Projecto de Mestrado em Engenharia Informática – Computação Móvel

Clientes de sincronização para plataformas móveis

Jorge Filipe Diogo Soares Miguel

Dissertação de Mestrado realizada sob a orientação da Doutora Catarina Helena Branco Simões Silva, Professora da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria.

Leiria, 2011

À Minha Família

Agradecimentos

Queria começar por agradecer a disponibilidade de todos aqueles que me ajudaram a concretizar este objectivo que marca a conclusão de mais uma etapa na minha vida académica.

Agradeço à WIT-Software por ter apostado em mim para agarrar este estágio e levá-lo até ao fim. Também de agradecer o facto de que, sempre que necessário, a WIT-Software ter disponibilizado o seu equipamento para que não me faltasse nada para a realização do trabalho durante o periodo de estágio.

Agradeço também ao meus orientadores, que me forneceram mais ajuda do que aquela que poderia esperar. Um obrigado à Doutora Catarina Silva pela ajuda prestada para a realização deste trabalho, especialmente no relatório e ao Engenheiro Luís Guilherme por toda a ajuda e tempo que dedicou a mim e a este projecto.

Também é de salientar a ajuda dos meus colegas de trabalho, nomeadamente os colegas do escritório de Leiria da WIT-Software, André Cravo, António Pereira, Marisa Martins, Miguel Nunes, Pedro Oliveira, Ricardo Carreira, Rui Costa e Tiago Marto, os quais me apoiaram durante o desenvolvimento deste trabalho, e em especial ao Alcides Jorge e ao Ricardo Nogueira que me acompanharam mais de perto durante os desenvolvimentos e me ensinaram muito do que aqui está presente.

Não poderia deixar de agradecer também à minha família, que sempre me apoiou e de tudo fez para que eu chegasse a este ponto. Também quero deixar aqui expresso um agradecimento especial à minha noiva pela ajuda e paciência que teve comigo durante o decorrer deste trabalho.

Resumo

Cada dia que passa novos dispositivos móveis são lançados no mercado, o que leva a que cada vez mais as pessoas tenham na sua posse um grande número de dispositivos, com capacidade de comunicação e armazenamento de informação. Neste cenário é frequente encontrar casos em que a informação fica tão dispersa pelos diversos dispositivos que o utilizador não sabe onde a encontrar de uma forma eficiente.

Com este trabalho pretende-se criar aplicações móveis através das quais seja possível proporcionar a um utilizador de diversas plataformas e dispositivos móveis a gestão da sua informação, através da sincronização entre todos os seus dispositivos.

As aplicações apresentadas foram desenvolvidas com base no protocolo de sincronização SyncML, desenvolvido por um grupo de empresas com o objectivo criar um standard de sincronização entre marcas.

Devido ao facto de todos os dias serem criadas novas formas de armazenamento de informação, as aplicações foram desenhadas de modo a serem escaláveis, garantindo assim o suporte para qualquer tipo de dados que lhe seja adicionado no futuro.

Palavras-chave: dispositivos móveis, sincronização de dados, SyncML, dispersão da informação, escalabilidade

Abstract

Everyday new devices are released to the market, which causes people to have an increasing number of devices with communication and data storage capabilities. Given such scenario, information can become so dispersed that people end up not knowing where to find it efficiently.

The aim of this work is to implement a set of mobile applications with which the user will be capable of synchronizing data across his devices.

These applications are based on the SyncML synchronization protocol, developed by a group of companies with the common goal of creating a synchronization standard that would be used across devices.

Due to the fact that new data types are being released every day, in these work we propose applications that are developed with the goal of being scalable; assuring the support to any new data type that could be added it in the future.

Key-Words: mobile devices, data synchronization, SyncML, data dispersion, scalability

Índice de Figuras

Índice de Quadros

Lista de Siglas

Android	Plataforma para dispositivos móveis comprada pela Google em 2005 e mantida por esta desde então.
Background (execução)	Execução de uma aplicação num modo no qual não existe interacção entre o utilizador e aplicação.
BlackBerry	Plataforma para dispositivos móveis da empresa Norte Americana RIM.
C/C++	Linguagem de programação.
Capabilities (SyncML)	Informação sobre as capacidades de um dispositivo.
cURL	Biblioteca de gestão e comunicação em rede.
Funambol	Nome de empresa e projecto open-source Italiano de sincronização.
iOS	Plataforma para dispositivos móveis da empresa Norte Americana Apple Inc.
iPad	Dispositivo móvel da empresa Norte Americana Apple Inc.
iPhone	Dispositivo móvel da empresa Norte Americana Apple Inc.
iPod	Dispositivo móvel da empresa Norte Americana Apple Inc.
Java	Linguagem de programação.
Java ME	Linguagem de programação orientada a dispositivos de móveis e de baixos recursos.
Java SE	Linguagem de programação orientada para ambientes de servidores e desktop.
JNI	Java Native Interface – Interface para programação em C/C++ em plataformas baseadas em Java.
Large Objects (SyncML)	Itens para sincronização de tamanho superior ao tamanho máximo de uma mensagem SyncML.
NDK	Native Development Kit – Conjunto de ferramentas para desenvolvimento em C/C++ na plataforma Android.
Open-Source	Aplicações cujo autor disponibiliza o código à comunidade.
PIM	Personal Information Managemente – Conjunto de ferramentas para manipulação de dados pessoais (contactos, calendários, notas, eventos...).
PostgreSQL	Motor de base de dados relacional frequentemente usado em projectos C/C++.
source-tree	Código de um projecto de uma aplicação.

SyncML	Protocolo de Sincronização baseado em XML orientado para plataformas móveis.
wxWidgets	Biblioteca para implementação de aplicações na linguagem C/C++.
XML	Extended Markup Language – Linguagem de suporte a dados frequentemente usada ser leve e de fácil manipulação.

Índice

1 Introdução

Desde o início da informática que existe a necessidade de replicar informação entre dispositivos. Inicialmente, a cópia de dados entre vários dispositivos foi a solução encontrada para manter a informação em segurança pois, ao ser replicada entre várias máquinas e em vários locais, a probabilidade de que esta sobreviva a um qualquer problema que ocorra com as plataformas em que se encontra armazenada aumenta consideravelmente, eliminando assim a possibilidade de falha única.

Mais recentemente, com a introdução da computação móvel na sociedade, temos vindo a assistir a um aumento significativo do número de dispositivos que possuem capacidade de armazenar e disponibilizar informação, como é o caso de telemóveis, computadores portáteis, tablets e até mesmo objectos como o carro ou o frigorífico, que até aos dias de hoje eram apenas direccionados para as suas tarefas específicas.

Com toda esta dispersão cada vez mais acentuada da informação, o utilizador sente cada vez mais a necessidade de mecanismos que mantenham a sua informação disponível em todos os seus dispositivos, de modo a que esta esteja disponível em qualquer parte e a qualquer momento.

1.1 Enquadramento

O trabalho documentado neste relatório tem como objectivo o desenvolvimento de um conjunto de clientes de sincronização, recorrendo ao protocolo SyncML para várias plataformas móveis. A motivação para o estágio foi a vontade da WIT Software, empresa onde o trabalho foi desenvolvido, de complementar o pacote de serviços de sincronização que já possuía e que é composto por um servidor de sincronização e uma *plug-in* para o MS Outlook, com um conjunto de clientes para dispositivos móveis que não suportam nativamente o protocolo SyncML. Em paralelo com os clientes, foi também objectivo da

WIT-Software obter um conjunto de bibliotecas de sincronização que possam ser integradas noutros projectos em que se justifique o uso do protocolo SyncML.

Em termos de conteúdo científico, este trabalho engloba um leque alargado de tecnologias, plataformas, linguagens de programação e arquitecturas, que normalmente não se encontram presentes com esta heterogeneidade num estágio. Tal deve-se ao facto de, ao contrário da maioria dos estágios, que se focam numa tecnologia para implementar ou estudar um tema, este estágio se basear na implementação de um protocolo em diversas plataformas com características bastante diferentes.

Em resumo, o trabalho apresentado neste documento envolveu as seguintes etapas:

- Desenvolver clientes de sincronização para as plataformas Android, BlackBerry e iOS (iPod, iPhone e iPad), que sejam escaláveis a diferentes tipos de dados para sincronização.
- Possibilitar a sincronização de contactos usando estes clientes.
- Criar bibliotecas de sincronização para as plataformas BlackBerry (Java SE), Android (Java ME) e iOS (Objective-C) que garantam uma ferramenta de fácil implementação que possa ser usada em outras aplicações para as quais faça sentido ter um serviço de sincronização.
- Na plataforma Android, ter em conta algumas especificidades existentes, como é o caso da estrutura dos contactos. Estas especificidades tem que ser suportadas não só no cliente mas também no servidor.
- Possibilitar a execução de sincronizações automáticas no cliente Android.
- Alterar a API de sincronização de modo a adicionar novas funcionalidades, como é o caso de suporte a itens demasiado grandes para serem enviados numa única mensagem.

O presente documento encontra-se dividido em 5 capítulos. Os dois primeiros introduzem o contexto do trabalho. Os restantes capítulos descrevem o trabalho realizado durante o estágio, sendo nestes apresentados os requisitos das aplicações, as arquitecturas utilizadas,

os processos de implementação dos clientes e as conclusões obtidas relativas às ferramentas das plataformas e aos resultados obtidos.

Após esta introdução, o Capítulo 2 irá fazer o enquadramento dos conceitos relativos à sincronização de dados e ao estado-da-arte dos clientes de sincronização.

No Capítulo 3 irá ser feita uma introdução ao trabalho realizado, o qual será posteriormente analisado em maior detalhe no Capitulo 4, onde irão ser descritas as abordagens, arquitecturas e Implementações referentes a cada uma das plataformas usadas durante a implementação dos clientes. A conclusão e o trabalho futuro serão descritos no Capítulo 5.

2 Estado-da-Arte

Existem diversos protocolos e normas que descrevem algoritmos para implementação de serviços de sincronização. No dia-a-dia usamos muitos destes protocolos, frequentemente sem que nos apercebamos desse facto. Um dos serviços de sincronização mais utilizado hoje em dia é a sincronização de e-mail, o qual é feito recorrendo habitualmente a um de dois protocolos, o POP ou o IMAP.

A sincronização representa assim uma maneira não só de guardar informação em locais descentralizados, com o objectivo de salvaguardar a informação, mas também uma garantia de que possamos ter sempre disponível a informação de que necessitamos perto de nós, quer estejamos em casa ou no trabalho, em frente ao computador ou num local em que apenas tenhamos acesso ao telemóvel.

Neste capítulo vai ser introduzido o protocolo utilizado no trabalho aqui documentado, o SyncML. Após a introdução ao serviço irá ser feita uma apresentação dos clientes SyncML que mais se destacam no mercado dos dispositivos móveis. Estes destacam-se por vários factores, entre eles o facto de suportarem várias plataformas, terem funcionalidades inovadoras ou por fazerem parte de pacotes de software de sincronização que incluem não só o cliente móvel mas também APIs ou servidores.

2.1 Sincronização recorrendo ao protocolo SyncML

O protocolo utilizado no desenvolvimento deste trabalho foi o "*Open Mobile Alliance Data Synchronization and Device Management*", ou SyncML. Este protocolo foi criado por um grupo denominado OMA (Open Mobile Alliance), constituído por marcas como a Samsung, a Motorola, a Nokia e a Sony Ericsson e tem como objectivo principal estandardizar a partilha de informação pessoal entre dispositivos de vários tipos e marcas. Ao ser desenvolvido por marcas do meio da computação móvel, este protocolo ganhou rapidamente uma presença

nativa em diversos tipos de dispositivos, especialmente em telemóveis. Além dos dispositivos com suporte nativo, há também disponíveis no mercado diversas soluções que garantem a dispositivos sem suporte nativo a compatibilidade com o protocolo.

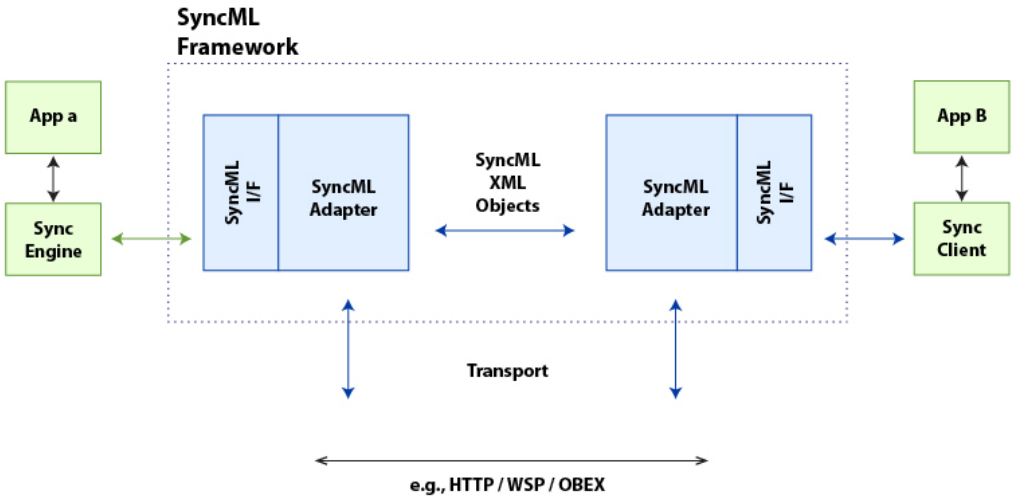


Figura 1. Implementação do protocolo SyncML

O SyncML, como se pode ver na Figura 1, é baseado numa arquitectura cliente-servidor. Apesar de ser possível encontrar alguns clientes que fazem alguma gestão da sincronização, a tarefa de gestão da sincronização é atribuída por definição ao servidor. As sincronizações feitas recorrendo a este protocolo podem ser completas, o que significa que todos os dados são sincronizados, ou incrementais, modo em que apenas as alterações aos dados previamente sincronizados são sincronizados. Para controlar as sincronizações incrementais, que precisam de ter uma referencia ao estado da última sincronização, e também para controlar possíveis erros de sincronização.

2.1.1 Âncoras

Para controlo das sincronizações incrementais, e para que seja possível identificar alguma incoerência que possa ocorrer entre duas sincronizações, o SyncML recorre ao uso de âncoras. Estas âncoras, baseadas em cadeias de caracteres (*strings*), são criadas e partilhadas entre os dispositivos no fim de cada sincronização bem sucedida. A validadação das âncoras é realizada no inicio de cada sincronização e é feita comparando se âncora do cliente é igual à do servidor.

2.1.2 Modos de Sincronização

O SyncML tem definidos na sua especificação seis modos de sincronização obrigatórios. Estes modos garantem a sincronização da informação em ambos os sentidos (*Two-Way*) ou apenas num sentido (*One-Way*). Para cada um destes tipos de sincronização podem ser executadas sincronizações completas, nas quais todos os dados são copiados, ou por outro lado sincronizações incrementais, onde apenas são copiadas as alterações efectuadas nos dados desde a última sincronização.

Os modos de sincronização definidos para o SyncML são:

- ***Slow-Sync:*** Modo de sincronização executado na primeira sincronização *Two-Way* entre um cliente e o servidor. Neste modo é executada uma sincronização completa em *Two-Way*, sendo enviados todos os dados existentes no cliente para o servidor, ao que se segue a recepção dos dados do servidor pelo cliente. A sincronização neste modo segue a seguinte sequência (caminho óptimo e simplificado):

    1. O cliente autentica-se perante o servidor enviando as suas credenciais e o tipo de itens que quer sincronizar;
    2. O servidor responde com sucesso;
    3. O cliente envia ao servidor os seus itens a sincronizar;
    4. O servidor recebe os itens e verifica se não os tem já presentes, adicionando os novos e mapeando os já existentes;
    5. Após receber e validar os itens do cliente, o servidor envia para o cliente um estado para cada item recebido. Envia também os itens que possuía antes da sincronização e que não corresponderam aos itens recebidos do cliente;
    6. O cliente guarda os itens que recebeu do servidor e envia para o servidor um estado para cada item adicionado e o mapeamento dos mesmos, utilizando para isso os ids criados no cliente e os ids recebidos do servidor;
    7. O servidor recebe os estados e os mapeamentos, guarda-os e termina a sincronização. Na mensagem final do servidor é enviada ao cliente uma âncora que será usada para negociar a próxima sincronização, se esta for incremental;
    8. O cliente guarda a âncora.

- ***Two-way Sync:*** Modo de sincronização pré-definido. Neste modo de sincronização é feita uma sincronização incremental em ambos os sentidos (cliente para servidor e servidor para cliente). A sincronização neste modo segue a seguinte sequência (caminho óptimo e simplificado):

    1. O cliente autentica-se perante o servidor enviando as suas credenciais, o tipo de itens a sincronizar e a âncora da última sincronização para o tipo de item correspondente;
    2. O servidor responde com sucesso;
    3. O cliente envia todas as alterações a que foi sujeito desde a última sincronização para o servidor.
    4. O servidor recebe as alterações do cliente e adiciona, actualiza ou apaga os itens sujeitos a actualizações. Após efectuar as alterações, envia para o cliente um estado para cada item recebido. Envia também para o cliente os itens que tenham sido actualizados no servidor desde a última sincronização e que não tenham sido actualizados pelo cliente durante a sincronização actual;
    5. O cliente guarda os itens que recebeu do servidor e envia para o servidor um estado para cada item adicionado e o mapeamento dos mesmos, utilizando para isso os ids criados no cliente e os ids recebidos do servidor;
    6. O servidor recebe os estados e os mapeamentos, guarda-os e termina a sincronização. Na mensagem final do servidor é enviada ao cliente uma âncora que será usada para negociar a próxima sincronização, se esta for incremental;
    7. O cliente guarda a âncora.

- ***Refresh From Server:*** Modo de sincronização executado na primeira sincronização *one-way from server* entre um cliente e o servidor. Neste modo é executada uma sincronização completa no sentido servidor para cliente. A sincronização neste modo segue a seguinte sequência (caminho óptimo e simplificado):

    1. O cliente autentica-se perante o servidor enviando as suas credenciais e o tipo de itens que quer sincronizar;
    2. O servidor responde com sucesso e envia todos os seus itens para o cliente;

3. O cliente apaga os seus itens e guarda os itens que recebeu do servidor. Em seguida envia para o servidor um estado para cada item adicionado e o mapeamento dos mesmos, utilizando para isso os ids criados no cliente e os ids recebidos do servidor;
4. O servidor recebe os estados e os mapeamentos, guarda-os e termina a sincronização. Na mensagem final do servidor é enviada ao cliente uma âncora que será usada para negociar a próxima sincronização, se esta for incremental;
5. O cliente guarda a âncora.

- ***One-way from Server:*** Neste modo é executada uma sincronização incremental no sentido servidor para cliente. A sincronização neste modo segue a seguinte sequência (caminho óptimo e simplificado):

    1. O cliente autentica-se perante o servidor enviando as suas credenciais e o tipo de itens que quer sincronizar. É enviada também a âncora da última sincronização;
    2. O servidor responde com sucesso e envia todos os itens que foram alterados ou adicionados desde a última sincronização para o cliente;
    3. O cliente guarda os itens que recebeu do servidor. Em seguida envia para o servidor um estado para cada item adicionado ou substituído e o mapeamento dos mesmos, utilizando para isso os ids no cliente e os ids recebidos do servidor;
    4. O servidor recebe os estados e os mapeamentos, guarda-os e termina a sincronização. Na mensagem final do servidor é enviada ao cliente uma âncora que será usada para negociar a próxima sincronização, se esta for incremental;
    5. O cliente guarda a âncora.

- ***Refresh From Client*****:** Modo de sincronização executado na primeira sincronização *one-way from client* entre um cliente e o servidor. Neste modo é executada uma sincronização completa no sentido cliente para servidor. A sincronização neste modo segue a seguinte sequência (caminho óptimo e simplificado):

    1. O cliente autentica-se perante o servidor enviando as suas credenciais e o tipo de itens que quer sincronizar;

2. O servidor responde com sucesso;
3. O cliente envia os seus itens para o servidor;
4. O servidor apaga os seus itens e guarda os itens que recebeu do cliente. Em seguida envia para o cliente um estado para cada item;
5. O cliente recebe os estados.
6. O servidor envia ao cliente uma âncora que será usada para negociar a próxima sincronização, se esta for incremental;
7. O cliente guarda a âncora.

- ***One-way from Client:*** Neste modo é executada uma sincronização incremental no sentido cliente para servidor. A sincronização neste modo segue a seguinte sequência (caminho óptimo e simplificado):

    1. O cliente autentica-se perante o servidor enviando as suas credenciais e o tipo de itens que quer sincronizar. É enviada também a âncora da última sincronização;
    2. O servidor responde com sucesso;
    3. O cliente envia todos os itens que foram alterados ou adicionados desde a última sincronização para o servidor;
    4. O servidor guarda os itens que recebeu do cliente. Em seguida envia para o cliente um estado para cada item;
    5. O cliente recebe os estados.
    6. O servidor envia ao cliente uma âncora que será usada para negociar a próxima sincronização, se esta for incremental;
    7. O cliente guarda a âncora.

2.2 Clientes móveis SyncML

De seguida irão ser apresentados os clientes de SyncML que mais se destacam actualmente. Estes clientes destacam-se devido às suas funcionalidades e/ou à diversidade de plataformas que suportam.

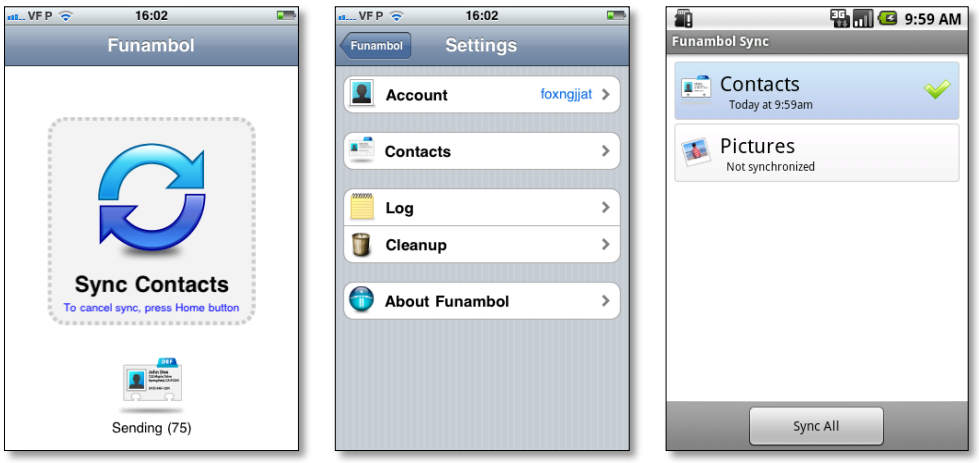
Os clientes apresentados variam nas suas características entre plataformas, pelo que as características apresentadas para cada cliente são o conjunto das características das várias

versões do mesmo cliente. Significa isto que as características apresentadas para cada cliente não existem necessariamente em todas as plataformas.

2.2.1 Funambol

Funambol[1] é um projecto *open-source* que actualmente detém o cliente que mais se destaca nesta área. As suas vantagens são a diversidade de dispositivos suportados e as funcionalidades que disponibiliza (Figuras 2 e 3).

Em paralelo com os clientes de sincronização, a Funambol mantém também um servidor e uma API de sincronização desenvolvida em várias linguagens (Java SE, Java ME, C++) e que é disponibilizada gratuitamente para uso pessoal e comercial (obrigando o código das aplicações em que é usada a ser fornecido à comunidade). A Funambol também disponibiliza uma versão paga da API que garante a sua utilização sem imposições. Todos os clientes da Funambol são baseados nas suas APIs.



a b c

Figura 2. Cliente Funambol nas versões iPhone (a, b) e Android (c)

[1] http://www.funambol.com

a b c d

Figura 3. Cliente Funambol nas versões Windows Mobile (a) e Symbian (b, c, d)

Como referido na Tabela 1, o cliente Funambol apresenta uma grande disparidade de funcionalidades entre plataformas, sendo a versão que menos funcionalidades possui a versão iPhone, que apenas sincroniza contactos, ao contrário da versão Windows Mobile, que não só suporta a sincronização de contactos como também sincronização de email, calendários, tarefas, notas e *Briefcases* (pastas especiais de partilha em ambiente Windows).

Na versão Android este cliente executa sincronizações automaticamente mas, para a sincronização de contactos, apenas sincroniza contactos do tipo Funambol, o que é uma desvantagem para quem possua outros contactos que não tenham sido criados com este tipo ou importados do servidor Funambol.

Positivo	**Negativo**
✓ Suporte para um largo número de dispositivos	× Na versão Android apenas sincroniza contactos do tipo Funambol
✓ Sincronização de dados pessoais	× Suporta poucos tipos de dados em iOS, Android e BlackBerry
✓ Sincronização de imagens	× Grande disparidade nos tipos de dados suportados

Tabela 1. Cliente Funambol

2.2.2 O3SIS – Personal Life Mobilizer – Full Edition

A O3SIS[2] é uma empresa alemã que fornece vários tipos de clientes de backup e sincronização sendo o seu cliente SyncML mais completo o "*Personal Life Mobilizer – Full Edition*"[3]. Este produto é orientado às operadoras que queiram disponibilizar aos seus clientes uma forma de estes fazerem backups da sua informação pessoal existente nos seus dispositivos.

a b c

Figura 4. Cliente O3SIS iPhone (a, b) e Windows Mobile (c)

O cliente, existente nas versões iPhone (Figuras 4a e 4b), BlackBerry, Windows Mobile (Figura 4c), Symbian e J2ME destingue-se por suportar a sincronização de vídeo, música e imagens na maioria das versões. No entanto, apenas as versão Symbian e Windows Mobile é suportada a sincronização de calendários, tarefas e e-mails.

Positivo	**Negativo**
✓ Suporte para um largo número de dispositivos	× Não sincroniza automaticamente
✓ Sincronização de dados pessoais	× Nas plataformas Android e iOS, dos tipos de dados pessoais apenas sincroniza contactos.
✓ Sincronização de imagem, vídeo e música	

Tabela 2. Cliente O3SIS

[2] http://www.o3sis.com

[3] http://www.o3sis.com/personal_life_mobilizer_full_edition.shtml

2.2.3 Pleex

Pleex[4] é uma empresa francesa que apresenta várias soluções de sincronização. Destaca-se principalmente por ter focado os seus serviços na partilha de informação directamente dos dispositivos móveis para as redes sociais.

O cliente disponibiliza nas várias versões sincronização de contactos, imagens, vídeo e áudio, ficando a faltar o suporte a outros tipos de dados pessoais como é o caso de calendários, tarefas e notas.

Figura 5. Cliente Pleex na versão iPhone

Este cliente destaca-se por suportar as mesmas funcionalidades em todas as plataformas para as quais foi desenvolvido. Outro destaque deste cliente, como se poe ver na Figura 5, é o facto de disponibilizar a sincronização de imagens directamente do dispositivo móvel para várias redes sociais. Fica a faltar a sincronização de tipos de dados pessoais, dos quais apenas a sincronização de contactos é suportada.

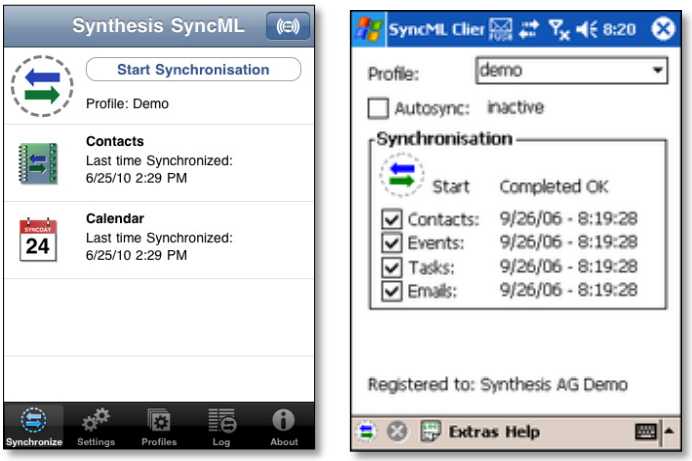
[4] http://en.pleex.com

Positivo	Negativo
✓ Sincronização de contactos	× Não sincroniza automaticamente
✓ Sincronização de Media (imagens, vídeo e áudio)	× Dos tipos de dados pessoais, apenas sincroniza contactos
✓ Partilha os media com redes sociais	
✓ Dispõe das mesmas funcionalidades em todas as plataformas que suporta	

Tabela 3. Cliente Pleex

2.2.4 Synthesis AG

A Synthesis AG[5] é uma empresa suíça que tem disponível várias soluções para sincronização baseada em SyncML. Estas soluções englobam um cliente de sincronização, disponível para várias plataformas, um servidor de sincronização e a API desenvolvida para várias linguagens (C, C++, Delphi, Java e .NET) e tecnologias.



a b

Figura 6. Cliente Synthesis AG nas versões iPhone (a) e Windows Mobile (b)

Este cliente disponibiliza sincronização de todos os tipos de dados pessoais, apesar de não o fazer em todas as plataformas. O destaque deste cliente está nas APIs de sincronização em

[5] http://www.synthesis.ch

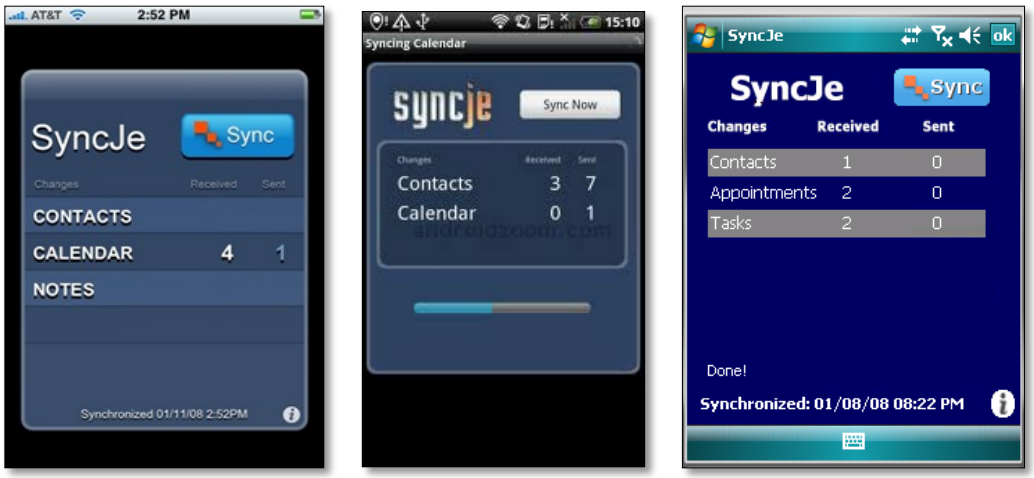
que é baseado e que a empresa disponibiliza e na capacidade de executar sincronizações automáticas na versão Windows Mobile (Figura 6b).

Positivo	Negativo
✓ Sincronização de dados pessoais	× Apenas sincroniza dados pessoais
✓ APIs de sincronização	
✓ Sincronização automática em Windows Mobile	

Tabela 4. Cliente Synthesis AG SyncML

2.2.5 Nexthaus SyncJe

A Nexthaus é uma empresa Norte-Americana que tem vindo a desenvolver um pacote de clientes de sincronização baseados em SyncML que já abrange um largo número de plataformas, tanto móveis como para computadores pessoais. As características nas versões deste cliente são semelhantes, havendo pouca variação nos tipos de dados suportados pelas diversas versões.



a b c

Figura 7. Cliente SyncJe nas versões iPhone (a), Android (b) e Windows Mobile (c)

O Cliente SyncJe centra-se apenas na sincronização de dados pessoais. Este cliente destaca-se por suportar em praticamente todas as versões os mesmos tipos de dados, o que se pode ver na Figura 7. Outro destaque deste cliente é a capacidade de executar sincronizações automaticamente na versão Windows Mobile.

Positivo	Negativo
✓ Sincronização de dados pessoais	× Apenas sincroniza dados pessoais
✓ Sincronização automática em Windows Mobile	

Tabela 5. Cliente Nexthaus SyncJe

2.2.6 Conclusões

Como pode ser constatado neste capítulo, existem diversas soluções no mercado para a sincronização de dados sobre o protocolo SyncML. No entanto, estas soluções conseguem diferenciar-se bastante umas das outras por se focarem em objectivos diferentes, quer seja a sincronização de dados pessoais, sincronização de média ou a integração com as redes sociais.

Nos clientes referidos neste estado-da-arte, também se pode verificar que apenas os contactos são suportados em todas as plataformas, sendo os outros tipos de dados apenas suportados em alguns dos clientes e versões. Na Tabela 6 podemos ver quais os tipos de dados que cada cliente suporta, para as versões Android, iPhone, BlackBerry e Windows Mobile.

	Contactos	Calendário	Tarefas	Notas	Ficheiros	Imagens	Vídeo	Música
O3SIS	✓	✓	✓	×	×	✓	✓	✓
Funambol	✓	✓	✓	✓	×	✓	×	×
Pleex	✓	×	×	×	✓	✓	✓	✓
Synthesis AG	✓	✓	✓	✓	×	×	×	×
Nexthaus SyncJe	✓	✓	✓	✓	×	×	×	×
WMB	✓	×	×	×	×	×	×	×

✓ - Disponível em todas as versões do cliente

✓ - Disponível em algumas versões do cliente

× - Não suportado pelo cliente em nenhuma versão

Tabela 6. Tipos de dados suportados pelos vários clientes

A nível de plataformas móveis suportadas pelos vários clientes SyncML (Tabela 7) o iPhone é a plataforma para o qual há mais clientes disponíveis, ao contrário do Symbian, que apenas é suportado por três dos clientes analisados. No entanto esta diferença não é muito

acentuada pois os outros clientes apenas falham numa das plataformas e, para o caso do Symbian, a pouca adesão a esta plataforma pode dever-se ao facto do SyncML ser suportado nativamente nesta plataforma.

	iPhone	Android	Black Berry	S60	Windows Mobile
O3SIS	✓	✓	✓	✓	✓
Funambol	✓	✓	✓	✓	✓
Pleex	✓	✓	✓	✓	✓
Synthesis AG	✓	✓	✗	✗	✓
Nexthaus SyncJe	✓	✗	✓	✗	✓
WMB	✓	✓	✓	✗	✗

Tabela 7. Plataformas móveis suportadas pelos clientes SyncML

3 Arquitectura proposta

Até ao momento foi introduzido o estado-da-arte e apresentado o protocolo SyncML. Neste capítulo irá ser descrita a arquitectura para os clientes de sincronização desenvolvidos.

Sendo o objectivo do trabalho criar um conjunto de aplicações e bibliotecas escaláveis, de modo a suportar a adição de novos tipos de dados, a arquitectura criada teve como premissas ser bastante flexível a nível de dependências e da gestão da informação sincronizável.

3.1 Arquitectura

De modo a garantir a flexibilidade desejada, a estrutura das aplicações foi dividida em três camadas distintas (Figura 8). Estas camadas encapsulam-se umas às outras, o que torna possível a escolha do nível de abstracção desejado. De modo a ser possível adicionar novos tipos de dados às bibliotecas, foi criado o conceito de módulos, que define uma interface à qual é possível adicionar novos módulos, sendo cada um responsável por gerir a informação de um tipo de dados diferente a sincronizar.

A conjugação destes dois conceitos, nomeadamente a modularidade e a divisão da aplicação por camadas e módulos leva à obtenção de uma arquitectura que não só é escalável, como também reutilizável em diversos tipos de aplicações.

Dadas às limitações e às diferenças estruturais entre plataformas, a implementação das várias bibliotecas variou em alguns aspectos, tendo sido necessário fazer alguns ajustes às camadas das aplicações e até mesmo criar novas camadas intermédias que foram encarregues de servir de ponte entre as camadas apresentadas na Figura 8.

Nas secções seguintes serão apresentadas as diversas diferenças que tiveram de ser implementadas na arquitectura devido às limitações das técnologias.

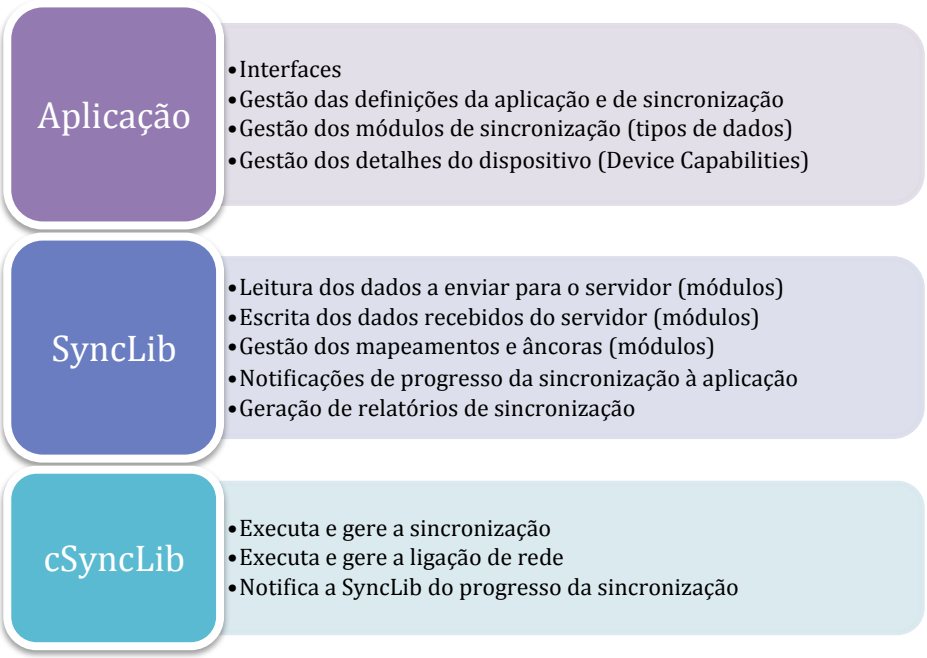


Figura 8. Camadas lógicas da arquitectura implementada

Como descrito na Figura 8, Cada camada tem um conjunto de funcionalidades associadas. A primeira camada da arquitectura, a cSyncLib, é a responsável por gerir toda a comunicação de rede necessária durante uma sincronização. É também nesta camada que o processo de sincronização, que engloba a preparação e leitura das mensagens SyncML, é realizado. Por esta ser a camada responsável pela gestão da ligação, é a este nível que o progresso da sincronização é processado, o qual é posteriormente enviado para a camada superior.

A segunda camada da arquitectura, a SyncLib, tem como objectivo gerir a informação que é sincronizada. Por poder haver vários tipos de dados a sincronizar, esta camada foi implementada em módulos, cada um responsável por um tipo especifico de dados. Por ser a camada que gere a informação, é a camada que mais de perto interage com as APIs de sistema, o que levou a que fosse desenvolvida nas linguagens nativas de cada plataforma para as quais foi implementada, tirando assim um maior partido das APIs existentes.

Para além dos dados sincronizáveis, esta camada gere também a informação necessária à implementação do protocolo SyncML, como o tipo de sincronização a executar e as âncoras de sincronização.

Ao gerir a informação sincronizada, a SyncLib está também encarregue de gerar os relatórios de sincronização, os quais fornecem informação sobre os itens que são transferidos durante

cada sincronização, especificando nestes o sentido em que os itens foram sincronizados (cliente para servidor ou servidor para cliente) e o tipo de acção a que foram sujeitos (adição, actualização ou remoção). Estes relatórios, assim como as notificações de progresso que a SyncLib recebe da camada inferior, são enviados para a camada superior de modo a poderem ser apresentados ao utilizador.

A terceira camada é a camada de interacção com o utilizador e de gestão de configurações. Esta recorre às outras duas camadas para executar sincronizações e é esta camada que está encarregue de mostrar as interfaces necessárias ao utilizador, assim como de gerir as definições de configuração para a sincronização. Estas definições vão desde a informação que o cliente tem de partilhar com o servidor sobre as suas capacidades, aos dados de configuração de servidor. Também é nesta camada que os módulos são geridos, dando assim a possibilidade ao utilizador de escolher os tipos de dados a serem sincronizados.

Ao serem implementadas desta forma, as aplicações finais podem ser decompostas em várias camadas independentes, o que aumenta a capacidade de reutilização do produto.

Também por ser baseada em módulos, a arquitectura é escalável aos mais diversos tipos de dados, os quais podem ser adicionadas à aplicação no futuro.

3.1.1 BlackBerry

Na plataforma BlackBerry foi necessário recorrer a uma biblioteca de sincronização desenvolvida em Java, a qual já englobava algumas das tarefas da segunda camada (SyncLib). Tendo-se aproveitado parte das tarefas já implementadas, a SyncLib na versão BlackBerry apenas ficou encarregue da gestão dos módulos de dados a sincronizar e dos relatórios de sincronização. Apesar de algumas tarefas desta camada terem passado para a biblioteca existente, esta teve também que ser alterada em alguns aspectos para que tal fosse possível.

3.1.2 Android

Na plataforma Android, baseada em Java, foi necessário recorrer ao JNI (Java-Native-Interface) para a criação de uma camada intermédia entre a cSyncLib e a syncLib, a qual possibilita a comunicação entre o Java da sycnLib com o C/C++ da cSyncLib.

3.1.3 iOS

Na plataforma iOS, devido ao facto da linguagem nativa ser o Objective-C, que por sua vez se é baseada em C/C++, não foi necessário criar nenhuma camada intermédia, tendo apenas sido necessário criar um pequeno wrapper que tem como objectivo servir de ponte entre ambas as linguagens para que o código esteja organizado.

3.2 cSyncLib

A criação da cSyncLib foi motivada pela necessidade de uma API que seja encarregue de executar e gerir toda a lógica de sincronização e comunicação. Esta biblioteca, representada na Figura 9, foi desenvolvida em C/C++, baseando-se nas bibliotecas *cURL* (biblioteca de comunicação), *smltoolkit* (parser de SyncML) e num wrapper para a *smltoolkit*, encarregue de gerir a sincronização.

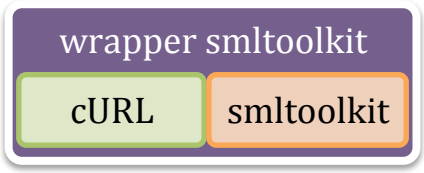


Figura 9. Estrutura da biblioteca cSyncLib

A biblioteca foi desenvolvida em C/C++ devido às dependências terem sido desenvolvidas em C e também pelo facto do C/C++ permitir à biblioteca ser utilizada num largo número de plataformas, como o Android e o iOS.

Devido ao facto da plataforma BlackBerry apenas possibilitar a programação em Java, a biblioteca não pôde ser utilizada nesta plataforma. A alternativa encontrada foi a utilização da biblioteca de sincronização da *Funambol,* que veio assim substituir esta biblioteca e parte da biblioteca SyncLib.

3.3 SyncLib

A biblioteca SyncLib foi implementada na linguagem nativa de cada plataforma, o que levou a que fosse desenvolvida de raiz para cada uma das plataformas, pois as plataformas utilizadas neste trabalho não partilham da mesma tecnologia ou linguagem. Assim, a

biblioteca foi desenvolvida para as plataformas BlackBerry (Java ME), Android (Java SE) e iOS (Objective-C).

A SyncLib encapsula a cSyncLib e adiciona-lhe toda a lógica necessária à gestão de dados a sincronizar. Também gere a informação de progresso, notificando a aplicação do estado da sincronização, e gera relatórios de sincronização, que são apresentados pela aplicação após cada sincronização bem sucedida.

Esta biblioteca consegue assim juntar num só ponto todo o trabalho necessário à realização de sincronizações, pelo que pode ser usada em qualquer tipo de projecto, nas plataformas para as quais foi desenvolvida, como uma API de sincronização para a qual não é necessário saber detalhes sobre a lógica de SyncML.

Em todas as versões da biblioteca foi criado o conceito de módulos. Os módulos representam os vários tipos de dados sincronizáveis e são geridos de modo a ser possível adicionar novos módulos à biblioteca. Cada módulo é responsável por:

- Obter/guardar os itens respectivos,
- Validar quais os itens que necessitam de ser sincronizados
- Gerir a sua âncora e modo de sincronização
- Fazer parse dos itens para o formato texto correspondente e vice-versa (por exemplo para contactos o formato a utilizar é o vCard)
- Gerir qualquer outra lógica ou informação que seja específica para o tipo de dados.

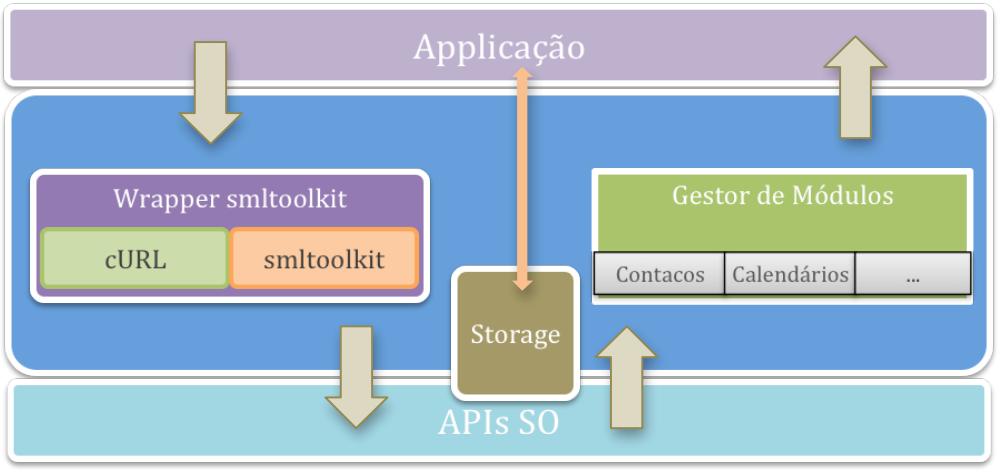


Figura 10. Estrutura da SyncLib

Como se pode ver na Figura 10, os componentes internos da biblioteca são a biblioteca cSyncLib, que gere toda a lógica de sincronização e comunicação, um data storage, implementado recorrendo a ferramentas dos SO, e o gestor de módulos que gere os módulos referentes aos tipos de dados sincronizáveis.

3.3.1 SyncLib na plataforma BlackBerry

No BlackBerry esta biblioteca não pôde recorrer à cSyncLib por limitações da plataforma, que apenas suporta Java ME, tendo esta sido substituída pela biblioteca de sincronização da Funambol. A biblioteca da Funambol gere a sincronização e a comunicação com o servidor assim como os dados a sincronizar.

Para efeitos de optimização, a biblioteca da Funambol foi alterada, de modo a tornar a criação e comparação dos mapeamentos de informação mais rápidos. Também outras alterações foram feitas para facilitar a integração com a biblioteca SyncLib.

3.3.2 SyncLib na plataforma Android

No cliente Android a biblioteca foi implementada como um serviço Android, o que possibilita a utilização da biblioteca em background e sem necessidade de recorrer a uma aplicação para ser executada. Deste modo foi possível implementar um sistema de sincronização automática que corre em segundo plano e que executa sincronizações automaticamente sempre que é detectada uma alteração nos dados ou em períodos pré-definido, sem que para isto seja necessário recorrer à aplicação.

3.4 Aplicações

As aplicações foram implementadas nas linguagens nativas das plataformas e apenas comunicam com a SyncLib. O objectivo geral das aplicações é disponibilizar uma interface para configuração do serviço de sincronização assim como fornecer uma interface para a execução da sincronização e para consulta dos relatórios de sincronizações. Para estas interfaces, pretendeu-se recorrer ao máximo aos componentes nativos de cada plataforma mas ao mesmo tempo criar uma “identidade” que fosse transversal a todas as plataformas. Com estes objectivos em mente, foram feitas várias iterações no desenvolvimento das interfaces.

As aplicações de Android e iOS para além das opções de sincronização e configuração, disponibilizam também a possibilidade de se executar uma limpeza à lista de contactos dos dispositivos, pois os sistemas operativos não disponibilizam esta opção de um modo transparente.

3.5 Serviço de monitorização de alterações

Para o cliente Android, de modo a possibilitar a execução de sincronizações automáticas baseadas em alterações aos dados, foi criado um serviço que tem como objectivo agendar sincronizações sempre que este é notificado da ocorrência de determinados eventos no sistema.

Este serviço, que está constantemente à escuta de eventos desde o arranque do sistema, reflecte a estrutura modular da SyncLib, pelo que lhe podem ser adicionados novos módulos de monitorização. No caso de não serem pretendidas sincronizações automáticas, o serviço pode ser desactivado através das definições da aplicação.

O serviço agenda também sincronizações em intervalos fixos. Este agendamento tem como objectivo verificar alterações no lado do servidor e pode também ser desactivado na aplicação.

3.6 Servidor de sincronização e cliente Android

No servidor de sincronização (Java SE) foi necessário fazer várias alterações de modo a torná-lo compatível com a estrutura dos contactos de Android. Devido ao facto destes contactos terem uma estrutura em árvore (Figura 11), em que cada contacto é composto por sub-contactos, e também ao facto de um dos objectivos do trabalho ser manter esta estrutura durante a sincronização, foi necessário fazer várias alterações para que tal se torna-se possível.

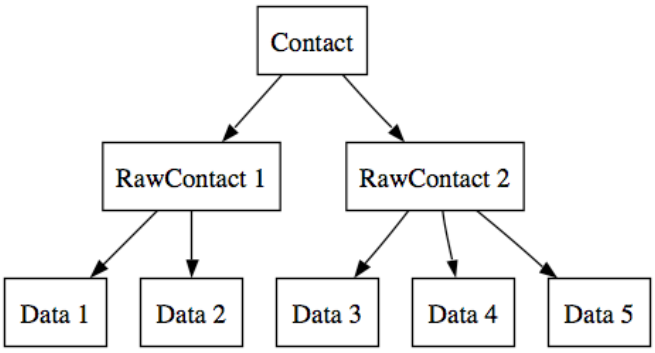


Figura 11. Estrutura de contactos Android [6]

Para manter a estrutura em árvore dos contactos (Figura 11) foi necessário mapear os ids dos sub-contactos do Android. Para a partilha de ids foi necessário criar um conjunto de extensões para o vCard, formato em que os contactos são partilhados sobre o SyncML, que adiciona capacidade de partilha dos ids. Na consequência destas alterações no formato do vCard, foi também necessário alterar os parsers de vCard, tanto do cliente como do servidor, para que ambos passassem a suportar estas extensões.

Para além do parser, foi também necessário alterar no servidor algumas estruturas de dados assim como a base de dados, de modo a possibilitar a persistência desta nova informação.

[6] http://developer.android.com/resources/articles/contacts.html

4 Clientes de sincronização para plataformas móveis

Neste capítulo vai ser descrito em maior detalhe o trabalho introduzido no capítulo 3. Aqui vão ser apresentadas em maior profundidade caracteristicas das bibliotecas utilizadas e implementadas neste trabalho. Irão tamém ser apresentadas as funcionalidades e interfaces das aplicações, o serviço de automatização de sincronizações para o Android e as alterações efectuadas no servidor de sincronização.

4.1 cSyncLib

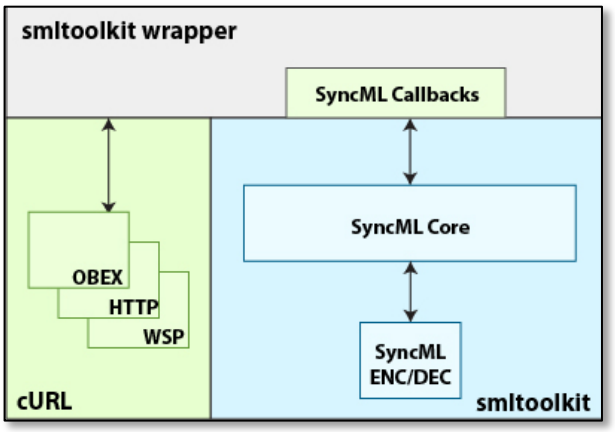


Figura 12. Arquitectura da biblioteca cSyncLib

A biblioteca cSyncLib (Figura 12) foi implementada com o propósito de ser uma API multi-plataforma com o objectivo de gerir toda a lógica de sincronização, tornando assim transparente a sincronização para quem a utilize como API de sincronização. A biblioteca foi também implementada de modo a ser escalável ao nível de tipos de dados sincronizáveis.

Devido ao facto de ter sido originalmente desenvolvida em ambiente Android, e este não ter disponível funcionalidades de *Debug* para C/C++, e também por ter sido baseada numa biblioteca orientada para *desktop*, houve algumas limitações arquitecturais e complicações no seu desenvolvimento, as quais acabaram por custar algum tempo ao projecto.

Estas limitações e complicações foram em grande parte devidas a dependências existentes na biblioteca original que não podem ser utilizadas em plataformas móveis e que devido a essa limitação tiveram de ser removidas ou substituídas. Outro factor de entropia foi o facto da biblioteca original ter várias funcionalidades que foram removidas para a nova biblioteca e que ao serem removidas deixaram alguma incoerência na estrutura da biblioteca. Durante o período de desenvolvimento deste trabalho, e sempre que foi necessário fazer alterações nesta biblioteca, tentou-se ir corrigindo estas incoerências, principalmente nas áreas em que se estava a trabalhar no momento, para deste modo melhorar a biblioteca e torná-la mais organizada.

4.1.1 smltoolkit

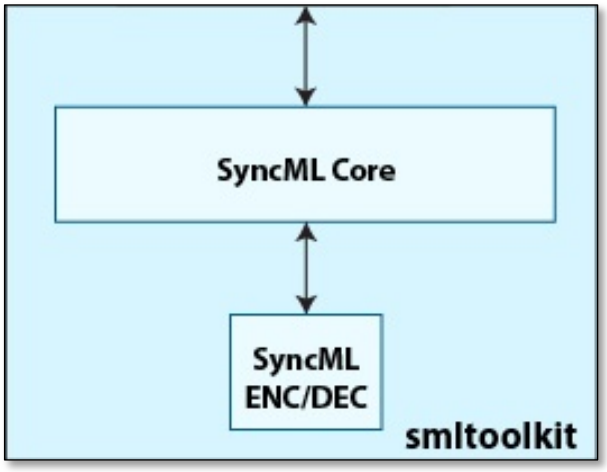


Figura 13. Arquitectura da biblioteca smltoolkit

A API smltoolkit (Figura 13) é a implementação da especificação de *SyncML* pela *SyncML Initiative, o* grupo que definiu a especificação. A definição e implementação desta API foi iniciada em 2000, ano em que foram lançadas quatro versões. A versão utilizada neste trabalho foi lançada no fim de 2003 e implementa a versão 1.1 do *SyncML*.

Esta API foi desenvolvida em C pois é objectivo da *SyncML Initiative* chegar ao maior número de dispositivos possível, o que o C garante pois é uma linguagem suportada por um largo número de plataformas e sistemas operativos.

A API apenas faz *parse* e composição de novas mensagens de *SyncML*, não tendo lógica de processamento de informação, lógica essa que foi implementada no *wrapper* da smltoolkit. A comunicação com o exterior é feita a partir de *callbacks*.

Esta API não apresentou muitos problemas nem limitações, tendo a única alteração à API sido a implementação do suporte para "*Large Objects*", a qual não estava completa.

O suporte para "*Large Objects*" [7] tem como objectivo possibilitar a sincronização de itens que são demasiado grandes para serem enviados numa única mensagem de *SyncM*L. Isto é conseguido dividindo estes itens em partes e enviando-as em várias mensagens sequenciais.

Na implementação do emissor é feita a divisão do objecto em partes menores do que o tamanho máximo que o cliente suporta, valor que é transmitido no inicio da sincronização. Estas partes são depois enviadas sequencialmente em mensagens compostas pela parte do item e o tamanho total do item (Figuras 14 e 15). Cada mensagem, excluindo a mensagem com a parte final do item, é concluida com a tag </MoreData>.

```
<Add>
    <CmdID>15</CmdID>
    <Meta>
        <Type>text/x-vcard</Type>
        <size>3000</size>
    </Meta>
    <Item>
        <Source>
            <LocURI>2</LocURI>
        </Source>
        <Data>BEGIN:VCARD
            VERSION:2.1
            FN:Bruce Smith
            N:Smith;Bruce
            TEL;WORK;VOICE:+1-919-555-1234
            TEL;WORK;FAX:+1-919-555-9876
            NOTE: here starts a huge note field, or icon etc...
        </Data>
        <MoreData/>
    </Item>
</Add>
```

Figura 14. Primeira parte de um LObject

[7] http://www.syncml.org/docs/syncml_sync_protocol_v11_20020215.pdf

```
<Add>
    <CmdID>28</CmdID>
    <Item>
        <Source>
            <LocURI>2</LocURI>
        </Source>
        <Data>here is the rest of the huge field.....
            blah, blah, blah.....
            END:VCARD
        </Data>
    </Item>
</Add>
```

Figura 15. Parte final de um LObject

Na implementação do receptor, este tem que detectar que está a receber partes de um item e processar a informação de modo a agrupar todos as partes num único objecto. Após todo o item ser transferido e reagrupado o receptor deve processar o item como que um item normal.

4.1.2 cURL

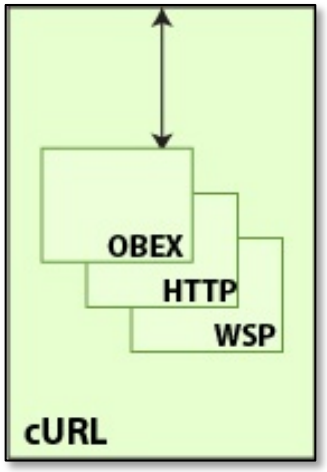


Figura 16. Biblioteca cURL (libcurl)

A API de transporte que foi utlizada no trabalho para executar as comunicações com o servidor foi a biblioteca cURL[8] (Figura 16). Esta API criada e mantida desde 2007 é frequentemente utilizada em projectos que necessitam de uma interface de rede. O cURL é implementado em C, o que o torna extremamente portável.

[8] http://curl.haxx.se/

Nenhuma alteração foi necessária à API mas, para a plataforma Android, foi necessário proceder a um conjunto de configurações[9][10], tanto na biblioteca como no ambiente de desenvolvimento, de modo a que fosse possível compilar a API. Para este processo, foi usado o Linux (Ubuntu) pois ao ser baseado em C/C++, este sistema operativo simplifica bastante a tarefa de compilação de projectos nesta linguagem.

Compilação do cURL para Android:

O processo de compilação da biblioteca libcurl para a plataforma Android, à data do trabalho, envolveu muita pesquisa a nível de como proceder e, como os próprios "tutoriais" indicam, muitas tentativas até finalmente se chegar a uma build. A pesquisa e tentativas necessárias tornaram uma tarefa que à partida tende a parecer rápida e trivial num processo algo moroso. Em seguida são descritos os passos mais importantes da compilação efectuada.

```
#!/bin/sh

ANDROID_ROOT="$HOME/android_src" &amp;amp;&amp;amp; \
TOOLCHAIN_VER="4.4.0"  \
PLATFORM_VER="5" \
CROSS_COMPILE=arm-eabi- \
PATH=$ANDROID_ROOT/prebuilt/linux-x86/toolchain/arm-eabi-$TOOLCHAIN_VER/bin:$PATH
&amp;amp;&amp;amp; \
CPPFLAGS="-I $ANDROID_ROOT/system/core/include -I$ANDROID_ROOT/bionic/libc/include
-I$ANDROID_ROOT/ndk/build/platforms/android-5/arch-arm/usr/include -
I$ANDROID_ROOT/bionic/libc/kernel/arch-arm -L $ANDROID_ROOT/prebuilt/linux-
x86/toolchain/arm-eabi-$TOOLCHAIN_VER/lib/gcc/arm-eabi/$TOOLCHAIN_VER/interwork -
L$ANDROID_ROOT/ndk/build/platforms/android-$PLATFORM_VER/arch-arm/usr/lib  -
L$ANDROID_ROOT/out/target/product/generic/system/lib " \
CFLAGS="-fno-exceptions -Wno-multichar -mthumb -mthumb-interwork -nostdlib -lc -ldl
-lm "  \
./configure CC=arm-eabi-gcc --host=arm-linux --disable-tftp --disable-sspi --
disable-ipv6 --disable-ldaps --disable-ldap --disable-telnet --disable-pop3 --
disable-ftp --without-ssl --disable-imap --disable-smtp --disable-pop3 --disable-
rtsp --disable-ares --without-ca-bundle --disable-warnings --disable-manual --
without-nss --enable-shared --without-zlib --without-random

# openssl/zlib version
#./configure CC=arm-eabi-gcc --host=arm-linux --disable-tftp --disable-sspi --
disable-ipv6 --disable-ldaps --disable-ldap --disable-telnet --disable-pop3 --
disable-ftp --with-ssl=$ANDROID_ROOT/external/openssl --disable-imap --disable-smtp
--disable-pop3 --disable-rtsp --disable-ares --without-ca-bundle --disable-warnings
--disable-manual --without-nss --enable-shared --with-
zlib=$ANDROID_ROOT/external/zlib --without-random
```

Figura 17. Script de exemplo para geração do ficheiro curl_config.h

O primeiro passo para a compilação da libcurl para a plataforma Android foi o download do código fonte do Android, o qual serve de "base" para a compilação da biblioteca. Após

[9] http://thesoftwarerogue.blogspot.com/2010/05/porting-of-libcurl-to-android-os-using.html

[10] http://osdir.com/ml/android-ndk/2010-03/msg00085.html

termos o código, adiciona-se o código da biblioteca libcurl na pasta *externals* do código fonte do Android.

De seguida é necessário criar o ficheiro curl_config.h, que define as configurações a serem aplicadas à biblioteca quando esta é compilada. Para facilitar este passo, vem juntamente com o código fonte da biblioteca um ficheiro Android.mk que contem um script de exemplo para a geração do curl_config.h, o qual serviu de base para a criação do script utilizado (Figura 17).

Após definir-se as propriedades para a biblioteca executa-se o script sob o código do Android. Este processo resulta em alguns erros, os quais têm de ser corrigidos, tornando este passo num processo de tentativa-erro até que o ficheiro seja finalmente criado com sucesso.

Após a criação do ficheiro curl_config.h estamos em condições de começar a tentar compilar a biblioteca. Para isto, na raiz da tree do Android, executa-se o comando "make curl". É preferível utilizar o comando "make curl" ao comando "make", pois os erros de compilação são mostrados e é mais rápido do que estar a compilar todo o Android. Neste passo é normal depararmo-nos com diversos erros de compilação, os quais tem de ser corrigidos, novamente num processo de tentativa-erro.

Após estes passos serem executados com sucesso temos uma build da libcurl configurada conforme as necessidades do projecto pronta a utilizar.

4.1.3 Wrapper smltoolkit

Figura 18. Wrapper smltoolkit

O wrapper smltoolkit (Figura 18), baseado numa biblioteca já existente, encapsula a API de parse e criação de mensagens de SyncML, a smltoolkit, assim como a API de comunicação, cURL. Este wrapper está deste modo encarregue de processar a informação a sincronizar e estabelecer e executar as comunicações com o servidor.

O wrapper foi feito em C/C++ para ser possível utilizar em Android, iOS e qualquer outra plataforma/SO que suporte C/C++.

Devido ao facto desta biblioteca ter sido baseada numa outra biblioteca já existente, houve alguma limitação na estruturação do código, o qual foi parcialmente reestruturado. Entre as alterações feitas foram retiradas algumas dependências ao wrapper, não só por serem demasiado pesadas e complexas para portar para dispositivos móveis, como é o caso da biblioteca wxWidgets (que estava entre outras coisas encarregue de gerir tipos de dados e notificações), mas também por se querer retirar algumas funcionalidades ao wrapper, como foi o caso do SQLite, que tinha como objectivo na biblioteca original gerir o mapeamento dos dados sincronizados.

Também a nível de reestruturação do código foi adicionado o conceito de módulos para facilitar a gestão dos vários tipos de dados a suportar durante as sincronizações.

Devido ao facto da obtenção e mapeamento de dados sincronizáveis terem sidos removidos, foi também necessário adicionar interfaces públicas que possibilitam a recepção e envio destes dados de e para a API. A comunicação com para o exterior é feita através de callbacks (Figura 19) que fornecem informações relevantes a quem a implementa, como é o caso dos dados recebidos do servidor e estado da sincronização.

```
typedef const char* (*ReceivedItemCallBack)(const char* id, const char* type, const
char* content, int action, void* user_data);

typedef void (*UpdateAnchorCallBack)(const char* anchor, const char* type, void*
user_data);

typedef void (*UpdateTotalReceiveItemsCallBack)(int count, void* user_data);

typedef void* (*FetchItemsCallBack)(int mode, const char* type, void* user_data);

typedef void (*UpdateSentItemsCountCallBack)(int count, void* user_data);

typedef void* (*ExecuteInsertsCallBack)(void* user_data);

typedef void (*ExecuteDeletesCallBack)(void* user_data);

typedef void (*ExecuteReplacesCallBack)(void* user_data);
```

Figura 19. Callbacks definidos na cSyncLib

```
/* native */
private native int sync(SyncConfigurations syncConfig);

public String sync() {
    L.d("wmb", "execute sync!");
    String s = String.valueOf(sync(syncConfig));
    L.d("wmb", "sync executed!");

    return s;
}
```

```
int sync(JNIEnv* env, jobject obj, jobject report){

     /* set settings into settings object */
     WMSClientSettings settings = setSettings(env, obj);

     /* initialize sync engine */
     syncEngine = new SyncEngine(&settings);

     /* run sync*/
     syncEngine->StartSyncEngine();

     int ret = syncEngine->GetErrorCode();
     DebugInt("return: %d", ret);

     /* sync report */
     if(ret == 0)
       fillReport(env, report, syncEngine->GetSyncReport(0));

     /* dump mem */
     SAFE_DELETE(syncEngine);

     return ret;
}
```

Figura 20. Exemplo de execução de métodos da cSyncLib no exterior (Android)

```
-(NSInteger)sync:(SLSyncSettings*)syncSettings {

    self.settings = syncSettings;

    /* set settings into settings object */
     WMSClientSettings *settings = new WMSClientSettings();
    [syncSettings toWMSClientSettings:settings];

    /* initialize sync engine */
    self.syncEngine = new SyncEngineWrapOpaque(settings);

    /* run sync*/
    self.syncEngine->syncEngine.StartSyncEngine();

    int ret = self.syncEngine->syncEngine.GetErrorCode();
    DebugInt("return: %d", ret);


    self.report = [SLReport reportWithSyncReport:(SyncReport*)self.syncEngine->sync
Engine.GetSyncReport(0)];


    /* dump mem */
    SAFE_DELETE(_syncEngine);
    SAFE_DELETE(settings);

    return ret;
}
```

Figura 21. Exemplo de execução de métodos da cSyncLib no exterior (iOS)

A comunicação para o interior é feita através da interface nativa do Java (Android), como pode ser visto na Figura 20, ou de acesso directo aos métodos públicos da biblioteca (iOS), como se pode ver nas Figura 21.

Outras funcionalidades também adicionadas ao wrapper foram a criação de relatórios de sincronização, notificações de progresso, o cancelamento da sincronização em curso e o suporte para "*Large Objects*".

4.2 SyncLib

A SyncLib foi idealizada como a API pública de sincronização que fornece todos os serviços de sincronização transparentemente para qualquer aplicação que necessite destes serviços. Para ser utilizada apenas é necessário indicar quais os tipos de dados a sincronizar e as *capabilities* do dispositivo que está a ser utilizado, sendo todo o restante processo tratado pela API.

Esta biblioteca foi implementada em três linguagens (Java ME (BlackBerry), Java SE (Android) e Objective-C (iOS)) de modo a integrar com as linguagens nativas dos dispositivos. É também esta a biblioteca que faz ponte entre a linguagem nativa da plataforma e o C/C++ da cSyncLib (nas versões Android e iOS).

Devido ao facto de ter sido implementada para plataformas diferentes, com diferentes tecnologias e arquitecturas, estas versões foram desenvolvidas com algumas diferenças entre si, diferenças estas que reflectem as limitações e funcionalidades especificas de cada plataforma.

4.2.1 SyncLib BlackBerry

A SyncLib para a plataforma BlackBerry não recorre à cSyncLib para efectuar sincronizações, pois nesta plataforma apenas é possível desenvolver recorrendo à linguagem do plataforma (Java) não sendo assim possível a utilização de código nativo (C/C++). Para contornar esta limitação da plataforma a solução encontrada foi procurar no mercado APIs de sincronização SyncML. Os requisitos para esta pesquisa foram a API ter de ser open-source e implementada em Java. Desta pesquisa apenas uma das APIs encontradas respeita ambas as premissas, pelo que a API da Funambol foi a escolhida para integrar no projecto.

Devido a esta limitação, a SyncLib para a plataforma BlackBerry foi implementada tirando partido dos atributos da API da Funambol, tendo apenas sido necessário implementar o gestor de módulos e alguma lógica para a gestão de relatórios e informação de progresso.

API Funambol

No decorrer do desenvolvimento do trabalho foram utilizadas duas versões desta API, pois este foi o primeiro cliente a ter a sua implementação iniciada e apenas veio a ser terminado após os clientes Android e iOS terem sido concluídos, altura em que já existia uma nova versão da API, mais completa.

Esta API é utilizada em vários dos clientes Funambol e tem muitas funcionalidades implementadas. No entanto, a documentação pública está num estado incompleto o que dificultou por vezes o uso da API. Também por ter sido feita a pensar nos clientes Funambol, a API suporta algumas tarefas que não foram necessárias para este trabalho, assim como não suporta algumas que eram requisito do trabalho, o que levou a que a API fosse alterada em alguns aspectos de modo a que seja possível suportar todas as tarefas necessárias a este trabalho.

De seguida vai ser descrito o processo de compilação da API e as alterações que foram feitas.

Compilação

Para compilar esta API houve a necessidade de se recorrer ao uso das seguintes ferramentas:

- **Apache Ant**: Ferramenta de compilação para Java, baseada em scripts;
- **Ant-Contrib Tasks**: Conjunto de *tasks* para Apache Ant;
- **Sun Java Wireless Toolkit**: *Toolbox* para desenvolvimento de aplicações baseadas em J2ME;
- **Apache Antenna**: Conjunto de *tasks* para Apache Ant;
- **BlackBerry JDE**: Ambiente de desenvolvimento para a plataforma BlackBerry;
- **bb-ant-tools**: Conjunto de *tasks* para Apache Ant para compilação de aplicações para a plataforma BlackBerry;
- **JavaCC**: gerador de *parsers* para Java.

Devido ao facto de a compilação depender de todas estas ferramentas, todas elas tiveram de ser configuradas de forma a funcionarem em conjunto. Também foi necessário alterar os script de compilação *ant*, fornecidos com a biblioteca, para que estes compilassem no ambiente montado.

Após a geração de todos os jar (ficheiros de dependências de Java), foi necessário executar uma pré-verificação (comando *preverify*) em cada um de modo a garantir que estes jar fossem utilizáveis num dispositivo BlackBerry.

Alterações

A API, que funciona de forma diferente à desejada para a SyncLib, teve que ser alterada em alguns aspectos. Estas alterações foram feitas com o objectivo de optimizar alguns processos, como é o caso do mapeamento de dados, ou para adicionar novas funcionalidades, como é o caso dos relatórios de sincronização.

- **Mapeamento de dados**

O mapeamento de dados da API é feito através do id e da hash obtida dos dados do item. Esta tarefa estava originalmente implementada tendo como dados de entrada para a criação das hashes as strings do conteúdo dos itens em utf-8.

O problema deste cenário está no facto de que os dados são gravados em bytes (byte[]), o que leva a que para a criação de uma hash, o item tenha que ser convertido para string em utf-8, conversão que num processo cíclico se torna muito pesado.

Após ser identificado este atraso, o algoritmo de criação de hash foi alterado de modo a receber os bytes dos itens, removendo assim a necessidade de conversão destes para string. O resultado da optimização foi uma diminuição drástica no tempo de geração de hashes, passando a geração de 100 hashes de perto de 2 minutos para 2 segundos.

Também para a validação de alterações aos dados locais, a ser executada no início de cada sincronização, foi necessário fazer alterações na API, as quais vieram possibilitar o mapeamento dos dados actuais para que, numa fase seguinte, este fosse comparado com o mapeamento do final da última sincronização, e assim detectar alterações aos itens.

- **Informação de progresso e relatórios**

Apesar de declarados, os métodos de notificação de progresso da API não estavam a funcionar, pelo que tiveram que ser alterados e em alguns casos, adicionados novos.

Também os relatórios fornecidos pela API não eram inteiramente os desejados, o que levou à necessidade da criação de um gestor de relatórios.

- **Storage**

A gestão de dados persistentes da SyncLib para BlackBerry é feita pela API da Funambol, pelo que não houve necessidade de implementar nenhuma persistência de dados a este nível.

Módulo de Contactos

O acesso à lista de contactos do BlackBerry é feito através da API de PIM (*Personal Information Management*) do Java ME, a qual possui estruturas próprias para manipulação de contactos. Após a obtenção destes dados, e devido à API de sincronização os receber na sua forma original, a comunicação entre a API de PIM e a de sincronização é feita de forma transparente, como pode ser visto nas Figuras 22 e 23.

```
protected PIMModule(IReportListener listener, Configurations configurations, int
pimType) {
    super(listener, configurations);

    store = storeFactory.getStringKeyValueStore(PIM_CONTACTS_MODULE);
    this.tracker = new PIMCacheTracker(store);

    this.pim = PIM.getInstance();
    PIMList list = null;
    try {
        list = pim.openPIMList(pimType, PIM.READ_WRITE);
    } catch(Exception ex) {
        Log.error(TAG, "Unable to get "+this.getDescription()+" list!", ex);
    }
    this.list = list;
}
```

Figura 22. Obtenção da lista de contactos

```
public int sync() {
    syncSource = new ContactSyncSource(config.getSourceConfig(), this.list,
getTracker());
    return super.sync();
}
```

Figura 23. Inicialização de uma sincronização de contactos

4.2.2 SyncLib Android

A SyncLib para a plataforma Android foi implementado recorrendo à cSyncLib como API de sincronização, através do uso do JNI/NDK que possibilita a comunicação entre código Java e código nativo (C/C++).

Devido ao facto de em Android ser possível criar serviços que correm em *background*, esta biblioteca foi criada como um serviço Android de modo a poder ser executada automaticamente sem intervenção do utilizador. Ao ser um serviço, a biblioteca pode também correr em segundo plano, mesmo quando a aplicação, por qualquer razão, é fechada a meio de uma sincronização.

"*A Service is an application component representing either an application's desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use.***"**, Android developer documentation[11]

Comunicação com a API cSyncLib

A comunicação da biblioteca com a API de sincronização é implementada recorrendo à interface nativa (JNI) e a um conjunto de ferramentas de criação de bibliotecas nativas disponível para Android (NDK).

Para utilizar esta interface foi necessário criar um conjunto de métodos (Figura 24) que possibilitam a comunicação no sentido Java → C/C++ e no sentido oposto, C/C++ → Java. Estes métodos estão encarregues de executar acções e passar informação entre as duas linguagens. Toda esta informação tem de ser devidamente convertida de modo a manter a integridade dos dados.

[11] http://developer.android.com/reference/android/app/Service.html

```
//-- receive
int Java_com_witsoftware_mb_synclib_SyncLib_sync(JNIEnv* env, jobject thiz, jobject
conf, jobject report);
void Java_com_witsoftware_mb_synclib_SyncLib_abortSync(JNIEnv* env, jobject thiz);


//-- send
//-- logger
void log_do_callback(int level, const char* message, void* user_data);

//-- received items
const char* receivedItem_do_callback(const char* id, const char* type, const char*
content, int action, void* user_data);

//-- update anchor
void updateAnchor_do_callback(const char* anchor, const char* type, void*
user_data);

//-- update total items from server
void updateTotalReceiveItems_do_callback(int count, void* user_data);

//-- fetch device item
void* fetchItems_do_callback(int mode, const char* type, void* user_data);

//-- sent items count
void updateSentItemsCount_do_callback(int count, void* user_data);

//-- execute inserts
void* executeInserts_do_callback(void* user_data);

//-- execute deletes
void executeDeletes_do_callback(void* user_data);

//-- execute replaces
void executeReplaces_do_callback(void* user_data);
```

Figura 24. Métodos declarados na camada de ligação para comunicação entre APIs

Comunicação com a aplicação

A comunicação entre o serviço e a aplicação é feita por mensagens enviadas em broadcast pelo serviço. Estas mensagens em broadcast são uma característica dos serviços de Android e funcionam como eventos que são enviados para toda a aplicação. Qualquer activity da aplicação pode estar à escuta destas mensagens, tendo que para isso estar registada como listener no manifest da aplicação ou dinamicamente na sua inicialização. Nas Figuras 25 e 26 é mostrado um exemplo de como estas mensagens são enviadas e recebidas.

```
public void sendReport(SyncReport report) {
     Intent intent = new Intent();
     intent.setAction(KEY_ACTION_PROGRESS_NOTIFICATION);
     intent.putExtra(KEY_REPORT, report.toArray());
     intent.putExtra(KEY_SYNC_DATE, report.getDateString());
     intent.putExtra(KEY_SYNC_IN_BACKGROUND, this.service.syncInBackground);
     this.service.sendBroadcast(intent);
}
```

Figura 25. Envio do relatório de sincronização por broadcast

```
public BroadcastReceiver progressReceiver = new BroadcastReceiver() {

  @Override
  public void onReceive(Context context, Intent intent) {
    if (intent.getAction().equals(ProgressBinder.KEY_ACTION_PROGRESS_NOTIFICATION))
    {

      //-- reset finished ?
      if(intent.getBooleanExtra(ProgressBinder.KEY_RESET_FINISHED, false)) {
        SynchronizationActivity.this.isWorking = false;
        SynchronizationActivity.this.showProgress(false);
        return;
      }

      //-- is sync finished?
      if(intent.getBooleanExtra(ProgressBinder.KEY_UNBIND, false)) {
        int state = intent.getIntExtra(ProgressBinder.KEY_STATUS, 0);
        int errCode = intent.getIntExtra(ProgressBinder.KEY_ERROR_CODE, 0);
        boolean isSilent = intent.getBooleanExtra(ProgressBinder.KEY_SYNC_IN_
BACKGROUND, false);

        SynchronizationActivity.this.showProgress(false);

        if(!isSilent)
          SynchronizationActivity.this.showDialog(errCode > 0 ? errCode : state);

        SynchronizationActivity.this.isWorking = false;
        return;
      }

      //-- report
      long[] report = intent.getLongArrayExtra(ProgressBinder.KEY_REPORT);
        if(report != null) {

          SynchronizationActivity.this.setReport(report, intent.getStringExtra(
ProgressBinder.KEY_SYNC_DATE));
          return;
        }

        //-- progress
        String label = intent.getStringExtra(ProgressBinder.KEY_LABEL);
        int progress = intent.getIntExtra(ProgressBinder.KEY_PROGRESS, 100);
        boolean isIndeterminated = intent.getBooleanExtra(ProgressBinder.KEY_IS_
INDETERMINATED, true);

        SynchronizationActivity.this.updateProgress(label, progress,
isIndeterminated);

        MobileBackupPreferences.setProgressLabel(SynchronizationActivity.this,
label);
        MobileBackupPreferences.setProgressValue(SynchronizationActivity.this,
progress);

        MobileBackupPreferences.setProgressIndeterminated(Synchronization
Activity.this, isIndeterminated);
      }
   }
};
```

Figura 26. Recepção do relatório de sincronização por broadcast

No sentido aplicação → serviço não é necessário haver comunicação, pois todas as definições de sincronização, configuráveis na aplicação, estão guardadas na storage do SO, o que possibilita o acesso à informação não só pela aplicação mas também pelo serviço.

Armazenamento de dados e configurações

O Android fornece um conjunto de ferramentas de armazenamento de dados que facilitam a gestão de informação persistente. A ferramenta utilizada neste projecto foi a classe *SharedPreferences* que fornece uma forma fácil de armazenar informação privada, de diversos tipos, recorrendo a um formato chave – valor (Figura 27). A informação armazenada é acedível a partir de qualquer activity da aplicação e apenas no contexto da aplicação, pelo que nenhuma outra aplicação tem acesso à informação guardada.

```
public static SharedPreferences getPreferences(Context context)
{
     return context.getSharedPreferences(PREFERENCES_FILE,Context.MODE_PRIVATE);
}

public static Editor getEditor(Context context)
{
     return getPreferences(context).edit();
}

public static boolean getBoolean(Context context, String key, boolean defaultValue)
{
     boolean result = getPreferences(context).getBoolean(key, defaultValue);
     L.v(TAG, "Fetched key: " + key + ", value: " + result);
     return result;
}


public static String getString(Context context, String key, String defaultValue)
{
     String result = getPreferences(context).getString(key, defaultValue);
     L.v(TAG, "Fetched key: " + key + ", value: " + result);
     return result;
}


public static void putBoolean(Context context, String key, boolean value)
{
     L.v(TAG, "Saving key: " + key + ", value: " + value);
     Editor editor = getEditor(context);
     editor.putBoolean(key, value);
     editor.commit();
}

public static void putString(Context context, String key, String value)
{
     L.v(TAG, "Saving key: " + key + ", value: " + value);
     Editor editor = getEditor(context);
     editor.putString(key, value);
     editor.commit();
}
```

Figura 27. Exemplo de utilização da classe SharedPreferences

4.2.3 SyncLib iOS

A SyncLib para a plataforma iOS foi também implementada recorrendo à cSyncLib como API de sincronização. No entanto, devido ao facto de o Objective-C, linguagem nativa do iOS, ser um *superset* do C/C++, não foi necessário recorrer ao JNI para fazer a ponte entre as duas linguagens, o que tornou a implementação mais fácil e compreensível.

Comunicação com a cSyncLib

A comunicação com a API de sincronização é feita através de um pequeno *wrapper* que foi definido de modo a isolar o C/C++ do Objective-C. Deste modo sempre que é necessário aceder a um método da API de sincronização, este é acedido através de um método de Objective-C. No sentido inverso, sempre que o *wrapper* recebe informação da API, através de um *callback*, este notifica a SyncLib através de um *delegater* responsável pela acção.

- **Ordem de cancelamento de sincronização (exemplo)**

Como pode ser visto na Figura 28, ao receber um pedido de cancelamento da sincronização em curso, a SyncLib delega à cSyncLib o pedido através do *wrapper*. Neste ponto é feita a transição entre Objective-C e C/C++. Ao recorrer a estes métodos de ponte entre as linguagens consegue-se evitar misturas de linguagens, mantendo o código simples e evitando alguns problemas, como por exemplo má gestão de memória.

```
-(void)abortSync {
     self.syncEngine->syncEngine.AbortSync();
}
```

Figura 28. Exemplo de execução de método C/C++ em Objective-C

- **Notificação de número de itens a receber (exemplo)**

Ao receber o número de alterações no servidor, a cSyncLib notifica a SyncLib deste valor, através de um *callback*. Esta informação é recebida no *wrapper* e enviada para o seu *delegater* (Figura 29). Deste modo é feita a transição entre o C/C++ e o Objective-C de um modo transparente.

```
//-- update total items from server
void updateTotalReceiveItems_do_callback(int count, void* user_data) {
    if([[self delegate] respondsToSelector:@selector(WPSyncEngine:update
ReceiveTotalWithValue:)]) {
        [[self delegate] WPSyncEngine:self updateReceiveTotalWithValue:count];
    }
}
```

Figura 29. Wrapper aos callbacks da cSyncLib

Comunicação com a aplicação

A comunicação entre a biblioteca e aplicação no sentido biblioteca → aplicação é feita através de notificações (*NSNotificaitons*). A vantagem principal das notificações perante os *delegaters* está em que podem ser escutadas em qualquer ponto da aplicação, e por mais do que um *listener*. Deste modo foi definido um conjunto de notificações que têm como objectivos fornecer informação sobre:

- **O estado da sincronização:**
  - GENERAL_PROGRESS_NOTIFICATION
  - SENT_ITEMS_PROGRESS_NOTIFICATION
  - RECEIVED_ITEMS_PROGRESS_NOTIFICATION
- **Relatório de sincronização (após esta terminar)**
  - SYNC_REPORT_NOTIFICATION
- **Estado da execução de *reset* à lista de contactos**
  - AB_RESET_FINISHED_NOTIFICATION

SYNC_REPORT_NOTIFICATION (exemplo)

Como pode ser visto na Figura 30, ao receber o relatório da sincronização, a biblioteca cria uma notificação com o relatório anexado. Esta notificação pode ser processada em qualquer ponto da aplicação e ser recebida por vários receptores simultaneamente.

```
-(void)WPSyncEngine:(WPSyncEngine *)wrapper syncReport:(SLReport *)syncReport
{
   //-- notification
    NSNotification* notification = [NSNotification notificationWithName:SYNC_
REPORT_NOTIFICATION object:syncReport];
    [[NSNotificationCenter defaultCenter] postNotification:notification];
}
```

Figura 30. Criação de notificação com o relatório da sincronização em anexo

Registo de *listener* para SYNC_REPORT_NOTIFICATION (exemplo)

Ao se registar um *listener* para uma notificação é definido o método que vai ser responsável pela tarefa associada à notificação, no exemplo da Figura 31, o método a ser executado é o p_syncReportNotification.

```
[[NSNotificationCenter defaultCenter] addObserver:self
          selector:@selector(p_syncReportNotification:)
              name:SYNC_REPORT_NOTIFICATION object:nil];
```

Figura 31. Registo de um listener para notificações

Armazenamento de dados

O armazenamento de definições e mapeamentos, à semelhança da aplicação Android, foi implementado recorrendo a ferramentas do sistema operativo, que permitem a persistência das definições, num formato chave – valor, de um modo standard.

"*The `NSUserDefaults` class provides a programmatic interface for interacting with the defaults system. The defaults system allows an application to customize its behavior to match a user's preferences. (…) Applications record such preferences by assigning values to a set of parameters in a user's defaults database. The parameters are referred to as defaults since they're commonly used to determine an application's default state at startup or the way it acts by default.*", NSUserDefaults class reference[12]

Deste modo é possível aceder a uma lista de informação persistente, no contexto da aplicação, da mesma forma que se acede a um simples dicionário de dados, como é mostrado na Figura 32.

```
[NSUserDefaults standardUserDefaults]; //-- initilization
[_settings setValue:obj forKey:idx];   //-- add object obj with key idx
[_settings synchronize];               //-- save changes
[_settings objectForKey:idx];          //-- read object with key idx
```

Figura 32. Exemplo de utilização da classe NSUserDefaults

Mapeamento de dados sincronizáveis

Para o módulo de sincronização implementado (contactos), e ao contrário do módulo de contactos implementado em Android, em que é necessário guardar o id e uma *hash* para

[12] http://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSUserDefaults_Class/Reference/Reference.html

cada contacto, no iOS apenas é necessário guardar o *id* de cada contacto existente. Isto deve-se ao facto de a lista de contactos do iOS ter para cada contacto o *timestamp* da criação do contacto e o *timestamp* da última alteração ao contacto.

Deste modo, o mapeamento dos itens é feito criando um *array* com os *ids* de todos os contactos existentes. Este *array* será usado apenas para detecção de remoções de contactos entre sincronizações, visto as adições e edições serem validadas através dos *timestamps* dos contactos.

Estes mapeamentos são actualizados e guardados na *storage* da aplicação no fim de cada sincronização através da interface descrita acima.

Para além dos mapeamentos dos ids, são também persistidos a âncora da sincronização, a data da sincronização (para comparação com os *timestamps* dos contactos na próxima sincronização), e o modo de sincronização utilizado.

4.3 Aplicações

As aplicações desenvolvidas no âmbito deste trabalho têm como objectivo disponibilizar uma interface simples e intuitiva para a execução de sincronizações de contactos baseadas em SyncML. As aplicações foram implementadas nas linguagens nativas das plataformas e recorrem às bibliotecas descritas neste documento para executar as sincronizações. De seguida serão apresentadas as aplicações e as suas funcionalidades.

4.3.1 Aplicação BlackBerry

As interfaces para a aplicação BlackBerry foram implementadas recorrendo a componentes desenvolvidos especificamente para esta aplicação. A razão que levou a que estes componentes fossem desenvolvidos prende-se com o facto de que os componentes nativos da plataforma BlackBerry são extremamente simplistas e pouco apelativos, o que iria dificultar a criação de uma imagem para o produto que se mantivesse minimamente relacionada nas três plataformas.

Requisitos mínimos

Sistema Operativo

Sendo requisito para a aplicação o suporte ao maior número de dispositivos possível, sem se comprometer deste modo a utilização da API mais recente, a versã0 4.1.6 do sistema operativo foi definida como requisito.

JRE - Java Runtime Environmet

Também com o intuito de suportar o maior número de dispositivos possível, a versão base da biblioteca de sistema escolhida para a aplicação foi a versão BlackBerry JRE 5.0.0.

Componentes criados

Botões

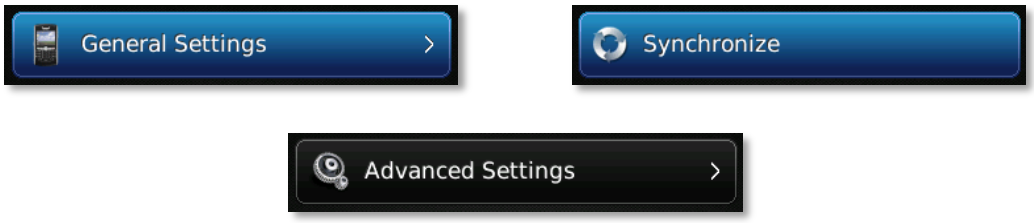


Figura 33. Botões criados para a aplicação BlackBerry

Um dos componentes obrigatórios numa aplicação é o botão. Para a aplicação BlackBerry foi criado um componente que, recorrendo às APIs da plataforma, desenha um botão completamente dinâmico (Figura 33). Este componente permite a configuração de diversos aspectos do botão, desde o tipo de letra do texto a apresentar, medidas e cores de fundo à existência de um icone no lado esquerdo do botão ou de uma seta do lado direito, utilizada na aplicação como indicação da existência de um novo nível de menu.

Como é habitual em botões, este componente suporta também vários estados, que são distinguidos visualmente pelo esquema de cores de fundo.

Uma caracteristica comum à maioria dos componentes criados é a possibilidade de estes se agruparem visualmente, de modo a criar uma ideia de grupos dentro de uma secção.

Caixa de seleção

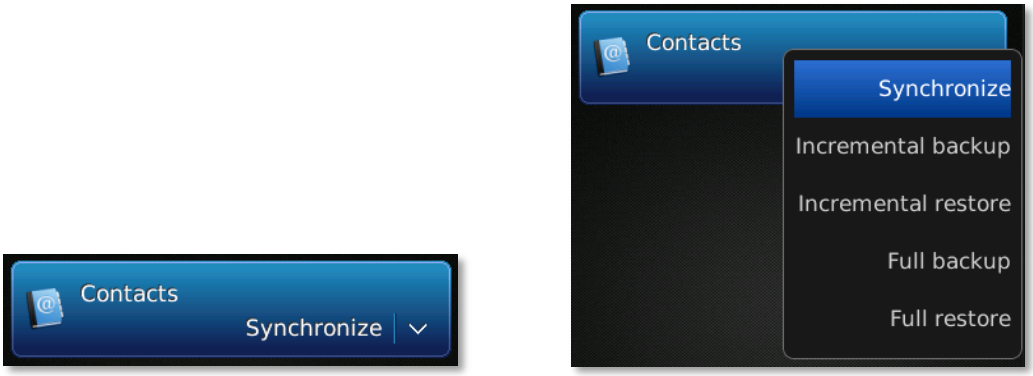


Figura 34. Caixas de selecção para a aplicação BlackBerry

Devido a haver a necessidade de pedir ao utilizador configurações que residem num domínio definido, foi também criado um componente de selecção que apresenta ao utilizador as opções existentes (Figura 34). Como o botão, este componente disponibiliza opções de configuração a nível de cores, texto e ícones. A opção seleccionada é apresentada no canto inferior direito do componente quando este se encontra fechado. Ao ser pressionado, o componente mostra uma vista, que se sobrepõe a todos os componentes vísiveis, com as diversas opções à escolha. Ao ser seleccionada uma opção, esta vista é removida e a opção seleccionada actualizada.

Devido a ser um componente que tem um objectivo e uma forma característicos, não pode ser agrupado com os outros componentes desenvolvidos.

Caixa de texto

Figura 35. Caixas de texto para a aplicação BlackBerry

Outro componente muito comum em qualquer aplicação é a caixa de texto, o qual permite a introdução de texto pelo utilizador. O componente criado para esta aplicação tem o mesmo nível de configuração dos outros componentes, assim como pode também ser agrupado. Este componente para além de aceitar a introdução de texto pelo utilizador pode também apresentar uma descrição, a qual é apresentado sobre a área de escrita (Figura 35).

Devido à necessidade de inserção de passwords na aplicação, foi também criada a opção de ocultar o texto inserido, substituindo a representação dos caracteres por asteriscos.

Toggles

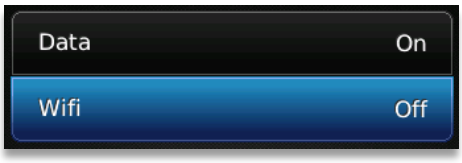


Figura 36. Toggles para a aplicação BlackBerry

Por haver a necessidade de disponibilizar escolhas binárias ao utilizador, foi também criado um componente que permite escolher entre duas opções pré-definidas (Figura 36). Este componente comporta-se como um botão, mas tem como diferença principal a existência de um texto indicativo de estado, configurável, no lado direito do botão. Este componente pode ser configurado em todos os atributos como um botão excepto à presença da imagem do lado direito do componente, assim como pode ser agrupado com outros componentes.

Label

Figura 37. Label para a aplicação BlackBerry

Por ter sido necessário apresentar informação estática, foi também criado um componente que pode mostrar um texto e uma imagem, como pode ser visto na Figura 37. É um elemento passivo, sem acções e que serve principalmente para criar divisões dentro de uma secção e situar o utilizador da aplicação no conteúdo da informação exposta.

Caixas de diálogo

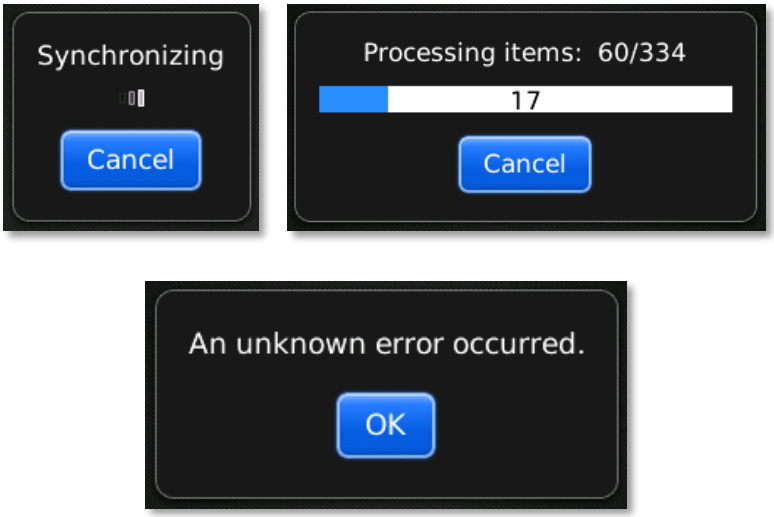


Figura 38. Caixas de diálogo para a aplicação BlackBerry

Um método habitual de apresentação de informação ao utilizador de um dispositivo móvel é a utilização de caixas de diálogo. Para esta aplicação criaram-se um conjunto de caixas de dialogo, baseadas nas caixas de dialogo do sistema, e que fornecem informação de erro, sucesso e progresso ao utilizador (Figura 38). Estas são apresentadas sobre a vista actual, bloqueando o acesso a esta enquanto estiver visível.

Secções da aplicação

A aplicação foi desenvolvida de modo a que em qualquer secção da aplicação se tenha acesso directo às secções principais. Este acesso é garantido através do menu que pode ser apresentado em qualquer uma das secções através da tecla Menu do dispositivo.

Sincronização

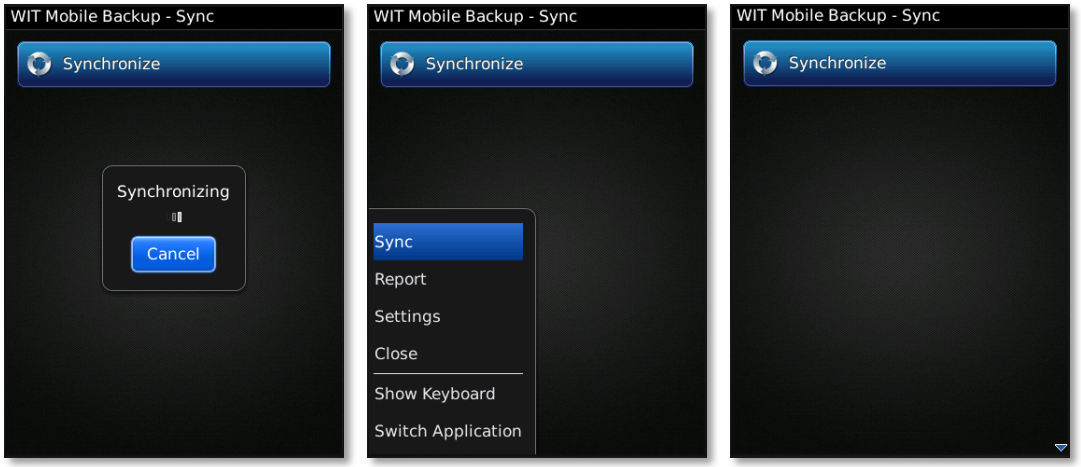


Figura 39. Secção de sincronização na aplicação BlackBerry

Esta secção é a secção apresentada por pré-definição quando a aplicação é iniciada e os dados de configuração foram previamente preenchidos. O objectivo desta secção é disponibilizar a opção de execução de uma sincronização ao utilizador, através do botão Sincronizar (Figura 39). Após cada sincronização terminada com sucesso é apresentado um pequeno relatório, por baixo do botão Sincronizar, com a data da última sincronização.

Menu de definições

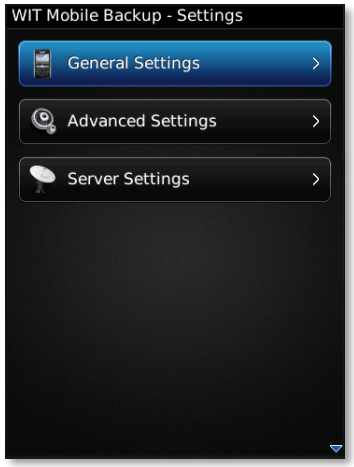


Figura 40. Secção de menu de definições na aplicação BlackBerry

Na secção de definições (Figura 40) pode-se navegar nos vários botões e escolher o sub-menu de definições pretendido. As opções são:

- **Definições gerais** – nesta secção define-se o tipo de rede a utilizar (sem aviso).
- **Definições avançadas** – nesta secção define-se o tipo de sincronização a utilizar. Está preparada para, ao serem adicionados novos tipos de dados, disponibilizar automaticamente as definições respectivas, não sendo necessário estar a alterar a secção para reflectir este tipo de alterações à aplicação.
- **Definições de servidor** – Nesta secção define-se o endereço do servidor a utilizar e os dados do utilizador para autenticação no servidor.

Definições gerais

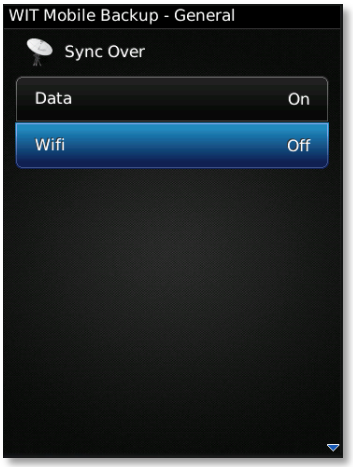


Figura 41. Secção de definições gerais na aplicação BlackBerry

Na secção definições gerais (Figura 41) é possível configurar os avisos de tipo de rede a ser utilizada durante a sincronização. Estas definições servem para que o utilizador, que tenha por exemplo um plano de tráfego limitado, saiba quando está prestes a fazer uma sincronização sobre 3G ou Wifi.

No caso de haver mais do que um módulo na aplicação é apresentada também um conjunto de opções para a activação ou desactivação dos diferentes módulos. Estas opções são automáticas pelo que não é necessário fazer nenhuma alteração na secção.

Definições avançadas

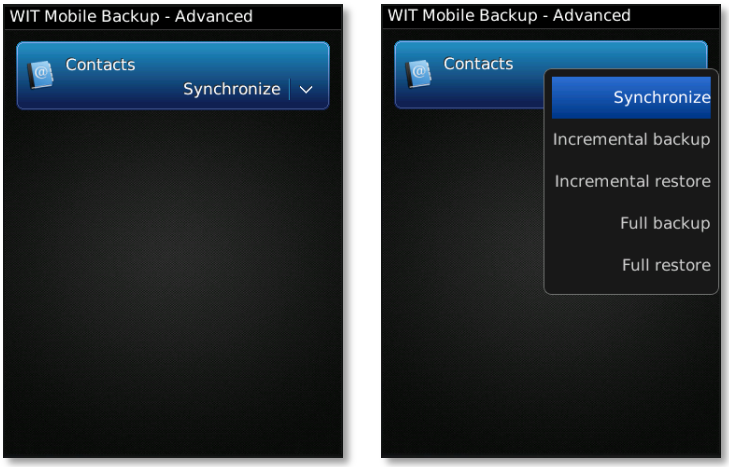


Figura 42. Secção de definições avançadas na aplicação BlackBerry

Nesta secção é possível ao utilizador configurar o tipo de sincronização que quer que seja executada para cada tipo de dados. Cada tipo de dados presente na aplicação é representado por uma caixa de selecção através da qual se procede à escolha do tipo desejado (Figura 42).

A adição de novos módulos à aplicação faz com que a secção se adapte automaticamente às alterações pelo que não é necessário fazer nenhuma alteração nesta secção para adicionar ou remover os tipos de dados.

Definições de servidor

Figura 43. Secção de definições de servidor da aplicação BlackBerry

Nesta secção, representada na Figura 43, o utilizador pode definir os endereço do servidor ao qual a aplicação se vai ligar para efectuar a sincronização. Também nesta secção o utilizador tem de configurar os dados necessário para a autenticação perante o servidor. Esta secção é apresentada por pré-definição na primeira execução da aplicação e sempre que a aplicação seja aberta e haja dados de servidor em falta.

Relatórios

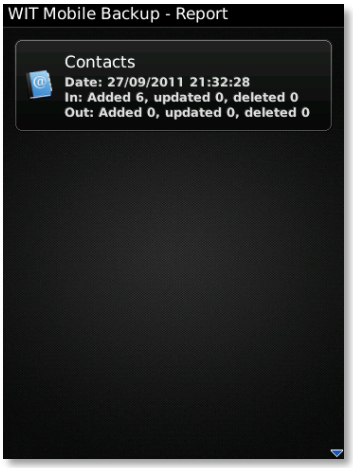


Figura 44- Secção de relatórios na Aplicação BlackBerry

Esta secção tem como objectivo fornecer informação sobre a última sincronização bem sucedida. Novamente, para cada módulo instalado na aplicação irá aparecer automaticamente uma secção com a informação relativa ao tipo de dados (Figura 44).

A informação apresentada é referente ao número de itens copiados durante a sincronização, detalhando o sentido, e o tipo de acção, quer esta tenha sido adicionada, editada ou removida.

4.3.2 Aplicação Android

A aplicação Android distingue-se das restantes por ser a única que recebeu a implementação de um serviço de sincronização automática. Nesta aplicação a UI foi também feita à base de componentes criados especialmente para aplicação, os quais são baseados em imagens, ao contrário dos componentes dinâmicos criados para o BlackBerry, que são desenhados pela API.

Esta aplicação disponibiliza as mesmas opções da aplicação BlackBerry com a adição das opções de configuração das sincronizações automática e agendada e da opção de execução de limpeza à lista de contactos.

Requisitos mínimos

Sistema Operativo

Estando o Android actualmente na versão 3.0, a versão 2.1 foi definida como requisito mínimo para o cliente, pois esta é a versão em que foi lançada a API de contactos actual.

Android API

A versão 7 da API de desenvolvimento para Android foi a escolhida para requisito mínimo por esta ser a API associada à versão 2.1 do Android.

NDK

A escolha da versão do NDK a utilizar neste projecto recaiu sobre a versão 4b, por ser a mais recente a quando do inicio dos desenvolvimentos para o cliente.

Secções

Sincronização

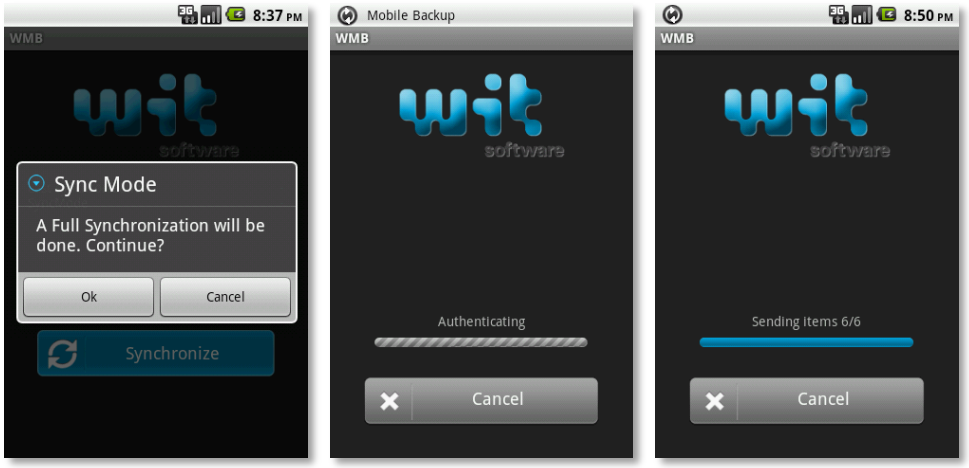


Figura 45. Secção de sincronização no cliente Android – sincronização

Na secção de sincronização é possível executar uma sincronização e aceder aos menus de configurações, através do menu de opções do Android. Sempre que uma sincronização completa ou de restauro, em que seja possível perder dados, a aplicação avisa o utilizador deste facto, pedindo em simultâneo uma confirmação de continuação. Durante uma sincronização é apresentada ao cliente informação sobre o estado da sincronização, o que é

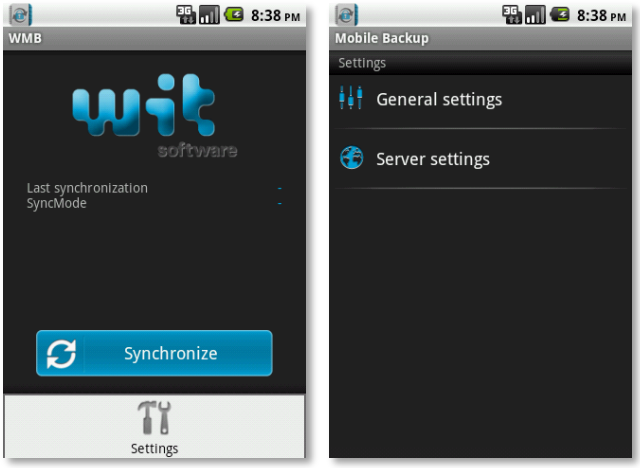
feito recorrendo a uma barra de progresso e a um texto descritivo da tarefa em execução (Figura 45). Em qualquer altura durante uma sincronização, o utilizador tem disponível a opção de cancelamento da sincronização através do botão Cancelar.

Ao ser executada durante uma sincronização automática, a aplicação assume o estado da sincronização, o qual é apresentado como se esta tivesse sido executada pelo utilizador. Em qualquer outro caso, a sincronização automática é sempre feita em background sem que o utilizador seja notificado.

Após a execução de uma sincronização com sucesso, é apresentado também nesta secção um relatório da ultima sincronização e que descreve as acções e volume de itens sincronizados, como pode ser visto na Figura 46.

Figura 46. Secção de sincronização no cliente Android – relatório

Definições



a b

Figura 47. Secções de sincronização (a) e definições (b) no cliente Android

A secção de definições, acedida através do menu de opções do Android (Figura 47 a) disponibiliza um menu onde o utilizador pode escolher o tipo de definições que quer configurar (Figura 47 b). As definições gerais têm definições como a configuração das sincronizações automática e agendada, tipo de sincronização a utilizar, alertas de rede e a opção de limpeza da lista de contactos. As definições de servidor possibilitam a configuração do dados de servidor de sincronização.

Definições gerais

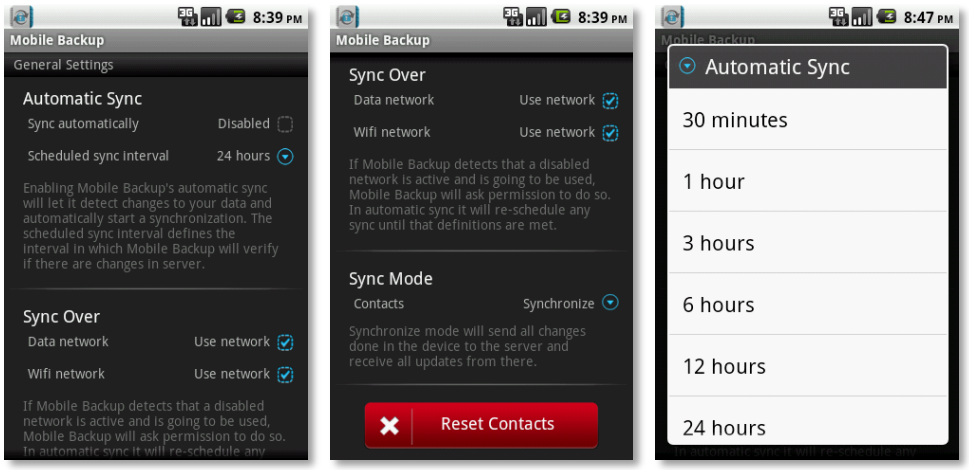


Figura 48. Definições gerais na aplicação Android

Nesta secção é possível configurar diversas definições da aplicação, nomeadamente as sincronizações automáticas, alertas e o tipo de sincronização a utilizar. Para além destas configurações é também possível executar a limpeza da lista de contactos. Esta funcionalidade foi adicionada à aplicação por nem todas as versões do Android a terem nativamente disponível.

A sincronização automática tem duas vertentes, a agendada, que é executada num intervalo definido pelo utilizador entre sincronizações, e uma mais ubíqua que é despoletada por eventos do sistema operativo, como por exemplo a actualização da base de dados de contactos do dispositivo. Estas funcionalidades podem ser activadas ou desactivadas pelo utilizador.

Os alertas de rede têm como objectivo avisar o utilizador de que uma sincronização está prestes a começar e que utilizar um determinado tipo de rede. Estes avisos podem ser úteis para utilizadores que tenham planos de tráfego limitados e queiram controlar os custos.

Nesta plataforma, esta configuração tem também outro objectivo, o qual é cancelar uma sincronização automática sempre que esta tente utilizar uma configurada com alerta activo. Deste modo a sincronização é adiada até voltar a haver uma conexão num outro tipo de rede que não esteja configurado para lançar alerta.

Definições de servidor

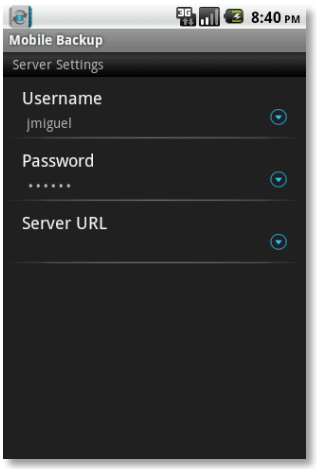


Figura 49. Secção de definições de servidor na aplicação Android

Nesta secção é possível ao utilizador configurar os dados de acesso ao servidor (Figura 49). Esta secção é apresentada por pré-definição ao utilizador sempre que esteja pelo menos um dos campos em falta. Caso contrário, a secção pré-definida da aplicação é a secção de sincronização.

Notificações

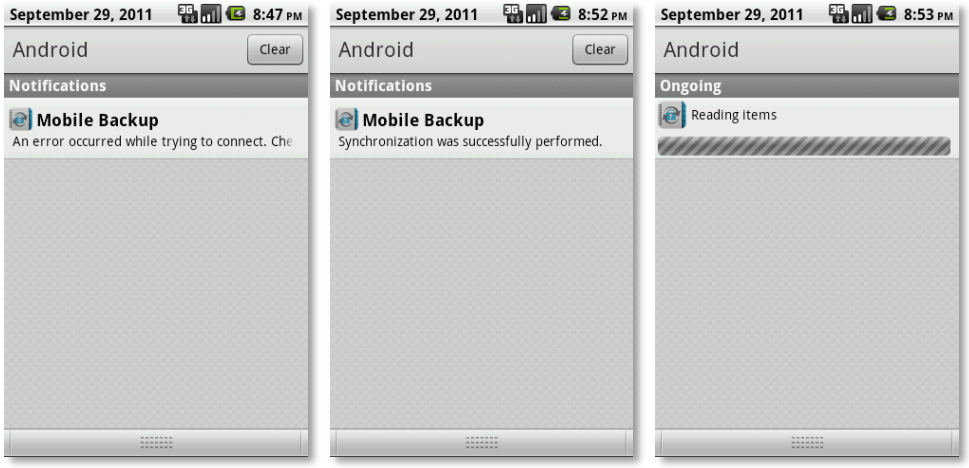


Figura 50. Notificações para a aplicação Android

Durante a utilização da aplicação é possível consultar o progresso de uma sincronização através das notificações do Android, como pode ser visto na Figura 50. Estas notificações são apresentadas sempre que uma sincronização é iniciada pelo utilizador e disponibilizam a informação do progresso da sincronização, apresentando também no fim de cada sincronização uma mensagem de sucesso ou erro. Ao ser seleccionada uma notificação, a aplicação é tornada activa, passando a aplicação aberta para background.

4.3.3 Aplicação iPhone / iPod

A aplicação para iPhone foi implementada recorrendo na maioria do casos a componentes de UI da plataforma. Todas as propriedades implementadas no cliente Android foram também aqui implementadas, à excepção da sincronização automática devido ao facto do iOS não suportar serviços em background.

Todas as secções da aplicação podem alternar entre a secção de sincronização e a de definições, recorrendo para isso ao *TabBarMenu* do fundo da interface. Esta opção é no entanto desactivada durante uma sincronização.

Requisitos mínimos

Sistema Operativo

Este cliente tem como requisito mínimo para a versão do sistema operativo a versão 3.1.2, por esta ser a versão correspondente à versão 3.2 do iPad.

Secções da aplicação

Sincronização

Figura 51. Secção de sincronização na aplicação iPhone

Esta secção possibilita ao utilizador da aplicação executar uma sincronização de contactos. Após cada sincronização executada com sucesso, é apresentado ao utilizador o relatório da última sincronização, o qual fica presente até à próxima execução bem sucedida de uma sincronização (Figura 51). Este relatório apresenta o tipo de sincronização efectuada, e o volume de contactos transferidos, especificando o sentido e o tipo de acção. Durante uma sincronização é apresentada uma barra de progresso com a informação das acções a decorres apresentada a cima desta barra. Também é apresentada ao utilizador a opção de cancelamento da sincronização em curso.

Definições

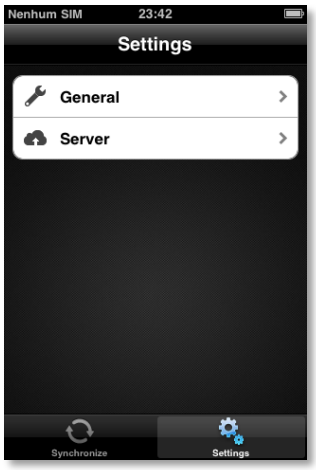


Figura 52. Secção de definições na aplicação iPhone

Como apresentado na Figura 52, A secção de definições disponibiliza um menu no qual é possível escolher o tipo de configurações que se pretende definir. Na secção definições gerais pode ser configurado o tipo de sincronização a efectuar assim como os alertas de rede a serem apresentados ao utilizador. Na secção definições de configuração o utilizador pode proceder à configuração dos dados para autenticação no servidor.

Definições gerais

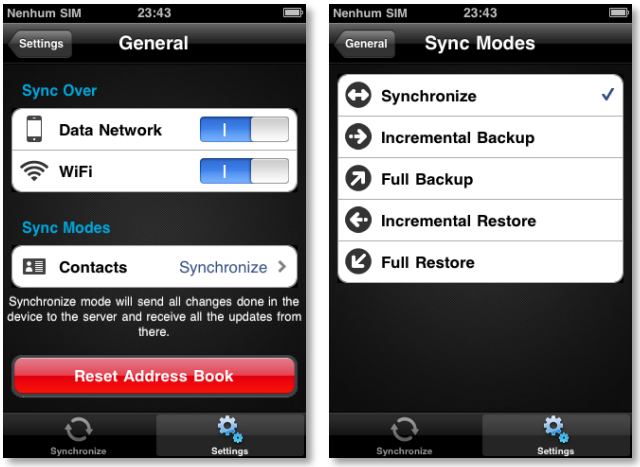


Figura 53. Secção de definições gerais e menu para escolha do tipo de sincronização

Esta secção disponibiliza ao utilizador a opção de configurar alertas de tipo de rede assim como a definição do tipo de sincronização a executar (Figura 53). Os alertas de tipo de rede são importantes para utilizadores que tenham planos de tráfego limitados e que apenas desejam utilizar a aplicação sobre Wifi. Nesta secção pode também ser executada uma limpeza à lista de contactos, opção que o sistema operativo não disponibiliza nativamente.

No caso da aplicação estar a ser utilizada num iPod, a opção de alertas não é apresentada, pois ao apenas ser suportado Wifi, não faz sentido estar a definir alertas de uso de rede.

Definições de servidor

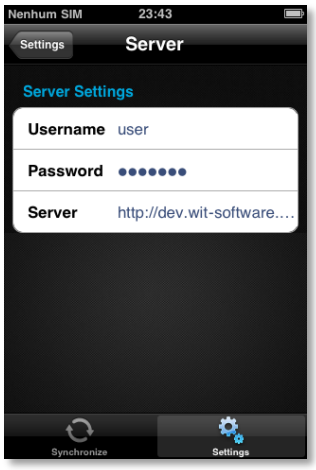


Figura 54. Secção de definições de servidor na aplicação iPhone

Esta secção disponibiliza ao utilizador a possibilidade de configurar o dados de autenticação para o servidor de sincronização a utilizar (Figura 54).

4.3.4 Aplicação iPad

A aplicação iPad foi implementada recorrendo ao projecto de iPhone, utilizando para este fim a funcionalidade Universal Application da plataforma iOS, que permite que um mesmo projecto seja utilizado para a implementação de aplicações para iPhone, iPod e iPad.

No entanto, para ter dois tipos de interfaces diferentes no mesmo projecto é necessário preparar o código para tal. Assim, após a aplicação de iPhone / iPod ter sido desenvolvida, foram adicionadas as interfaces para iPad ao projecto e feitas as alterações necessárias no código de modo a que ambas as interfaces conseguissem conviver num mesmo projecto sem que uma delas falha-se. Isto implica a validação de comportamentos específicos das versões das interfaces, como por exemplo o caso da rotação do ecrã que apenas foi activada na versão iPad, ou a existência de componentes, como é o caso do SplitView, um componente de UI que apenas está disponível no iPad.

Requisitos Mínimos

Sistema Operativo

Este cliente tem como requisito mínimo para a versão do sistema operativo a versão 3.2 por ser esta a primeira versão do iOS para iPad.

Secções da aplicação

Sincronização

Figura 55. Secção de sincronização na aplicação iPad, versão vertical

Figura 56. Secção de sincronização na aplicação iPad, versão horizontal

À semelhança da secção correspondente na versão iPhone, a secção de sincronização (Figuras 55 e 56) disponibiliza um relatório da ultima sincronização bem sucedida, assim como informação de progresso durante uma sincronização em curso. A navegação entre secções nesta secção é também semelhante à navegação na aplicação iPhone, a qual é feita através dos botões no fundo do ecrã.

Definições

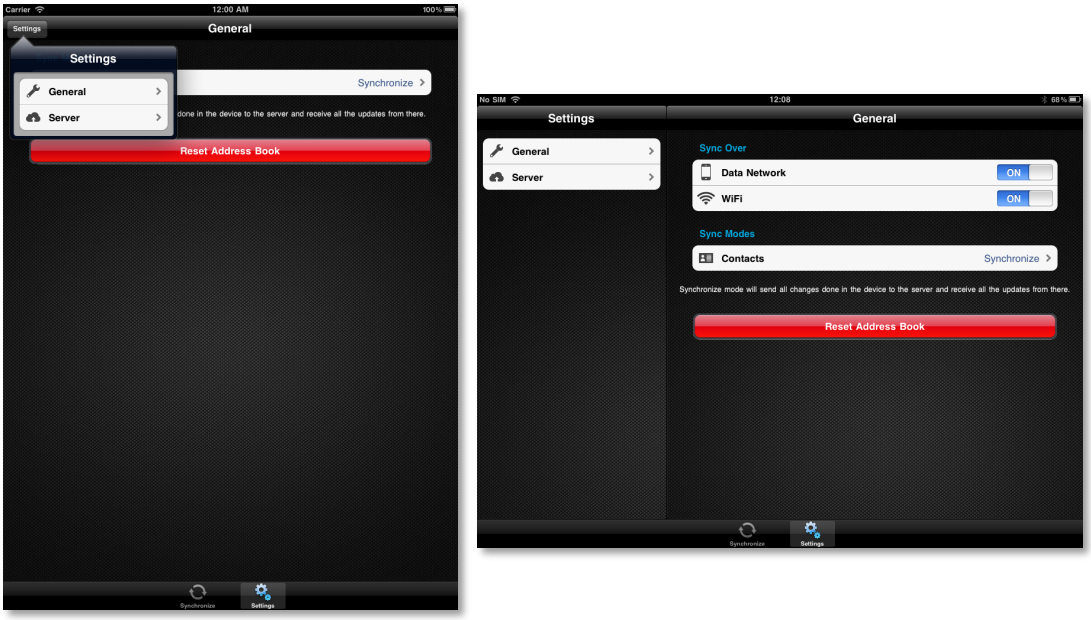


Figura 57. Definições na aplicação iPad

Para o cliente iPad, devido à maior área de ecrã disponível, não foi criada uma secção especial para o menu de definições, tendo-se optado por implementar as definições num estilo idêntico ao das interfaces nativas de iPad, ou seja, na vista horizontal usar aproximadamente um terço da área à esquerda do ecrã para o menu enquanto que o restante espaço disponibiliza a sub-secção (Figura 57). Para a vista vertical, o menu está oculto por pré-definição e é apresentado ao ser pressionado o botão definições que está presente na barra de topo desta vista.

Definições gerais

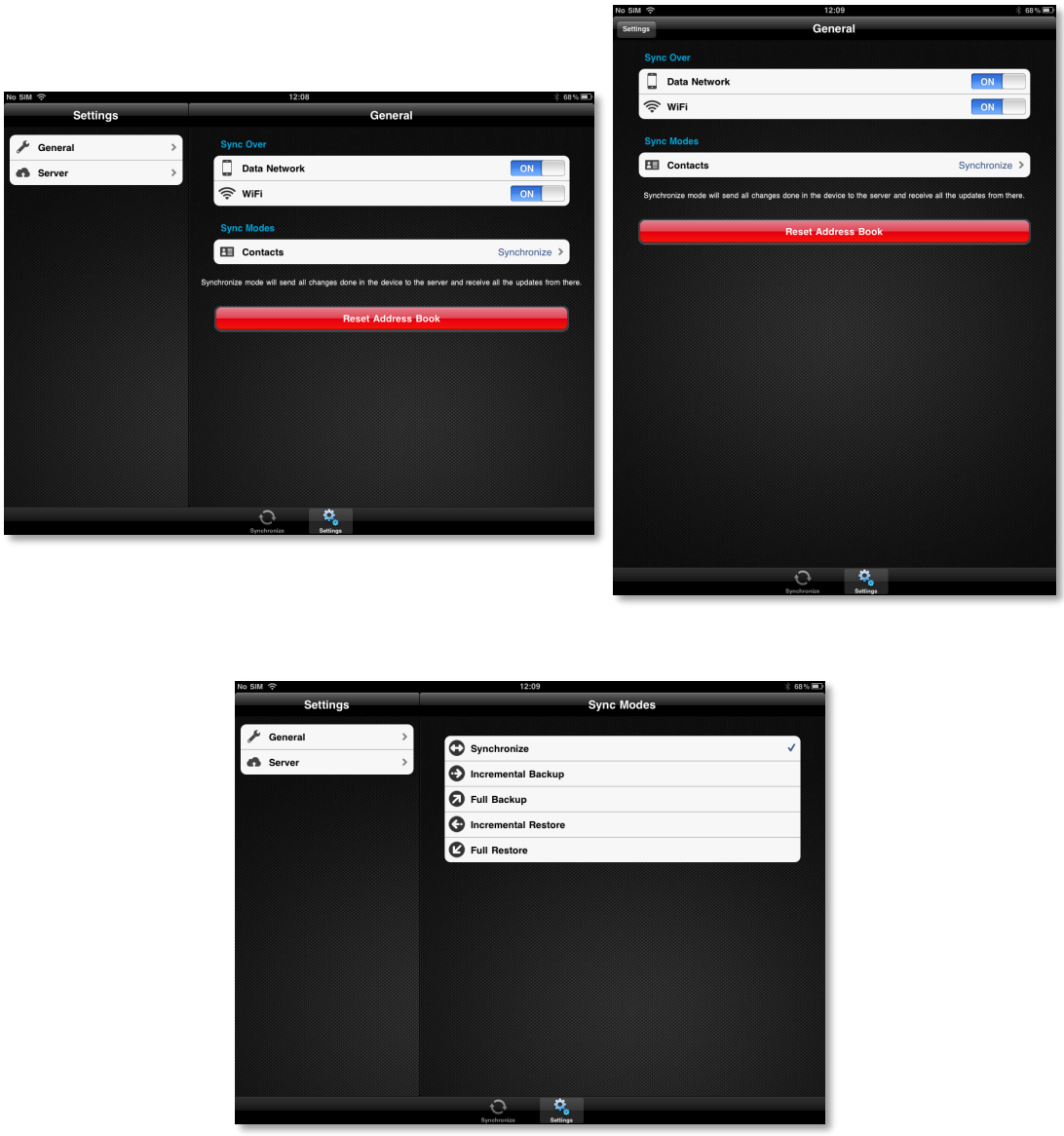


Figura 58. Secção de definições gerais na aplicação iPad

A secção de definições gerais permite ao utilizador, á semelhança da aplicação para iPhone, escolher o tipo de sincronização a utilizar, os alertas de rede e executar uma limpeza à lista de contactos (Figura 58).

Também para iPad, devido ao facto de haver versões da plataforma sem tráfego de dados, o menu de alertas apenas é apresentado nas versões que suportam redes Wifi e de dados.

Definições de servidor

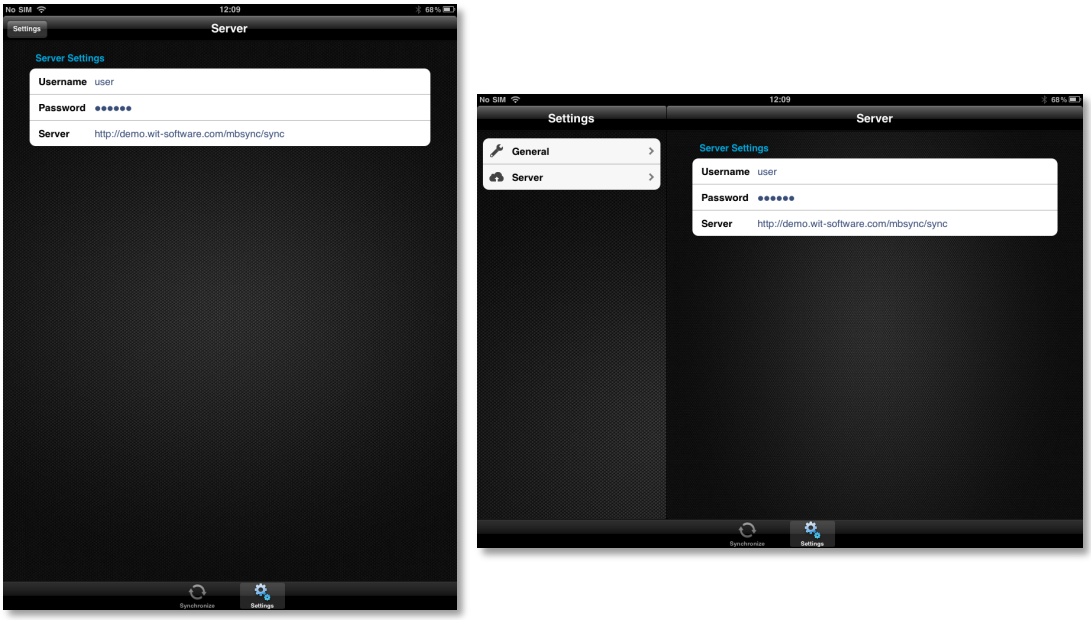


Figura 59. Secção de definições de servidor na aplicação iPad

Novamente, a secção de definições de servidor possibilita ao utilizador a configuração dos dados de acesso ao servidor de sincronização (Figura 59). Também neste cliente é apresentada por pré-definição em caso de pelo menos um dos campos não estar preenchido. Caso contrário, a secção de sincronização é apresentada.

4.4 Serviço de agendamento de sincronizações (Android)

O serviço de monitorização de eventos e agendamento de sincronizações desenvolvido para o cliente Android partilha com a aplicação de sincronização o facto de suportar a adição de novos tipos de dados na sua arquitectura. Esta escalabilidade do serviço é devida também à implementação de módulos que se registam perante o SO para estarem à escuta de diversos eventos do sistema que lhes sejam úteis. Deste modo, ao adicionar-se um novo tipo de dados à aplicação, facilmente se adiciona um novo módulo a este serviço, tornando este

capaz de processar novos eventos que irão agendar a sincronização para o novo tipo de dados.

O serviço é executado automaticamente no arranque do sistema, e ao validar se a utilização de sincronizações automáticas está activa, este regista os listeners para cada módulo, caso esteja activa, ou , em caso contrário, deixa-se terminar. Sempre que a opção de sincronização automática é activada na aplicação, o serviço é arrancado.

O módulo implementado, que monitoriza alterações a contactos, regista-se para escuta de alterações à base de dados de contactos do Android. Sempre que uma alteração é detectada na base de dados, este agenda uma sincronização de modo a que seja executada passados trinta segundos, intervalo de tempo que serve de margem para que o utilizador possa efectuar novas alterações. Se o utilizador voltar a alterar um contacto dentro destes trinta segundos, o intervalo de tempo até ao inicio da sincronização é reiniciado. No caso do serviço não conseguir executar a sincronização cinco vezes seguidas, esta deixa de ser reagendada e a próxima tentativa apenas será feita a quando da próxima sincronização agendada pela aplicação.

Este serviço tem também em conta os requisitos disponíveis do dispositivo no momento do inicio da sincronização, o que leva a que se o dispositivo estiver a ser usado exaustivamente no momento em que uma sincronização está prestes a começar, esta irá ser reagendada para evitar causar algum atraso ao uso normal do dispositivo.

A monitorização é feita ao nível da bateria e da utilização de processador e da memória. Se a bateria estiver a menos de 15% da sua carga total, o processador a mais de 50% de uso ou a memória tiver menos de 20MB livres, a sincronização não é executada e é reagendada para mais tarde.

4.5 Servidor de sincronização

O servidor de sincronização que foi sujeito a alterações neste trabalho foi implementado em Java SE e corre sobre o servidor TomCat. As alterações que foram feitas baseiam-se na necessidade que houve de suportar a estrutura em árvore dos contactos de Android, a qual é especifica desta plataforma.

4.5.1 vCard

Para suportar esta estrutura, as alterações tiveram de começar no formato da mensagem partilhada entre o cliente e o servidor através de SyncML, o vCard[13]. Este teve que sofrer algumas alterações ao seu formato original, tendo sido criadas algumas extensões para o vCard, para que fosse possível a partilha desta informação adicional.

```
BEGIN:VCARD
VERSION:2.1
N:Miguel;Jorge;;;
X-N-RAW_ID:1
FN:Jorge Miguel
X-FN-RAW_ID:1
X-N;X-RAW_ID=2:Miguel;Jorge;Filipe;;
X-FN;X-RAW_ID=2:Jorge Filipe Miguel
TEL;CELL;X-RAW_ID=1:918765432
TEL;WORK;X-RAW_ID=1:912345678
TEL;WORK;X-RAW_ID=2:912345678
TEL;FAX;WORK;X-RAW_ID=1: 938765431
EMAIL;X-RAW_ID=1:email@normal.pt
EMAIL;X-RAW_ID=2:email@home.pt
EMAIL;X-RAW_ID=3:email@work.pt
URL;X-RAW_ID=1;ENCODING=QUOTED-PRINTABLE;CHARSET=UTF-8:http=3A//web
ORG;ENCODING=QUOTED-PRINTABLE;CHARSET=UTF-8:W=E2=82=ACt
X-ORG-RAW_ID:1
X-PHOTO-RAW_ID:1
PHOTO;ENCODING=BASE64;CHARSET=UTF-8;TYPE=JPEG:
/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAYEBQYFBAYGBQYHBwYIChAKCgkJChQODwwQFxQYGBc
UFhYaHSUfGhsjHBYawgIyYnKSopGR8tMC0oMCUoKSj/2wBDAQcHBwoIChMKChMoGhYaKCgoKC
goKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCj/wAARCAA2AEgDA
SIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUF
==

X-PHOTO;X-RAW_ID=2;ENCODING=BASE64;CHARSET=UTF-8;TYPE=JPEG:
KjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/
8QAHwEAAwEBAQEBAQEBAQAAAAAawQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAEC
AxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkN
ER==

END:VCARD
```

Figura 60. vCard criado pelo cliente Android

No exemplo da Figura 60, é notória a presença de extensões no vCard, as quais estão identificadas com o formato 'X-'. O formato adoptado para este conjunto de extensões foi a adição de uma propriedade extra ao vCard por cada propriedade nativa, com o objectivo de identificar a origem da informação. Um exemplo deste formato pode ser visto na Figura 61, na qual se pode ver que para a propriedade N:, original do vCard, foi adicionada uma propriedade X-N-RAW_ID, a qual identifica a origem do valor de N:

[13] http://tools.ietf.org/html/rfc2426

```
N:Miguel;Jorge;;;
X-N-RAW_ID:1
FN:Jorge Miguel
X-FN-RAW_ID:1
```

Figura 61. Exemplo de extensão para vCard

O formato descrito a cima garante que os mapeamentos são mantidos, mas, por haverem propriedades únicas no vCard, como é o caso do N: e do FN:, este não garante que não seja perdida informação. Para contornar esta limitação do vCard, outras extensões foram criadas, de modo a conseguir garantir a persistência dos dados repetidos.

```
N:Miguel;Jorge;;;
X-N-RAW_ID:1
X-N;X-RAW_ID=2:Miguel;Jorge;Filipe;;
```

Figura 62. Exemplo de extensão para vCard

No exemplo presente na Figura 62, pode-se encontrar uma propriedade N:, nativa do vCard, que se encontra emparelhada com uma extensão criada, o X-N-RAW_ID, para garantir a identificação da origem do N:. A novidade neste exemplo é a extensão X-N: que não só identifica a origem de uma propriedade, através do parâmetro X-RAW_ID, como também contêm o valor da propriedade.

Devido ao facto das extensões não serem processadas pelos parsers de vCard, sendo apenas mantidas pelos servidores para que possam ser devolvidas ao dispositivo originador, a popriedade X-N: não só não se sobrepõe à N:, como também é mantida pelo servidor e só é sincronizada com o dispositivo que a criou.

Com estes dois tipos de extensões ao vCard foi possível gerir toda a informação necessária à sincronização de contactos de Android, mantendo a sua estrutura original.

4.5.2 Servidor

Devido a estas alterações no formato do vCard, não só os parsers do cliente Android tiveram que ser modificados para suportarem esta nova formatação, como também os parsers do servidor tiveram que os suportar. Para isto, fez-se uma versão especial dos parsers do servidor, que mediante identificação do dispositivo como um dispositivo Android, é executado de modo a ser possível recolher esta informação.

Devido ao facto de haver nova informação, houve outras alterações que tiveram de ser implementadas no servidor, nomeadamente a nível de estrutura de dados, que tiveram que ser alteradas de modo a reflectir os novos campos de id, e a nível de base de dados, a qual teve também receber novas tabelas e colunas em algumas tabelas existentes para que esta informação possa ser persistida.

4.6 Testes

Os testes realizados a este trabalho foram executados pela equipa de testes da WIT-Software, a qual seguiu uma planificação criada para os testes. Estes testes foram executados sobre as várias aplicações desenvolvidas neste trabalho e os bugs encontrados durante foram reportados na plataforma de tracking de bugs utilizada na empresa.

A planificação dos testes foi desenvolvida pelo orientador deste trabalho e tiveram foco sobre as funcionalidades de sincronização, comportamentos das interfaces, erros de conexão ao servidor, falha de rede, entre outros. As aplicações foram também dadas a testar a algumas pessoas da empresa, as quais indicaram as suas dificuldades de interacção e os problemas que encontraram. Todos os erros reportados e muitas das sugestões dadas foram levadas em conta e corrigidas ou alteradas.

5 Conclusão

Os clientes de sincronização, como vimos ao longo deste trabalho, sempre tiveram um papel extremamente importante no dia-a dia da informática. Desde cedo que a sincronização tem sido utilizada como meio de salvaguarda de dados, estando este cenário a metamorfarse aos poucos e poucos à medida que a sincronização entra cada vez mais e mais na vida de cada um de nós, criando um cenário em que o objectivo principal passou da salvaguarda de dados para a centralização destes, garantido assim o fácil e rápido acesso a dados em qualquer parte e através de qualquer dispositivo.

Esta alteração de paradigma tem sido motivada pelo aparecimento de novos dispositivos móveis, capazes de comunicar e transportar dados como nunca outros o conseguiram antes, o que leva a que o público cada vez mais se mostre interessado por soluções na nuvem.

Para que isto fosse possível, foi efecutuada primeiramente uma analise ao problema proposto, analise que resoltou na organização e estruturação de uma arquitectura para implementação das aplicações que supera diversos desafios, como a reutilização, flexibilidade e suporte a multi-plataformas.

Com base nesta arquitectura, foram desenvolvidas três aplicações, que apesar das diferenças forçadas pelas limitações de cada plataforma para as quais foram desenvolvidas, têm um modo de funcionamento e utilização tranversal, e que se pode dizer que são equivalentes entre si.

Com este trabalho, mostrou-se que é possível criar aplicações extremamente versáteis, capazes de correr em diversas plataformas e dispositivos, e que conseguem a cima de tudo partilhar informação entre si.

Como trabalho futuro, novos módulos de sincronização poderam ser adicionados às aplicações de modo a tornar possível o suporte a um maior número de tipos de dados.

Também as bibliotecas criadas para este projecto poderam vir a servir de base para novos clientes de sincronização para estas ou outras plataformas.

6 Bibliografia

- Uwe Hansmann,Riku M. Mettala,Apratim Purakayastha,Peter Thompson (2003) SyncML: synchronizing and managing your mobile data. Pearson Education Inc.
- http://en.wikipedia.org/wiki/VCard (29 de Setembro)
- http://en.wikipedia.org/wiki/SyncML (29 de Setembro)