interface gráfica com o utilizador. Esta janela ocupa, tipicamente, todo o ecrã, mas pode ser mais pequena, sobrepondo-se a outras janelas.

Geralmente, cada actividade implementa um ecrã na aplicação. Por exemplo, uma das actividades da aplicação pode implementar um ecrã de configuração/preferências, enquanto que outra actividade implementa um ecrã para seleccionar uma imagem[25].

Ciclo de vida de uma actividade

Para navegar nas transições entre os vários estados do ciclo de vida de uma actividade, a classe `Activity` implementa seis métodos nucleares: `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()` e `onDestroy()`. O sistema invoca cada um destes métodos quando uma actividade transita de estado, como se pode constatar pela observação da ilustração simplificada, na Fig. 2.

Quando o utilizador deixa a actividade, o sistema invoca métodos para a desmantelar. Nalgumas situações, esse desmantelamento é apenas parcial; a actividade ainda reside na memória (por exemplo, quando o utilizador comuta para outra aplicação) e pode ainda retornar ao primeiro plano. Se o utilizador regressa à actividade, esta continua a partir do estado em que o utilizador a deixou. A probabilidade de o sistema exterminar um dado processo — e, com ele, as suas actividades — depende do estado corrente da actividade. Dependendo da complexidade da actividade, pode não ser necessário implementar todos os métodos do ciclo de vida, apenas os suficientes para garantir que a aplicação se comporta da forma esperada[26].

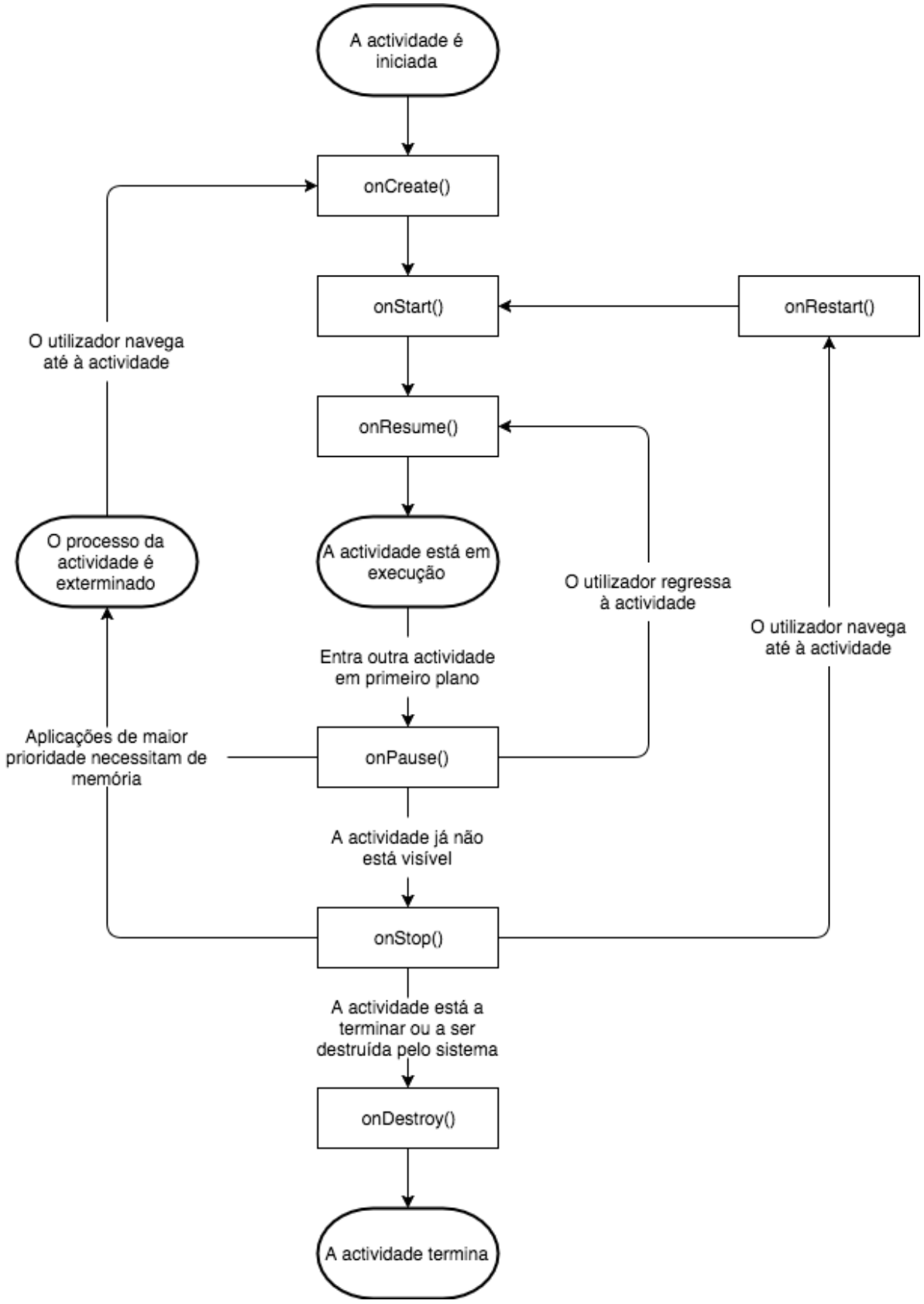

Fig. 2 — Ilustração simplificada do ciclo de vida de uma actividade[27].

Serviços

Um serviço é um componente da aplicação que pode desempenhar tarefas demoradas em segundo plano, sem providenciar uma interface com o utilizador. Outro componente da aplicação pode iniciar um serviço e este permanece em execução, ainda que o utilizador comute para outra aplicação. Adicionalmente, um componente pode vincular-se a um serviço para interagir com este e até efectuar IPC. Por exemplo, um serviço pode tratar transacções de rede, reproduzir música, desempenhar operações de entrada/saída em ficheiros ou interagir com um fornecedor de conteúdos, tudo em segundo plano[28].

Ciclo de vida de um serviço

O ciclo de vida de um serviço é bastante mais simples do que o de uma actividade. No entanto, é ainda mais importante ter em atenção a forma como um serviço é criado e destruído, dado que este pode ser executado em segundo plano, sem o conhecimento do utilizador. Os serviços podem ser vinculados (*bound*) ou iniciados (*started*). Este projecto recorre a um serviço iniciado cujo funcionamento, em conformidade com o nível da API usado, se encontra representado de forma algorítmica na Fig. 3.

Quando um componente invoca `startService()`, o serviço é criado e mantém-se, indefinidamente, em execução, até que pare por iniciativa própria (aquando de uma invocação interna de `stopSelf()`), ou até outro componente o parar (invocando, para essa finalidade, `stopService()`). Os serviços parados são destruídos pelo sistema.

A totalidade do tempo de vida de um serviço está contida entre o início da invocação de `onCreate()` e do fim da execução de `onDestroy()`. Tal como uma actividade, o serviço executa as acções de inicialização necessárias em `onCreate()` e liberta os recursos adquiridos em `onDestroy()`. O tempo de vida activo de um serviço inicia-se com uma invocação de `onStart()` (`onStartCommand()`, em níveis superiores da API). Este método recebe como argumento o intento passado na invocação de `startService()`. No caso particular de um serviço iniciado, o tempo de vida total coincide com o tempo de vida activo[29].

Fig. 3 — Representação algorítmica do ciclo de vida de um serviço[30].

Engenharia da Solução

Neste capítulo levar-se-á a cabo a engenharia da solução, começando-se por uma análise às aplicações existentes, com base na qual serão definidos os requisitos do sistema. Partir-se-á para a arquitectura da aplicação, justificando-se todas as decisões de implementação tomadas.

Análise das Aplicações Existentes

Na União Europeia, a engenharia inversa é legalmente admissível ao abrigo do artigo 3.º (1) (b) da Directiva Europeia 2016/943 de 2016:

> *1. A aquisição de um segredo comercial é considerada legal quando o segredo comercial é obtido por um dos seguintes meios: […] b) observação, estudo, desmontagem ou teste de um produto ou objeto que tenha sido disponibilizado ao público ou que esteja legalmente na posse do adquirente da informação, não estando este sujeito a qualquer dever legalmente válido de limitar a aquisição do segredo comercial;*

Consequentemente, com o auxílio de ferramentas adequadas[31], as aplicações da própria Fon[32] e dos fornecedores de acesso parceiros NOS wi-fi[33] (Portugal), Oi WiFi[34] (Brasil), BT Wifi[35] (Reino Unido), e SFR WIFI[36] (França, aparentemente descontinuada, dado que não se encontra na Play Store) foram descompiladas e o seu comportamento analisado, tanto quanto possível (dado que as boas práticas de desenvolvimento em Android recomendam o recurso a ferramentas de obscurecimento do código[37]). Deste modo, foi possível retirar um conjunto de conclusões relevantes.

Em primeiro lugar, nenhuma das aplicações analisadas, com presença na Play Store, suporta Android abaixo da versão 2.3.3 (NOS wi-fi). Tal leva a suspeitar que exista uma impossibilidade tecnológica fundamental no desenvolvimento e/ou publicação de uma solução compatível com versões anteriores.

Em segundo lugar, a aplicação da SFR é a única que suporta Android 2.2 e cujo código fonte foi pré-processado/obscurecido por ProGuard, antes da geração do APK final o que, ainda que não impossibilitando, dificultou consideravelmente a análise (obrigando à

leitura de *bytecode* Dalvik *disassembled*[1], nas secções do programa impossíveis de descompilar). Esta é também a única que permite definir um limite mínimo de intensidade de sinal, abaixo do qual se rejeitam pontos de acesso.

Em terceiro lugar, todas as aplicações seguem um padrão idêntico de desenvolvimento do sistema de autenticação (o relevante para este projecto), com base na aplicação original de Joan Fisbein, incluindo as mesmas classes, os mesmos identificadores e até os mesmos erros de desenvolvimento. Sendo que esta é disponibilizada sob a licença GPLv3, que proíbe a reutilização do código fonte em aplicações fechadas, é seguro concluir que o código foi usado nestas aplicações com autorização explícita do(s) seu(s) autor(es).

Por último, e pela análise da aplicação da Fon, é possível encontrar uma API SOAP[38] complexa, capaz de responder a casos de uso como, por exemplo, a aquisição de *vouchers* para acesso temporário à Internet, por parte de qualquer pessoa que deseje fazê-lo. Este caso de uso não se enquadra no âmbito deste projecto, assumindo-se que o cliente já dispõe de credenciais válidas.

Requisitos

Considera-se fundamental o requisito de desenvolver uma aplicação para autenticação automática nas redes wi-fi da Fon e seus parceiros, capaz de ser executada em dispositivos que implementem desde o nível 7 (versão 2.1)[39] ou inferior da API de Android, até ao nível máximo que seja possível suportar, sem recurso a bibliotecas externas (incluindo a biblioteca de compatibilidade de Android disponibilizada pela Google) nem a execução condicional (ramificação para caminhos independentes de código, em função do nível da API de Android detectado em tempo de execução).

Adicionalmente, definem-se como requisitos do projecto a implementação de uma funcionalidade de reconexão (isto é, a possibilidade da aplicação tomar a iniciativa de se desligar da rede Fon e se ligar a uma rede privada, pré-configurada no dispositivo e detectada na vizinhança) e da possibilidade de rejeição de pontos de acesso cuja intensidade do sinal se encontre abaixo de um limite de sustentabilidade da ligação, configurável pelo utilizador.

Considera-se ainda como requisito a independência de API extras fornecidas pela Google, não presentes no sistema base de Android. As aplicações existentes na Play Store recorrem a API da Google para funcionalidades como georreferenciação (visualização num mapa de pontos de acesso das redes Fon na vizinhança). Tal funcionalidade é considerada

[1] Não existe, em português, uma palavra para designar a tradução homomórfica de um código máquina para as suas mnemónicas.

supérflua. Adicionalmente, existem dispositivos no mercado, a sua vasta maioria proveniente da China, em cuja base de *software* instalada não se incluem as API da Google.

O último requisito consiste na frugalidade na utilização dos recursos computacionais do dispositivo móvel, limitada à estritamente necessária e suficiente para o correcto funcionamento da aplicação (consequência do requisito inicial de suporte para versões obsoletas de Android). Por conseguinte, não será usada qualquer biblioteca externa (inclusive a biblioteca de suporte de Android), dado que estas contribuem de sobremaneira para o tamanho final da aplicação.

Arquitectura do Sistema (Alto Nível)

Em termos de engenharia de *software* e no contexto de Android, a aplicação é composta por uma actividade de configuração e um serviço. Cabe a este último a tarefa de responder a acções despoletadas por eventos recebidos (intentos), difundidos pelo sistema operativo ou sintetizados pela própria aplicação. A perspectiva de alto nível pode ser simplificada de acordo com o seguinte diagrama de blocos.

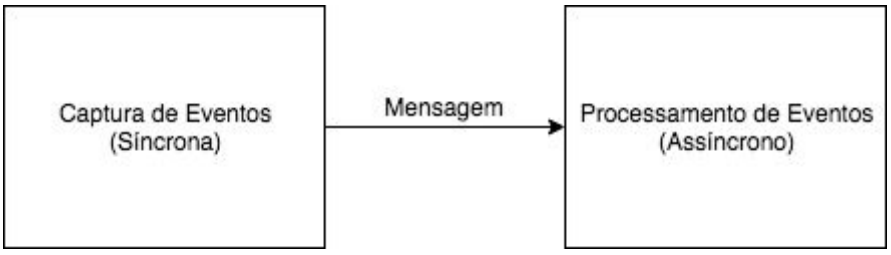

Fig. 4 — Esquema de alto nível da arquitectura do sistema.

Diagrama de Estados

Durante a sua execução, o sistema pode encontrar-se em três estados possíveis, como se pode observar na Fig. 5.

Inicialmente no estado Desconectado (sem conectividade wi-fi), este pode associar-se a um ponto de acesso, comutando para o estado Conectado (associado a um ponto de acesso wi-fi e com conectividade IP). Aí, se a autenticação não for bem-sucedida, retorna ao estado Desconectado. Caso contrário, transita para o estado Autenticado e aí permanece até a conectividade com o ponto de acesso se perder, regressando, finalmente, ao estado Desconectado.

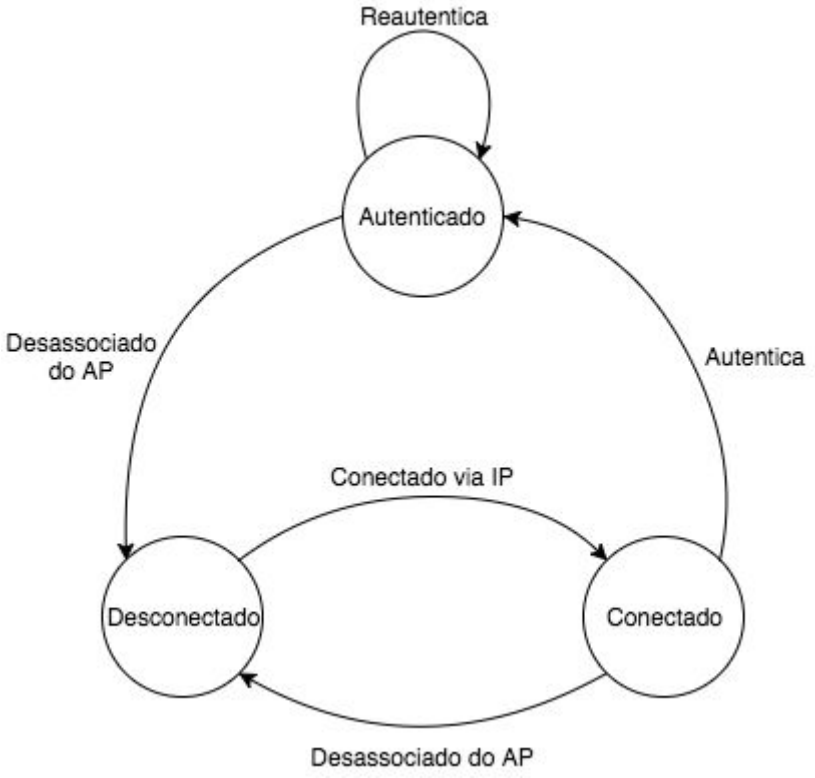

Fig. 5 — Diagrama de estados do sistema.

Diagramas de Casos de Uso

Cabe à aplicação permitir responder a um conjunto de casos de uso definidos *a priori*, considerados indispensáveis e/ou desejáveis. É ainda pertinente considerar uma distinção entre casos de uso do utilizador (nos quais este intervém directamente) e casos de uso do sistema (tratados pela própria aplicação como resposta a um estímulo externo).

Casos de uso do utilizador

Tal como se pode constatar pela observação da Fig. 6, a configuração da aplicação é necessária para o seu funcionamento. Tal implica, necessariamente, preencher as credenciais (nome de utilizador e palavra-chave) usadas para a autenticação na rede Fon. A aplicação pode ser configurada em modo activo (efectuando automaticamente ligações a pontos de acesso wi-fi Fon) ou passivo (apenas efectuando a autenticação, sendo da responsabilidade do utilizador a associação ao ponto de acesso wi-fi). O utilizador pode ser notificado do sucesso/insucesso do processo de autenticação por via acústica e/ou háptica. Adicionalmente, o utilizador pode configurar a aplicação para se desconectar da rede Fon e se ligar a uma outra rede conhecida, caso esta se encontre na vizinhança. Por último, há a

possibilidade avançada de se definir um limite mínimo (em dBm) de intensidade de sinal, sendo que apenas as redes wi-fi cuja intensidade do sinal exceda este limite serão consideradas.

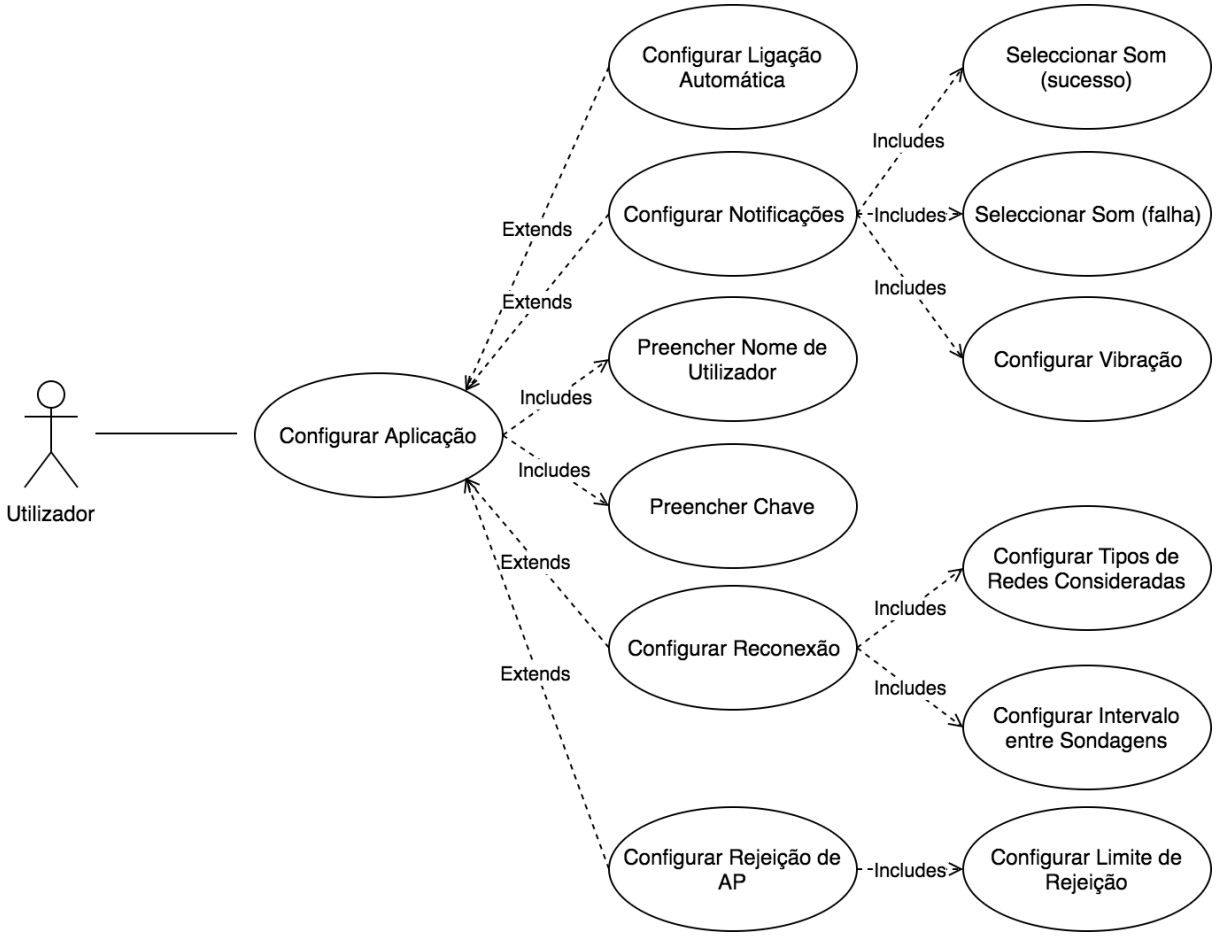

Fig. 6 — Diagrama de casos de uso do utilizador.

Casos de uso do sistema

É da responsabilidade da aplicação efectuar a ligação a uma rede wi-fi (ponto de acesso), quando configurada nesse sentido, tal como demonstrado na Fig. 7. Não obstante, a ligação a uma rede Fon é estabelecida se, e só se, não havendo conectividade wi-fi e após uma sondagem, não se encontrar outra rede conhecida na vizinhança. Efectuada a ligação wi-fi a uma rede Fon, é necessário proceder à autenticação no respectivo RADIUS e verificar periodicamente o seu estado. Se tal for desejado, a aplicação pode ainda tomar a iniciativa de se desconectar da rede Fon e se reconectar a uma outra rede wi-fi conhecida. Para tal, no decorrer de uma sessão iniciada numa rede Fon, são feitas sondagens periódicas às redes wi-fi disponíveis.

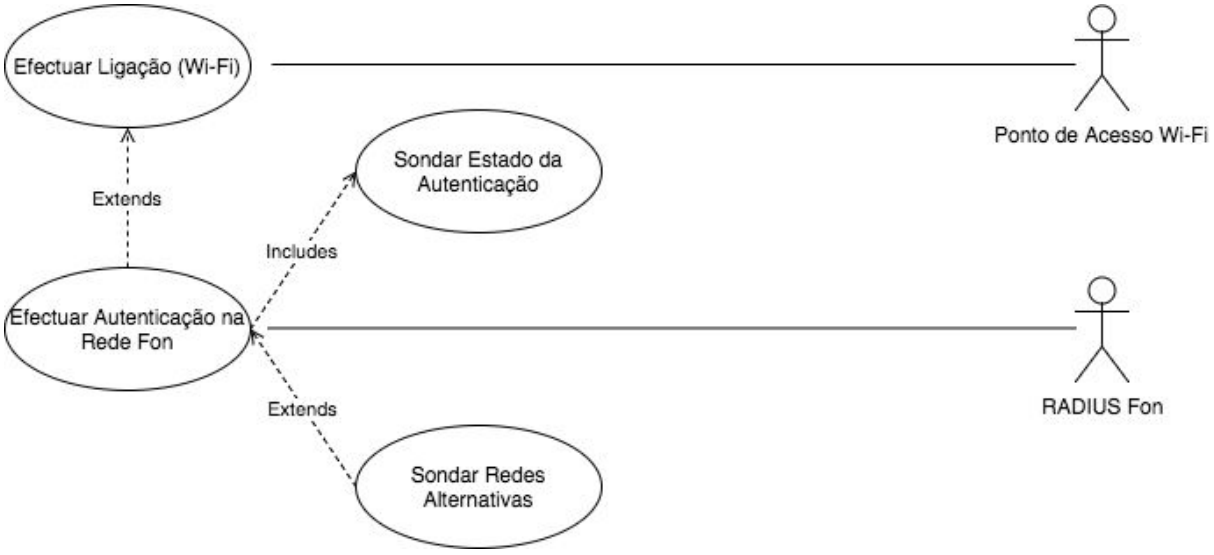

Fig. 7 — Diagrama de casos de uso do sistema.

Diagrama de Sequência (Alto Nível)

Pela análise das mensagens XML recebidas e das respostas do protocolo HTTP, aquando do início da sessão numa rede Fon, é possível criar algo semelhante a um diagrama de sequência de alto nível, para uma operação de autenticação bem-sucedida, tal como se apresenta na Fig. 8. Podem ainda encontrar-se, no Anexo 1, exemplos e uma explicação mais detalhada de cada uma das mensagens.

O cliente (aplicação) começa por efectuar um pedido HTTP GET a um servidor remoto de teste cuja resposta, estando autenticado (com acesso livre à Internet), é conhecida *a priori*. Não estando autenticado, o ponto de acesso Fon intercepta este pedido e responde, ele próprio, com o código HTTP 302, redireccionando o cliente para o RADIUS da Fon, do qual é descarregada a página HTML UAM (*Universal Access Method*). No corpo (elemento `<body>`) desta encontra-se embebida a mensagem XML com a informação necessária para proceder à autenticação.

Seguidamente, o cliente efectua um pedido HTTP POST ao URL especificado na mensagem XML anterior, enviando as credenciais (nome de utilizador e palavra-chave) num formulário HTML, cujo conteúdo é codificado em MIME como application/x-www-form-urlencoded[40].

Por fim, se a autenticação for bem sucedida e após (pelo menos) dois redireccionamentos HTTP 302, o RADIUS responderá com o código HTTP 200 (OK), concluindo-se o processo de autenticação e ficando estabelecido o acesso à Internet.

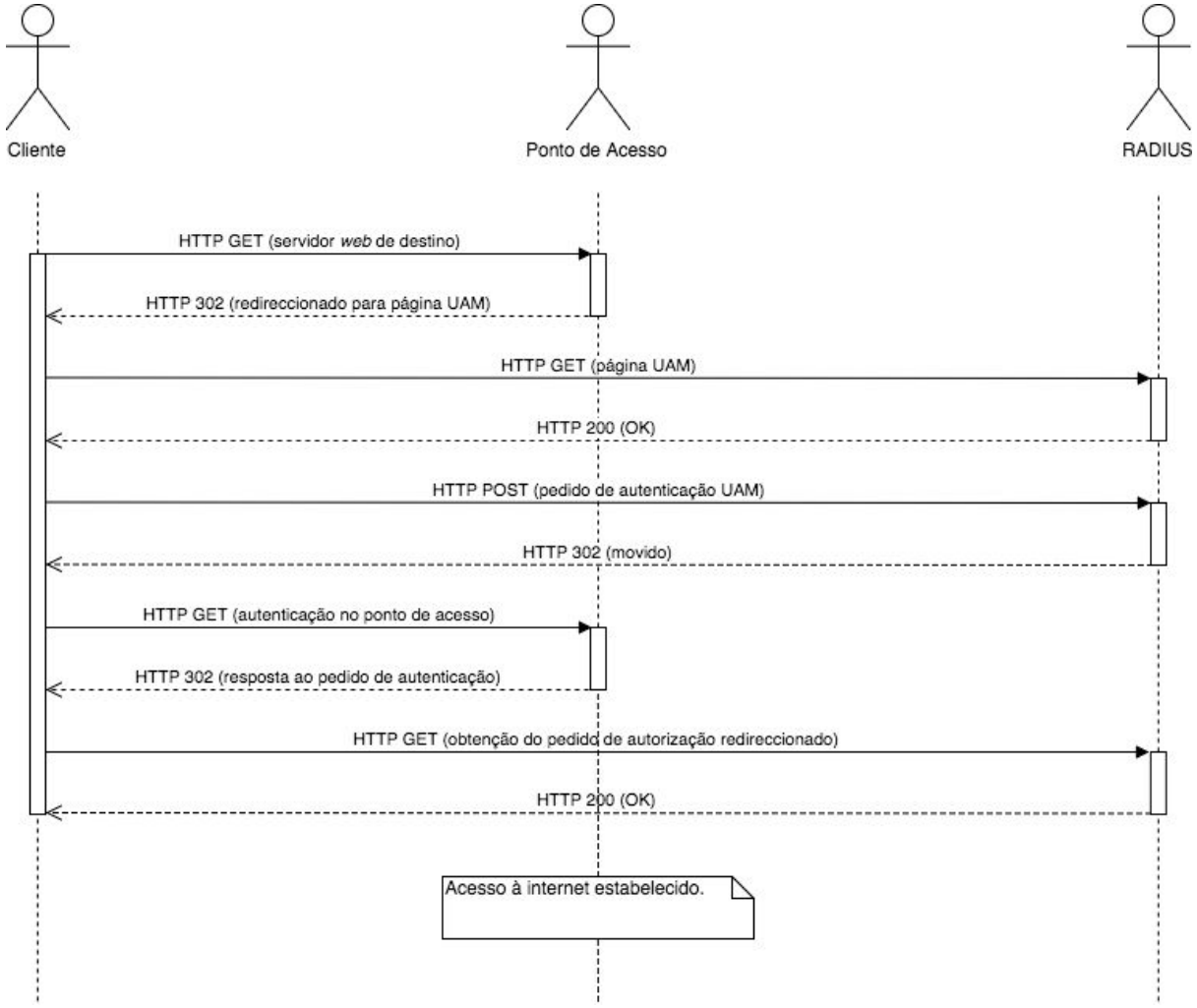

Fig. 8 — Diagrama de sequência de alto nível da operação de autenticação.

Justificação das Permissões Declaradas

Com o intuito de proteger a privacidade dos utilizadores, o sistema operativo Android obriga a que as certas operações do sistema necessitem de permissão explícita e, dado o nível alvo da API, declarada *a priori*[41]. Deste modo, e de acordo com os casos de uso definidos, constata-se que a aplicação requer, para o seu correcto funcionamento, seis permissões do sistema. A justificação para cada permissão declarada é apresentada de seguida.

android.permission.ACCESS_WIFI_STATE

Permissão de acesso ao estado da ligação wi-fi[42]. Esta permissão é necessária para a obtenção de instâncias de `WifiManager` e consequente acesso à informação sobre o estado da ligação wi-fi.

android.permission.CHANGE_WIFI_STATE

Permissão de alteração do estado da ligação wi-fi[43]. Esta permissão é necessária para estabelecer uma ligação wi-fi, efectuar sondagens às redes wi-fi disponíveis, criar, modificar e eliminar ligações wi-fi no sistema.

android.permission.INTERNET

Permissão de criação de *sockets* de rede[44]. É necessária para comunicar através da Internet, um requisito do processo de autenticação.

android.permission.ACCESS_COARSE_LOCATION

Permissão de acesso à localização imprecisa. Trata-se de uma permissão considerada sensível[45]. Apesar de não estritamente necessária nos níveis da API em que a aplicação é executada, considera-se boa prática declará-la por duas razões: em primeiro lugar, a aplicação requer acesso à Internet para efectuar a autenticação; em segundo lugar, requer acesso à lista de pontos de acesso wi-fi sondados (e respectivos metadados, que incluem, para cada ponto de acesso, a intensidade do seu sinal). Qualquer aplicação que necessite destas permissões tem, por definição, acesso a uma forma de localização que, apesar de classificada como imprecisa pode, efectivamente, atingir um grau de precisão muito elevado[46], na medida em que nada a impede de usar um serviço externo (por exemplo, Mozilla Location Service[47]) para correlacionar a informação obtida a partir dos pontos de acesso sondados. Por este motivo, no nível 23 da API, a Google tornou obrigatória a declaração desta permissão para todas as aplicações que necessitem de aceder aos resultados de uma sondagem de pontos de acesso wi-fi, sob a pena de as invocações de `getScanResults()` sobre uma instância da classe `WifiManager` resultarem sempre na devolução de uma lista de resultados vazia[48].

android.permission.VIBRATE

Permissão de acesso ao vibrador do dispositivo[49]. Esta permissão é necessária para notificação háptica, caso o utilizador a deseje activar.

android.permission.WAKE_LOCK

Permissão de obtenção de `WakeLock` a partir do `PowerManager`[50].

A gestão de energia em Android tem por base uma técnica designada por suspensão oportunística[51]. Por outras palavras, se não houver nenhuma tarefa importante a ser executada pelo sistema, este entra, logo que possível, em suspensão. Qualquer secção crítica de código (que requeira execução ininterrupta) tem que ser protegida pela aquisição prévia de uma `WakeLock`. Num dado intervalo de tempo, a suspensão é impedida se, e só se, existir pelo menos uma `WakeLock` adquirida durante esse intervalo de tempo[52].

Diagramas de Classes

A totalidade da aplicação é implementada em cinco classes, sendo que uma não é instanciada, apenas contendo métodos de classe (estáticos). Consequentemente, são instanciados quatro objectos, dos quais um persiste durante todo o tempo em que a aplicação se encontra em execução.

FonManWifiReceiver e FonManAlarmReceiver

Os «motores» de todo o funcionamento, na Fig. 9. São responsáveis pela intercepção dos intentos que despoletam as acções a executar pelo serviço. `FonManWifiReceiver` intercepta intentos do sistema relacionados com o estado da ligação wi-fi, enquanto que `FonManAlarmReceiver` intercepta intentos correspondentes a acções futuras, sintetizados pelo serviço, no momento em que devem ser tratados.

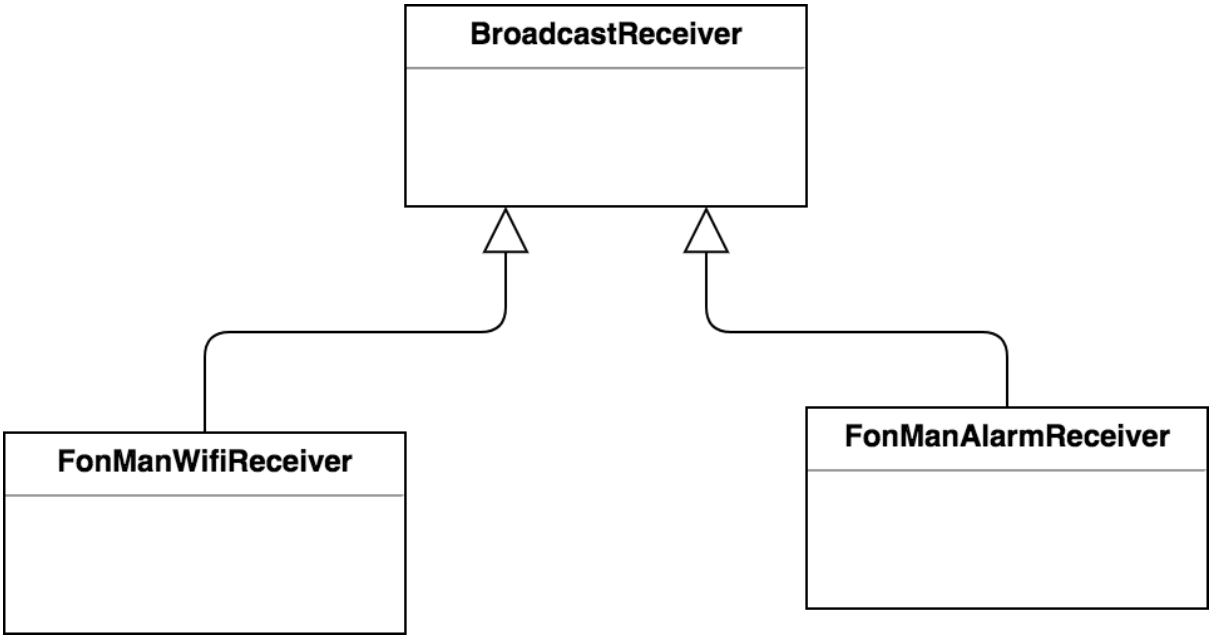

Fig. 9 — Diagrama de classes dos `BroadcastReceiver` (componente síncrona).

FonManService

Trata-se do serviço responsável pelo tratamento de todos os intentos interceptados por ambos os `BroadcastReceiver`. Como é possível constatar, pela observação da Fig. 10, é constituído por um `Handler`, responsável pela gestão de uma fila de mensagens e implementa ele próprio a interface `Callback`, cujo método `handleMessage()` é invocado no tratamento de cada mensagem. De modo a limitar o número de instâncias mantidas em tempo de execução, implementa também a interface `Comparator<ScanResult>`, cujo método `compareTo()` é usado na ordenação (por ordem decrescente de intensidade de sinal) da lista de resultados de uma sondagem wi-fi. Contém ainda a única `WakeLock` da aplicação, necessária para garantir a execução ininterrupta de secções temporalmente críticas.

Agrega, por fim, `LoginManager`, a classe responsável pelo processo de autenticação (cuja instanciação é tanto desnecessária como impossível, apenas contendo métodos de classe e um construtor por omissão declarado como privado) e a «lista negra». Esta última consiste num `HashMap` cujas chaves são `String` (o BSSID do ponto de acesso, ou seja, o seu endereço MAC) e os valores são `Long` (o instante temporal, em milissegundos, calculado no momento da inserção, até que o BSSID correspondente possa ser removido do `HashMap`), onde são colocadas, durante um período mínimo de cinco minutos, as redes wi-fi Fon detectadas como não funcionais.

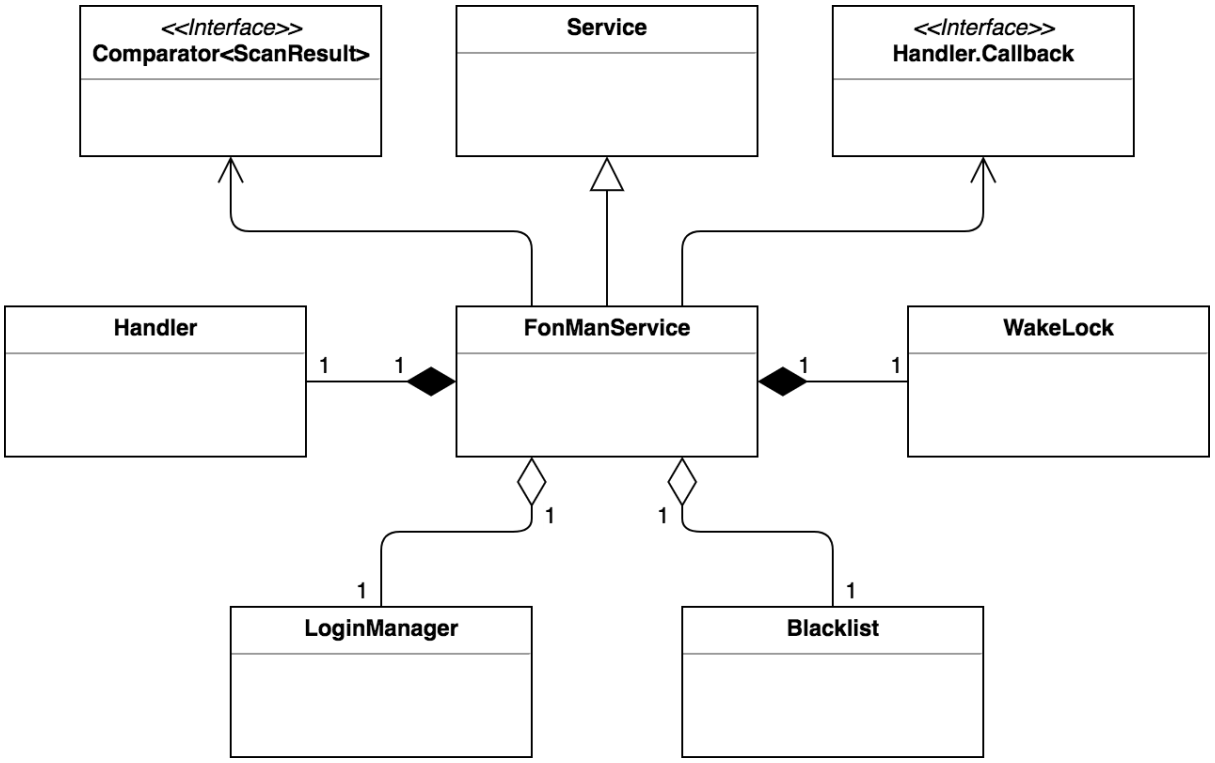

Fig. 10 — Diagrama de classes do serviço (componente assíncrona).

SettingsActivity

A API de Android disponibiliza a classe `PreferenceActivity` (derivada de `Activity`), cuja função é a de facilitar a criação de actividades específicas para a configuração das aplicações. `SettingsActivity` é a única actividade existente na aplicação, como se pode observar na Fig. 11. A sua finalidade é a de responder aos casos de uso do utilizador ou, por outras palavras, a todo o processo de configuração. Esta implementa três `Listener`, necessários para o preenchimento dinâmico de descrições e apresentação da janela *pop-up* com a informação de *copyright*.

Actualmente, por questões de segurança, é veementemente desencorajada a derivação directa de `PreferenceActivity`, dada a existência de uma vulnerabilidade que permite o carregamento arbitrário de classes (e consequente execução dos inicializadores de classe) e injecção de fragmentos[53]. No entanto, os fragmentos de preferências foram introduzidos apenas no nível 11 da API (Android 3.0), pelo que não são utilizados neste projecto. Adicionalmente, o tipo de permissões requerido pela aplicação torna-a num vector de ataque pouco interessante para escalada de privilégios.

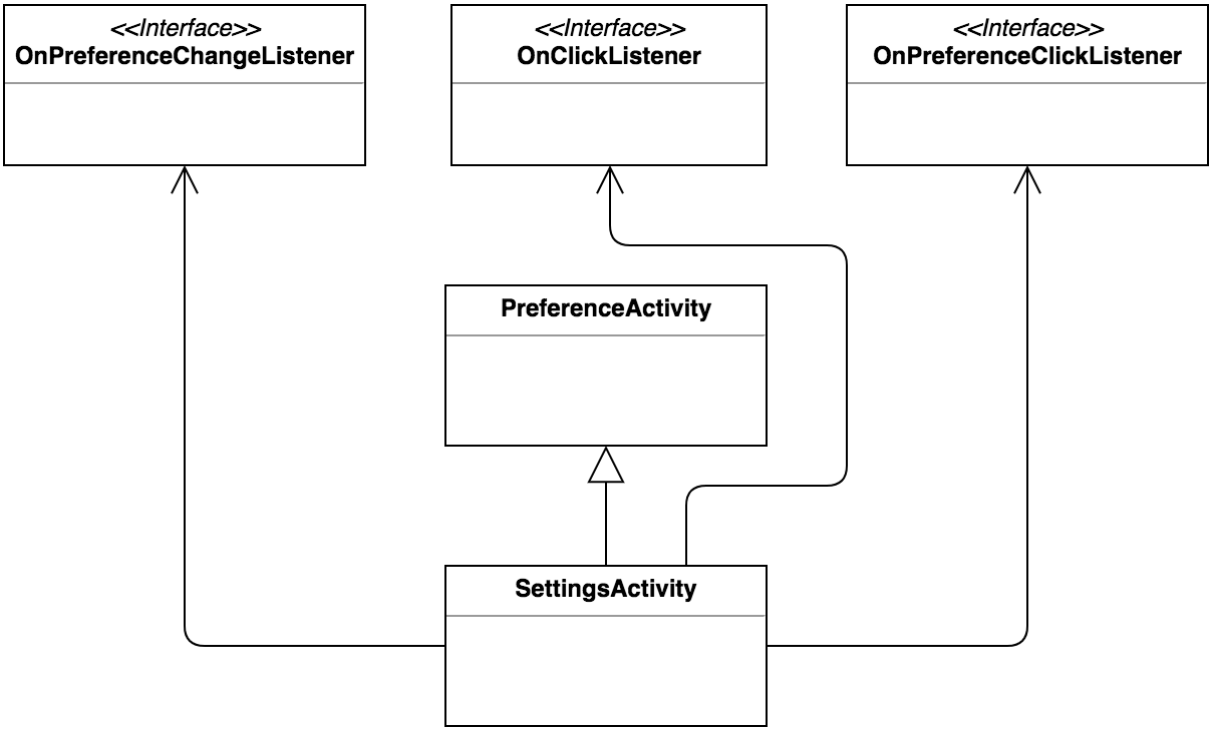

Fig. 11 — Diagrama de classes da actividade de configuração.

Detalhes da Implementação

Neste subcapítulo apresenta-se, de forma mais detalhada, o funcionamento geral da aplicação e, posteriormente, aspectos técnicos relevantes na prossecução dos objectivos propostos, nomeadamente em termos de eficiência (utilização de recursos computacionais, em termos de espaço e de tempo de execução) e de minimização de dependências.

Descrição geral

A classe `FonManWifiReceiver` manifesta, perante o sistema operativo, o interesse em interceptar eventos de mudança de estado de uma ligação e de disponibilidade dos resultados de uma sondagem wi-fi ocorrida. Quando um intento deste tipo é interceptado, é enviado ao serviço em execução (`FonManService`) para ser tratado, libertando o processamento do *thread* principal da aplicação (não o fazer poderia dar origem a um evento ANR, *application not responding*).

`FonManService` é a classe responsável por todo o processamento de intentos. Consiste num serviço (`Service`) em execução permanente (apesar de tal parecer um contra-senso, de acordo com as boas práticas documentadas[54]), num *thread* separado. Este é criado com o auxílio de uma instância de `HandlerThread`, da qual se extrai o

`Looper` (o *thread* de execução cíclica) usado para criar o `Handler` (fila de mensagens). A persistência do serviço é justificada pela necessidade de manter em memória uma determinada quantidade de estado (nomeadamente, a «lista negra» de pontos de acesso não funcionais), pela minimização do tempo de resposta ao tratamento de eventos (não sendo necessário recriar o serviço a cada intento recebido) e pela redução do número de alocações/desalocações de memória[55], responsável por uma maior fragmentação interna da *heap* e consequente aumento de probabilidade da intervenção do *garbage collector*, com consequências nefastas no que diz respeito à latência da execução[56].

Todas as operações efectuadas pela aplicação estão sujeitas a constrangimentos temporais, pelo que é necessário garantir que são executadas ininterruptamente. A execução do método `onReceive()` de um `BroadcastReceiver` é considerada uma tarefa de primeiro plano, pelo que, garantidamente, nunca será interrompida (excepto em situações anómalas de pressão de memória extrema)[57]. Finda a execução de `onReceive()`, é expectável que o dispositivo entre em suspensão a qualquer momento, pelo que é necessário tomar medidas para evitar que tal aconteça, adquirindo-se uma `WakeLock` antes do envio do intento a ser processado a `FonManService`.

O leitor que possua uma experiência mais profunda de desenvolvimento em Android reconhecerá, indubitavelmente, um padrão de execução semelhante ao disponibilizado pela classe `WakefulBroadcastReceiver`[58], existente na biblioteca de suporte de Android[59], quando combinada com um `IntentService`[60]. De forma a não ser necessário utilizar a biblioteca de suporte, a classe `WakefulBroadcastReceiver` foi inicialmente copiada e integrada directamente na aplicação. Verificou-se, no entanto, que esta não está programada de uma forma tão eficiente quanto o desejável, tanto em termos de tempo de execução como de memória usada (cria uma nova `WakeLock` sem contagem de referências a cada invocação do serviço e guarda-a numa matriz até expirar), pelo que a sua funcionalidade foi simplificada, optimizada (apenas existe uma única `WakeLock`, com contagem de referências, adquirida na invocação do serviço e libertada na sua finalização) e absorvida pela classe `FonManService`. Adicionalmente, os `IntentService` invocam `stopSelf()` no final do processamento de cada mensagem podendo, finda essa invocação, ser destruídos pelo sistema a qualquer momento, não se adequando às necessidades supra referidas de manutenção de estado. É pertinente salientar que nenhuma das aplicações analisadas no início do terceiro capítulo recorre à aquisição/libertação de `WakeLock` no processo de autenticação. FonMan é a única aplicação que garante a execução ininterrupta de secções críticas.

No decorrer da execução, certos eventos poderão dar origem à síntese de intentos a serem tratados no futuro. A autenticação bem-sucedida numa rede Fon, por exemplo, dá origem a uma acção periódica de verificação do estado da ligação. A intercepção dos intentos sintetizados por `FonManService` é da responsabilidade da classe `FonManAlarmReceiver`. Apesar de ser uma classe derivada de `BroadcastReceiver` (tal como `FonManWifiReceiver`), não declara um filtro de intentos em AndroidManifest.xml, o que limita o seu âmbito de resposta a intentos sintetizados pela própria aplicação.

O processo de autenticação é da responsabilidade da classe `LoginManager`. Se as credenciais estiverem devidamente configuradas (nome de utilizador e palavra-chave) e a ligação a um ponto de acesso wi-fi da Fon tiver sido estabelecida com sucesso, o método `login()` de `LoginManager` é invocado para que seja efectuado o início de sessão. Esta é, provavelmente, a classe mais complexa da aplicação na medida em que, de modo a evitar a utilização de bibliotecas externas, toda a comunicação HTTP/HTTPS (métodos GET e POST) é implementada a um nível mais baixo, à custa de `HttpURLConnection`[61]. Todas as respostas, incluindo redireccionamentos, são tratadas explicitamente. Tal decisão de engenharia permite evitar o recurso a bibliotecas externas (OkHttp[62], por exemplo) ou a componentes da API desencorajados/eliminados (Apache HTTP Client[63]).

Numa fase inicial do desenvolvimento procedeu-se à avaliação das mensagens XML recebidas com recurso a dois interpretadores (*parsers*) SAX[64], dado que o ambiente de avaliação XPath[65], mais intuitivo (na opinião do autor) e menos verboso na sua utilização, apenas foi introduzido no nível 8 da API (Android 2.2). No entanto, a simplicidade estrutural das mensagens (como se pode constatar pela leitura do Anexo 1) permitiu uma reimplementação da sua interpretação com recurso exclusivo à manipulação directa de `String`, poupando a criação/destruição frequente (a cada autenticação) de dois objectos (os respectivos interpretadores). De notar que esta foi uma decisão de engenharia tomada em consciência, num âmbito muito limitado e não constitui, de forma alguma, uma abordagem encorajada pelo autor.

Restrições ao intervalo entre níveis da API suportados

Com base nos requisitos definidos inicialmente e durante o processo de desenvolvimento, foi possível identificar com absoluta exactidão quais as funcionalidades da API de Android usadas que estabelecem os limites inferior e superior do intervalo entre os níveis da API suportados.

De modo a reduzir o número de classes e instâncias, por oposição à criação de uma subclasse de `Handler`, optou-se por implementar a interface `Handler.Callback`. Esta

apenas está disponível a partir do nível 3 da API (Android 1.5). Dado que o primeiro dispositivo Android no mercado (HTC Dream[66]) suporta até, pelo menos, a versão 1.5 do sistema operativo[67], este foi considerado um compromisso aceitável.

O limite superior é definido pela invocação do método `setLatestEventInfo()`, usado na construção das notificações da aplicação. Este foi removido no nível 23 da API (Android 6.0)[68], passando a ser obrigatória a construção de notificações com recurso a `Notification.Builder`, introduzido no nível 11 da API (Android 3.0), estabelecendo-se como limite superior o nível 22 da API (Android 5.1).

É ainda importante referir que, como resultado destas restrições e dados os requisitos mínimos actuais da Google, em termos de nível da API alvo, a publicação da aplicação na Play Store não é permitida[69]. Existem, no entanto, lojas de aplicações alternativas (F-Droid[70] e Aptoide[71], por exemplo) em que esta limitação não se aplica.

Gestão de Código Fonte

A gestão do código fonte e o controlo de revisões do projecto foram levadas a cabo com o auxílio de Git[72]. Trata-se de um sistema de gestão de código fonte concebido em 2005, por Linus Torvalds, com o objectivo de suportar a taxa de modificações (quantidade de código adicionada/removida/alterada por unidade de tempo) e os *workflows* adoptados no desenvolvimento do *kernel* Linux, o maior projecto de *software* de sempre[73]. É, actualmente, mantido por Junio Hamano e por uma comunidade de desenvolvimento que envolve, além de entusiastas, diversas empresas.

As suas vantagens como sistema distribuído, o facto de garantir a integridade da informação armazenada e o seu desempenho inigualável, tornam-no numa ferramenta indispensável no desenvolvimento de qualquer projecto de *software* não trivial[74].

No contexto específico deste projecto, houve diversas alterações de complexidade e impacto profundos (por exemplo, a reescrita da comunicação HTTP para eliminar as classes do cliente Apache HTTP, usando apenas `HttpURLConnection`) que, na ausência de uma ferramenta adequada de gestão de código fonte, teriam sido substancialmente mais difíceis de executar.

Ocorrências imprevistas

Encontra-se temporariamente alojada em GitHub uma versão preliminar do código fonte[75]. Esta foi, desde então, substancialmente modificada, encontrando-se a versão final no Anexo 2. Dada a perda acidental do segundo factor de autenticação, é impossível a

submissão de novas alterações, o que invalida este repositório como referência (Git é um sistema distribuído, pelo que nenhum repositório é, por definição, mais importante do que outro). Trata-se apenas de um contratempo, na medida em que está planeada a criação de uma nova conta, noutra plataforma aberta (GitLab[76]) e a importação de toda a árvore Git actualmente existente em GitHub, não havendo qualquer perda de informação.

Lições aprendidas

Factores de autenticação extras constituem camadas de segurança adicionais e devem, sempre que disponíveis, ser usados. No entanto, há que salvaguardar a informação necessária para efectuar a recuperação de uma conta, caso um ou mais factores se percam.

No caso particular de GitHub, é impossível recuperar uma conta caso se perca o segundo factor de autenticação e não se tenham salvaguardado os códigos de segurança fornecidos aquando da activação da autenticação de dois factores.

Conclusão

Consideram-se integralmente cumpridos todos os requisitos definidos. FonMan pode ser executado por dispositivos que implementem a API de Android entre os níveis 3 e 22, inclusive, sendo a interoperabilidade total garantida até ao nível 19 (Android 4.4.x). A execução entre os níveis 21 (Android 5.0) e 22 é possível se, e só se, a comunicação de dados via GSM estiver desactivada.

A partir no nível 23, além dos constrangimentos relacionados com a construção de notificações, foi implementada uma API específica para autenticação em portais cativos[77], impedindo o correcto funcionamento da aplicação desenvolvida neste projecto. A documentação não é clara quanto à existência de todas as primitivas necessárias à construção de aplicações para autenticação em portais cativos, em conformidade com esta nova API, pelo que o suporte a versões mais recentes de Android poderá estar comprometido.

A verificação do correcto funcionamento da aplicação está limitada por constrangimentos temporais e financeiros. Apesar de esta suportar, teoricamente, os fornecedores de acesso parceiros da Fon conhecidos, a nível internacional, apenas foram efectuados testes em Portugal, em dispositivos com as versões de Android 2.1, 4.0 e 4.3, tendo-se comprovado a sua correcção. Há, no entanto, indicação de que a aplicação funciona correctamente no Reino Unido, na rede Fon da BT[78].

A aplicação foi ainda instalada num dispositivo emulado, com a versão 1.5 de Android. Ainda que não tenha sido possível testar as funcionalidades de ligação e autenticação, pelo facto de dependerem de uma ligação wi-fi, comprovou-se o funcionamento da actividade de configuração, não tendo sido detectados erros em tempo de execução.

Este projecto, quando interpretado como uma prova de conceito, demonstra ainda a viabilidade de, em âmbitos restritos, desenvolver aplicações minimalistas para dispositivos Android obsoletos, sem prejuízo de compatibilidade com versões mais recentes do sistema.

Bibliografia

[1] Balasubramanian, N., Balasubramanian, A., & Venkataramani, A. (2009, November). Energy consumption in mobile phones: a measurement study and implications for network applications. *In Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement* (pp. 280-293). ACM.

[2] Joshi, P., Colombi, D., Thors, B., Larsson, L. E., & Törnevik, C. (2017). Output power levels of 4G user equipment and implications on realistic RF EMF exposure assessments. *IEEE Access, 5*, 4545-4550.

[3] Fon is the global WiFi network with millions of hotspots | Fon. (2018). Retrieved 23 November 2018, from https://fon.com/.

[4] Anton, B., Bullock, B., & Short, J. (2003). *Best Current Practices for Wireless Internet Service Provider (WISP) Roaming. Marcelotoledo.com.* Retrieved 23 November 2018, from http://marcelotoledo.com/wp-content/uploads/2007/12/wispr_v10.pdf.

[5] Android Open Source Project. (2018). Retrieved 23 November 2018, from https://source.android.com/.

[6] Fisbein, J. (2014). jfisbein/androidwisprclient. Retrieved 23 November 2018, from https://github.com/jfisbein/androidwisprclient.

[7] Joan Fisbein - Principal Engineer at Fon. (2018). Retrieved 23 November 2018, from https://es.linkedin.com/in/joan-fisbein-1b515.

[8] The GNU General Public License v3.0. (2007). Retrieved 23 November 2018, from https://www.gnu.org/licenses/gpl-3.0.en.html.

[9] Onda Vi40 Ultimate Android Slate 9.7 inch IPS Screen 1024×768 WIFI Direct 2160P HDMI 32GB. (2012). Retrieved 23 November 2018, from http://www.onda-tablet.com/onda-vi40-ultimate-android-slate-9-7-inch-ips-screen-1024-768-wifi-direct-2160p-hdmi-32gb.html.

[10] ZTE Racer - Full phone specifications. (2010). Retrieved 23 November 2018, from https://www.gsmarena.com/zte_racer-3423.php.

[11] Huawei Ascend Y530 - Full phone specifications. (2014). Retrieved 23 November 2018, from https://www.gsmarena.com/huawei_ascend_y530-6103.php.

[12] Android-x86 - Porting Android to x86. (2018). Retrieved 23 November 2018, from http://www.android-x86.org/.

[13] platform/bionic - Git at Google. (2018). Retrieved 23 November 2018, from https://android.googlesource.com/platform/bionic.

[14] platform/external/apache-harmony - Git at Google. (2018). Retrieved 23 November 2018, from https://android.googlesource.com/platform/external/apache-harmony/.

[15] Dalvik bytecode | Android Open Source Project. (2018). Retrieved 23 November 2018, from https://source.android.com/devices/tech/dalvik/dalvik-bytecode.

[16] Android System Architecture. (2018). Retrieved 23 November 2018, from https://developer.android.com/images/system-architecture.jpg.

[17] 6.4 System V IPC. (1996). Retrieved 23 November 2018, from https://www.tldp.org/LDP/lpg/node21.html.

[18] Love, R. (2011). *Anonymous shared memory (ashmem) subsystem* [LWN.net]. Retrieved 23 November 2018, from https://lwn.net/Articles/452035/.

[19] mmap(2) - Linux manual page. (2018). Retrieved 23 November 2018, from http://man7.org/linux/man-pages/man2/mmap.2.html.

[20] Aleksandar, G. (2013). *Deep Dive into Android IPC/Binder Framework at Android Builders Summit 2013*. Retrieved 23 November 2018, from https://events.static.linuxfound.org/images/stories/slides/abs2013_gargentas.pdf.

[21] Intents and Intent Filters | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/guide/components/intents-filters.

[22] Hackborn, D. (2009). LKML: Dianne Hackborn: Re: [PATCH 1/6] staging: android: binder: Remove some funny && usage. Retrieved 23 November 2018, from https://lkml.org/lkml/2009/6/25/3.

[23] Dalvik VM Internals - 2008 Google I/O Session Videos and Slides. (2008). Retrieved 23 November 2018, from https://sites.google.com/site/io/dalvik-vm-internals.

[24] Shi, Y., Casey, K., Ertl, M. A., & Gregg, D. (2008). *Virtual machine showdown: Stack versus registers. ACM Transactions on Architecture and Code Optimization (TACO), 4*(4), 2.

[25] The concept of activities | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/guide/components/activities/intro-activities.

[26] Understand the Activity Lifecycle | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/guide/components/activities/activity-lifecycle.

[27] A simplified illustration of the activity lifecycle. (2018). Retrieved 23 November 2018, from https://developer.android.com/guide/components/images/activity_lifecycle.png.

[28] Services overview | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/guide/components/services.

[29] Managing the lifecycle of a service | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/guide/components/services#Lifecycle.

[30] The service lifecycle. (2018). Retrieved 23 November 2018, from https://developer.android.com/images/service_lifecycle.png.

[31] APK decompiler - decompile Android .apk ✓ ONLINE ✓. (2018). Retrieved 23 November 2018, from http://www.javadecompilers.com/apk.

[32] Fon WiFi app – app wi fi map with unlimited access. (2018). Retrieved 23 November 2018, from https://play.google.com/store/apps/details?id=com.fon.wifiapp.

[33] NOS wi-fi. (2018). Retrieved 23 November 2018, from https://play.google.com/store/apps/details?id=pt.zon.fon.

[34] Oi WiFi. (2018). Retrieved 23 November 2018, from https://play.google.com/store/apps/details?id=br.com.mobicare.oiwifi.

[35] BT Wi-fi. (2018). Retrieved 23 November 2018, from https://play.google.com/store/apps/details?id=com.bt.mnie.wispr.

[36] SFR WiFi for Android - APK Download. (2013). Retrieved 23 November 2018, from https://apkpure.com/sfr-wifi/com.sfr.android.sfrwifi.

[37] Shrink your code and resources | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/studio/build/shrink-code.

[38] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). (2018). Retrieved 23 November 2018, from https://www.w3.org/TR/soap12/.

[39] Codenames, Tags, and Build Numbers | Android Open Source Project. (2018). Retrieved 23 November 2018, from https://source.android.com/setup/start/build-numbers.

[40] Nebel, E., & Masinter, L. (1995). RFC 1867 - Form-based File Upload in HTML. Retrieved 23 November 2018, from https://www.ietf.org/rfc/rfc1867.txt.

[41] Permissions overview | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/guide/topics/permissions/overview.

[42] ACCESS_WIFI_STATE | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/reference/android/Manifest.permission#ACCESS_WIFI_STATE.

[43] CHANGE_WIFI_STATE | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/reference/android/Manifest.permission#CHANGE_WIFI_STATE.

[44] INTERNET | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/reference/android/Manifest.permission#INTERNET.

[45] ACCESS_COARSE_LOCATION | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/reference/android/Manifest.permission#ACCESS_COARSE_LOCATION.

[46] Kotaru, M., Joshi, K., Bharadia, D., & Katti, S. (2015, August). Spotfi: Decimeter level localization using wifi. In *ACM SIGCOMM Computer Communication Review* (Vol. 45, No. 4, pp. 269-282). ACM.

[47] Mozilla Location Service - Overview. (2018). Retrieved 23 November 2018, from https://location.services.mozilla.com/.

[48] Access to Hardware Identifier | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/about/versions/marshmallow/android-6.0-changes#behavior-hardware-id.

[49] VIBRATE | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/reference/android/Manifest.permission#VIBRATE.

[50] WAKE_LOCK | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/reference/android/Manifest.permission#WAKE_LOCK

[51] Corbet, J. (2012). Autosleep and wake locks [LWN.net]. Retrieved 23 November 2018, from https://lwn.net/Articles/479841/.

[52] Keep the device awake | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/training/scheduling/wakelock.

[53] Hay, R. (2013). A New Vulnerability in the Android Framework: Fragment Injection. Retrieved 23 November 2018, from https://securityintelligence.com/new-vulnerability-android-framework-fragment-injection/.

[54] Use services sparingly | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/topic/performance/memory#Services.

[55] Avoid memory churn | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/topic/performance/memory#churn.

[56] Avoid creating unnecessary objects | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/training/articles/perf-tips#ObjectCreation.

[57] Effects on process state | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/guide/components/broadcasts#effects-process-state.

[58] v4/java/android/support/v4/content/WakefulBroadcastReceiver.java - platform/frameworks/support - Git at Google. (2013). Retrieved 23 November 2018, from https://android.googlesource.com/platform/frameworks/support/+/android-support-lib-19.1.0/v4/java/android/support/v4/content/WakefulBroadcastReceiver.java.

[59] WakefulBroadcastReceiver | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/reference/android/support/v4/content/WakefulBroadcastReceiver.

[60] core/java/android/app/IntentService.java - platform/frameworks/base - Git at Google. (2008). Retrieved 23 November 2018, from https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/app/IntentService.java.

[61] HttpURLConnection | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/reference/java/net/HttpURLConnection.

[62] OkHttp. (2018). Retrieved 23 November 2018, from http://square.github.io/okhttp/.

[63] Apache HTTP Client Removal | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/about/versions/marshmallow/android-6.0-changes#behavior-apache-http-client.

[64] SAXParser | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/reference/javax/xml/parsers/SAXParser.

[65] XPath | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/reference/javax/xml/xpath/XPath.

[66] Wilson, M. (2008). T-Mobile G1: Full Details of the HTC Dream Android Phone. Retrieved 23 November 2018, from https://gizmodo.com/5053264/t-mobile-g1-full-details-of-the-htc-dream-android-phone.

[67] Bettiol, M. (2009). Rogers' HTC Dream and Magic to be deprived of Donut. Retrieved 23 November 2018, from https://bgr.com/2009/12/19/rogers-htc-dream-and-magic-to-be-deprived-of-donut/.

[68] Notifications | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/about/versions/marshmallow/android-6.0-changes#behavior-notifications.

[69] Meet Google Play's target API level requirement | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/distribute/best-practices/develop/target-sdk.

[70] F-Droid - Free and Open Source Android App Repository. (2018). Retrieved 23 November 2018, from https://f-droid.org/.

[71] Aptoide - Own Your Android Market. (2018). Retrieved 23 November 2018, from https://www.aptoide.com/.

[72] Git. (2018). Retrieved 23 November 2018, from https://git-scm.com/.

[73] Keynote: State of the Linux Kernel, Greg Kroah-Hartman. (2016). Retrieved 23 November 2018, from https://www.youtube.com/watch?v=SIQr2-Dh0es.

[74] Tech Talk: Linus Torvalds on git. (2007). Retrieved 23 November 2018, from https://www.youtube.com/watch?v=4XpnKHJAok8.

[75] Salvaterra, R. (2016). FonMan — Yet another Fon authentication app for Android (loosely based on Joan Fisbein's FONAccess). Retrieved 23 November 2018, from https://github.com/rsalvaterra/Fon.

[76] The first single application for the entire DevOps lifecycle - GitLab. (2018). Retrieved 23 November 2018, from https://about.gitlab.com/.

[77] CaptivePortal | Android Developers. (2018). Retrieved 23 November 2018, from https://developer.android.com/reference/android/net/CaptivePortal.

[78] available SSIDs in LoginManager.java · Issue #2 · rsalvaterra/Fon. (2015). Retrieved 23 November 2018, from https://github.com/rsalvaterra/Fon/issues/2.

Anexo 1 — Mensagens XML

Apresentam-se neste anexo exemplos de mensagens XML que a aplicação obtém do RADIUS Fon. Neste caso particular, trata-se do portal cativo do fornecedor de acesso NOS, parceiro da Fon em Portugal. As mensagens são embebidas nos corpos (elemento `<body>`) das páginas HTML usadas no início de sessão normal, através de um navegador *web*. Por esse motivo, antes de serem interpretadas como XML, necessitam de pré-processamento, sendo substituídas as entidades HTML reservadas pelos respectivos caracteres. As mensagens apresentadas neste anexo encontram-se já pré-processadas.

Redireccionamento UAM

Esta mensagem encontra-se embebida na página de início de sessão obtida a partir do RADIUS (o segundo pedido do cliente no diagrama de sequência), após o redireccionamento inicial. Tem a estrutura de uma mensagem WISP padrão, de acordo com o *draft* de referência.

O elemento `<AccessProcedure>` identifica a versão do protocolo WISPr (1.0). `<LoginURL>` é o elemento que contém o URL ao qual será efectuado o pedido POST, sendo enviadas as credenciais num formulário HTML codificado em MIME como application/x-www-form-urlencoded. O elemento `<AbortLoginURL>` seria, supostamente, o URL ao qual um pedido GET interromperia a operação de autenticação (apenas está presente para manter a estrutura padrão da mensagem, dado que não é usado). Os elementos `<MessageType>` e `<ResponseCode>` contêm, respectivamente, o tipo de mensagem XML com que estamos a lidar (100, redireccionamento inicial) e o código de resposta (0, nenhum erro). Por fim, o elemento `<AccessLocation>` contém o identificador do dispositivo através do qual será efectuado o acesso.

```
<?xml version="1.0" encoding="UTF-8"?>
<WISPAccessGatewayParam xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.acmewisp.com/WISPAccessGatewayParam.xsd">
  <Redirect>
    <AccessProcedure>1.0</AccessProcedure>

<LoginURL>https://captiveportal.nos.pt/noswifi?hmac=ce3fb53e6519862b2e9c8d06e0b8693
22d5ff55ba4700380c8c02cd280f0a870&res=notyet&uamip=192.168.3.1&uamport=80&userurl=&
```

```
challenge=0144cd54f91b522aa25060cc8a6abb22&nasid=84-94-8C-80-D4-85&mac=6C-71-D9-FF-
7B-79</LoginURL>
    <AbortLoginURL>http://192.168.3.1:80/logoff</AbortLoginURL>
    <MessageType>100</MessageType>
    <ResponseCode>0</ResponseCode>
    <AccessLocation>FonZON:PT</AccessLocation>
  </Redirect>
</WISPAccessGatewayParam>
```

Resposta da Autenticação (Bem-sucedida)

Esta mensagem é obtida na resposta ao pedido de autorização redireccionado (o último pedido do cliente no diagrama de sequência), após a autenticação no RADIUS. Também ela tem a estrutura de uma mensagem WISP padrão, de acordo com o *draft* de referência.

Mais simples que a anterior, esta apenas contém três elementos. `<MessageType>` e `<ResponseCode>` consistem, novamente, no tipo de mensagem XML corrente (120, notificação de autenticação) e no código de resposta (50, início de sessão efectuado com sucesso). O elemento `<LogoffURL>` contém o URL ao qual um pedido HTTP GET termina a sessão (trata-se de uma funcionalidade opcional, dado que a perda de conectividade com o ponto de acesso wi-fi termina, implicitamente, a sessão).

```
<?xml version="1.0" encoding="UTF-8"?>
<WISPAccessGatewayParam xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.acmewisp.com/WISPAccessGatewayParam.xsd">
  <AuthenticationReply>
    <MessageType>120</MessageType>
    <ResponseCode>50</ResponseCode>
    <LogoffURL>http://192.168.3.1:80/logoff</LogoffURL>
  </AuthenticationReply>
</WISPAccessGatewayParam>
```

Anexo 2 — Código Fonte

Apresenta-se, neste anexo, o conteúdo do código fonte da aplicação, separado por cada um dos ficheiros que a constituem.

A ausência de comentários é deliberada, na medida em que não se trata de uma biblioteca fechada cuja API é exposta (requerendo, necessariamente, documentação), mas de uma aplicação completa, cujo código fonte é aberto e se encontra integralmente disponível para ser analisado e estudado. Por esse motivo, foi tomado o máximo cuidado possível para garantir que todos os identificadores (interfaces, classes, constantes, variáveis, procedimentos, predicados e métodos) apresentem a carga semântica necessária e suficiente para tornar autoevidente o seu propósito à audiência alvo (programadores com experiência de desenvolvimento em Android).

Constants.java

```
package org.rsalvaterra.fon;

interface Constants {

  int SECONDS = 1000;

  int CRC_ALREADY_AUTHORISED = 1000;
  int CRC_CREDENTIALS_ERROR = 1001;
  int CRC_WISPR_NOT_PRESENT = 1002;

  int FRC_GENERAL_ERROR = 900;
  int FRC_NOT_ENOUGH_CREDIT = 901;
  int FRC_BAD_CREDENTIALS = 902;
  int FRC_BLACKLISTED = 903;
  int FRC_USER_LIMIT_EXCEEDED = 905;
  int FRC_HOTSPOT_LIMIT_EXCEEDED = 906;
  int FRC_NOT_AUTHORIZED = 907;
  int FRC_PARTNER_ERROR = 908;
  int FRC_INTERNAL_ERROR = 909;
  int FRC_UNKNOWN_ERROR = 910;
  int FRC_INVALID_TEMPORARY_CREDENTIAL = 911;
  int FRC_AUTHORIZATION_CONNECTION_ERROR = 912;
  int FRC_BANNED_REALM = 913;
```

```
int FRC_BANNED_USER = 914;
int FRC_FLOOD_LIMIT_EXCEEDED = 915;
int FRC_DATACAP_EXCEEDED = 916;
int FRC_USER_SUSPENDED = 917;
int FRC_DEVICE_NOT_AUTHORIZED = 918;
int FRC_NOT_CREDENTIALS = 920;
int FRC_INSERT_PROMO_CODE = 922;
int FRC_VIEW_PROMO_CODE = 923;
int FRC_NOT_PASS_PURCHASE_AVAILABLE = 924;
int FRC_NOT_ENOUGH_CREDIT_2 = 925;
int FRC_INTERNAL_ERROR_2 = 926;
int FRC_NOT_AUTHORIZED_OWN_HOTSPOT = 927;

int WMT_INITIAL_REDIRECT = 100;
int WMT_PROXY_NOTIFICATION = 110;
int WMT_AUTH_NOTIFICATION = 120;
int WMT_LOGOFF_NOTIFICATION = 130;
int WMT_RESPONSE_AUTH_POLL = 140;
int WMT_RESPONSE_ABORT_LOGIN = 150;

int WRC_NO_ERROR = 0;
int WRC_LOGIN_SUCCEEDED = 50;
int WRC_LOGIN_FAILED = 100;
int WRC_RADIUS_ERROR = 102;
int WRC_NETWORK_ADMIN_ERROR = 105;
int WRC_LOGOFF_SUCCEEDED = 150;
int WRC_LOGING_ABORTED = 151;
int WRC_PROXY_DETECTION = 200;
int WRC_AUTH_PENDING = 201;
int WRC_ACCESS_GATEWAY_INTERNAL_ERROR = 255;

String DEFAULT_PERIOD = "300";
String DEFAULT_MINIMUM_RSSI = "-80";

String APP_ID = "org.rsalvaterra.fon";

String ACT_CLEANUP = "0";
String ACT_CHECK = "1";
String ACT_CONNECT = "2";
String ACT_LOGIN = "3";
String ACT_SCAN = "4";
```

```
}
```

FonManAlarmReceiver.java

```
package org.rsalvaterra.fon;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public final class FonManAlarmReceiver extends BroadcastReceiver {

  @Override
  public void onReceive(final Context c, final Intent i) {
     FonManService.execute(c, i.getAction());
  }

}
```

FonManService.java

```
package org.rsalvaterra.fon;

import android.app.AlarmManager;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.net.Uri;
import android.net.wifi.ScanResult;
import android.net.wifi.SupplicantState;
import android.net.wifi.WifiConfiguration;
import android.net.wifi.WifiConfiguration.KeyMgmt;
import android.net.wifi.WifiInfo;
import android.net.wifi.WifiManager;
import android.os.Handler;
import android.os.Handler.Callback;
import android.os.HandlerThread;
```

```
import android.os.IBinder;
import android.os.Message;
import android.os.PowerManager;
import android.os.PowerManager.WakeLock;
import android.os.SystemClock;
import android.preference.PreferenceManager;

import java.util.Arrays;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;

public final class FonManService extends Service implements Callback,
Comparator<ScanResult> {

  private static final int NOTIFICATION_ID = 1;
  private static final int REQUEST_CODE = 1;
  private static final int CONNECTIVITY_CHECK_PERIOD = 60 * Constants.SECONDS;
  private static final int BLACKLIST_PERIOD = 300 * Constants.SECONDS;

  private static final long WAKELOCK_TIMEOUT = 60 * Constants.SECONDS;

  private static final long[] VIBRATE_PATTERN_FAILURE = { 100, 250, 100, 250 };
  private static final long[] VIBRATE_PATTERN_SUCCESS = { 100, 250 };

  private static final String WAKELOCK_ID = Constants.APP_ID + ":wakelock";

  private static final HashMap<String, Long> BLACKLIST = new HashMap<>();

  private static volatile WakeLock WAKELOCK;

  private final Handler messageHandler;

  {
     final HandlerThread ht = new HandlerThread(Constants.APP_ID);
     ht.start();
     messageHandler = new Handler(ht.getLooper(), this);
  }

  private static void addToBlacklist(final String bssid) {
     FonManService.BLACKLIST.put(bssid, SystemClock.elapsedRealtime() +
FonManService.BLACKLIST_PERIOD);
```

```
  }

  private static boolean getPreference(final Context c, final int id, final boolean
v) {
     return FonManService.getPreferences(c).getBoolean(c.getString(id), v);
  }

  private static SharedPreferences getPreferences(final Context c) {
     return PreferenceManager.getDefaultSharedPreferences(c);
  }

  private static boolean isBlacklisted(final String bssid) {
     final Long t = FonManService.BLACKLIST.get(bssid);
     if (t != null) {
        if (t > SystemClock.elapsedRealtime()) {
           return true;
        }
        FonManService.BLACKLIST.remove(bssid);
     }
     return false;
  }

  private static boolean isBt(final String ssid) {
     return ssid.equals("BTFON") || ssid.equals("BTWiFi") ||
ssid.equals("BTWifi-with-FON") || ssid.equals("BTWiFi-with-FON") ||
ssid.startsWith("BTOpenzone");
  }

  private static boolean isConnected(final SupplicantState ss) {
     return (ss == SupplicantState.COMPLETED);
  }

  private static boolean isDisconnected(final SupplicantState ss) {
     return (ss == SupplicantState.INACTIVE) || (ss == SupplicantState.DORMANT) ||
(ss == SupplicantState.DISCONNECTED) || (ss == SupplicantState.SCANNING);
  }

  private static boolean isDowntownBrooklyn(final String ssid) {
     return ssid.equals("DowntownBrooklynWiFi_Fon");
  }

  private static boolean isDt(final String ssid) {
```

```
      return ssid.equals("Telekom_FON");
   }

   private static boolean isGenericFon(final String ssid) {
      return ssid.startsWith("FON_");
   }

   private static boolean isHt(final String ssid) {
      return ssid.equals("HotSpot Fon");
   }

   private static boolean isInsecure(final ScanResult sr) {
      return !(sr.capabilities.contains("WEP") || sr.capabilities.contains("PSK") ||
sr.capabilities.contains("EAP"));
   }

   private static boolean isInsecure(final WifiConfiguration wc) {
      for (final String s : wc.wepKeys) {
         if (s != null) {
            return false;
         }
      }
      return wc.allowedKeyManagement.get(KeyMgmt.NONE);
   }

   private static boolean isJt(final String ssid) {
      return ssid.equalsIgnoreCase("JT Fon");
   }

   private static boolean isKpn(final String ssid) {
      return ssid.equals("KPN Fon");
   }

   private static boolean isMt(final String ssid) {
      return ssid.equals("Telekom Fon");
   }

   private static boolean isMweb(final String ssid) {
      return ssid.equals("@MWEB FON");
   }

   private static boolean isNos(final String ssid) {
```

```
      return ssid.equals("NOS_WIFI_Fon");
    }

    private static boolean isOi(final String ssid) {
      return ssid.equals("Oi WiFi Fon") || ssid.equals("OI_WIFI_FON") ||
ssid.equals("Coca-Cola WiFi");
    }

    private static boolean isOte(final String ssid) {
      return ssid.equals("OTE WiFi Fon");
    }

    private static boolean isOtherFon(final String ssid) {
      return ssid.startsWith("Fon WiFi");
    }

    private static boolean isProximus(final String ssid) {
      return ssid.equals("PROXIMUS_FON");
    }

    private static boolean isRomtelecom(final String ssid) {
      return ssid.equals("Romtelecom Fon");
    }

    private static boolean isSfr(final String ssid) {
      return ssid.equals("SFR WiFi FON");
    }

    private static boolean isSoftbank(final String ssid) {
      return ssid.equals("FON");
    }

    private static boolean isSupported(final String ssid) {
      return FonManService.isNos(ssid) || FonManService.isGenericFon(ssid) ||
FonManService.isVodafoneSpain(ssid) || FonManService.isBt(ssid) ||
FonManService.isJt(ssid) || FonManService.isSfr(ssid) ||
FonManService.isVodafoneItaly(ssid) || FonManService.isProximus(ssid) ||
FonManService.isKpn(ssid) || FonManService.isDt(ssid) || FonManService.isHt(ssid)
|| FonManService.isMt(ssid) || FonManService.isOte(ssid) ||
FonManService.isRomtelecom(ssid) || FonManService.isOi(ssid) ||
FonManService.isDowntownBrooklyn(ssid) || FonManService.isMweb(ssid) ||
```

```
FonManService.isOtherFon(ssid) || FonManService.isSoftbank(ssid) ||
FonManService.isTelstra(ssid);
  }

  private static boolean isTelstra(final String ssid) {
     return ssid.equals("Telstra AIR");
  }

  private static boolean isVodafoneItaly(final String ssid) {
     return ssid.equals("Vodafone-WIFI");
  }

  private static boolean isVodafoneSpain(final String ssid) {
     return ssid.equals("_ONOWiFi");
  }

  private static String stripQuotes(final String ssid) {
     final int length = ssid.length();
     if ((ssid.charAt(0) == '"') && (ssid.charAt(length - 1) == '"')) {
        return ssid.substring(1, length - 1);
     }
     return ssid;
  }

  private static void wakeLockAcquire(final Context c) {
     if (FonManService.WAKELOCK == null) {
        synchronized (FonManService.class) {
           if (FonManService.WAKELOCK == null) {
              final PowerManager pm = (PowerManager)
c.getSystemService(Context.POWER_SERVICE);
              if (pm != null) {
                 // Paranoia
                 FonManService.WAKELOCK =
pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, FonManService.WAKELOCK_ID);
              }
           }
        }
     } else if (!FonManService.WAKELOCK.isHeld()) {
        FonManService.WAKELOCK.acquire(FonManService.WAKELOCK_TIMEOUT);
     }
  }
```

```
  private static void wakeLockRelease() {
     if ((FonManService.WAKELOCK != null) && FonManService.WAKELOCK.isHeld()) {
        FonManService.WAKELOCK.release();
     }
  }

  static void execute(final Context c, final String a) {
     FonManService.wakeLockAcquire(c);
     c.startService(new Intent(c, FonManService.class).setAction(a));
  }

  static String getPreference(final Context c, final int id, final String v) {
     return FonManService.getPreferences(c).getString(c.getString(id), v);
  }

  static boolean isAutoConnectEnabled(final Context c) {
     return FonManService.getPreference(c, R.string.kautoconnect, true);
  }

  private WifiConfiguration[] getConfiguredNetworks(final WifiManager wm) {
     final List<WifiConfiguration> wcl = wm.getConfiguredNetworks();
     final WifiConfiguration[] wca;
     if (wcl == null) {
        wca = new WifiConfiguration[] {};
     } else {
        wca = wcl.toArray(new WifiConfiguration[] {});
     }
     return wca;
  }

  private int updateOrAddFonConfiguration(final WifiManager wm, final
WifiConfiguration[] wca, final ScanResult sr) {
     final String ssid = '"' + sr.SSID + '"';
     if (wca.length != 0) {
        for (final WifiConfiguration wc : wca) {
           if (FonManService.isInsecure(wc) && wc.SSID.equals(ssid)) {
              wc.BSSID = sr.BSSID;
              return wm.updateNetwork(wc);
           }
        }
     }
     final WifiConfiguration wc = new WifiConfiguration();
```

```
      wc.SSID = ssid;
      wc.BSSID = sr.BSSID;
      wc.allowedKeyManagement.set(KeyMgmt.NONE);
      return wm.addNetwork(wc);
   }

   private void check(final WifiManager wm) {
      login(wm, true);
   }

   private void cleanUp() {
      stopPeriodicConnectivityCheck();
      stopPeriodicScan();
      removeNotification();
   }

   private void connect(final WifiManager wm) {
      final WifiInfo wi = wm.getConnectionInfo();
      final SupplicantState ss = wi.getSupplicantState();
      if (FonManService.isDisconnected(ss)) {
         final WifiConfiguration[] wca = getConfiguredNetworks(wm);
         final ScanResult[] sra = getScanResults(wm);
         int id = getOtherId(wca, sra, false);
         if (id == -1) {
            id = getFonId(wm, wca, sra);
            if ((id != -1) && wm.enableNetwork(id, true) && isReconnectEnabled()) {
               startPeriodicScan();
            }
         }
      } else if (FonManService.isConnected(ss) && isReconnectEnabled() &&
FonManService.isSupported(FonManService.stripQuotes(wi.getSSID()))) {
         final int id = getOtherId(getConfiguredNetworks(wm), getScanResults(wm),
isSecureEnabled());
         if (id != -1) {
            wm.enableNetwork(id, true);
         }
      }
   }

   private String getFailureTone() {
      return FonManService.getPreference(this, R.string.kfailure, "");
   }
```

```
  private int getFonId(final WifiManager wm, final WifiConfiguration[] wca, final
ScanResult[] sra) {
     final int mr = getMinimumRssi();
     for (final ScanResult sr : sra) {
        if (sr.level < mr) {
           break;
        }
        if (FonManService.isSupported(sr.SSID) && FonManService.isInsecure(sr) &&
!FonManService.isBlacklisted(sr.BSSID)) {
           return updateOrAddFonConfiguration(wm, wca, sr);
        }
     }
     return -1;
  }

  private int getMinimumRssi() {
     if (FonManService.getPreference(this, R.string.kreject, false)) {
        return Integer.parseInt(FonManService.getPreference(this, R.string.krssi,
Constants.DEFAULT_MINIMUM_RSSI));
     }
     return Integer.MIN_VALUE;
  }

  private int getOtherId(final WifiConfiguration[] wca, final ScanResult[] sra,
final boolean secure) {
     if (wca.length != 0) {
        final HashMap<String, Integer> wcm = new HashMap<>();
        for (final WifiConfiguration wc : wca) {
           final String ssid = FonManService.stripQuotes(wc.SSID);
           if (!((secure && FonManService.isInsecure(wc)) ||
FonManService.isSupported(ssid))) {
              wcm.put(ssid, wc.networkId);
           }
        }
        final int mr = getMinimumRssi();
        for (final ScanResult sr : sra) {
           if (sr.level < mr) {
              break;
           }
           final Integer id = wcm.get(sr.SSID);
           if (id != null) {
```

```
                return id;
            }
        }
    }
    return -1;
  }

  private String getPassword() {
    return FonManService.getPreference(this, R.string.kpassword, "");
  }

  private int getPeriod() {
    return Integer.parseInt(FonManService.getPreference(this, R.string.kperiod,
Constants.DEFAULT_PERIOD));
  }

  private ScanResult[] getScanResults(final WifiManager wm) {
    final List<ScanResult> srl = wm.getScanResults();
    final ScanResult[] sra = srl.toArray(new ScanResult[] {});
    Arrays.sort(sra, this);
    return sra;
  }

  private String getSuccessTone() {
    return FonManService.getPreference(this, R.string.ksuccess, "");
  }

  private String getUsername() {
    return FonManService.getPreference(this, R.string.kusername, "");
  }

  private void handleError(final WifiManager wm, final WifiInfo wi, final String[]
lr) {
    if (FonManService.isAutoConnectEnabled(this)) {
      FonManService.addToBlacklist(wi.getBSSID());
      wm.removeNetwork(wi.getNetworkId());
    } else {
      notifyFonError(lr);
    }
  }

  private void handleSuccess(final String ssid, final boolean check) {
```

```
        if (check) {
            return;
        }
        notify(getString(R.string.started), FonManService.VIBRATE_PATTERN_SUCCESS,
Notification.FLAG_NO_CLEAR | Notification.FLAG_ONLY_ALERT_ONCE |
Notification.FLAG_ONGOING_EVENT, getSuccessTone(), getString(R.string.connected,
ssid), PendingIntent.getActivity(this, 0, new Intent(),
PendingIntent.FLAG_UPDATE_CURRENT));
        startPeriodicConnectivityCheck();
    }

    private boolean isReconnectEnabled() {
        return FonManService.getPreference(this, R.string.kreconnect, false);
    }

    private boolean isSecureEnabled() {
        return FonManService.getPreference(this, R.string.ksecure, true);
    }

    private boolean isSoundEnabled() {
        return FonManService.getPreference(this, R.string.knotify, true);
    }

    private boolean isVibrationEnabled() {
        return FonManService.getPreference(this, R.string.kvibrate, false);
    }

    private void login(final WifiManager wm) {
        login(wm, false);
    }

    private void login(final WifiManager wm, final boolean check) {
        final WifiInfo wi = wm.getConnectionInfo();
        String ssid = wi.getSSID();
        if (ssid == null) {
            return;
        }
        ssid = FonManService.stripQuotes(ssid);
        if (!FonManService.isSupported(ssid)) {
            return;
        }
        final String[] lr = LoginManager.login(getUsername(), getPassword());
```

```
      switch (Integer.parseInt(lr[0])) {
         case Constants.WRC_LOGIN_SUCCEEDED:
         case Constants.CRC_ALREADY_AUTHORISED:
            handleSuccess(ssid, check);
            break;
         case Constants.WRC_RADIUS_ERROR:
         case Constants.WRC_NETWORK_ADMIN_ERROR:
         case Constants.FRC_HOTSPOT_LIMIT_EXCEEDED:
         case Constants.FRC_UNKNOWN_ERROR:
         case Constants.CRC_WISPR_NOT_PRESENT:
            handleError(wm, wi, lr);
            break;
         case Constants.WRC_ACCESS_GATEWAY_INTERNAL_ERROR:
            wm.removeNetwork(wi.getNetworkId());
            break;
         case Constants.FRC_BAD_CREDENTIALS:
         case Constants.CRC_CREDENTIALS_ERROR:
            notifyCredentialsError();
            break;
         default:
            notifyFonError(lr);
            break;
      }
   }

   private void notify(final String title, final long[] vibratePattern, final int
flags, final String ringtone, final String text, final PendingIntent pi) {
      final Notification n = new Notification();
      n.flags |= flags;
      n.icon = R.drawable.ic_stat_fon;
      if (isSoundEnabled()) {
         n.sound = Uri.parse(ringtone);
         if (isVibrationEnabled()) {
            n.vibrate = vibratePattern;
         }
      }
      n.setLatestEventInfo(this, title, text, pi);
      final NotificationManager nm = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
      if (nm != null) {
         // Paranoia
         nm.notify(FonManService.NOTIFICATION_ID, n);
```

```
    }
  }

  private void notifyCredentialsError() {
    notifyError(getString(R.string.cred_error));
  }

  private void notifyError(final String title) {
    notify(title, FonManService.VIBRATE_PATTERN_FAILURE, Notification.FLAG_NO_CLEAR
| Notification.FLAG_ONGOING_EVENT, getFailureTone(), getString(R.string.configure),
PendingIntent.getActivity(this, FonManService.REQUEST_CODE, new Intent(this,
SettingsActivity.class).addFlags(Intent.FLAG_ACTIVITY_NEW_TASK),
PendingIntent.FLAG_UPDATE_CURRENT));
  }

  private void notifyFonError(final String[] lr) {
    notifyError(getString(R.string.fon_error, Integer.parseInt(lr[0]), lr[1]));
  }

  private void removeNotification() {
    final NotificationManager nm = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
    if (nm != null) {
      // Paranoia
      nm.cancel(FonManService.NOTIFICATION_ID);
    }
  }

  private void startPeriodicAction(final int milliseconds, final String action) {
    final AlarmManager am = (AlarmManager)
getSystemService(Context.ALARM_SERVICE);
    if (am != null) {
      // Paranoia
      am.setRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
SystemClock.elapsedRealtime() + milliseconds, milliseconds,
PendingIntent.getBroadcast(this, FonManService.REQUEST_CODE, new Intent(this,
FonManAlarmReceiver.class).setAction(action), PendingIntent.FLAG_UPDATE_CURRENT));
    }
  }

  private void startPeriodicConnectivityCheck() {
```

```
      startPeriodicAction(FonManService.CONNECTIVITY_CHECK_PERIOD,
Constants.ACT_CHECK);
   }

   private void startPeriodicScan() {
      startPeriodicAction(getPeriod() * Constants.SECONDS, Constants.ACT_SCAN);
   }

   private void stopPeriodicAction(final String action) {
      final AlarmManager am = (AlarmManager)
getSystemService(Context.ALARM_SERVICE);
      if (am != null) {
         am.cancel(PendingIntent.getBroadcast(this, FonManService.REQUEST_CODE, new
Intent(this, FonManAlarmReceiver.class).setAction(action),
PendingIntent.FLAG_UPDATE_CURRENT));
      }
   }

   private void stopPeriodicConnectivityCheck() {
      stopPeriodicAction(Constants.ACT_CHECK);
   }

   private void stopPeriodicScan() {
      stopPeriodicAction(Constants.ACT_SCAN);
   }

   @Override
   public int compare(final ScanResult sr1, final ScanResult sr2) {
      return sr2.level - sr1.level;
   }

   @Override
   public boolean handleMessage(final Message m) {
      final String s = (String) m.obj;
      if (s.equals(Constants.ACT_CLEANUP)) {
         cleanUp();
      } else {
         final WifiManager wm = (WifiManager)
getApplicationContext().getSystemService(Context.WIFI_SERVICE);
         if (wm != null) {
            // Paranoia
            switch (s) {
```

```
                case Constants.ACT_CHECK:
                    check(wm);
                    break;
                case Constants.ACT_CONNECT:
                    connect(wm);
                    break;
                case Constants.ACT_LOGIN:
                    login(wm);
                    break;
                case Constants.ACT_SCAN:
                    wm.startScan();
                    break;
            }
        }
    }
    FonManService.wakeLockRelease();
    return true;
  }

  @Override
  public IBinder onBind(final Intent i) {
    return null;
  }

  @Override
  public void onDestroy() {
    messageHandler.getLooper().quit();
  }

  @Override
  public void onStart(final Intent i, final int id) {
    final Message m = Message.obtain();
    m.obj = i.getAction();
    messageHandler.sendMessage(m);
  }

}
```

FonManWifiReceiver.java

```
package org.rsalvaterra.fon;
```

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.net.NetworkInfo.State;
import android.net.wifi.WifiManager;

public final class FonManWifiReceiver extends BroadcastReceiver {

  @Override
  public void onReceive(final Context c, final Intent i) {
     final String a = i.getAction();
     if (a == null) {
        return;
     }
     if (a.equals(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION) &&
FonManService.isAutoConnectEnabled(c)) {
        FonManService.execute(c, Constants.ACT_CONNECT);
     } else if (a.equals(WifiManager.NETWORK_STATE_CHANGED_ACTION)) {
        final NetworkInfo ni =
i.getParcelableExtra(WifiManager.EXTRA_NETWORK_INFO);
        if (ni.getType() == ConnectivityManager.TYPE_WIFI) {
           final State s = ni.getState();
           if (s == State.CONNECTED) {
              FonManService.execute(c, Constants.ACT_LOGIN);
           } else if (s == State.DISCONNECTED) {
              FonManService.execute(c, Constants.ACT_CLEANUP);
           }
        }
     }
  }

}
```

LoginManager.java

```
package org.rsalvaterra.fon;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;

final class LoginManager {

  private static final int MAX_REDIRECTS = 5;
  private static final int HTTP_CONNECT_TIMEOUT = 30 * Constants.SECONDS;
  private static final int HTTP_READ_TIMEOUT = 30 * Constants.SECONDS;

  private static final String CONNECTED = "CONNECTED";
  private static final String CONNECTION_TEST_URL =
"http://cm.fon.mobi/android.txt";
  private static final String CONTENT_LENGTH = "Content-Length";
  private static final String CONTENT_TYPE = "Content-Type";
  private static final String CONTENT_TYPE_STRING =
"application/x-www-form-urlencoded";
  private static final String FON_WISPR_PREFIX = "FON_WISPR/";
  private static final String FON_ROAM_PREFIX = "FON_ROAM/";
  private static final String LOCATION = "Location";
  private static final String SAFE_PROTOCOL = "https://";
  private static final String TAG_FON_RESPONSE_CODE = "FONResponseCode";
  private static final String TAG_LOGIN_URL = "LoginURL";
  private static final String TAG_MESSAGE_TYPE = "MessageType";
  private static final String TAG_REPLY_MESSAGE = "ReplyMessage";
  private static final String TAG_RESPONSE_CODE = "ResponseCode";
  private static final String TAG_WISPR = "WISPAccessGatewayParam";
  private static final String USER_AGENT = "User-Agent";
  private static final String USER_AGENT_STRING = "FonMan; wispr; (Linux; U;
Android)";
  private static final String USER_NAME = "UserName=";
  private static final String UTF_8 = "UTF-8";
  private static final String PASSWORD = "&Password=";

  private static final String[] VALID_SUFFIX = { ".nos.pt", ".fon.com",
".btopenzone.com", ".btfon.com", ".wifi.sfr.fr", ".hotspotsvankpn.com",
".portal.vodafone-wifi.com" };

  private LoginManager() {}
```

```
  private static HttpURLConnection buildConnection(final String url) throws
IOException {
     final HttpURLConnection conn = (HttpURLConnection) new
URL(url).openConnection();
     conn.setRequestProperty(LoginManager.USER_AGENT,
LoginManager.USER_AGENT_STRING);
     conn.setConnectTimeout(LoginManager.HTTP_CONNECT_TIMEOUT);
     conn.setReadTimeout(LoginManager.HTTP_READ_TIMEOUT);
     return conn;
  }

  private static String getElementText(final String s, final String e) {
     final int start = s.indexOf(">", s.indexOf(e));
     if (start != -1) {
        final int end = s.indexOf("</" + e, start);
        if (end != -1) {
           return s.substring(start + 1, end);
        }
     }
     return "";
  }

  private static int getElementTextAsInt(final String s, final String e) {
     return Integer.parseInt(LoginManager.getElementText(s, e));
  }

  private static String getPrefixedUserName(final String host, final String user) {
     if (!(host.contains("belgacom") || host.contains("telekom"))) {
        if ((host.contains("portal.fon.com") || host.contains("wifi.sfr.fr") ||
host.equals("www.btopenzone.com"))) {
           return LoginManager.FON_WISPR_PREFIX + user;
        } else if (host.contains("portal.vodafone-wifi.com")) {
           return LoginManager.FON_WISPR_PREFIX + LoginManager.FON_ROAM_PREFIX +
user;
        }
     }
     return user;
  }

  private static String getTestUrl() {
     try {
        String target = LoginManager.CONNECTION_TEST_URL;
```

```
        int redirects = 0;
        do {
           final HttpURLConnection conn = LoginManager.buildConnection(target);
           if (LoginManager.isNoRedirect(conn.getResponseCode())) {
              return LoginManager.readStream(conn);
           }
           target = conn.getHeaderField(LoginManager.LOCATION);
           conn.disconnect();
        } while (++redirects != LoginManager.MAX_REDIRECTS);
     } catch (final IOException x) {
        // Nothing to do.
     }
     return null;
  }

  private static boolean isNoRedirect(final int rc) {
     return rc != HttpURLConnection.HTTP_MOVED_PERM && rc !=
HttpURLConnection.HTTP_MOVED_TEMP && rc != HttpURLConnection.HTTP_SEE_OTHER;
  }

  private static String login(final String url, final String user, final String
pass) {
     if (url.startsWith(LoginManager.SAFE_PROTOCOL)) {
        for (final String s : LoginManager.VALID_SUFFIX) {
           final int b = LoginManager.SAFE_PROTOCOL.length();
           final int e = url.indexOf("/", b);
           if (e > b) {
              final String h = url.substring(b, e);
              if (h.endsWith(s)) {
                 try {
                    HttpURLConnection conn =
LoginManager.buildConnection(LoginManager.replaceAmpEntities(url));
                    conn.setDoOutput(true);
                    final byte[] pc = (LoginManager.USER_NAME +
URLEncoder.encode(LoginManager.getPrefixedUserName(h, user), LoginManager.UTF_8) +
LoginManager.PASSWORD + URLEncoder.encode(pass, LoginManager.UTF_8)).getBytes();
                    conn.setRequestProperty(LoginManager.CONTENT_LENGTH,
Integer.toString(pc.length));
                    conn.setRequestProperty(LoginManager.CONTENT_TYPE,
LoginManager.CONTENT_TYPE_STRING);
                    final OutputStream os = conn.getOutputStream();
                    os.write(pc);
```

```
                    os.close();
                    int redirects = 0;
                    do {
                       if (LoginManager.isNoRedirect(conn.getResponseCode())) {
                          return LoginManager.readStream(conn);
                       }
                       final String target =
conn.getHeaderField(LoginManager.LOCATION);
                       conn.disconnect();
                       conn = LoginManager.buildConnection(target);
                    } while (++redirects != LoginManager.MAX_REDIRECTS);
                    conn.disconnect();
                 } catch (final IOException x) {
                    // Nothing to do.
                 }
                 break;
              }
           }
        }
     }
     return null;
  }

  private static String readStream(final HttpURLConnection conn) throws IOException
{
     final BufferedReader br = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
     String s;
     final StringBuilder sb = new StringBuilder();
     while ((s = br.readLine()) != null) {
        sb.append(s);
     }
     br.close();
     conn.disconnect();
     return sb.toString();
  }

  private static String replaceAmpEntities(final String s) {
     return s.replace("&amp;", "&");
  }

  static String[] login(final String user, final String pass) {
```

```
    int rc = Constants.WRC_ACCESS_GATEWAY_INTERNAL_ERROR;
    String rm = "";
    if ((user.length() != 0) && (pass.length() != 0)) {
       String c = LoginManager.getTestUrl();
       if (c != null) {
          if (!c.equals(LoginManager.CONNECTED)) {
             c = LoginManager.getElementText(c, LoginManager.TAG_WISPR);
             if ((LoginManager.getElementTextAsInt(c,
LoginManager.TAG_MESSAGE_TYPE) == Constants.WMT_INITIAL_REDIRECT) &&
(LoginManager.getElementTextAsInt(c, LoginManager.TAG_RESPONSE_CODE) ==
Constants.WRC_NO_ERROR)) {
                c = LoginManager.login(LoginManager.getElementText(c,
LoginManager.TAG_LOGIN_URL), user, pass);
                if (c != null) {
                   c = LoginManager.getElementText(c, LoginManager.TAG_WISPR);
                   final int mt = LoginManager.getElementTextAsInt(c,
LoginManager.TAG_MESSAGE_TYPE);
                   if ((mt == Constants.WMT_AUTH_NOTIFICATION) || (mt ==
Constants.WMT_RESPONSE_AUTH_POLL)) {
                      rc = LoginManager.getElementTextAsInt(c,
LoginManager.TAG_RESPONSE_CODE);
                      if ((rc == Constants.WRC_LOGIN_FAILED) || (rc ==
Constants.WRC_ACCESS_GATEWAY_INTERNAL_ERROR)) {
                         rc = LoginManager.getElementTextAsInt(c,
LoginManager.TAG_FON_RESPONSE_CODE);
                         rm = LoginManager.getElementText(c,
LoginManager.TAG_REPLY_MESSAGE);
                      }
                   }
                }
             } else {
                rc = Constants.CRC_WISPR_NOT_PRESENT;
             }
          } else {
             rc = Constants.CRC_ALREADY_AUTHORISED;
          }
       }
    } else {
       rc = Constants.CRC_CREDENTIALS_ERROR;
    }
    return new String[] { Integer.toString(rc), rm };
 }
```

```
}
```

SettingsActivity.java

```
package org.rsalvaterra.fon;

import android.app.AlertDialog.Builder;
import android.content.Context;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.media.RingtoneManager;
import android.net.Uri;
import android.os.Bundle;
import android.preference.Preference;
import android.preference.Preference.OnPreferenceChangeListener;
import android.preference.Preference.OnPreferenceClickListener;
import android.preference.PreferenceActivity;
import android.text.Html;
import android.text.method.LinkMovementMethod;
import android.widget.TextView;

public final class SettingsActivity extends PreferenceActivity implements
OnClickListener, OnPreferenceChangeListener, OnPreferenceClickListener {

  private Preference getPreference(final int id) {
     return getPreferenceScreen().findPreference(getString(id));
  }

  private void setOnPreferenceChangeListener(final int id, final String v) {
     final Preference p = getPreference(id);
     p.setOnPreferenceChangeListener(this);
     onPreferenceChange(p, FonManService.getPreference(p.getContext(), id, v));
  }

  @Override
  public void onClick(final DialogInterface d, final int w) {
     d.dismiss();
  }

  @Override
  public void onCreate(final Bundle b) {
```

```
        super.onCreate(b);
        addPreferencesFromResource(R.layout.settings);
        getPreference(R.string.kabout).setOnPreferenceClickListener(this);
        setOnPreferenceChangeListener(R.string.kperiod, Constants.DEFAULT_PERIOD);
        setOnPreferenceChangeListener(R.string.krssi, Constants.DEFAULT_MINIMUM_RSSI);
        setOnPreferenceChangeListener(R.string.ksuccess, "");
        setOnPreferenceChangeListener(R.string.kfailure, "");
    }

    @Override
    public boolean onPreferenceChange(final Preference p, final Object v) {
        final Context c = p.getContext();
        final String k = p.getKey();
        String s;
        if (k.equals(c.getString(R.string.kperiod))) {
            s = c.getString(R.string.periodSummary, v);
        } else if (k.equals(c.getString(R.string.krssi))) {
            s = c.getString(R.string.rssiSummary, v);
        } else {
            s = (String) v;
            if (s.length() != 0) {
                s = RingtoneManager.getRingtone(c, Uri.parse(s)).getTitle(c);
            }
        }
        p.setSummary(s);
        return true;
    }

    @Override
    public boolean onPreferenceClick(final Preference p) {
        final Context c = p.getContext();
        ((TextView) new
Builder(c).setIcon(R.drawable.ic_launcher).setTitle(R.string.app_name).setMessage(Ht
ml.fromHtml(c.getString(R.string.app_credits, c.getString(R.string.app_copyright),
c.getString(R.string.app_source)))).setNeutralButton(R.string.accept,
this).show().findViewById(android.R.id.message)).setMovementMethod(LinkMovementMeth
od.getInstance());
        return true;
    }

}
```

settings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">

  <PreferenceCategory
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:title="@string/basic">

     <CheckBoxPreference
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:defaultValue="true"
        android:key="@string/kautoconnect"
        android:summary="@string/autoconnectSummary"
        android:title="@string/autoconnect" />

     <EditTextPreference
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="textEmailAddress"
        android:key="@string/kusername"
        android:summary="@string/usernameSummary"
        android:title="@string/username" />

     <EditTextPreference
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
        android:key="@string/kpassword"
        android:summary="@string/passwordSummary"
        android:title="@string/password" />

  </PreferenceCategory>

  <PreferenceCategory
     android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
    android:title="@string/notifications">

    <CheckBoxPreference
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:defaultValue="true"
        android:key="@string/knotify"
        android:title="@string/notify" />

    <RingtonePreference
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:defaultValue="content://settings/system/notification_sound"
        android:dependency="@string/knotify"
        android:key="@string/ksuccess"
        android:layout="?android:attr/preferenceLayoutChild"
        android:ringtoneType="notification"
        android:title="@string/success" />

    <RingtonePreference
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:defaultValue="content://settings/system/notification_sound"
        android:dependency="@string/knotify"
        android:key="@string/kfailure"
        android:layout="?android:attr/preferenceLayoutChild"
        android:ringtoneType="notification"
        android:title="@string/failure" />

    <CheckBoxPreference
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:defaultValue="false"
        android:dependency="@string/knotify"
        android:key="@string/kvibrate"
        android:layout="?android:attr/preferenceLayoutChild"
        android:title="@string/vibrate" />

</PreferenceCategory>

<PreferenceCategory
```

```
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:title="@string/advanced">

<CheckBoxPreference
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:defaultValue="false"
   android:key="@string/kreconnect"
   android:summary="@string/reconnectSummary"
   android:title="@string/reconnect" />

<CheckBoxPreference
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:defaultValue="true"
   android:dependency="@string/kreconnect"
   android:key="@string/ksecure"
   android:layout="?android:attr/preferenceLayoutChild"
   android:summary="@string/secureSummary"
   android:title="@string/secure" />

<EditTextPreference
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:defaultValue="300"
   android:dependency="@string/kreconnect"
   android:inputType="number"
   android:key="@string/kperiod"
   android:layout="?android:attr/preferenceLayoutChild"
   android:maxLength="4"
   android:title="@string/period" />

<CheckBoxPreference
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:defaultValue="false"
   android:key="@string/kreject"
   android:summary="@string/rejectSummary"
   android:title="@string/reject" />

<EditTextPreference
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:defaultValue="-80"
        android:dependency="@string/kreject"
        android:inputType="numberSigned"
        android:key="@string/krssi"
        android:layout="?android:attr/preferenceLayoutChild"
        android:maxLength="3"
        android:title="@string/rssi" />

  </PreferenceCategory>

  <Preference
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:key="@string/kabout"
     android:title="@string/about" />

</PreferenceScreen>
```

strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string
     name="app_name"
     translatable="false">FonMan</string>
  <string name="app_label">FonMan Settings</string>
  <string name="app_credits"><![CDATA[Automatic authenticator for Fon
networks.<p>%1$s</p>This application is free software: you can redistribute it
and/or modify it under the terms of the GNU General Public License (GPL) version 3,
as published by the Free Software Foundation.<p>Source code available on
%2$s.</p>]]></string>
  <string
     name="app_copyright"
     translatable="false">Copyright © 2018 Rui Salvaterra</string>
  <string
     name="app_source"
     translatable="false"><![CDATA[<a
href="https://github.com/rsalvaterra/Fon">GitHub</a>]]></string>
  <string name="advanced">Advanced settings</string>
  <string name="basic">Basic settings</string>
```

```
  <string name="about">About</string>
  <string
     name="kabout"
     translatable="false">about</string>
  <string name="accept">Accept</string>
  <string
     name="kusername"
     translatable="false">username</string>
  <string name="username">User name</string>
  <string name="usernameSummary">Your user name.</string>
  <string
     name="kpassword"
     translatable="false">password</string>
  <string name="password">Password</string>
  <string name="passwordSummary">Your password.</string>
  <string
     name="kautoconnect"
     translatable="false">autoconnect</string>
  <string name="autoconnect">Connect automatically</string>
  <string name="autoconnectSummary">Automatically connect to any available Fon
access point.</string>
  <string
     name="kreconnect"
     translatable="false">reconnect</string>
  <string name="reconnect">Allow reconnection</string>
  <string name="reconnectSummary">Disconnect from Fon and connect to a known
network, if found.</string>
  <string
     name="ksecure"
     translatable="false">secure</string>
  <string name="secure">Secure only</string>
  <string name="secureSummary">Only consider secure networks for
reconnection.</string>
  <string
     name="kperiod"
     translatable="false">period</string>
  <string name="period">Period between scans</string>
  <string name="periodSummary">%s seconds</string>
  <string
     name="kreject"
     translatable="false">reject</string>
  <string name="reject">Reject weak APs</string>
```

```
  <string name="rejectSummary">Avoid connections to APs with weak signal.</string>
  <string
     name="krssi"
     translatable="false">rssi</string>
  <string name="rssi">Minimum power</string>
  <string
     name="rssiSummary"
     translatable="false">%s dBm</string>
  <string name="notifications">Notifications</string>
  <string
     name="knotify"
     translatable="false">notify</string>
  <string name="notify">Notify</string>
  <string
     name="ksuccess"
     translatable="false">success</string>
  <string name="success">Tone (success)</string>
  <string
     name="kfailure"
     translatable="false">failure</string>
  <string name="failure">Tone (error)</string>
  <string
     name="kvibrate"
     translatable="false">vibrate</string>
  <string name="vibrate">Vibrate</string>
  <string name="connected">Connected to \"%s\".</string>
  <string name="fon_error">Error %1$d: \"%2$s\"</string>
  <string name="cred_error">Invalid credentials</string>
  <string name="configure">Touch to configure.</string>
  <string name="started">Session started</string>
</resources>
```

strings.xml [PT]

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_label">Definições FonMan</string>
  <string name="app_credits"><![CDATA[Autenticador automático para redes
Fon.<p>%1$s</p>Esta aplicação é software livre: pode redistribuí-la e/ou
modificá-la segundo os termos da GNU General Public License (GPL) versão 3, tal
como publicada pela Free Software Foundation.<p>Código fonte disponível no
%2$s.</p>]]></string>
```

```
  <string name="advanced">Definições avançadas</string>
  <string name="basic">Definições básicas</string>
  <string name="about">Acerca</string>
  <string name="accept">Aceitar</string>
  <string name="username">Nome de utilizador</string>
  <string name="usernameSummary">O seu nome de utilizador.</string>
  <string name="password">Palavra-chave</string>
  <string name="passwordSummary">A sua palavra-chave.</string>
  <string name="autoconnect">Ligação automática</string>
  <string name="autoconnectSummary">Ligar automaticamente a uma rede Fon
detectada.</string>
  <string name="reconnect">Permitir reconexão</string>
  <string name="reconnectSummary">Desligar da rede Fon e ligar a outra rede
conhecida, se detectada.</string>
  <string name="secure">Apenas seguras</string>
  <string name="secureSummary">Apenas considerar redes seguras para
reconexão.</string>
  <string name="period">Período entre sondagens</string>
  <string name="periodSummary">%s segundos</string>
  <string name="reject">Rejeitar PA fracos</string>
  <string name="rejectSummary">Evitar ligações a PA com sinal fraco.</string>
  <string name="rssi">Potência mínima</string>
  <string name="notifications">Notificações</string>
  <string name="notify">Notificar</string>
  <string name="success">Toque (sucesso)</string>
  <string name="failure">Toque (erro)</string>
  <string name="vibrate">Vibrar</string>
  <string name="connected">Ligado a «%s».</string>
  <string name="fon_error">Erro %1$d: «%2$s»</string>
  <string name="cred_error">Credenciais inválidas</string>
  <string name="configure">Toque para configurar.</string>
  <string name="started">Sessão iniciada</string>
</resources>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="org.rsalvaterra.fon"
  android:versionCode="1"
  android:versionName="1.0">
```

```
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
  <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.VIBRATE" />
  <uses-permission android:name="android.permission.WAKE_LOCK" />

  <uses-feature android:name="android.hardware.wifi" />

  <application
     android:allowBackup="false"
     android:icon="@drawable/ic_launcher"
     android:label="@string/app_name">

     <activity
        android:name=".SettingsActivity"
        android:label="@string/app_label">
        <intent-filter>
           <action android:name="android.intent.action.MAIN" />
           <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
     </activity>

     <receiver android:name=".FonManWifiReceiver">
        <intent-filter>
           <action android:name="android.net.wifi.SCAN_RESULTS" />
           <action android:name="android.net.wifi.STATE_CHANGE" />
        </intent-filter>
     </receiver>

     <receiver android:name=".FonManAlarmReceiver" />

     <service android:name=".FonManService" />

  </application>

</manifest>
```

build.gradle

```
apply plugin: 'com.android.application'
android {
```

```
    compileSdkVersion 22
    defaultConfig {
        applicationId "org.rsalvaterra.fon"
        minSdkVersion 3
        targetSdkVersion 22
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles 'proguard.txt'
        }
    }
    lintOptions {
        checkReleaseBuilds false
    }
    productFlavors {
    }
}
dependencies {
}
```