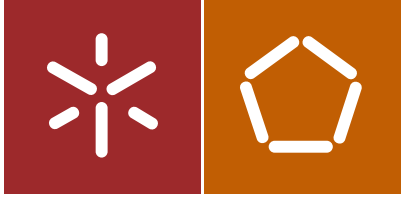
Universidade do Minho
Escola de Engenharia

Rui Filipe Tavares Nogueira

Aplicação Android para configuração e acesso direto a câmaras ONVIF

Aplicação Android para configuração e acesso direto a câmaras ONVIF
Rui Filipe Tavares Nogueira
UMinho | 2014

dezembro de 2014



Universidade do Minho
Escola de Engenharia

Rui Filipe Tavares Nogueira

Aplicação Android para configuração e acesso direto a câmaras ONVIF

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao
Grau de Mestre em Engenharia de Comunicações

Trabalho efetuado sob a orientação do
Professor Doutor Sérgio Adriano Fernandes Lopes

dezembro de 2014

DECLARAÇÃO

Nome: Rui Filipe Tavares Nogueira

Endereço eletrónico: a58669@alunos.uminho.pt

Telemóvel: 915309033

Número do Bilhete de Identidade: 14003605

Título da dissertação

Aplicação Android para configuração e acesso direto a câmaras ONVIF

Orientador:

Professor Doutor Sérgio Adriano Fernandes Lopes

Ano de conclusão: 2014

Designação do Mestrado:

Ciclo de Estudos Conducentes ao Grau de Mestre em Engenharia de Comunicações

Universidade do Minho, ___/___/_____

Assinatura: _____

Agradecimentos

Chegado ao fim de este longo percurso, gostaria de aproveitar a oportunidade para agradecer a todas as pessoas que contribuíram, para o meu sucesso e desenvolvimento pessoal.

Gostaria de agradecer ao Professor Sérgio Lopes pela orientação e total disponibilidade durante a realização deste trabalho.

A todos os amigos que me acompanharam durante estes cinco anos e que me marcaram de uma forma ou de outra.

Por último, tendo consciência que nada disto seria possível de realizar sozinho, um agradecimento especial aos meus pais, Celina e Fernando Nogueira, pelo apoio incondicional que me deram, pela amizade e paciência demonstrada ao longo de toda a minha vida. Ao José, ao Nuno e à Telma por toda a ajuda e conselhos oferecidos durante todo este tempo. À Rita por todos os momentos partilhados, pelo apoio e carinho oferecidos.

O meu muito obrigado!

Resumo

O ONVIF é uma norma para comunicação entre dispositivos multimédia IP, incluindo câmaras de vídeo, que tem amadurecido e são cada vez mais os dispositivos que o suportam [5]. Nos dispositivos móveis, o sistema operativo *Android* lidera o mercado e por isso são os melhores candidatos para integrarem funcionalidades de configuração e receção de *streaming* de câmaras de vídeo ONVIF.

No entanto, o ONVIF é baseado em serviços web SOAP e por isso computacionalmente pesado. A implementação de serviços SOAP é normalmente feita recorrendo a *toolkits* para geração de código cliente e servidor, no entanto, os *toolkits* com maturidade são para as versões *Standard* e *Enterprise* de Java. Atualmente, existem apenas 2 aplicações cliente ONVIF no Google Play.

Esta dissertação estuda os *toolkits* SOAP disponíveis para *Android*, e conclui que nenhum deles é capaz de gerar código correto e completo para serviços ONVIF. A melhor alternativa encontrada consiste em utilizar apenas uma biblioteca que facilita a troca de mensagens SOAP. São comparadas duas alternativas de implementação comunicação com as câmaras, uma utilizando o SDK [6] e outra utilizando o NDK [7] do *Android*. Foi clara a vantagem em termos de desempenho computacional da implementação NDK, contabilizando já o custo dos cruzamentos da camada JNI [8].

Os resultados dos estudos realizados conduziram a uma solução de arquitetura da aplicação em que a comunicação é implementada em C e a restante parte da aplicação em Java. A comunicação com as câmaras ONVIF envolve várias dezenas de operações, para as quais existia já uma biblioteca C, não sendo por isso necessário gerar e utilizar *stubs* C. Foi no entanto preciso desenvolver a camada JNI para todas essas funções. Essa tarefa foi realizada utilizando a ferramenta SWIG [9].

O desenvolvimento da parte Java da aplicação consistiu no desenho de uma estrutura adequada às funcionalidades ONVIF, que por um lado fosse intuitiva, alinhando com a estrutura da própria norma, e de utilização fácil. Na implementação foram incorporadas soluções que minimizam o número de pedidos feitos às câmaras, tonando a aplicação mais eficiente.

Este trabalho produz assim vários resultados importantes para o estado da arte da implementação de serviços SOAP em dispositivos *Android*, e uma aplicação eficiente que permite a configuração de câmaras ONVIF e a capacidade de monitorização remota de espaços.

Abstract

ONVIF is a standard for communication with multimedia IP devices, including video cameras, which is becoming more widely supported in the market [5]. Android is the leader OS in mobile devices, which are then the best candidate to integrate functionalities of ONVIF camera configuration and stream play.

However, ONVIF is based on SOAP web services, and therefore it is computationally heavy. The development of SOAP applications usually involves the use of a toolkit to generate client and server code, but the ones currently mature target standard and enterprise Java editions. Presently, there are only two ONVIF client applications in Google Play.

This work analyses the available SOAP toolkits targeting Android, and concludes that none of them is able to generate correct and complete code for ONVIF services. The best option that was found is the use of a library that helps to exchange SOAP messages. Two alternatives are compared for implementing communication with cameras, one using Android's SDK and another using the NDK. There is a clear advantage of the NDK implementation in terms of computing performance, including already the cost of calls across the JNI layer.

The results, of the research work that was conducted, led to an architectural where the communication with cameras is implemented with the NDK and the rest of the application is developed with the SDK. ONVIF communication involves several dozens of operations for which there is already a C library, and therefore it was not needed to generate C stubs. Still, it was necessary to develop the JNI layer for all those functions, task which was completed using the SWIG tool.

The development of the application's Java part consisted in the design and implementation of a structure suitable for the ONVIF functionalities. On one hand, it should be intuitive, by following the standard's structure, and on the other hand it should be user friendly. The implementation includes solutions to minimize the number of request that are sent to cameras, thus making the application more efficient.

This work produces several results that are important to the state of the art on SOAP web services implementation in Android, and an efficient application that enable the configuration of ONVIF cameras and the remote monitoring of places.

Índice de conteúdos

Lista de figuras

1. Introdução

1.1 Enquadramento

Os constantes desenvolvimentos na produção e utilização de câmaras de vídeo IP fez surgir um aumento na diversidade das especificidades e protocolos de configuração criados pelos fabricantes das mesmas. Devido a este aumento, a necessidade de normalizar o modo de comunicação entre estes dispositivos é cada vez maior. Foi com este objetivo que o *Open Network Video Interface* (ONVIF) [5] foi criado. O ONVIF é uma organização aberta à indústria com o objetivo de proporcionar uma norma para a comunicação entre dispositivos IP multimédia e garantir a interoperabilidade entre dispositivos independentemente do seu fabricante.

O *Android* é atualmente o sistema operativo mais utilizado no mundo em *smartphones* e *tablets* e o número de dispositivos com capacidades de processamento adequadas a conteúdos multimédia é cada vez maior. Além disso, este sistema operativo permite um nível de comodidade que o torna muito atrativo para a criação de aplicações que lidam com câmaras de vídeo.

Para o desenvolvimento de aplicações, a plataforma *Android* oferece duas arquiteturas base: utilizar o SDK e a linguagem de programação Java ou utilizar o NDK e a linguagem de programação C/C++. A utilização simultânea das duas é também possível utilizando o JNI, um *framework* que permite a interação entre a linguagem Java e código nativo como o C.

Atualmente existem diversas aplicações *Android* que permitem o acesso e controlo a câmaras de vídeo IP sendo estas, na maioria dos casos, dependentes do modelo da câmara para a qual foram construídas. Ou seja, estas aplicações apenas lidam com as especificidades de cada câmara, modelo ou fabricante, sendo necessária a implementação de funcionalidades para o suporte de outro tipo de câmaras.

O ONVIF é baseado em *web services* SOAP [23] que definem formas normalizadas de descrever protocolos de rede, formatos de transmissão de dados e operações a realizar. Uma vez que são descritos utilizando ficheiros *Web Service Definition Languague* (WSDL) [3] [4], os serviços definidos nestes *web services* podem ser gerados recorrendo a *toolkits* de geração de código de *stubs* de clientes e de esqueletos do servidor. Através da utilização de *toolkits* é evitada a escrita manual de código dos processos de serialização e de-serialização do XML das mensagens SOAP, código esse que para além de ser repetitivo, para os vários casos definidos pelo *web service,* é favorável à ocorrência de erros de programação.

1.2 Motivação

Uma vez que o ONVIF é baseado em normas de *web services* SOAP, este é bastante pesado em termos computacionais, cujas consequências são mais graves em dispositivos móveis devido às suas limitações em termos de capacidade computacional e energia. A necessidade da serialização e de-serialização de dados XML para efetuar a comunicação com as câmaras é um dos processos mais críticos em termos de processamento.

A grande vantagem da utilização do NDK na programação *Android* é o facto deste, como utiliza código nativo, ter uma melhor performance no tratamento de operações computacionalmente pesadas.

Existe já uma biblioteca C para comunicação com câmaras ONVIF, que constitui então uma oportunidade para implementação da comunicação utilizando o NDK. No entanto, as invocações através da fronteira JNI também têm um custo em termos de processamento. Por isso não é evidente qual das opções é a melhor.

Por outro lado, os *toolkits* SOAP maduros (e.g, *Axis2* [10]) para Java geram *stubs* compatíveis para as *frameworks Java2SE* [11] e *Java2EE* e portanto não são compatíveis com a API *Android*. Existem alguns *toolkits* para *Android* cuja maturidade é necessária aferir pois são individuais com curto tempo de vida ou mal documentados.

1.3 Objetivos

O objetivo geral desta dissertação é o estudo e exploração de soluções que tornem a utilização de câmaras IP menos pesadas em termos computacionais e a criação de uma aplicação *Android* que proporcione uma simples configuração de acesso a câmaras *ONVIF*, demonstrando com esta aplicação as funcionalidades da norma e capacidade de monitorização remota de espaços.

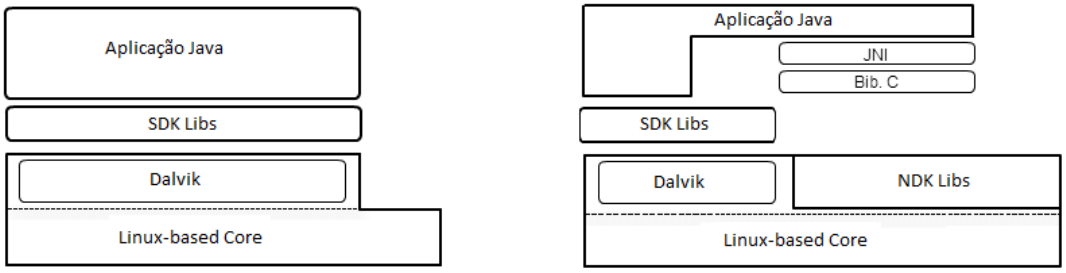


Figura 1 - Ilustração das duas opções de implementação

Os objetivos específicos são:

- Aferir a maturidade das bibliotecas e *toolkits* que suportam *web services SOAP* em *Android* e a sua viabilidade para desenvolvimento de clientes.

- Comparar o desempenho de uma aplicação desenvolvida recorrendo exclusivamente ao *Android SDK* e outra que utilize o *NDK* para fazer o processamento dos dados relativos à comunicação *ONVIF*.

- Desenvolvimento de uma aplicação de configuração e acesso aos dados multimédia de câmaras que não dependa das especificidades das câmaras e que seja computacionalmente leve, sendo o mais otimizada possível ao nível da comunicação com as câmaras.

1.4 Estrutura da dissertação

No capítulo 1 é realizada uma introdução ao tema da dissertação. Neste capítulo é descrito o enquadramento do trabalho assim como os seus objetivos. Este capítulo é finalizado com a apresentação da estrutura desta dissertação.

No capítulo 2 (Fundamentos) é realizada uma introdução ao sistema operativo *Android* assim como uma breve descrição sobre funcionalidades e conceitos que foram importantes na realização deste trabalho.

No capítulo 2 foi também realizada uma descrição sobre os conjuntos de ferramentas de desenvolvimento de *software* utilizados, nomeadamente o *Software Development Kit* (SDK) e o *Native Development Kit* (NDK), assim como o *Simplefied Wrapper and Interface Generator* (SWIG).

Para finalizar é também realizada uma análise introdutória ao *Open Network Video Interface* (ONVIF).

No capítulo 3 (Estado da arte), é realizada uma análise de *toolkits* para geração de código Java compatível com o *Android* assim como de bibliotecas que suportam serviços SOAP.

No capítulo 4 (Desenho) é apresentado um estudo comparativo entre as alternativas para o desenvolvimento da aplicação final obtidas com o estudo dos *toolkits* e de bibliotecas realizadas no capítulo 3.

O capítulo 5 (Desenvolvimento) retrata o processo de construção da aplicação final. Neste capítulo é descrito o processo de escolha da estrutura da aplicação assim como as dificuldades enfrentadas e as soluções adotadas para as mesmas.

No capítulo 6 (Resultados e discussão) são apresentados os resultados. Neste capítulo é realizada uma descrição do resultado final da aplicação. Para além da descrição da interface da aplicação, neste capítulo é realizada uma discussão sobre os restantes objetivos deste trabalho, sendo realizada uma breve análise sobre, a maturidade dos *toolkits* e bibliotecas estudadas, assim como das aplicações de teste desenvolvidas.

As conclusões da realização deste trabalho assim como a discussão sobre trabalho futuro a desenvolver são, por fim, apresentadas no capítulo 7.

2. Fundamentos

2.1 Programação Android

O *Android* é um sistema operativo *open-source* para dipositivos móveis desenvolvido pela Google sendo, atualmente, o mais utilizado no mundo.

O facto de o *Android* ser open-*source* e a sua utilização ser bastante popular motivou o constante crescimento de comunidades de programadores que partilham conhecimentos sobre o desenvolvimento de aplicações neste sistema operativo [12]. Juntando a este fator a existência de uma boa documentação sobre a API [13] e funcionalidades, a introdução e obtenção de conhecimentos necessários à criação e desenvolvimento de aplicações *Android* é bastante facilitada.

Para desenvolver o trabalho desta dissertação foi necessário estudar o Sistema Operativo *Android* e as suas funcionalidades. Os tópicos de maior importância são:

- *Activity*;
- *Fragment*;
- *User Interface Layout*;
- *Application Manifest*;

2.1.1 Activity

Activity [15] é um componente básico da programação *Android* que permite fornecer ao utilizador uma interface de interação com a aplicação.

No arranque de uma aplicação, é sempre iniciada pelo menos uma *Activity*. Esta funciona como um ponto de referência na aplicação podendo, por sua vez, iniciar outras *Activities*. A

transição para outra *Activity* irá corresponder a uma modificação na *User Interface* da aplicação uma vez que cada Activity tem apenas um *layout* de interface associado.

A transição entre *Activities* é realizada recorrendo a um *Intent*. Um *Intent* é uma descrição abstrata de uma operação a ser realizada. No caso da inicialização de uma nova *Activity*, a informação necessária à operação é guardada num *Intent*, sendo este posteriormente utilizado como argumento do método *startActivity* que desencadeia o processo de transição. É também através da utilização de *Intents* que se realiza a passagem de informação de dados entre a *Activity* atual e a seguinte.

Quando uma *Activity* é iniciada, a anterior é parada temporariamente, sendo preservada numa *stack* ("*back stack*"). Uma vez que esta *stack* funciona segundo o princípio de "*last in first out*", quando o "botão" de retorno é acionado, a *Activity* atual é substituída pela anterior e destruída. Por isso, é impossível a existência de duas *Activities* em funcionamento simultâneo. No entanto, o ciclo de vida de uma *Activity* é composto por diversas fases, sendo possível programar alterações na aplicação conforme a fase em se encontra. Os métodos da classe Activity que definem o comportamento de cada fase são os seguintes:

- *onCreate* – executado durante a criação;
- *onStart* – executado quando a *Activity* está prestes a ser colocada no ecrã;
- *onPause* – executado durante a transição da *User Interface* para outra *Activity*;
- *onStop* – executado após a transição da *Activity* para fora da User Interface;
- *onResume* – executado com o retorno da *Activity* para a *User Interface*;
- *onRestart* – executado caso o utilizador volte a iniciar a *Activity*;
- *onDelete* - executado na destruição da *Activity*;

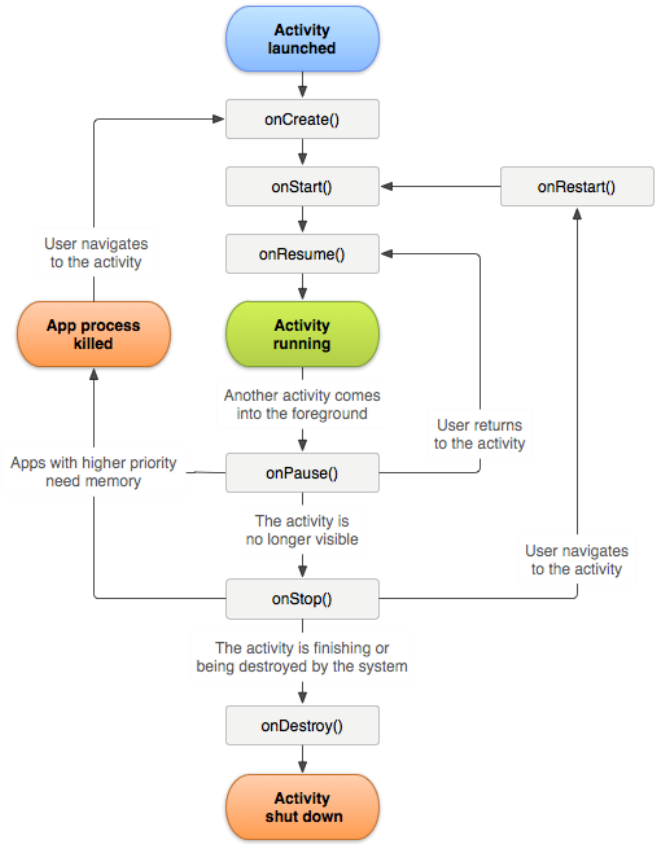


Figura 2 – Ciclo de vida de uma Activity [14]

A figura 2 apresenta o ciclo de vida completo de uma *Activity* com as fases descritas anteriormente.

2.1.2 Fragment

Um *Fragment* [15] é um componente do *Android* normalmente utilizado na gestão de uma parte da *User Interface*. Os *Fragments* são iniciados e geridos por *Activities* sendo que, cada *Activity* pode ser responsável pela utilização de mais do que um *Fragment*. Uma vez que os *Fragments* podem ser utilizados paralelamente entre si, é permitida a divisão dos componentes da *User* Interface em diferentes *Fragments*. Com isto é possível manipular individualmente os *layouts* da interface, por exemplo trocar um por outro, tornando a aplicação mais flexível. Assim, os *Fragments* permitem reutilizar partes da interface que se mantêm de uma forma mais simples e otimizadas, uma vez que apenas necessitam de ser inicializadas uma única vez e não sempre que se realiza uma modificação do resto do *layout* da Interface.

Tal como uma *Activity*, um *Fragment* possui o seu próprio ciclo de vida, sendo este dependente do ciclo de vida da *Activity* pela qual foi iniciado. Tal como acontece na utilização de *Activities*, é possível efetuar operações durante as transições entre os estados de um *Fragment*.

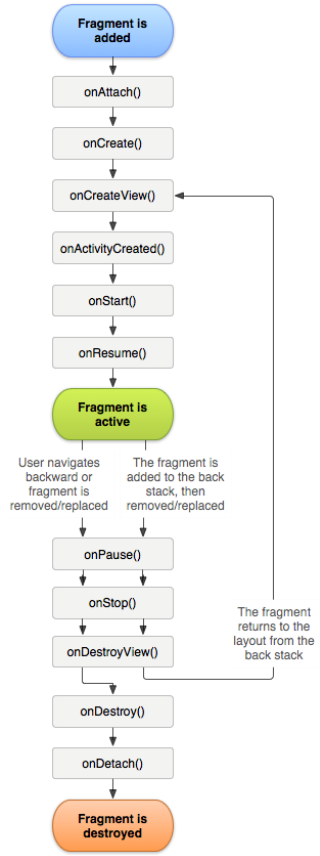


Figura 3 - Ciclo de vida de um Fragment [15]

Tal como representado na figura 3, para além dos que tem em comum para com o ciclo de vida de uma *Activity*, um *Fragment* possui os seguintes métodos:

- *onAttach* – Inicializado ao adicionar um *Fragment* à *Activity*;
- *onActivityCreated* – Inicializado após o retorno do método *onCreate* da *Activity*;
- *onCreateView* - Inicializado na criação da *user interface* associada ao *Fragment*;
- *onDestroyView* – Inicializado na destruição da *user* interface associada ao *Fragment*;

- *onDetach* – Inicializado durante a remoção do *Fragment* da *Activity*.

2.1.3 User Interface Layout

Tal como descrito anteriormente, a *User* Interface é construída utilizando *Activities* e *Fragments*. É nestes componentes que os ficheiros de *layout* [16] (com as declarações dos elementos da interface) são carregados.

Apesar de ser possível a criação de *layouts* para as aplicações através do código Java, o método recomendado é que estes sejam definidos em ficheiros XML e que sejam posteriormente invocados pelas *Activities*/ *Fragments*.

Os elementos de *user interface* das aplicações são criados recorrendo a dois componentes:

- *View*;
- *ViewGroup*.

Uma *View* é um objeto responsável por partes da interface em contato com o utilizador. Um *ViewGroup* é responsável por um conjunto de *Views* ou de outros *ViewGroups, conforme ilustrado na figura 4,* que, juntos, formam o *layout* da *User Interface.*

É através de subclasses destes objetos que é possível a utilização de diferentes tipos de *layout* como Linear ou *Relative Layouts*. É também com a utilização destes objetos que se realiza o preenchimento destes tipos de *layouts* com a utilização de componentes como caixas de texto, botões, imagens, etc.

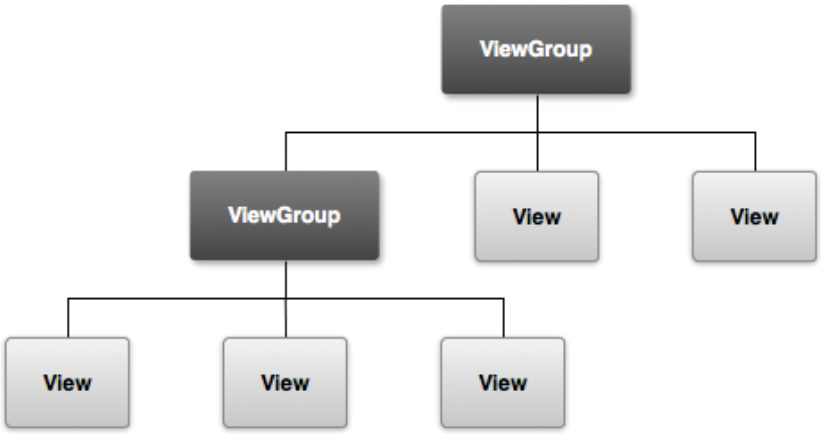


Figura 4 - Exemplo da hierarquia de *views* de uma aplicação [16]

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

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="166dp"
        android:text="@string/button" />

</RelativeLayout>
```

Figura 5 - Exemplo de definição de um *XML*

Na figura 5 é demonstrado um exemplo de um ficheiro XML na qual está implementado um botão. Este *layout* deverá depois ser carregado, no método *onCreate* de uma *Activity*, ou no método *onCreateView* de um *Fragment*, fazendo com que apareça na *User Interface* o botão criado.

2.1.4 Manifest file

O *Manifest file* [17] é um ficheiro obrigatório em todas as aplicações *Android* que guarda informação essencial ao funcionamento de uma aplicação. Algumas das mais importantes funções deste ficheiro são definir:

- *Activities* – É necessário definir de todas as *Activities* utilizadas na aplicação;
- Permissões de utilização – Para o seu funcionamento, algumas aplicações necessitam de permissões específicas. Um exemplo é a permissão *android.permission.INTERNET* que permite o acesso à Internet;
- Nome do package da aplicação – Identificativo único da aplicação;
- Nível mínimo da API do *Android* que a aplicação necessita;

2.2 ONVIF

O ONVIF (Open Network *Video* Interface *Forum*) surgiu com a necessidade de encontrar uma normalização entre a comunicação de dispositivos multimédia. Fundado pela Axis, Bosch e Sony em 2008, rapidamente entrou e cresceu no mercado com a sua utilização por parte dos grandes fabricantes de equipamentos de videovigilância. Atualmente o ONVIF é a norma mais adotada no mercado [18].

Dispositivos ONVIF podem ser classificados pelos seguintes tipos: *Network Video Transmitter* (NVT), *Network Video Storage* (NVS) e *Network Video Analytics* (NVA). Este trabalho é realizado sobre câmaras que enviam dados através de uma rede IP, ou seja sobre *Network Video Transmitter* (NVT).

De modo a definir os protocolos de comunicação entre os dispositivos, o ONVIF define várias especificações. A *Core Specification* [19] contextualiza a utilização de *WS-Discovery*, *WS-Security* [20] e *WS-BaseNotification* [21] [22]. Este também define os serviços *Device Management* e *Events*, serviços estes, comuns a todos os dispositivos. Cada uma das restantes especificações define por sua vez um dos restantes serviços do ONVIF. Estes serviços são classificados numa de três classificações:

- Serviços Obrigatórios;
- Serviços Obrigatórios dependendo de características da câmara;
- Serviços Opcionais.

Os serviços obrigatórios são serviços que estão implementados em todos os dispositivos que utilizam o ONVIF. Estes são os seguintes: *Device Management*, *Events*, *Media*, *Streaming*, *Device IO*.

O *Device Management* é um serviço que define configurações gerais do dispositivo. Este está dividido nas seguintes categorias: *Capabilities*, *Network*, *System* e *Security*. Através do acesso à categoria *Capabilities* é possível consultar informação sobre os serviços implementados no dispositivo. Por sua vez o Network permite consultar e alterar atributos como o DNS, DDNS e NTP. O *System* permite o acesso a informação de sistema do dispositivo tal como a manipulação da data e hora e realização de um reinício forçado do sistema. Através do *Security* é possível aceder a configurações de segurança como por exemplo realizar a gestão de utilizadores com permissões de acesso ao dispositivo.

O serviço de Media permite a consulta das configurações de media do dispositivo. É através deste serviço que se acede a perfis de media e se obtém as URIs necessárias para a realização de *stream* de vídeo e de *snapshots* da imagem atual capturada pelo dispositivo.

Os serviços obrigatórios dependendo de característica da câmara são serviços que, são obrigatórios caso o dispositivo em questão tenha implementado uma funcionalidade que permita a utilização do serviço. O único serviço nesta classificação é o PTZ.

O serviço PTZ é o serviço responsável pela configuração dos movimentos da *pan tilt zoom* de cada câmara. Caso este serviço não esteja implementado no dispositivo, é impossível a interação entre utilizadores e movimentos da câmara.

Os serviços Opcionais são serviços que não necessitam de estar implementados nos dispositivos que utilizam ONVIF nem dependem da implementação de outras funcionalidades. Nesta categoria encontram-se os seguintes serviços: *Imaging* e *Video Analytics*.

O serviço *Imaging* é um serviço oferece acesso a informação conforme o fornecimento de um vídeo *source* de um dispositivo. Este permite consultar informação de imagem como o contraste, brilho, focagem, etc.

2.3 Web services SOAP

Os *web services* (WS) definem formas normalizadas de descrever protocolos de rede, nomeadamente ficheiros WSDL que definem formatos de transmissão de dados e operações a suportadas. São, na maioria dos casos, baseados em normas bastante utilizadas como o HTTP para o transporte e o XML para o envio dos dados.

Atualmente existem dois tipos de implementação de web services bastante populares: o SOAP e o REST [23]. Nesta dissertação o estudo coincidiu sobre a utilização de web services SOAP.

O *Simple Object Access Protocol* (SOAP) é um protocolo que define serviços de troca de mensagens utilizando o formato XML. Os serviços SOAP são descritos utilizando ficheiros WSDL.

Uma vez que os serviços ONVIF são definidos em ficheiros WSDL, estes podem ser gerados automaticamente através da utilização de *toolkits* que geram *stubs* Java para a implementação da comunicação com cada serviço do ONVIF. Através dos stubs gerados é possível a receção e envio de troca de mensagens entre a aplicação e os *web services*.

2.4 Ferramentas de desenvolvimento Android

Nesta secção são identificadas as ferramentas utilizadas durante a realização deste trabalho.

2.4.1 Ambiente de desenvolvimento

Apesar de existirem várias alternativas, o *Integrated Development Environment (*IDE) recomendado pela Google para desenvolvimento de aplicações *Android* é o Eclipse com a utilização do *plugin Android Development Tools* (ADT) [24]. Por esse motivo foi com recurso ao

Eclipse ADT que o código para a realização das aplicações de teste e da aplicação final foram criados.

2.4.2 (Android) Software Development Kit

O *Android Software Development Kit* (SDK) é o conjunto de ferramentas oficial para desenvolvimento de aplicações para o Sistema Operativo *Android*. O SDK contém um compilador, um depurador, um emulador e bibliotecas essenciais para o desenvolvimento de aplicações. As aplicações SDK são desenvolvidas na linguagem de programação Java e executadas pelo *Dalvik*, a máquina virtual de processamento do *Android*.

Uma vez que o SDK é essencial para o desenvolvimento de aplicações *Android*, sempre que exista uma atualização na API do *Android*, existe também uma atualização no SDK.

2.4.3 Native Development Kit

O *Native Development Kit* (NDK) é, tal como o SDK, um conjunto de ferramentas oficial de desenvolvimento de software para o Sistema Operativo *Android.* Este permite o desenvolvimento nas linguagens de programação C e C++.

Uma vez que a programação com NDK é realizada através de linguagens de programação nativas, é uma melhor opção para operações computacionalmente pesadas, como por exemplo processar um grande volume de dados. Contudo, segundo a Google, a utilização do NDK não é benéfica para a maioria das aplicações, devendo ser utilizado apenas quando é necessário ou benéfico, por exemplo para reutilizar componentes existentes em C/C++ [25].

2.4.4 SWIG

De maneira a ser possível a troca de dados entre o código C e o *Android*, é utilizado o SWIG para a criação do meio de comunicação entre as duas linguagens de programação.

O *Simplified Wrapper and Interface Generator* (SWIG) é uma ferramenta de desenvolvimento de *software* que torna possível a conexão de código nativo escrito na linguagem de programação C e C++ em código de alto nível de programação como Java.

O SWIG é geralmente utilizado de modo a permitir a linguagens de programação de alto nível a troca de tipos de dados complexos assim como realizar invocações a código escrito nas linguagens de programação C e C++. O que o SWIG faz é criar o código de interligação que permite este tipo de comunicação entre as linguagens.

3. Estado da arte

Neste capítulo é descrita a análise realizada sobre as tecnologias existentes no âmbito desta dissertação. Foram analisadas as aplicações *Android* que permitem comunicação com câmaras de vídeo ONVIF. Foi também realizado, tal como especificado nos objetivos, um estudo com o fim de conhecer a maturidade de *toolkits* e bibliotecas existentes que ajudam na implementação da comunicação com *web services* SOAP.

3.1 Aplicações existentes

Foram estudadas aplicações existentes que permitissem comunicar com as câmaras. Apesar de existirem várias aplicações para este fim, apenas duas realmente utilizavam a norma para a troca de dados.

3.1.1 Onvif IP Camera viewer/explore

Ao iniciar o *Onvif IP Camera viewer* [26], tal como apresentado na figura 6 a), a aplicação permite adicionar uma câmara através de um clique no *ícone* "+" presente no cabeçalho.

Para adicionar uma câmara é necessário fornecer o endereço IP do dispositivo assim como um *username* e a *password* de acesso. Um exemplo de inserção de dados nesta aplicação é demonstrado na figura 6 b).

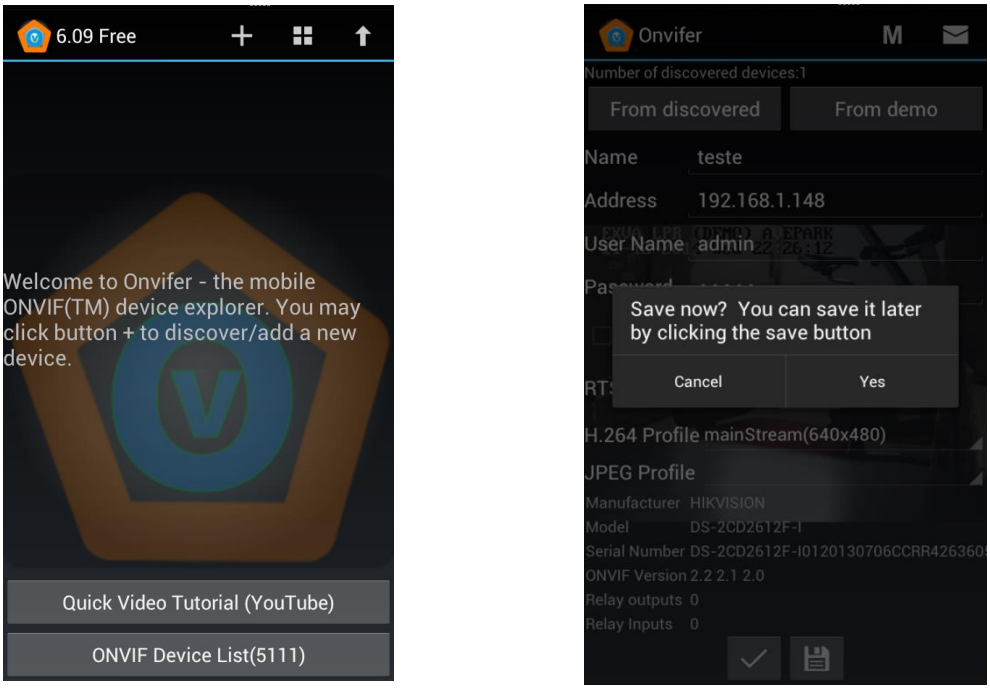


Figura 6 - a) - Menu Principal; b) - Adicionar câmara

Após aceitar que a informação seja guardada, a aplicação obtém todos os serviços ONVIF suportados pela câmara exceto o *Imaging* e o *Events*, sendo necessário esperar cerca de 40 segundos para que as funções de obtenção de dados terminem de comunicar com a câmara.

Durante o acesso ao menu "*Edit*", sempre que se modifica a orientação da aplicação (*portrait* para *landscape* e vice versa), os pedidos "*SystemDateAndTime*", "*getSnapshot*" e "*getMediaProfiles*" são repetidos. Para além destes pedidos, ao realizar uma alteração de um parâmetro de um perfil de media, a aplicação tenta eliminar o perfil alterado e cria um novo com as alterações efetuadas, repetindo posteriormente todas as comunicações com a câmara, mesmo dos pedidos sem relação com o parâmetro alterado. Mesmo realizando todos estes passos a alteração não é realizada com sucesso na câmara.

O acesso aos dados de configuração de uma câmara é realizado através da seleção da câmara no menu principal demonstrado pela figura 7 a) e b). Através do menu Explore é possível visualizar a informação anteriormente obtida e alterar as configurações de Media.

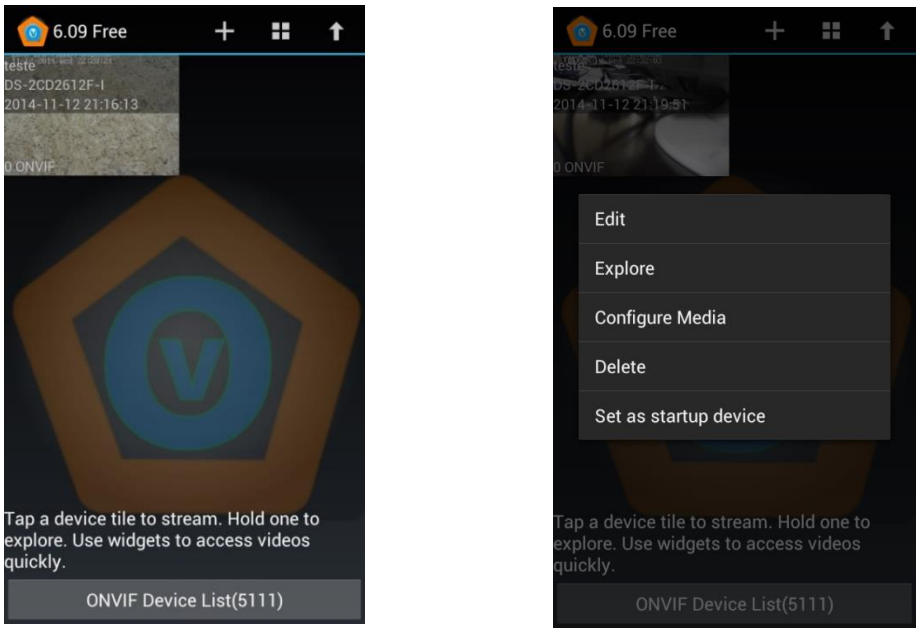


Figura 7 - a) - Acesso à câmara; b) - Configurações;

Após a seleção de visualização do *streaming* da câmara, a aplicação, apesar de informar que encontra o URI de *stream*, não foi capaz de o reproduzir.

Na figura 8 está presente o ecrã correspondente ao teste executado.

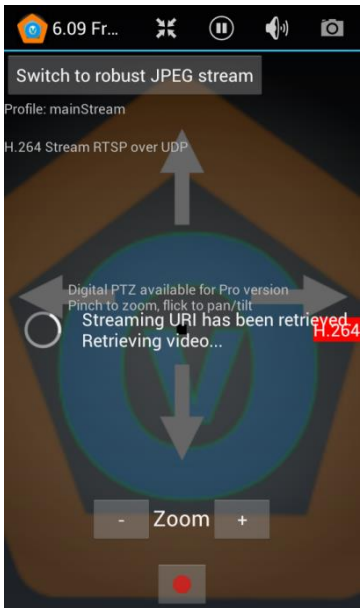


Figura 8 - Tentativa de reprodução de stream de vídeo

Após entrar em contacto com os criadores da aplicação foi possível saber que esta foi desenvolvida recorrendo apenas à utilização do SDK e código Java.

3.1.2 IP Cam Viewer Basic

Ao iniciar o *IP Cam Viewer* [27], tal como apresentado na figura 9 a), a aplicação permite adicionar uma câmara através de um clique no *ícone* "+" situado no centro inferior da aplicação.

Para adicionar uma câmara é necessário fornecer o endereço IP do dispositivo assim como um *username* e a *password* de acesso. Um exemplo de inserção de dados nesta aplicação é demonstrado na figura 9 b).

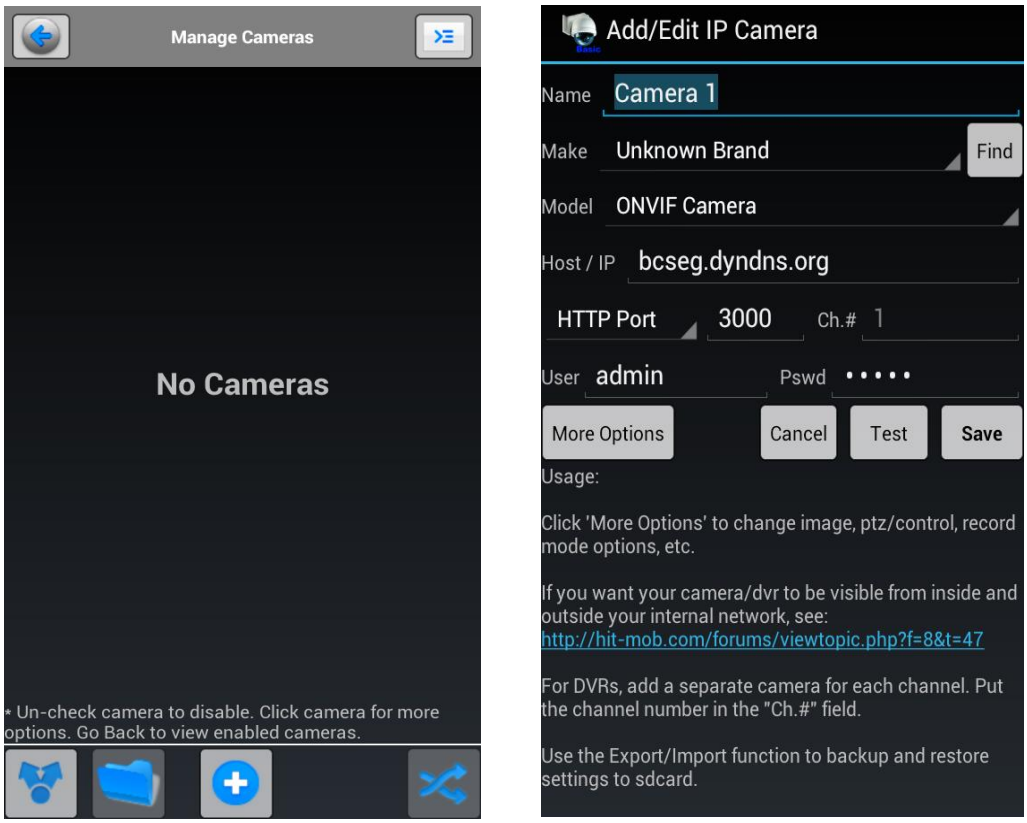


Figura 9 – a) Menu inicial; b) Adicionar câmara

Ao aceder ao menu "*More Options*" é permitida a modificação de atributos de imagem/som, ptz e de modos de gravação de imagem, como se pode observar pela figura 10.

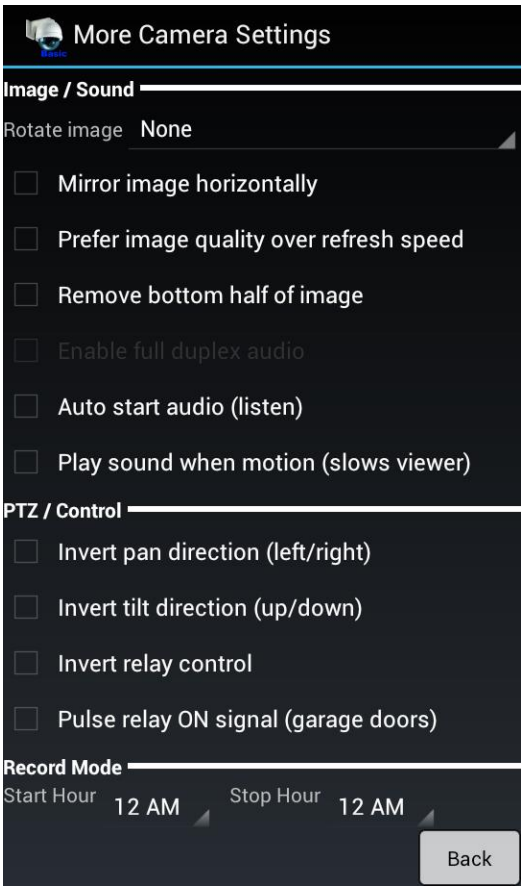


Figura 10 – Menu More Options

A Aplicação não permite o acesso a muitos dos serviços (mais concretamente do *System*, *Events* e vária informação dos restantes) do ONVIF.

A tentativa de *streaming* de vídeo foi, ao contrário do teste com a aplicação anterior, bem-sucedida.

Após entrar em contacto com os criadores da aplicação de modo a saber qual a arquitetura de desenvolvimento utilizada na aplicação, e apesar de várias tentativas, não foi obtida qualquer resposta.

3.2 Estudo de Toolkits e Bibliotecas

Um dos objetivos desta dissertação é o estudo da maturidade dos *toolkits* existentes para geração de código Java compatível com o *Android*.

Neste estudo foram utilizados os ficheiros onvif.xsd [28] e *devicemgmt.wsdl* que contêm as definições a gerar do serviço *Device Management* do ONVIF.

Para além dos *toolkits* foram também pesquisadas bibliotecas existentes que facilitassem o processo de comunicação com as câmaras.

3.2.1 EasyWSDL

O *EasyWSDL* [29] é um gerador de *stubs online* que gera classes Java ou *Objective*-C para aplicações *Android* ou *iOS*, respetivamente, consumirem *WebServices* SOAP. O código *Java* é baseado na biblioteca *Ksoap2-Android*.

Esta ferramenta gerou classes Java à primeira vista compatíveis com o *Android*, mas não foi possível a implementação de uma aplicação através da utilização deste gerador. Durante o estudo desta ferramenta, o modelo de negócio do *EasyWSDL* foi alterado sendo que, a versão gratuita não gera todas as classes necessárias.

3.2.2 Android Web Service Client (Jinouts)

O *Android Web Service Client* [30] permite a geração de *stubs Java* para o consumo de *web services* SOAP compatíveis com aplicações *Android*, que utiliza a *framework Apache CXF* [31]. Esta ferramenta não suporta serviços cujo *endpoint* é desconhecido (i.e., cujo endereço do serviço pode ser um qualquer), tal como acontece com os serviços ONVIF.

Para que o *Android Web Service Client* pudesse gerar código foi necessário adicionar ao ficheiro *devicemgmt.wsdl* um elemento *service* com uma porta com *binding* do tipo do serviço *Device* do ONVIF (definido nesse mesmo ficheiro) e definir o respetivo endereço como sendo de uma câmara.

Após a geração das classes *Java*, foi criado um projeto *Android* sendo estas classes importadas para o mesmo. A compilação do código detetou alguns erros de incompatibilidade, todos eles em classes que utilizavam o pacote "*javax.xml.ws.**". A situação foi facilmente resolvida ao modificar a importação desses pacotes para "*org.jinouts.xml.ws.**". Após esta correção o código gerado compilou sem erros.

Outra característica deste *toolkit* é que o código gerado depende do WSDL em tempo de execução, concretamente lê o ficheiro WSDL para obter o endereço do serviço. Por isso, o ficheiro foi colocado na memória do dispositivo de teste de modo a estar sempre acessível pela aplicação.

Foi implementada uma aplicação de teste simples que que fez a invocação da operação *getSystemDateAndTime*. A aplicação falhou com uma "*Fatal Exception*" que não foi possível corrigir, pois era originado numa classe cujo código fonte não estava disponível e envolvia reflexão [40].

3.2.3 wsdl2-ksoap2-android

O *wsdl2-Ksoap2-Android* [32] é uma ferramenta para a geração de classes Java compatíveis com o sistema operativo *Android*. Tal como o *EasyWsdl* esta ferramenta utiliza o *Ksoap2-Android* para a gestão dos pedidos e respostas SOAP e é baseado na *framework Apache Axis2*.

Esta ferramenta gerou classes sem problemas, contudo ao adicioná-las a um projeto causaram erros de compilação tal como aconteceu com o *Android Web Service Client*. Neste caso, faltava importar para o projeto *Java ARchives* específicos:

- *Axis2-1.5.1.jar;*
- *Log4j-1.2.17.jar;*
- *Ksoap2-android-assembly-3.3.0.jar.*

Com a inclusão destes ficheiros, alguns dos erros iniciais foram corrigidos mas outros mantiveram-se.

Após uma análise mais aprofundada dos erros, foi descoberta a falta de certas classes. Com o objetivo de tentar perceber a lógica para a não geração das mesmas foi realizado um levantamento das classes em falta e estas foram comparadas com certas classes geradas. Nesta análise foi possível verificar que as classes em falta eram todas relativas a elementos do tipo *xs:simpleType* do *onvif.xsd*. No entanto, havia elementos cuja definição era igual e as respetivas classes foram geradas.

Algumas das soluções possíveis passavam por:

- Implementação das classes em falta tendo como base as classes semelhantes que foram geradas;
- Substituir diretamente os elementos necessários por tipos nativos.

Foram feitas algumas tentativas, mas as modificações no código gerado inicialmente já eram bastantes. Mesmo resolvendo estes erros seria ainda necessário resolver os outros erros. De facto, as correções ao código obtido com o *wsdl2-ksoap2-android* não aparentavam ter um fim. Além disso, nenhuma das soluções consideradas garantia que o código final fosse correto e funcionasse como o esperado.

3.2.4 Ksoap2-Android

Com o não funcionamento dos *toolkits* estudados procedeu-se ao estudo da biblioteca *Ksoap2-Android* [33], biblioteca utilizada por dois dos *toolkits* analisados (*EasyWSDL* e *wsdl2-ksoap2-android*).

O *Ksoap2-Android* é apenas uma biblioteca, ou seja, não tem uma ferramenta de geração de código, que possibilita o envio e receção de mensagens SOAP em aplicações *Android*. Fornece classes e métodos necessários a todo este processo, desde a criação, serialização e envio do pedido à receção e de-serialização da resposta.

3.2.4.1 Envio do Pedido

Para ser possível realizar um pedido de comunicação com o *Ksoap2-Android*, é necessário em primeiro lugar a construção do pedido *HTTP* a enviar.

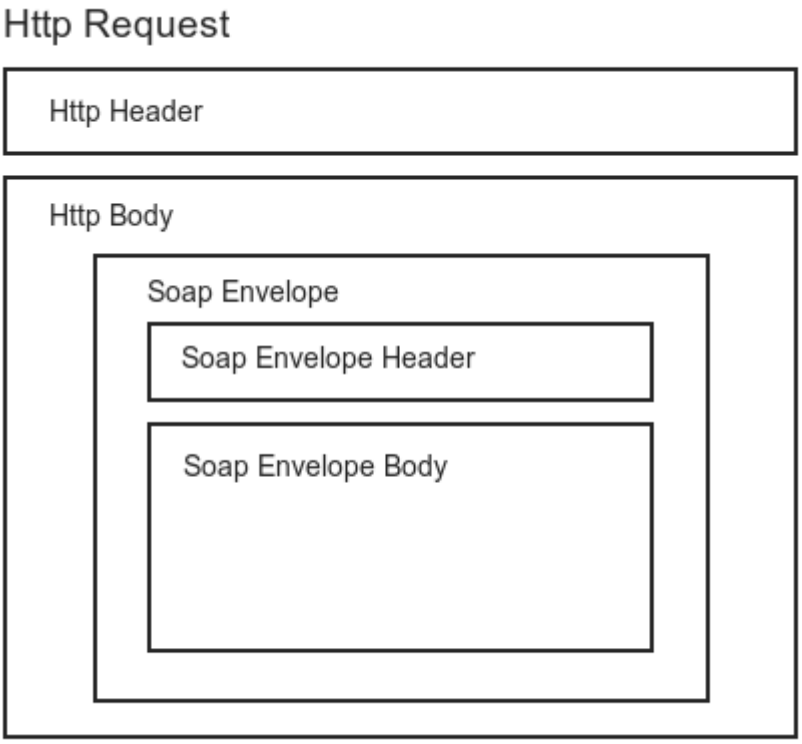


Figura 11 - Estrutura de um pedido HTTP a realizar

A figura 11 demonstra a estrutura de um pedido *HTTP* a enviar às câmaras. Para a realização do mesmo, o *Ksoap2-Android* oferece as seguintes classes:

- *HttpTransportSE* - Permite enviar pedidos *HTTP* nomeadamente definindo a URI no construtor e passando o cabeçalho e o corpo no método *call*;
- *SoapObject* – Permite a criação de um Objeto SOAP assim como a sua inicialização com o namespace da operação a realizar.
- *SoapSerializationEnvelope* - Permite a criação de um envelope SOAP. A inicialização deste é realizada com a utilização do método *setOutputSoapObject*, sendo-lhe fornecido o SoapObject correspondente ao corpo do pedido.
- *HeaderProperty* – Permite a criação de uma lista de *HeaderProperties* na qual é colocada informação de autenticação utilizando o método *add*.

3.2.4.2 (Construção do) Corpo do pedido

Apesar de existirem operações SOAP cujo corpo do pedido é vazio, existem várias que contêm dados a enviar.

```
<Test>
  <version>1</version>
  <note>Test1</note>
  <name>Ksoap Test</name>
</Test>
```

Figura 12 - Exemplo de uma estrutura de um ficheiro XML

A construção do XML a enviar no corpo do pedido é feita utilizando a classe *SoapObject* e os métodos:

- *addProperty* - Permite adicionar ao *SoapObject* elementos que não contêm *sub*e*lementos* (i.e., filhos);
- *addSoapObject* - Permite adicionar outros *SoapObject*, ou seja, elementos com subelementos (criando uma hierarquia XML);

A construção de pedidos que envolvem mais do que um *namespace*, faz-se criando *SoapObjects* inicializados com os respetivos *namespaces*.

3.2.4.3 Receção e processamento da resposta

Após o envio de um pedido, o corpo do envelope SOAP é substituído com o corpo da resposta ao mesmo.

Para o processamento dos dados da resposta o *Ksoap2-Android* oferece as seguintes classes:

- *SoapSerializationEnvelope* – Permite obter o *SoapOject* da resposta com o método *getResponse*;
- *SoapObject* – Permite a obtenção de elementos filho e folha da resposta através dos métodos *getProperty* e *getPrimitivePropertySafely*, respetivamente;

Desta forma é possível o acesso a todos os dados da resposta recebida. A utilização desta biblioteca não levantou problemas.

3.2.5 – Conclusão da análise de toolkits e bibliotecas

Previamente ao estudo dos *toolkits* descritos neste documento era esperado encontrar pelo menos um que permitisse a realização, com sucesso, de testes de comunicação entre um *web service* SOAP e uma aplicação Android. Por esta mesma razão, a análise destes *toolkits* e bibliotecas foi uma das tarefas que mais tempo consumiu na realização deste trabalho.

Para além do descrito, era esperado realizar a aplicação de teste desenvolvida somente com SDK recorrendo a um destes *toolkits*. Tal não foi possível uma vez que as incompatibilidades dos *toolkits* não foram totalmente ultrapassadas.

Com o não funcionamento dos *toolkits* estudados procedeu-se ao estudo da biblioteca Ksoap2-Android, biblioteca utilizada por dois dos *toolkits* analisados (EasyWSDL e wsdl2-ksoap2-android). Com esta biblioteca foi possível a realização, com sucesso, de testes de comunicação com as câmaras. Apesar da necessidade de se realizar mais trabalho em termos de programação na utilização desta biblioteca para comunicar com as câmaras, esta foi escolhida para a criação de uma das aplicações de teste.

4. Escolha da arquitetura

Tendo em conta a análise aos *toolkits* e bibliotecas realizada nos capítulos anteriores, procedeu-se à realização das aplicações de teste ilustradas na figura 1:

- Aplicação totalmente desenvolvida com SDK e que utilize o *Ksoap2-Android* para o auxílio na comunicação com as câmaras;
- Aplicação desenvolvida recorrendo ao NDK e à linguagem de programação C para a realização de comunicações com as câmaras;

4.1 Teste comparativo

O teste consistiu na realização de vários pedidos idênticos de uma maneira sequencial às câmaras.

Para efeitos comparativos entre as aplicações, foram avaliados os seguintes aspetos:

- Número de pedidos realizados pelas aplicações num determinado tempo;
- Consumo de bateria que o aparelho teve durante a realização dos pedidos;
- Facilidade de programação da comunicação (i.e., construção do pedido e processamento dos dados obtidos).

Uma vez que um dos grandes problemas no tratamento de pedidos SOAP é a serialização e de-serialização de dados, foi decidido que o pedido a ser realizado nos testes seria o *getCapabilities* uma vez que a este pedido corresponde uma resposta de grande volume de informação.

4.2 Estrutura de um pedido com *Ksoap2-Android*

Em seguida são exemplificados os passos utilizados para construir e enviar um pedido *getCapabilities* a uma câmara utilizando o *Ksoap2-Android*.

```
1 String SOAP_ACTION = "http://www.onvif.org/ver10/device/wsdl" +
          "/GetCapabilities";
  String METHOD_NAME = "GetCapabilities";
  String SOAP_NAMESPACE = "http://www.onvif.org/ver10/device/wsdl/";
  String URL = "http://192.168.1.148/onvif/device_service";

2 SoapSerializationEnvelope envelope;
  envelope = new SoapSerializationEnvelope(SoapEnvelope.VER12);

3 SoapObject request = new SoapObject(SOAP_NAMESPACE, METHOD_NAME);
  envelope.setOutputSoapObject(request);

4 List<HeaderProperty> headers = new ArrayList<HeaderProperty>();
  headers.add(new HeaderProperty
  ("Authorization", "Basic "+Base64.encode("admin:12345".getBytes())));

5 HttpTransportSE httpTransport = new HttpTransportSE(URL);

6 httpTransport.call(SOAP_ACTION, envelope, headers);
```

Figura 13 - Pedido ksoap2-Android

1. Inicialização de variáveis auxiliares que contêm informação como por exemplo a ação da operação SOAP e endereço do serviço da câmara.
2. Criação do envelope SOAP a enviar.
3. Criação do *SoapObject* que irá conter o corpo da mensagem.
4. Criação de lista de propriedades para o cabeçalho SOAP e adição de informação de autenticação.
5. Criação do objeto HTTP que irá transportar o pedido.
6. Invocação do método *call* da classe *HttpTransportSE* de modo a enviar o pedido.

É possível visualizar na figura 13 os passos realizados na criação e envio de um pedido com o *ksoap2-Android*.

4.2.1 Estrutura da resposta ao pedido

Depois do envio do pedido é necessário a receção da resposta e fazer o processamento/tratamento dos dados:

```
1 SoapObject response = (SoapObject) envelope.getResponse();
2 Capabilities_details capabilities = new Capabilities_details();
3 SoapObject device, network, system, io, security, events, imaging;
  SoapObject media, streamingCapabilities, analytics;
4 analytics = (SoapObject) result.getProperty("Analytics");
5 capabilities.getAnalytics().setXAddr(analytics
        .getPrimitivePropertySafelyAsString("XAddr"));
  capabilities.getAnalytics().setRuleSupport(analytics
        .getPrimitivePropertySafelyAsString("RuleSupport"));
  capabilities.getAnalytics().setAnalyticsModuleSupport(analytics
        .getPrimitivePropertySafelyAsString("AnalyticsModuleSupport"));
```

Figura 14 - Resposta ksoap2-Android

1. Obtenção do *SoapObject* com o elemento raiz da resposta.

Com a obtenção da resposta é então inicializado o processo de de-serialização.

2. Criação de um objeto para guardar a informação de-serializada (cuja classe contém atributos correspondentes aos dados do tipo *GetCapabilitiesResponse* do ONVIF).
3. Declaração de *SoapObjects* auxiliares para guardar os subelementos da resposta.
4. Obtenção dos subelementos da resposta. Na figura 14 exemplifica-se apenas o subelemento *Analytics*.

5. Obtenção dos elementos *folha* (sem filhos) da árvore XML da resposta. Estes dados são guardados no objeto da classe auxiliar previamente criada, para futura utilização. Exemplificam-se os elementos folha do elemento *Analytics*.

Uma vez que os dados foram obtidos como sendo *Strings*, é nos métodos set da classe *Capabilities* que se realiza a conversão do tipo de elementos obtidos para os tipos primitivos Java (por exemplo, "12345" para o tipo "int").

4.3 NDK e utilização de código nativo

Para implementar a invocação de funções C da biblioteca ONVIF existente a partir da aplicação *Android* através do JNI, foi utilizado o *Simplified Wrapper and Interface Generator* (SWIG). O SWIG é uma ferramenta de desenvolvimento da camada de *software* que faz a ponte entre código nativo escrito nas linguagens C/C++ e código de várias linguagens de alto nível como o Java. O SWIG é geralmente utilizado de modo a permitir a troca de tipos de dados complexos e o que faz é criar o código de interligação que permite a comunicação entre as linguagens.

O SWIG gera o código C que encapsula as funções C, fazendo a tradução dos tipos JNI para os tipos C, e classes Java que são *proxies* de tipos C, como por exemplo estruturas. Esta geração de código é automática em muitos casos, mas o SWIG permite definir num dialeto C/C++ próprio, para tratar situações específicas ou que fogem do previsto, como por exemplo devolver resultados por argumento do tipo endereço de apontador. O SWIG baseia-se no conceito de mapeamento de tipos, que consiste em definir regras para processar os vários tipos de variáveis globais e parâmetros, e até aplicar diferentes mapeamentos de forma seletiva distinguindo as variáveis e parâmetros pelo nome.

O pedido *getCapabilities* é implementado na biblioteca C pela seguinte função:

get_Capabilities_LL(const char address, const char* user, const char* pass, Capabilities** cap);*

O resultado é devolvido no último argumento, que é o endereço de um apontador. A forma recomendada para tratar estes tipos de dados em SWIG é esconder a função dentro do construtor da classe Java correspondente à estrutura *Capabilities*. O resultado é a seguinte classe:

```
public class Capabilities {
    public Capabilities() {...};
    public Capabilities(String address, String user, String pass) {...}
    public AnalyticsC getAnalytics() {...}
    public DeviceC getDevice() {...}
    public EventC getEvents() {...}
    public ImagingC getImaging() {...}
    public MediaC getMedia() {...}
    public PTZC getPTZ() {...}
}
```

Em seguida são exemplificados os passos utilizados para realizar um pedido a uma câmara ONVIF.

```
1 String ip = "http://192.168.1.148";
  String username = "admin";
  String password = "12345";

2 Capabilities c_capabilities;

3 c_capabilities = new Capabilities(ip, username, password);
```

Figura 15 - Pedido e resposta com o NDK

1. Criação de variáveis com identificação da câmara e informação de autenticação do utilizador.
2. Declaração do objeto auxiliar que irá conter a informação obtida.
3. Execução do pedido utilizando a informação criada anteriormente.

Tal como referido anteriormente, a comunicação e a de-serialização dos pedidos são realizados pelo código nativo, não sendo necessário mais nenhum passo para ter acesso aos elementos da resposta.

4.4 Aplicação de teste

A aplicação de teste foi realizada com o objetivo de ser o mais simples possível. De modo a se poder controlar o teste foi criado um botão que o permite iniciar e terminar a qualquer momento, como demonstrados pelas figuras 16 e 17 a).

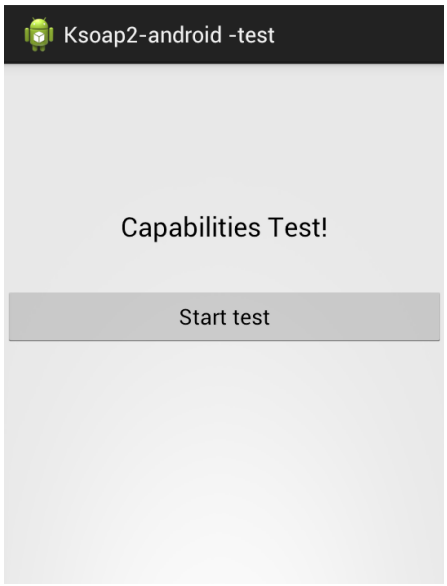


Figura 16 - Iniciar teste.

Após a finalização do teste, é inicializada uma nova *Activity* que apresenta o resultado do número de pedidos efetuados e o número de resultados processados pela aplicação, tal como demonstrado pela figura 17 b).

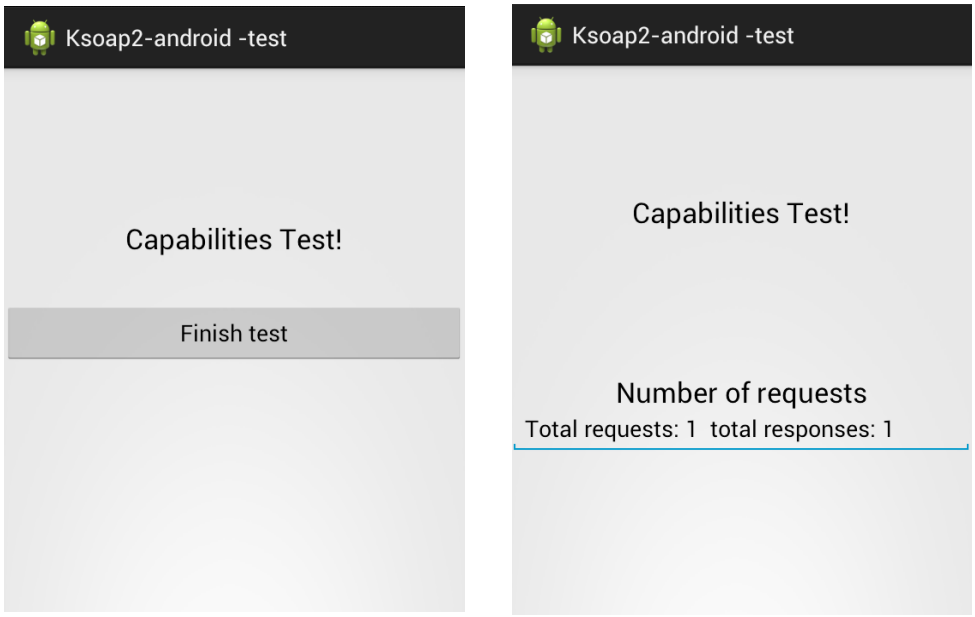


Figura 17 - a) - Finalizar teste; b) – Resultados.

4.5 Conclusões

Os testes das duas aplicações revelaram que a que utiliza o NDK é claramente mais rápida que a que utiliza a biblioteca Ksoap2-Android.

Para além disso, como é possível concluir a partir da descrição dos passos necessários à utilização de cada uma das soluções, a realização de um pedido com o NDK é bastante mais simples em termos de programação. Com base nestes resultados foi decidido proceder à criação da aplicação final através da utilização do NDK e da linguagem de programação C para comunicar com as câmaras.

5. Desenvolvimento da aplicação

5.1 Introdução

Uma vez que um dos objetivos propostos era a implementação de uma aplicação que fosse o mais eficiente possível, foi necessário ter em consideração alguns fatores importantes na sua estruturação.

Na construção da aplicação foi necessária a implementação de várias funções de comunicação com as câmaras, que envolvem o envio e receção de uma quantidade de dados significativa. A aplicação foi desenvolvida com o objetivo de proporcionar um acesso simples a esta informação, sendo este o aspeto mais importante no seu planeamento e implementação.

Neste capítulo é discutido o processo de desenvolvimento da aplicação final, a sua estrutura, as principais dificuldades enfrentadas e as soluções adotadas para os problemas encontrados.

5.2 Estrutura da aplicação

De forma a gerir a quantidade de informação a recolher a aplicação foi projetada para que a sua *User Interface* fosse flexível e simples, facilitando o acesso de um modo organizado às funcionalidades das câmaras. Com este aspeto em mente a aplicação foi criada utilizando maioritariamente *Fragments*. Desta forma era possível, para além da reutilização simplificada de certos componentes da *User Interface*, uma customização simples da mesma.

No total foram utilizadas duas *Activities*. Uma foi utilizada na criação do menu da lista de câmaras enquanto a segunda foi utilizada para o menu principal que realiza a gestão dos *Fragments* com informação relativa aos serviços do ONVIF.

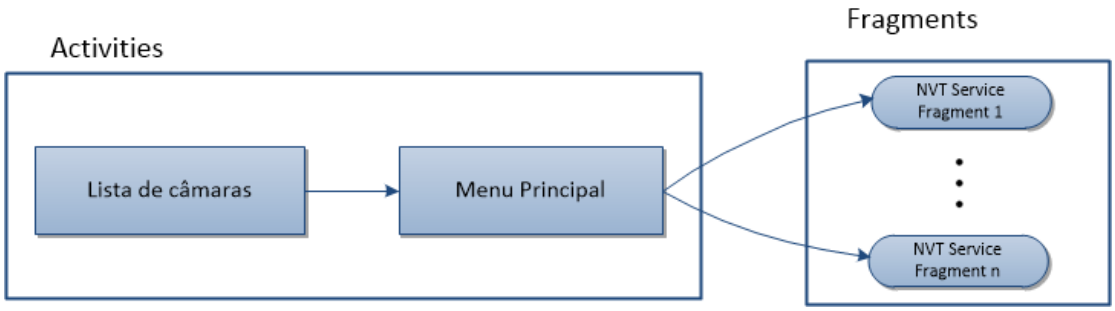


Figura 18 - Estrutura geral da aplicação

5.2.1 Lista de câmaras

Como se trata de uma aplicação que permite comunicar com câmaras de vídeo, a aplicação oferece, no primeiro menu, a possibilidade de gerir uma lista de câmaras. Esta gestão é permitida através do acesso às seguintes funcionalidades:

- Adicionar uma câmara;
- Remover uma câmara;
- Editar uma câmara.

A informação sobre as câmaras é guardada em ficheiros texto na memória interna do dispositivo na qual a aplicação está a ser executada. Isto é realizado uma vez que se pretende evitar que o utilizador tenha de adicionar as câmaras sempre que inicie a aplicação.

Figura 19 – Processo de adição de uma câmara

A figura 19 descreve o processo de adição de uma nova câmara na aplicação.

Após a inserção e validação dos dados, é realizado um teste de modo a verificar se a câmara, acabada de introduzir, se encontra *online* e disponível para ser acedida. Este teste é realizado ao efetuar um pedido para obter dados identificativos da câmara como: a marca, o modelo e o número de série. Caso o teste tenha uma resposta positiva, é possível informar o utilizador que a câmara que adicionou se encontra *online* e pronta a ser utilizada.

Figura 20 - Verificação do estado da câmara

A utilização deste pedido, como processo de verificação, foi escolhida uma vez que a informação recolhida é utilizada como um meio extra de identificação das câmaras no menu de seleção.

Na figura 20 é possível visualizar um fluxograma do teste de verificação realizado após a adição de uma nova câmara na aplicação.

5.2.2 Menu Principal

Após a seleção da câmara, a *User* Interface é transferida da *Activity* do menu de seleção para a *Activity* que disponibiliza o acesso à informação da câmara. É esta *Activity* que serve como base para a utilização dos *Fragment*s que irão controlar a *User* Interface.

De modo a inicializar e gerir os *Fragment*s, foi projetado e criado um *Navigation Drawer* [34]. Um *Navigation Drawer* é um painel que permite a navegação pelos menus principais da

aplicação. Este é implementado diretamente na *Activity* que gere os *Fragments* de modo a garantir que, mesmo que se mude de menu, este continue acessível.

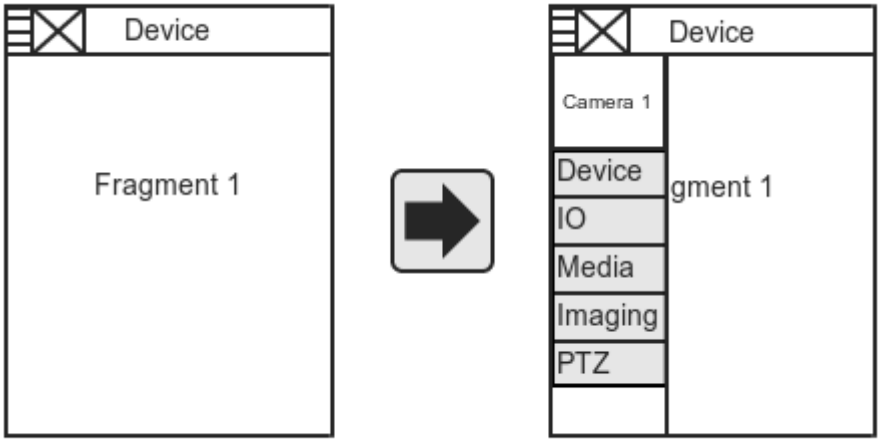


Figura 21 - *Navigation Drawer* projetado

Como podemos visualizar na figura 21, o *Navigation Drawer* é acedido com um clique no logotipo da aplicação ou ao realizar um "*swipe*" da direita para a esquerda do ecrã.

Outro motivo para a implementação desta funcionalidade foi o facto de esta não necessitar de ocupar a interface a todo o momento. Cada entrada corresponde a um serviço ONVIF.

Com a seleção de um dos menus do *Navigation Drawer*, o *Fragment* que corresponde a esse mesmo menu é inicializada e colocada na *User* Interface.

5.2.3 "Swipe Menus"

Uma vez que há menus que disponibilizam uma grande quantidade de dados, nomeadamente os menus *Device*, Media e *Imaging*, estes não devem ser apresentados aos utilizadores utilizando apenas uma único *Fragment*. A informação deve ser apresentada por partes e permitir ao utilizador navegar entre essas partes. De maneira a resolver este problema, e a manter a estrutura definida no *Navigation Drawer*, foram criados *Fragment*s auxiliares, denominados *Fragment Adapters* [35], responsáveis pela gestão dos *Fragment*s que trocam informação com o utilizador. Os *Fragment Adapters* inicializam os *Fragments* de Informação e permitem a transição entre elas.

A figura 22 mostra a estrutura adotada para a navegação nos menus da aplicação, exemplificando o caso de acesso ao serviço *Device Management*.

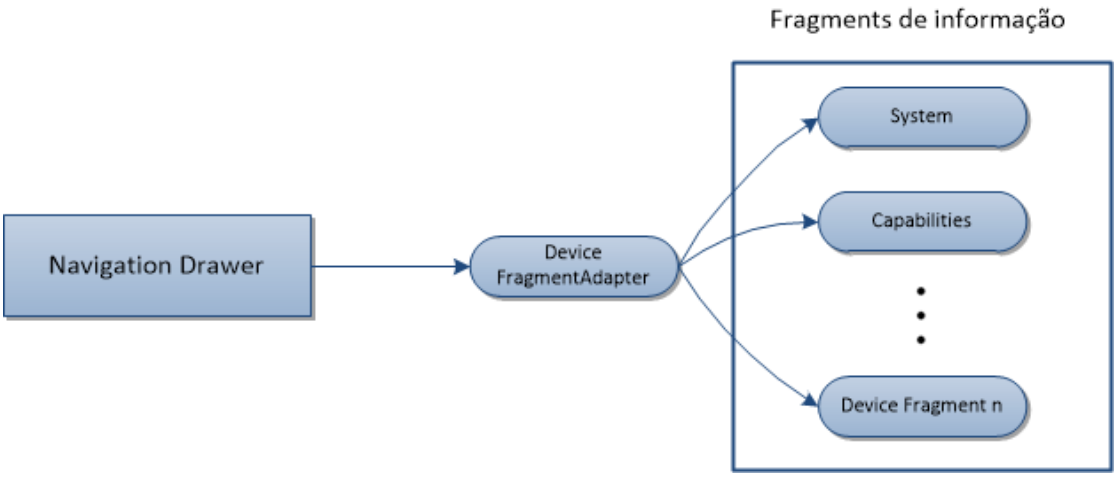


Figura 22 – Gestão da User Interface recorrendo a um Fragment auxiliar

Os *Fragment Adapters* são inicializados pelo do *Navigation Drawer* e permitem a transição entre *Fragments* de Informação através de um "swipe" na interface.

Na figura 23 é possível visualizar o modo de funcionamento da aplicação recorrendo a esta funcionalidade.

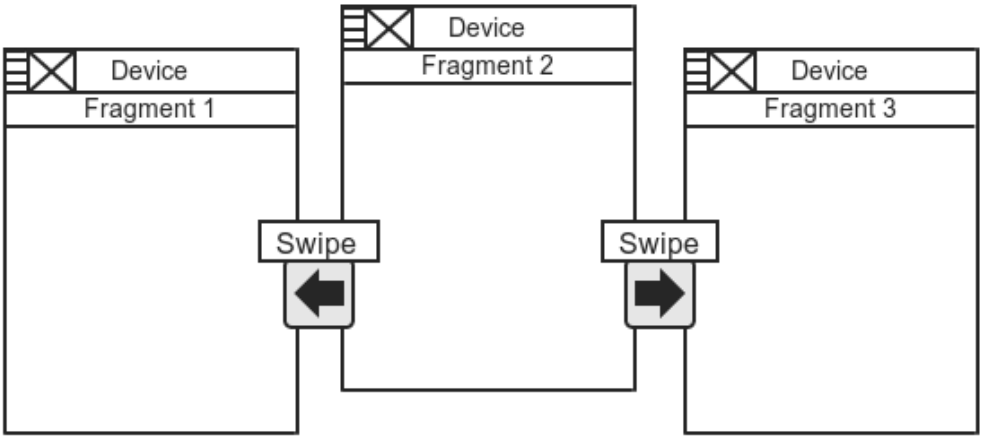


Figura 23 – Aplicação com Swipe Views

Com esta solução foi então possível criar uma aplicação que, apesar de conter menus que apresentam a bastante informação, estejam organizados e sejam de simples acesso.

5.3 Dificuldades no desenvolvimento

Durante o desenvolvimento da aplicação surgiram algumas dificuldades que foram necessárias ultrapassar de modo a conseguir uma aplicação que fosse de encontro aos objetivos definidos.

Neste subcapítulo são explicados os problemas enfrentados assim como as soluções adotadas e o impacto que estas tiveram na estrutura da aplicação.

5.3.1 Inicialização dos Fragments

Para implementar a gestão dos *Fragments* da *User* Interface utilizando *swipe views* foi necessária a utilização de um *PagerAdapter*. Existem dois tipos de *PagerAdapter* para utilizar em situações como esta:

- *FragmentPagerAdapter* [36];
- *FragmentStatePagerAdapter* [37].

A diferença entre estes consiste no modo como gerem as *Views* pelas quais são responsáveis. O *FragmentPagerAdapter* é aconselhado para a navegação entre um número fixo e pequeno de *Fragments*. Este, quando é criado, inicia todos os Fragment*s* por si controladas. O *FragmentStatePagerAdapter* é aconselhado para a navegação entre um número maior de *Fragments*. Este realiza a destruição de *Fragments* antigos enquanto o utilizador navega pelos mais recentes.

Por defeito, um *FragmentStatePagerAdapter* inicia o *Fragment* que preencherá a *User* Interface e os *Fragment*s que se encontram imediatamente antes e depois. Por exemplo, na figura 22, o *Fragment* ativo é o *Fragment* 2, mas, os *Fragment*s 1 e 3 também são iniciados uma vez que se encontram ao lado deste. O *Fragment* 4, como não se encontra imediatamente ao lado do *Fragment* ativo não é inicializado.

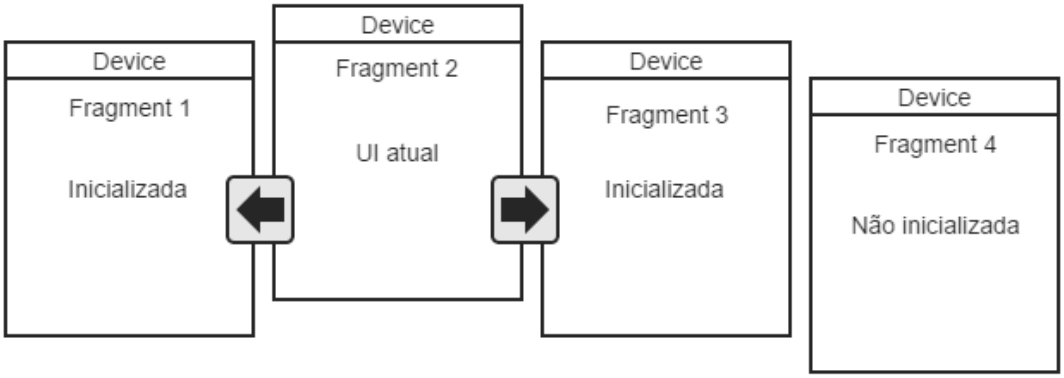


Figura 24 - Inicialização dos Fragments

Um eventual *swipe* do *Fragment* atual para o *Fragment* 3 faz com que o *Fragment* 4 seja iniciada e o *Fragment* 1 seja eliminada pelo *adapter*. Caso um utilizador realize vários *swipes* em ambas as direções, os *Fragment*s são iniciadas e destruídas várias vezes.

Para o seu funcionamento, a aplicação necessita de dados que são obtidos comunicando com as câmaras. Para além do processamento relativo à interface, que foge ao controlo do programador, a comunicação com as câmaras é o aspeto principal da otimização para tornar a aplicação mais eficiente.

Partindo do princípio que a maioria dos utilizadores não deseja aceder ou visualizar toda a informação que a aplicação fornece de cada vez que a utilizam, excluiu-se a hipótese de realizar todos os pedidos possíveis no arranque da aplicação. Essa implementação seria ineficiente e, no caso de a câmara não estar na rede local, faria com que a aplicação demorasse algumas dezenas de segundos a "arrancar". Decidiu-se colocar os pedidos distribuídos pelos *Fragments* que implementam as funcionalidades dos diferentes serviços. Mesmo assim, para cada serviço existiam duas alternativas:

- Configurar a aplicação para iniciar todas os *Fragment*s, independentemente de estas pretenderem ser visualizadas ou não, com a utilização do *FragmentPagerAdapter*.
- Utilizar o *FragmentStatePagerAdapter* e criar um mecanismo que verifique se o Fragment já foi iniciado antes e, em caso positivo, disponibilize os resultados das comunicações anteriormente realizadas.

Apesar da primeira opção ser a mais simples, a realização de todos os pedidos de informação não só torna a aplicação mais lenta a entrar no serviço, como provavelmente/possivelmente faz pedidos inúteis (por exemplo, o utilizador pode não querer aceder a todos as partes de um serviço).

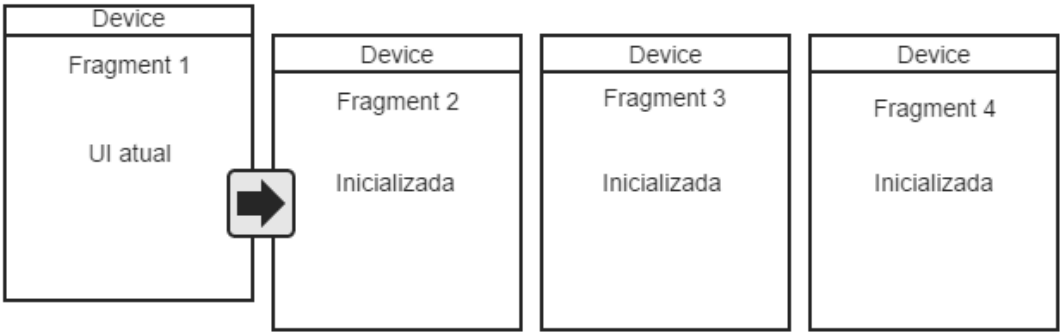


Figura 25 - Inicialização de todas os Fragments com o FragmentPagerAdapter

Chegando a esta conclusão optou-se por desenvolver uma solução com base no segundo método descrito. Apesar de não se evitar a possibilidade de repetidas criações e destruições da mesmo Fragment, cada pedido à câmara selecionada só é efetuado uma vez. Desta forma, o impacto na fluidez da aplicação é minimizado, evitam-se pedidos redundantes e minimizam-se os desnecessários, aumentando a eficiência.

Para tal foi criado um mecanismo de verificação baseado no armazenamento das respostas ONVIF num *HashMap*. Este é passado nos argumentos de todas as classes que compõem os *Fragments* (controlados pelos *Fragment adapters)* permitindo que os dados resultantes dos pedidos já realizados possam ser acedidos em qualquer altura.

No arranque de cado *Fragment*, a aplicação tenta obter os dados necessários a partir do *Hasmap* e, só se não os encontrar é que envia o(s) pedido(s) à câmara.

Garante-se assim que, apesar dos *Fragments* poderem ser iniciadas várias vezes, as comunicações com a câmara atual só são efetuadas na primeira criação.

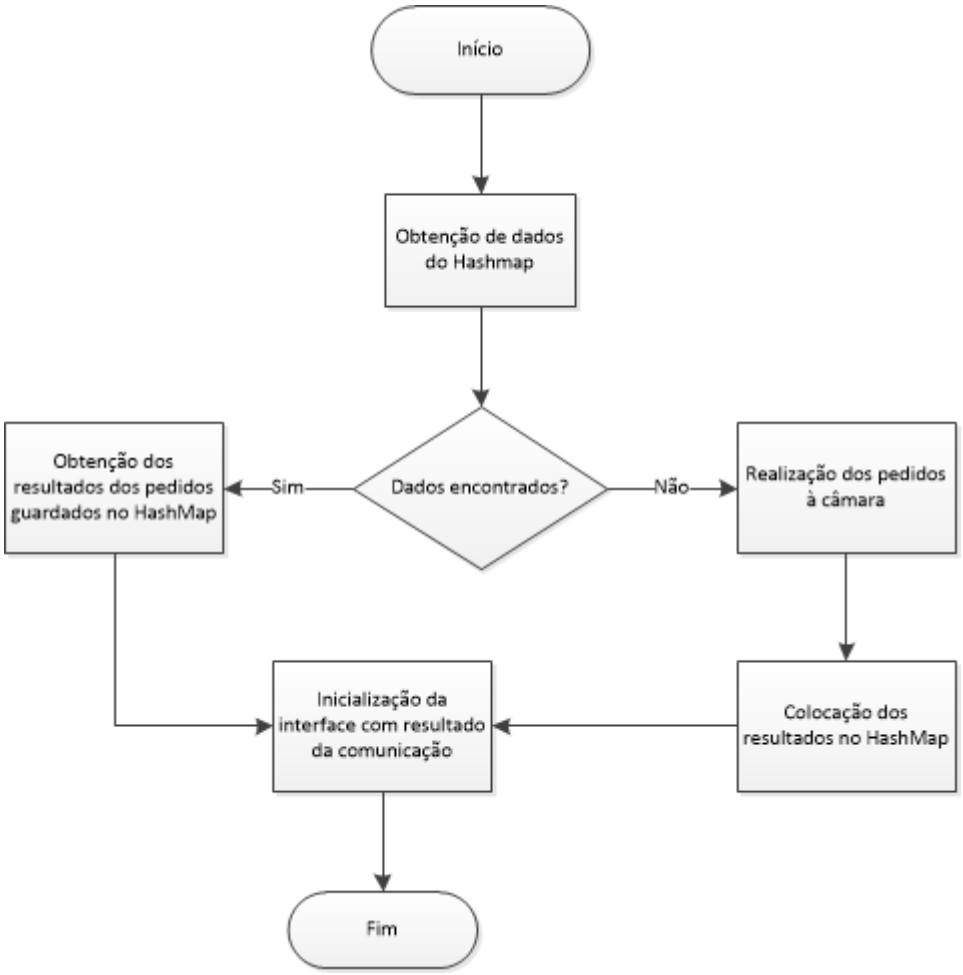


Figura 26 - Inicialização dos *Fragments*

A utilização desta solução permite resolver o problema encontrado uma vez que evita a inicialização de *Fragments* não desejadas assim como uma eventual repetição de comunicações com as câmaras.

5.3.2 Alteração dos dados da câmara

A implementação da alternativa descrita no subcapítulo anterior faz surgir um problema. Uma vez que a recolha de dados das câmaras só é realizada uma vez, quando o utilizador modifica alguma configuração com sucesso e haja um reinício do respetivo *Fragment*, a alteração não será visível na *User Interface*. Isto acontece porque os dados da câmara guardados no *HashMap* estão desatualizados.

Repetir o pedido à câmara correspondente à configuração alterada seria um desperdício de tempo e CPU uma vez que a aplicação dispõe de todos os dados alterados e também da confirmação da alteração por parte da câmara.

Uma alternativa que foi considerada foi a alteração direta das estruturas de dados obtidas das câmaras. No entanto, a comunicação com as câmaras é realizada através do NDK, e algumas das estruturas de dados não são copiadas para o Java. A camada JNI mantém apenas uma ligação para os dados C e os objetos Java são apenas proxys. Isto não é um problema para grande parte das configurações, nem para todos os dados que são constantes mas, o resultado de certos pedidos é um *array* de objetos. Apesar de ser possível alterar os elementos do *array*, refletindo-se na memória C, não é possível aumentar/diminuir o número de elementos.

Este problema foi solucionado criando classes auxiliares (concretamente, apenas quatro: *Scopes*, *Users*, *Profiles* e *Presets*) para guardar exclusivamente em Java os dados provenientes das câmaras que sejam arrays. Como essas classes não têm memória C associada, é possível a adição e remoção de objetos de contentores Java sem problemas. São estes contentores que são guardados no *HashMap* (de modo a estarem acessíveis durante o funcionamento da aplicação) em vez dos obtidos diretamente das câmaras.

Naturalmente, os dados guardados nas classes auxiliares só são utilizados para preencher a *User Interface*. Por sua vez, os dados obtidos da comunicação com as câmaras são também úteis para fornecer aos pedidos de alteração das configurações. Isto acontece quando o tipo dos argumentos dos respetivos métodos (geralmente, do tipo set/*add*) é igual ao dos métodos que obtêm as configurações (geralmente, do tipo *get*). Embora a otimização que resulta dessa reutilização possa não ser muito significativa, ela tornou-se obrigatória pois o mapeamento SWIG que trata os referidos tipos não permite a criação de objetos pelo Java.

Quando se alteram essas configurações, os dados obtidos das câmaras são alterados sendo depois adicionados nas classe auxiliar. Desta forma, mesmo que os dados originais das comunicações anteriores sejam alterados, estes continuam a ser acessíveis pois foram previamente guardados nas classes auxiliares. Esta solução é ilustrada na figura 27.

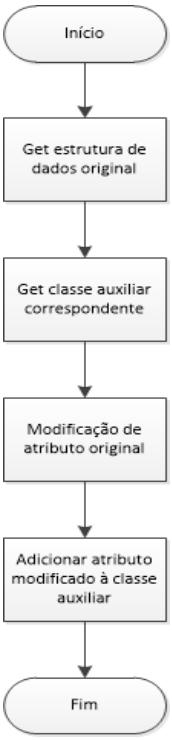


Figura 27 - Adição de dados à câmara

6. Resultados e discussão

6.1 Maturidade dos toolkits

Dos *toolkits* analisados (EasyWSDL, Android Web Service Client e esdl2-ksoap2-android) concluiu-se que nenhum revela maturidade suficiente para a sua utilização na aplicação. Apesar de se ter superado vários erros, não foi possível a geração de código correto, que compilasse, para serviços ONVIF. Posto isto, foi utilizada uma biblioteca SOAP de auxílio ao processo de construção de pedidos e obtenção das respetivas respostas mais simples de implementar.

6.2 Desempenho NDK vs SDK

6.2.1 Ambiente de teste

Durante o desenvolvimento das aplicações de teste (descritas no cap. 4) e da aplicação final foi necessário realizar testes. Estes podiam ser feitos de uma das seguintes formas:

1. Num dispositivo virtual simulado pelo ADT;
2. Num *smartphone* com uma versão do *Android* compatível com a aplicação;

Após a realização de alguns testes com simuladores do ADT, foi possível verificar que os mesmos são bastante lentos a iniciar, sendo também muito "pesados" em termos de processamento. Devido a essa razão, decidiu-se realizar os testes num dispositivo físico.

O dispositivo utilizado foi um telemóvel HTC *Sensation* com a versão *IceCreamSandwish do Android*. Desta forma, o tempo perdido com o lançamento do aparelho virtual é eliminado e a compilação e execução do programa são muito mais rápidas.

Apesar da última versão do *Android* ser o *Android* 5.0 *Lollipot* (API *level* 21), a aplicação foi realizada de modo a funcionar da melhor forma no *Android IceCreamSandwish* (API *level* 14) uma vez que, esta era o nível de API máximo suportada pelo sistema operativo do dispositivo no

qual os testes da aplicação foram realizados. Contudo, uma vez que o *Android* garante compatibilidade para com as versões anteriores, a aplicação funcionará de igual modo nas versões mais recentes.

A versão mínima do *Android* necessária ao funcionamento da aplicação é o *Android HoneyComb* (API level 11) pois foi a partir desta versão que foi possível a utilização de *Fragments*.

6.2.2 Resultados dos testes

Numa primeira abordagem, os testes das aplicações de comparação NDK vs. SDK foram realizados utilizando uma câmara acedida através da Internet durante cerca de 90 minutos. Após a realização de alguns testes com a mesma aplicação foi possível chegar à conclusão que estes apresentavam uma variação do número de pedidos efetuados relativamente alta. Em alguns dos testes realizados a variação do número de pedidos foi até 58.5%. Isto foi um claro indicador que os fatores que poderiam influenciar os resultados estavam a ter um papel preponderante, nomeadamente a variação de tráfego em qualquer ponto da rede entre a aplicação e a câmara.

Desta forma era necessário arranjar uma configuração que permitisse a realização de testes com menor variação.

Para isso, criou-se uma *Local Area Network* (LAN) (figura 28) na qual se conectaram a câmara e o dispositivo com as aplicações de teste, através de um *router*. Desta forma foi possível utilizar um meio controlado que permitiu recolher resultados conclusivos.

Após a realização de novos testes com as duas aplicações em estudo, a variação entre o número de pedidos efetuados de cada aplicação ficou abaixo dos 5.5%.

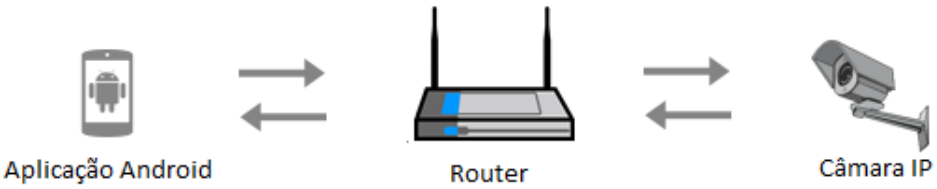


Figura 28 - Local Area Network dos testes realizados

No total foram realizados 16 testes com a duração de 30 minutos, sendo claramente visível, através do número de comunicações realizadas por cada aplicação, qual a mais eficiente no processamento. Na figura 29 é possível observar a diferença no número de pedidos processados por cada uma das aplicações.

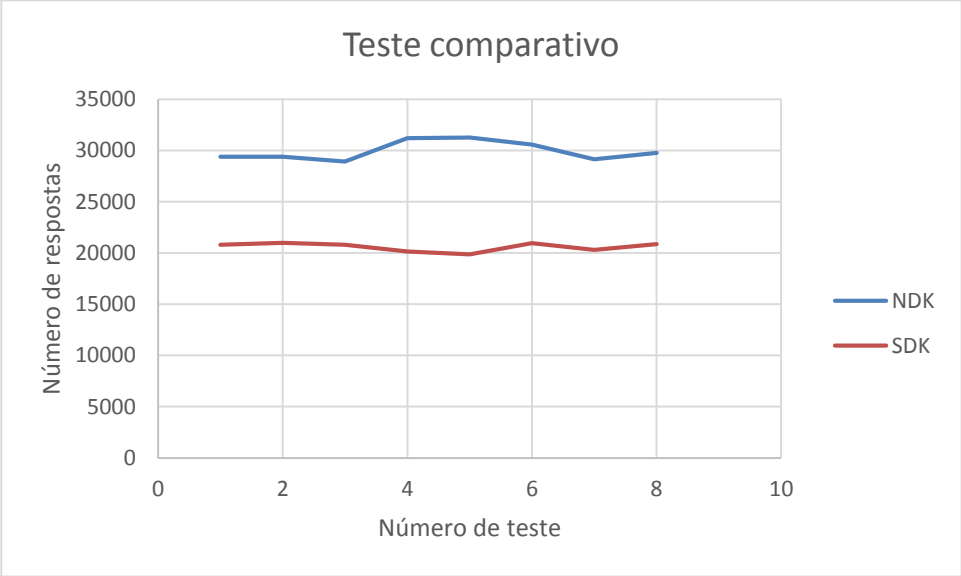


Figura 29 - Resultados das aplicações de teste

A conclusão sobre a aplicação mais rápida a processar pressupõe que o tempo gasto nas comunicações é constante. Para comprovar isso, foi realizada uma análise mais aprofundada, nomeadamente para conhecer o tempo gasto no envio e receção dos pacotes trocados com a câmara, e dessa forma medir o real tempo gasto por cada aplicação no processamento, conforme ilustrado na figura 30.

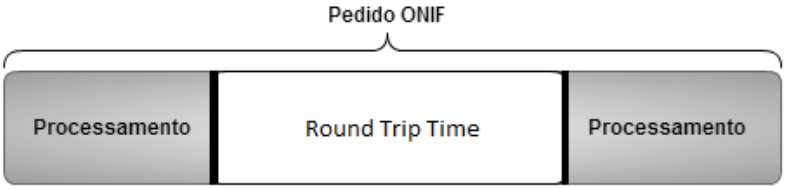


Figura 30 – Round Trip Time (RTT) e processamento de um pedido

Uma vez que a API de comunicação em JAVA não pode ser comparada com a API de comunicação em rede utilizada pelas bibliotecas C, a medição do tempo de rede teve de ser realizada de uma forma externa às aplicações. Com a utilização do Wireshark, um *software* de análise de tráfego de rede, foi possível a análise dos pacotes trocados entre as aplicações e a câmara. Através desta análise foi possível verificar que o Round Trip Time (RTT) de cada comunicação. Verificadas as primeiras quatro comunicações, registaram-se os seguintes tempos: 16.7ms; 13.6ms; 17.2ms; 14.2ms; Ficou assim comprovado que o tempo gasto na rede durante o envio e receção de cada mensagem é praticamente constante e equivale a aproximadamente 15.4 milissegundos.

Com a chegada ao valor correspondente ao tempo de rede dos pedidos foi possível o cálculo do tempo de processamento dos mesmos utilizando a seguinte equação:

$$Nndk * (RTT + PTndk) = Nsdk * (RTT + PTsdk) \approx Tempo\ Total$$

em que Nsdk e Nndk correspondem ao número de pedidos executados pela aplicação realizada com o ksoap2-Android e NDK, respetivamente; PTndk e PTsdk correspondem ao tempo de processamento que cada aplicação consumiu por pedido; E, Tempo Total é a duração de um teste.

Com a substituição dos valores na equação foi possível obter como valores médios dos vários testes:

TPndk = 44 ms;

TPsdk = 73.1 ms;

6.3 Aplicação final

6.3.1 Gestão das câmaras

Ao iniciar a aplicação é apresentado ao utilizador o menu de gestão de câmaras. Neste é permitido a adição, modificação e remoção de câmaras disponíveis na aplicação.

De modo a adicionar uma câmara o utilizar necessita de premir o botão sinalizado com um "+".

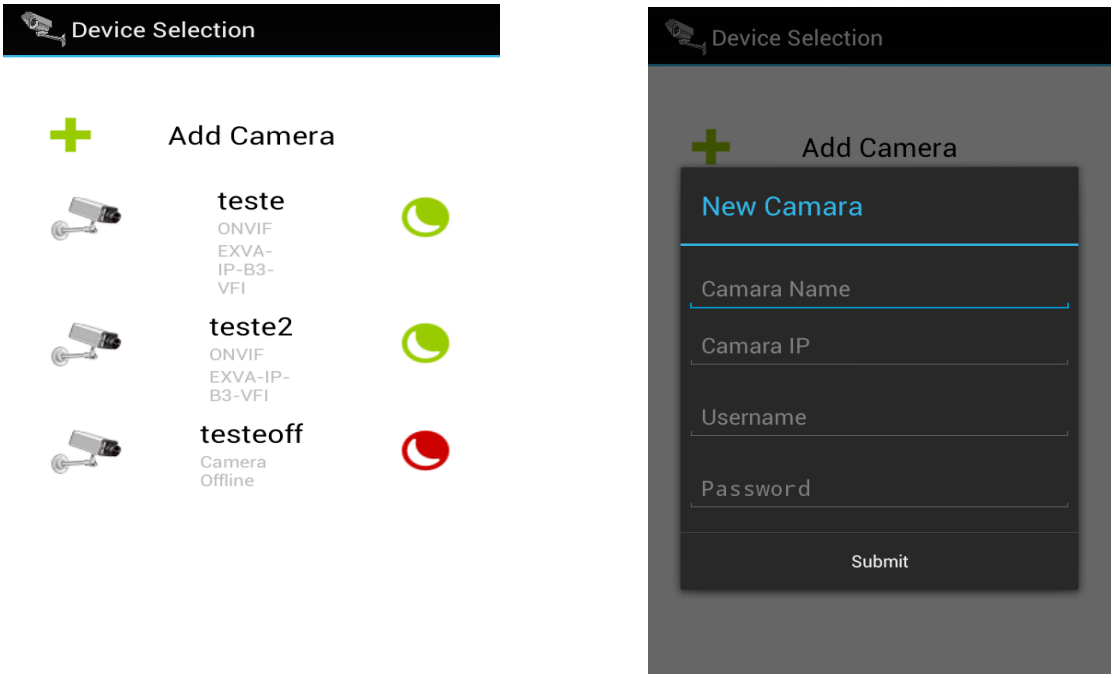


Figura 31 - Menu de seleção de câmara; b) – Adicionar câmara

Como é possível observar na figura 31 esta interação irá fazer aparecer no ecrã, uma caixa de diálogo para o utilizador inserir os dados da câmara a adicionar.

A alteração ou remoção de uma câmara da lista são possíveis ao tocar prolongadamente sobre a câmara pretendida. O resultado desta interação pode ser visualizado na figura 32.

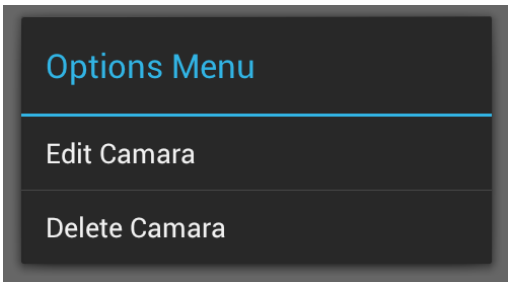


Figura 32 - Opções de gestão de uma câmara

6.3.2 Menu de serviços

Quando se seleciona uma câmara a aplicação transita a *user interface* para o serviço *Device Management*. Para aceder aos outros serviços ONVIF é preciso clicar no lado esquerdo do ecrã para fazer aparecer o menu de serviços, conforme ilustrado na figura 33 a).

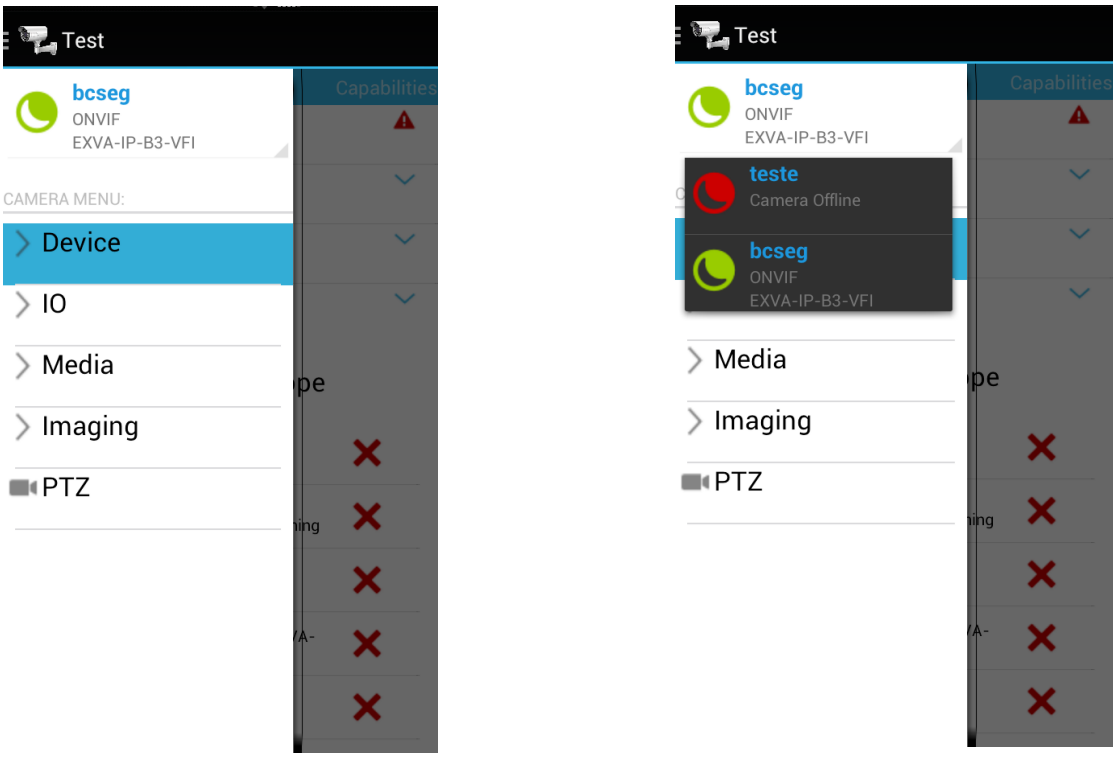


Figura 33 - a) - *Navigation Drawer*, b) - Selecção de câmara

Através deste menu é também possível alterar a câmara selecionada sem voltar ao menu de gestão das câmaras. Desta forma, é possível trocar de câmara de uma forma simples e rápida. Na figura 33 b) ilustra-se o funcionamento do menu de serviços.

6.3.3 Device

O serviço *Device Managenment* é composto por quatro diferentes submenus: *System*, *Capabilities*, *Network* e *Users*.

6.3.3.1 Device System

O submenu *Device System* é o inicial do serviço. Neste o utilizador tem acesso a várias informações sobre o sistema da câmara.

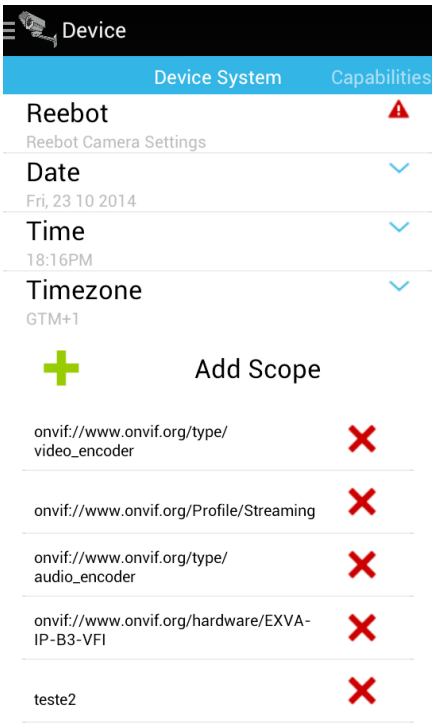


Figura 34 - Device System

Como é visível na figura 34 neste menu é possível realizar um reinício da câmara. Como se pode observar pela figura 35 a), quando isto acontece o utilizador terá de confirmar a operação.

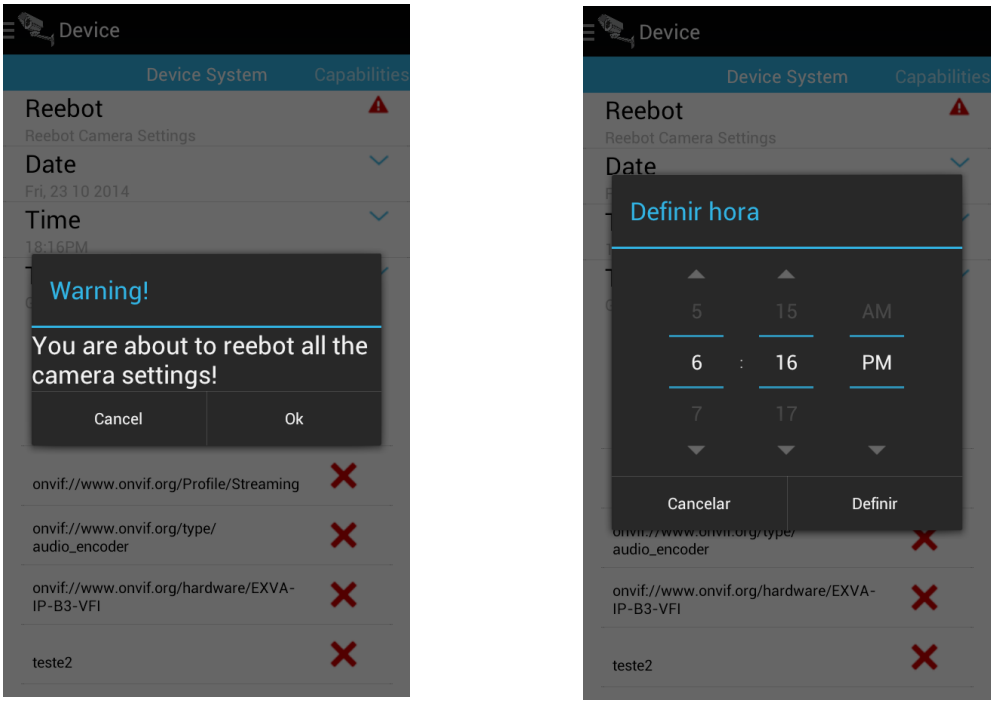


Figura 35 - a) – Reiniciar câmara; b) - Alterar hora

A informação de relógio e data da câmara é também alterável. Para tal é apresentado na interface um *Dialog* que permite ao utilizador realizar as modificações que pretende. Um exemplo de modificação de dados de relógio das câmaras é exemplificado na figura 35 b).

Neste submenu são também apresentados parâmetros de scope da câmara.

Os parâmetros de scope são utilizados na descoberta de dispositivos. Estes podem ser fixos ou configuráveis. Parâmetros fixos correspondem a características permanentes da câmara e não podem ser alterados. Parâmetros configuráveis podem ser geridos pelos utilizadores.

A gestão da lista de scopes da câmara é bastante semelhante à gestão da lista de câmaras disponíveis no menu de seleção de câmara. A única diferença é que neste menu a eliminação de um scope é realizada ao pressionar o ícone “x” no item correspondente.

6.3.3.2 Capabilities

Neste submenu é apresentada informação sobre configurações dos diferentes serviços da câmara. Esta informação é dividida em grupos conforme o serviço a que pertencem.

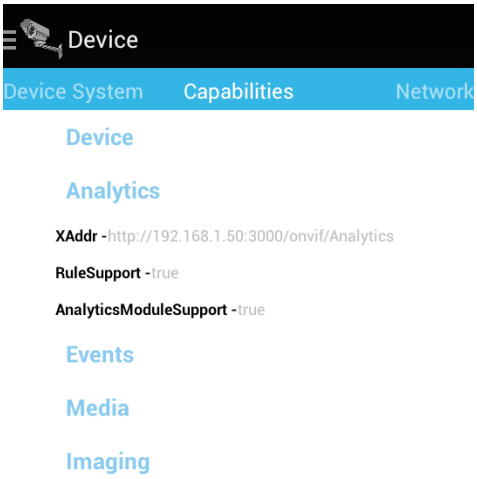


Figura 36 – *Capabilities*

Uma vez que se trata de uma *view* com bastante informação, esta foi organizada, como é possível observar pela figura 36, recorrendo a *ExpandableLists*. Ou seja, de modo a visualizar a informação sobre um determinado grupo, o utilizador necessita de carregar no título do respetivo grupo fazendo com que uma lista de atributos e respetivos valores seja visível.

6.3.3.3 Network

Neste submenu é possível a visualização e alteração do NTP (*Network Time Protocol*), DNS (*Domain Name System*) e DynDNS (*Dynamic DNS*). De forma a editar estes campos é necessário um clique no atributo que se deseja alterar, aparecendo na interface, um *dialog* que permite a alteração do mesmo.

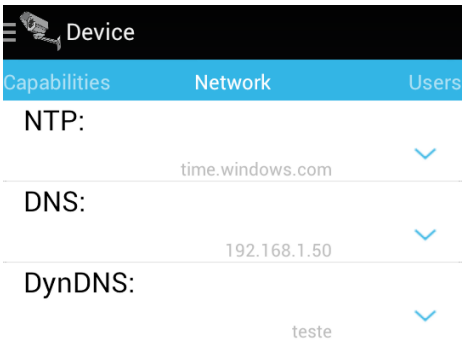


Figura 37 – Network

Na figura 37 é possível observar o acesso aos recursos oferecidos pelo Network.

6.3.3.4 Users

Este submenu é dedicado à gestão de utilizadores com registo na câmara selecionada. Neste submenu é permitido adicionar, editar ou eliminar perfis de utilizadores. A gestão destes é realizada do mesmo modo que a gestão da lista de scopes descrita anteriormente.

Na criação de um perfil de utilizador é apenas necessário indicar o *username* e a *password* do mesmo.

Na figura 39 é possível observar o aspeto deste submenu.

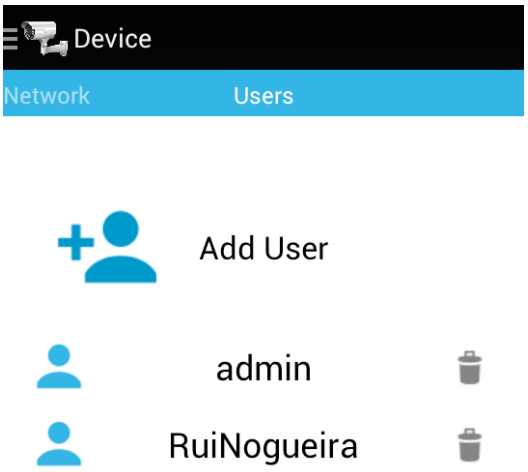


Figura 38 – Utilizadores

6.3.4 Input/Output

Este serviço permite comandar as saídas digitais das câmaras, caso existam.

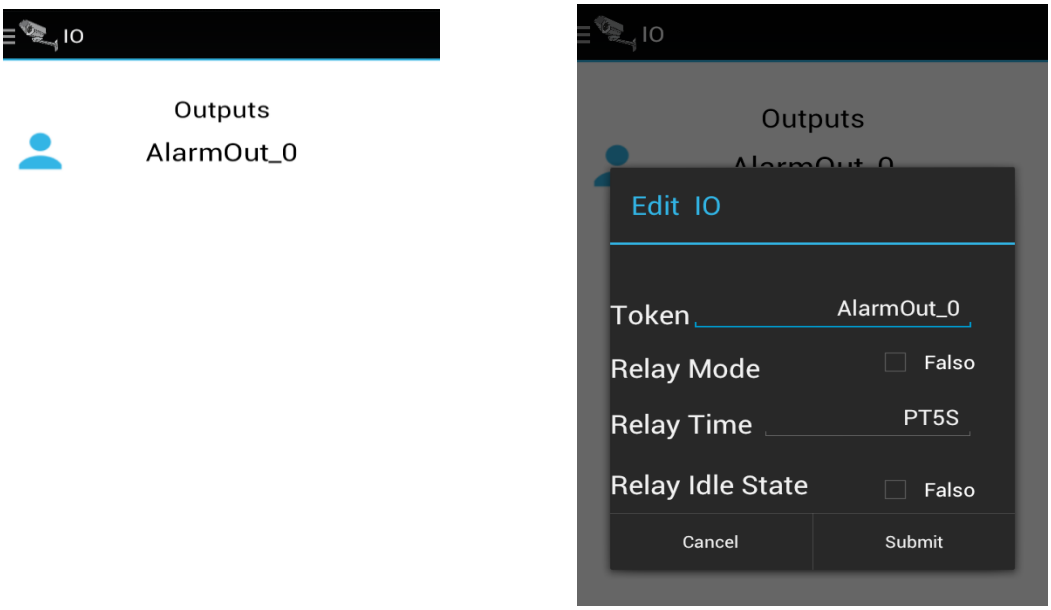


Figura 39 – a) - Input/Output; b) – Edit input/Output

Como é visível na figura 39 é possível alterar o *Token* do componente assim como o *relay mode*, o *relay time* e o *relay idle state*.

6.3.5 PTZ

Este menu é responsável pela reprodução de *streaming* da câmara. Para a reprodução de vídeo o utilizador necessita de selecionar o perfil de media cujo *stream* pretende visualizar. Isso acontece antes da alteração de interface (da interface atual para a de *streaming*) através do aparecimento de um *dialog box*, como demonstrado pela figura 40.

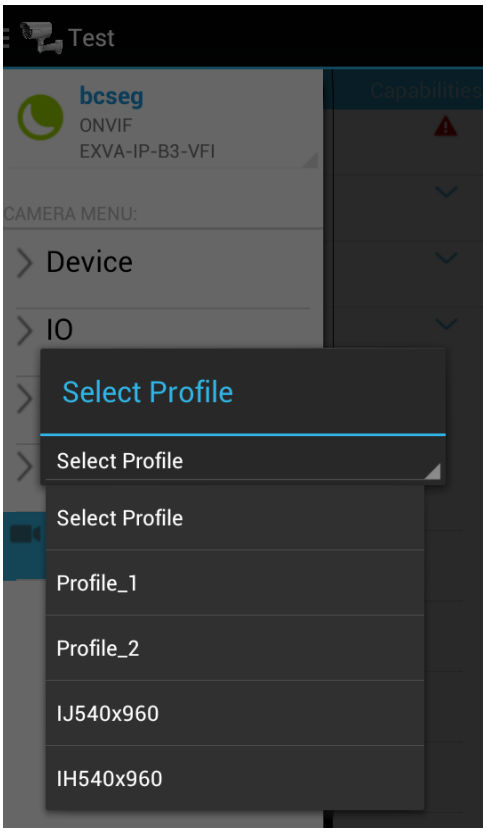


Figura 40 - Seleção de perfil de media para reprodução

Após a seleção do perfil a utilizar o *streaming* é iniciado, tal como é mostrado na figura 41.

Figura 41 - *Streaming* de vídeo

6.3.6 Imaging

Neste menu é possível o acesso às configurações de imagem que afetam um determinado *video source* da câmara, escolhido tal como no caso do PTZ. Este é composto por três diferentes submenus: *Imaging Status* e *Move Options*, *Imaging Settings*.

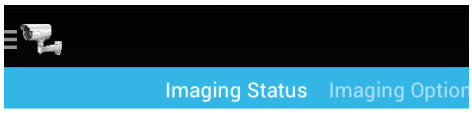


Image Status

Move Options

Figura 42 - *Imaging status e Move Options*

No submenu *Imaging Status* e *Move Options* foi agrupada a consulta de dois tipos de informação, por serem de tamanho reduzido (e ambas de consulta). Na primeira é possível a consulta do estado atual da focagem da imagem de um *video source* específico. A segunda permite a consulta das opções para o movimento da lente da câmara, nomeadamente intervalos de velocidades e posições possíveis.

6.3.7 Imaging Options

Através do menu *Imaging Options* é possível consultar as opções de configuração das propriedades de imagem de um determinado *video source*. Neste submenu é possível visualizar as seguintes informações:

- *Brightness;*
- *ColorSaturation;*
- *Contrast;*
- *Sharpness;*
- *Expusure options.*
- *Focus options;*
- *Wide Dynamic Range options;*
- *White Balance options.*

No menu *Imaging Settings* é possível a visualização das propriedades de imagem de um determinado vídeo source. Neste é possível visualizar as configurações descritas no *Imaging Options*. Uma vez que os últimos dois submenus descritos apresentam um grande número de informações, o *layout* implementado para estes é o mesmo utilizado no exemplo da figura 37, utilizado no *Capabilities*.

6.3.8 Media

Este serviço permite o acesso e alteração das configurações de media da câmara. Este é composto por dois diferentes submenus: *media options* e media *profiles*.

O *Media Options* é a secção inicial do serviço *Media*, e disponibiliza todas as opções para os vários parâmetros de media suportados pela câmara:

- *Video source configurations;*
- *Video encoder configurations;*
- *Audio source configurations;*
- *Audio encoder configurations;*
- *Metadata configurations;*
- *PTZ configurations;*
- *PTZ spaces.*

Esta funcionalidade não foi implementada, pois envolve um total de 16 elementos de *user interface* que é necessário processar. No entanto, esta implementação nada acrescenta em termos de conhecimento, pois é igual à do menu *Capabilities* do serviço *Device Management* (permite a consulta de uma grande quantidade de informação).

O *media profiles* lista os *tokens* dos perfis (figura 43). A gestão destes perfis é realizada do mesmo modo que a gestão de scopes e de utilizadores.

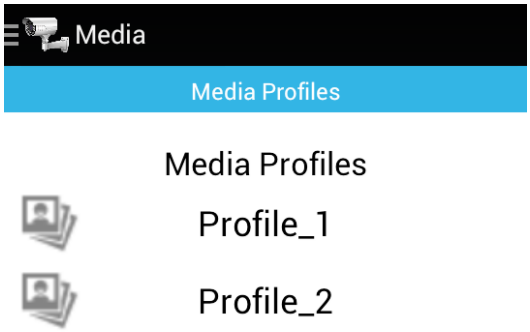


Figura 43 - Media Profiles

Selecionando um perfil a aplicação disponibiliza um novo ecrã. Esta funcionalidade não foi mais além, mas tal como a anterior consiste num grande volume de elementos (concretamente 37) de *user interface* que é preciso disponibilizar e não acrescenta nada de novo. A diferença para a anterior é que será necessário permitir a edição dos campos. Estas edições são limitadas

às opções de media existentes e por isso todas consistem na disponibilização de listas de opções, tal como é feito por exemplo, com a seleção de um perfil de *streaming* no serviço de PTZ.

Foi decidido disponibilizar separadamente o Media Options, cujas opções poderiam aparecer apenas durante a edição dos perfis, para que o utilizador pudesse conhecer de uma forma mais simples todas as opções disponíveis.

6.3.9 Mensagens de confirmação

Na utilização da aplicação, sempre que se efetue uma operação que altere o estado de alguma configuração, é apresentado ao utilizador uma mensagem de confirmação sobre a operação efetuada.

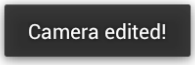


Figura 44 - Mensagem de navegação

Na figura 44 é possível observar um exemplo de uma mensagem deste tipo.

7. Conclusão

Neste trabalho foram estudados *toolkits* de geração de *stubs* clientes para serviços SOAP no sistema operativo *Android*. Por diferentes razões e mesmo depois de superados vários erros que foram encontrados, nenhum dos *toolkits* foi capaz de gerar código correto para serviços ONVIF. Foi então estudada e aplicada uma biblioteca SOAP que facilita a construção de pedidos e obtenção das respetivas respostas para implementar a comunicação com as câmaras.

Foi realizado um estudo comparativo do desempenho computacional de duas aplicações de teste, uma foi implementada utilizando o SDK *Android* e outra utilizando o NDK para a parte de comunicação com as câmaras. Este estudo revelou uma clara vantagem na utilização do NDK. Foi assim desenvolvida a aplicação utilizando o NDK para a comunicação com as câmaras e a restante parte utilizando o SDK.

O desenvolvimento da aplicação final envolveu um estudo aprofundado do *Android* e foram encontrados vários problemas. Estes problemas prenderam-se com dois aspetos: por um lado a definição de uma estrutura de interface adequada às funcionalidades de configuração e acesso ao *streaming* de câmaras ONVIF e por outro evitar o fácil esgotamento de recursos do dispositivo devido aos pedidos ONVIF. Para estes problemas foram desenhadas soluções que, no primeiro aspeto, proporciona uma utilização fácil e intuitiva da aplicação (sobretudo considerando que o utilizador está minimamente familiarizado com o ONVIF) e, no segundo aspeto, oferece um desempenho otimizado da aplicação. Foi ainda utilizado o código gerado pela ferramenta SWIG para fazer a interligação entre Java e C, que envolve variadas formas de mapeamento dos dados e também da API gerada do lado Java.

O trabalho atingiu praticamente todos objetivos propostos inicialmente. Concretamente, a pesquisa e avaliação de maturidade dos *toolkits* SOAP para *Android* e a comparação de desempenho das alternativas NDK e SDK foram completamente conseguidas. A aplicação final apenas não implementa a gestão de perfis ONVIF nem o PTZ. Cada perfil envolve algumas dezenas de variáveis e a implementação é muito demorada. Quer esta funcionalidade, quer o PTZ, não acrescentam nenhuma novidade em termos de desenvolvimento.

7.1 Trabalho futuro

Como melhorias na aplicação podem apontar-se, além de completar as funcionalidades de configuração que não foram implementadas, a adição de funcionalidades de videovigilância, incluindo a possibilidade de receber vários streams e até gravação dos mesmos.

Como trabalho mais fundamental, poder-se-á explorar a otimização da camada constituída pela parte da biblioteca acima dos stubs C, a camada JNI e os proxys Java de modo a evitar duplicação de cópia de dados e a maximizar o processamento que é feito na parte NDK.

Bibliografia

[1] Christian Gehrmann: ONVIF Security Recommendations. Axis Communications AB and Swedish Institute of Computer Science, Version 1.0, March 2010.

[2] James Snell, Doug Tidwell, Pavel Kulchenko: Programming Web Services with SOAP. O'Reilly Media, December 2001.

[3] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana: Mukhi, and Sanjiva Weerawarana: Unraveling the Web ServicesDescription Language (WSDL) 1.1. W3C Note, March 2001.

[4] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi and Sanjiva Weerawarana: Unraveling the Web Services Web – An Introduction to SOAP WSDL, and UDDI. Watson Research Center, March 2002.

[5] ONVIF, Dezembro 2013. *ONVIF™ Core Specification, Version 2.2.1.* [Online]. Available: http://www.onvif.org/specs/DocMap.html

[6] Android SDK, Dezembro 2013. [Online].

Available: https://developer.android.com/sdk/index.html?hl=i

[7] Android NDK, Fevereiro 2014. [Online].

Available: https://developer.android.com/tools/sdk/ndk/index.html

[8] Java Native Interface, Fevereiro 2014. [Online].

Available: https://docs.oracle.com/javase/6/docs/technotes/guides/jni/

[9] SWIG, Fevereiro 2014. [Online]. Available: http://www.swig.org/exec.html

[10] Axis2, Março 2014. [Online]. Available: http://axis.apache.org/axis2/java/core

[11] Java API, Dezembro 2013. [Online]. Available: http://www.oracle.com/technetwork/java/api-141528.html

[12] Android popularity, Dezembro 2013. [Online]. Available: http://developer.android.com/about/index.html

[13] Android API, Janeiro 2014. [Online]. Available: http://developer.android.com/reference/packages.html

[14] Android Activity, Janeiro 2014. [Online]. Available: http://developer.android.com/reference/android/app/Activity.html

[15] Android Fragment, Janeiro 2014. [Online]. Available: http://developer.android.com/guide/components/fragments.html

[16] Android Layouts, Janeiro 2014. [Online]. Available: http://developer.android.com/guide/topics/ui/declaring-layout.html

[17] Manifest File, Janeiro 2014. [Online]. Available: http://developer.android.com/guide/topics/manifest/manifest-intro.html

[18] ONVIF vs PSIA: Guide to Standards In Video Surveillance, Maio 2014. [Online]. Available: http://www.ifsecglobal.com/guide-to-standards-in-video-surveillance-onvif-v-psia/

[19] ONVIF, Dezembro 2013. *ONVIF™ Core Specification, Version 2.2.1.* [Online]. Available: http://www.onvif.org/specs/DocMap.html

[20] OASIS, Março 2014. Web Services Security: SOAP Message Security 1.1 (WS-Security). [Online]. Available: http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[21] OASIS, Março 2014. Web Services Base Notification 1.3 (WS-BaseNotification). [Online]. Available: http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf

[22] OASIS, October 2006. Web Services Topics 1.3 (WS-Topics). [Online]. Available: http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf

[23] Web Services SOAP, introduction, Novembro 2013. [Online]. Available: http://www.w3schools.com/webservices/ws_soap_intro.asp

[24] Android Developer Tools, Novembro 2013. [Online]. Available: http://developer.android.com/tools/sdk/eclipse-adt.html

[25] NDK performance, Janeiro 2014. [Online]. Available: https://developer.android.com/tools/sdk/ndk/index.html

[26] Onvif IP Camera Viewer/Explorer, November 2013. [Online]. Available: https://play.google.com/store/apps/details?id=net.biyee.onvifer&hl=en

[27] IP Cam Viewer Basic, November 2013. [Online]. Available: https://play.google.com/store/apps/details?id=com.rcreations.ipcamviewerBasic&hl=en

[28] W3C, April 2012. XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. [Online]. Available: http://www.w3.org/TR/xmlschema11-2

[29] EasyWSDL, Março 2014. [Online]. Available: http://easywsdl.com/

[30] Android Web Service Client, Março 2014. [Online]. Available: https://code.google.com/p/android-ws-client/

[31] Apache CXF, Março 2014. [Online]. Available: http://cxf.apache.org/

[32] Wsdl2-ksoap2-Android, Abril 2014 [Online]. Available: https://code.google.com/p/wsdl2ksoap2-android/

[33] Ksoap2-Android, Fevereiro 2014. [Online]. Available: https://code.google.com/p/ksoap2-android/

[34] Android Navigation Drawer, Maio 2014. [Online]. Available: https://developer.android.com/design/patterns/navigation-drawer.html

[35] Android FragmentAdapter, Maio 2014. [Online]. Available: http://developer.android.com/training/implementing-navigation/lateral.html

[36] Android FragmentPagerAdapter, Maio 2014. [Online]. Available: http://developer.android.com/reference/android/support/v4/app/FragmentPagerAdapter.html

[37] Android FragmentStatePagerAdapter, Maio 2014. [Online]. Available: http://developer.android.com/reference/android/support/v4/app/FragmentStatePagerAdapter.html

[38] Zigurd Mednieks, Laird Dornin Meike, Masumi Nakamura: Java Programming for the New Generation of Mobile Devices, 2011

[39] Ian F. Darwin: Android Cookbook, Problems and Solutions for Android Developers. O'Reilly Media, April 2012

[40] Patrick Niemeyer, Daniel Leuck, Learning Java, 4th Edition, O'Reilly Media, June 2013.