Universidade do Minho
Escola de Engenharia

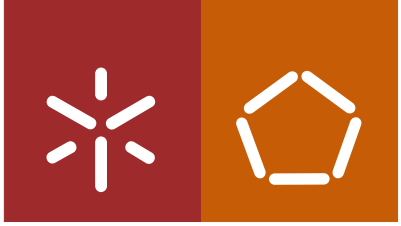
Hélder Filipe da Silva Machado Dias

Comunicação Android com câmaras ONVIF e gestão de Perfis de Media

outubro de 2015

Hélder Filipe da Silva Machado Dias **Comunicação Android com câmaras ONVIF e gestão de Perfis de Media**

UMinho | 2015



Universidade do Minho
Escola de Engenharia

Hélder Filipe da Silva Machado Dias

Comunicação Android com câmaras ONVIF e gestão de Perfis de Media

Dissertação de Mestrado
Mestrado Integrado em Engenharia de Telecomunicações e Informática

Trabalho efetuado sob orientação do
Professor Doutor Sérgio Adriano Fernandes Lopes

outubro de 2015

Declaração

Nome: Hélder Filipe da Silva Machado Dias

Endereço electrónico: a58658@alunos.uminho.pt

Telefone: +351 914 903 368

Número do Bilhete de Identidade: 13830165

Título dissertação: Comunicação Android com câmaras ONVIF e gestão de Perfis de Media

Orientador(es): Professor Doutor Sérgio Adriano Fernandes Lopes

Ano de conclusão: 2015

Designação do Mestrado: Mestrado Integrado em Engenharia de Telecomunicações e Informática

Universidade do Minho, ___/___/_____

Assinatura: _____

Agradecimentos

No final deste percurso, não queria deixar de agradecer a todas as pessoas que de alguma forma me ajudaram, acompanharam e aconselharam.

Em primeiro lugar, aos meus pais que sem a ajuda deles nada disto seria possível, por todo o apoio e sacrifício que fizeram em prol do meu sucesso, por todos os bons conselhos que me deram e por sempre me apoiarem nos bons e maus momentos.

À Juliana, que incondicionalmente esteve do meu lado a partilhar os bons e maus momentos, por toda a compreensão e encorajamento e pelo tempo que não pude estar ao lado dela.

Aos meus avós, que me acolheram durante este percurso académico e que, desde sempre, se preocuparam com o meu bem-estar.

A todos os amigos e colegas que fizeram deste percurso uma experiencia inesquecível com muitos bons momentos.

Ao Professor Doutor Sérgio Adriano Fernandes Lopes pela orientação e pelos momentos em que foi incansável no suporte á realização desta dissertação.

Por fim, gostaria de dedicar esta dissertação a todos que de uma forma ou outra contribuíram para que este objetivo fosse possível.

Resumo

A normalização da comunicação com dispositivos de vigilância IP conseguida através do protocolo ONVIF (Open Network Video Interface Forum), levou que fossem criadas novas formas de interagir com estes dispositivos. Para facilitar a comunicação com estes dispositivos abstraindo a comunicação SOAP utilizada nativamente pelo protocolo ONVIF surgiu uma biblioteca escrita em C, chamada UMOC.

Suportado pela biblioteca UMOC foi desenvolvido um servidor web que disponibiliza uma interface REST, introduzindo uma nova filosofia na utilização do protocolo ONVIF.

Uma vez que nos últimos anos temos assistido a um crescimento das plataformas móveis, surgiu a necessidade de criar uma aplicação para Android que permitisse que fosse capaz de fazer a gestão de perfis de Media no contexto do protocolo ONVIF.

Com isto passaram a estar disponíveis várias alternativas para a comunicação com as câmaras, uma delas por comunicação direta com os dispositivos, através de uma JNI (Java Native Interface) para a UMOC ou através da utilização de uma biblioteca que permite a realizar pedidos SOAP, a outra alternativa seria utilizar o servidor REST intermediando as comunicações entre as câmaras e o Android.

Esta dissertação analisa estas alternativas, para isso foi criado um ambiente de testes que visa isolar os vários intervenientes nas comunicações ONVIF numa rede estanque e uma aplicação de testes na qual foram implementadas duas operações ONVIF nos três modos de comunicação mencionados.

Para a realização dos testes foram estabelecidos dois limites para os testes a realizar, um que define um período de quebra da carga da bateria do telemóvel e mede o número de pedidos realizados durante esse período, e outro define um número fixo de pedidos e mede o tempo necessário para esse valor ser alcançado, sendo que assim conseguiu-se avaliar o desempenho e a eficiência energética proporcionada pelos diferentes modos de comunicação.

Os resultados obtidos dos testes conduziram á utilização da JNI para UMOC na aplicação final de gestão de perfis de Media ONVIF. Os perfis de Media são compostos por diversas entidades que possuem vários parâmetros que podem ser configurados através da aplicação. Na aplicação foi utilizada persistência para que o utilizador possa guardar várias câmaras numa base de dados, para além de ter acesso a funcionalidades de gestão de perfis pode também aceder aos diferentes *streams* de vídeo gerados pelos diferentes perfis que o utilizador crie ou modifique.

Com o trabalho desenvolvido na fase de testes foram avaliados os diferentes modos de comunicação que para além deste trabalho podem servir de guia para outros projetos futuros no contexto ONVIF para plataformas móveis. A aplicação desenvolvida oferece uma solução visual e tecnicamente eficiente para a utilização do serviço de Media disponibilizado pelo protocolo ONVIF.

Abstract

The interoperability in surveillance IP devices brought by ONVIF (Open Network Video Interface Forum), takes us to the point where new ways to interact with this devices have been created. To abstract the SOAP native communication of ONVIF protocol a new C library named UMOC was developed.

Supported by the UMOC library a web server was developed bringing a REST interface that introduces a new philosophy to the ONVIF protocol.

In regard of the latest years mobile platforms up growth, there was a need for the development of an Android App that was capable of doing the management of Media profiles on the ONVIF context.

With that, other communication modes to the cameras are available, one is by either interact directly from devices to cameras through a JNI for UMOC or using a library the helps in the SOAP requests, the other one is by placing a REST server intermediating the communication between Android and the cameras.

This dissertation analyses this alternatives. For that, a test environment was created in order to isolate all the parties involved on ONVIF communication in a closed network and also a an application for the test was developed, where two ONVIF operations were implemented in all the three communication modes mentioned above.

For the tests two limits were established one that defines a battery charge downswing and measures the number of requests made in that period, and another that defines a fixed number of requests and measures the time needed for reaching that. Through this it's possible to evaluate the performance and energy efficiency of all the different communication modes.

The obtained results with the tests lead us to use the JNI with UMOC in the final application for management of ONVIF Media profiles. The profiles are a set of various entities that have many parameters which can be configured by the application. Persistence was implemented in order to give the user the ability to save various cameras in a database, plus the Media profiles management it can be used for watch all the streams generated by the profiles that the user creates or modifies.

With the work done in the tests phase the different communications modes were evaluated, that results are helpful not only for this work but also for futures projects in the mobile platforms ONVIF context. The application developed offers a good looking and efficient solution for using the Media service brought by ONVIF protocol.

Índice

Agradecimentos i

Resumo iii

Abstract v

Índice vii

Lista de Figuras xi

Lista de Tabelas xiv

Acrónimos xv

1. Introdução 1

1.1. Enquadramento 1

1.2. Motivação 2

1.3. Objetivos 3

1.4. Estrutura da dissertação 3

2. Fundamentos 5

2.1. Protocolo ONVIF 5

2.2. Biblioteca UMOC 7

2.3. Servidor Web Apache 7

2.4. Servidor REST ONVIF 9

2.5. Interface JNI para UMOC 10

2.6. Biblioteca ksoap2-Android 10

2.7. Ferramentas Utilizadas no Desenvolvimento 11

2.7.1. Postman 11

2.7.2. Online JSON Viewer 12
2.7.3. Ambiente de Desenvolvimento 12
2.8. Aplicações Existentes 12
2.8.1. ONVIF Client for Android 13
2.8.2. Ocular IP Camera 13
3. Comparação de Modos de Comunicação 15
3.1. Análise de Requisitos 15
3.2. Configurações de Rede 16
3.2.1. Comunicação através de um servidor REST 16
3.2.2. Comunicação direta com a câmara 17
3.3. *Deployment* do Servidor 18
3.4. Aplicação de Teste 21
3.4.1. Estrutura da aplicação 23
3.4.2. Descrição das funções de teste 24
3.4.2.1. Operação de SetSystemDateAndTime através do servidor REST 24
3.4.2.2. Operação de GetProfiles através do servidor REST 25
3.4.2.3. Operação SetDateAndTime utilizando a API JNI 26
3.4.2.4. Operação GetProfiles utilizando a API JNI 27
3.4.2.5. Operação SetSystemDateAndTime utilizando a biblioteca SOAP 28
3.4.2.6. Operação GetProfiles utilizando a biblioteca SOAP 29
3.5. Realização dos Testes e Resultados 29
4. Configuração de Perfis de Media 33
4.1. Perfil de Media ONVIF 33
4.1.1. Entidades de um perfil de Media 33

4.1.2. Serviço Media 35
4.2. Cliente Android para Gestão de Perfis de Media 36
4.2.1. Introdução 36
4.2.2. Prototipagem da User Interface 36
4.2.3. Descrição dos Componentes da Aplicação 39
4.2.3.1. Autenticação 39
4.2.3.2. Persistência 40
4.2.3.3. Correspondência entre as Activities/fragments e a UI 42
4.2.3.4. API JNI UMOC 46
5. Resultados e Discussão 49
5.1. Resultados dos Testes 49
5.2. Aplicação Final 51
5.2.1. Menu Inicial e Painel de navegação lateral 52
5.2.2. Menu de Configuração dos Media Profiles de uma Câmara 53
5.2.3. Activities de Configuração das entidades 54
5.2.3.1. Video Configurations 54
5.2.3.2. Audio Configurations 55
5.2.3.3. Metadata Configurations 56
5.2.3.4. PTZ Configurations 57
5.2.3.5. Stream Video 57
5.2.4. Notas sobre a aplicação final 58
6. Conclusão 59
6.1. Conclusões 59
6.2. Trabalho Futuro 60

7. Bibliografia 63

Lista de Figuras

Figura 1 – Estrutura de ficheiros e diretórios do Apache2 8

Figura 2 – Diagrama Temporal de uma operação ONVIF via REST 10

Figura 3 – Enquadramento da JNI entre aplicação java e biblioteca C 11

Figura 4 – Menu Principal da Aplicação ONVIF Client 14

Figura 5 – Menu de Configuração do Encoder da câmara 14

Figura 6 – Screenshot Ocular IP Camera 14

Figura 7 – Diferenças de saltos entre configuração REST e comunicação Direta 17

Figura 8 – Configuração de rede ajustada para modos de comunicação direta 17

Figura 9 – Diagrama de Classes da aplicação de Testes 22

Figura 10 – Plano de execução simplificado da aplicação de testes 23

Figura 11 – Formato do JSON Abstrato 24

Figura 12 – Construção JSON para setDateTime através API REST 24

Figura 13 – Estabelecer Conexão HTTPS com Servidor REST 25

Figura 14 – Receção Resposta do Servidor REST 25

Figura 15 – Parsing do Objecto Multicast JSON e criação de objeto JAVA Multicast 26

Figura 16 – Instanciar objeto “datetime” e envio para câmara através da JNI 26

Figura 17 – Método JNI para operação setDateAndTime 27

Figura 18 – Declaração de Media Profile (JNI) e Método Java para getProfiles através da JNI 27

Figura 19 – Aceder aos dados guardados em UMOC via JNI e criar objeto JAVA 27

Figura 20 – Definição de envelope SOAP e Construção do cabeçalho do pedido 28

Figura 21 – Construção do Nó “year” e inclusão no envelope SOAP 28

Figura 22 – Parsing XML recebido na operação ONVIF GetProfiles 29

Figura 23 – Perfil de Media Completo 34
Figura 24 – Protótipo Menu Inicial 37
Figura 25 – Protótipo Menu Configurações Perfil de Media 37
Figura 26 – Navigation Drawer 38
Figura 27 – Protótipo dos Dialogs 38
Figura 28 – Protótipo Menu Configuração de uma entidade de um Perfil 39
Figura 29 – Modelo de Base de Dados utilizado na aplicação 41
Figura 30 – Fluxograma processo de recuperação do estado da aplicação 42
Figura 31 – Integração entre Activities/Fragments na Interface Gráfica 43
Figura 32 – Exemplos Layout's template utilizados nas Activities das entidades Media Profile 45
Figura 33 – Criar um perfil novo através da cópia de um existente 46
Figura 34 – Diagrama do processo de alteração de um perfil de media 47
Figura 35 – Resultados do teste de eficiência 49
Figura 36 – Resultados do teste de Performance 50
Figura 37 – Arquitetura de Software da Aplicação 51
Figura 38 – Menu Inicial da Aplicação 52
Figura 39 – Dialog de introdução de nova câmara 52
Figura 40 – Painel de Navegação Lateral 53
Figura 41 – Dialog para apagar Câmara 53
Figura 42 – Menu de Configuração dos Media Profiles 54
Figura 43 – Activity Video Configuration 56
Figura 44 – Activity Audio Configuration 56
Figura 45 – Activity Metadata Configurations 57
Figura 46 – Activity PTZ Configuration 57

Figura 47 – Activity de Stream Video.. 58

Lista de Tabelas

Tabela 1 – Resultados dos testes com limite pela diferença da carga da bateria 30

Tabela 2 – Resultados dos testes com limite de 5000 operações 31

Acrónimos

API	Application Programming Interface
CGI	Common Gateway Interface
UI	User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Ambiente de desenvolvimento integrado
IP	Internet Protocol
JNI	Java Native Interface
JSON	JavaScript Object Notation
NVT	Network Video Transmitter
NVR	Network Video Recorder
NVD	Network Video Display
NVS	Network Video Storage
NVA	Network Video Analytics
ONVIF	Open Network Video Interface Forum
PTZ	Pan, Titl and Zoom
REST	Representational State Transfer
RTSP	Real Time Streaming Protocol
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
XML	eXtensible Markup Language

1. Introdução

1.1. Enquadramento

Hoje em dia, estamos a viver uma evolução tecnológica tão rápida e abrangente que se não houvesse uma uniformidade entre os vários produtores tecnológicos no que se refere às comunicações, iriamos ficar limitados a produtos que são construídos apenas por um determinado produtor. No entanto, para quase todos os sistemas de comunicações surgem protocolos que permitem a integração e homogeneidade entre diferentes produtos produzidos por diversos fabricantes de tecnologia. Foi isto que a norma Open Network Video Interface Forum (ONVIF) [1] trouxe para o caso das câmaras IP. Desta forma, todas as câmaras que suportem ONVIF utilizam as mesmas interfaces de comunicação e de operação independentemente do fabricante. O fórum ONVIF teve três grandes objetivos na sua criação que foram a uniformização da comunicação entre equipamentos de vídeo em rede, a interoperabilidade entre os vários equipamentos independentemente do seu produtor e, por fim, a abertura a todas as companhias e organizações (independentemente do seu tamanho).

A norma ONVIF baseia-se em serviços SOAP e é constituída por uma especificação base ("core") comum a todos os tipos de equipamento e na qual são englobadas áreas funcionais como a descoberta de dispositivos, gestão de dispositivos e a geração de eventos. Além disso, a norma especifica muitos mais serviços que podem ou não estar presentes dependendo do tipo de equipamento em questão. Os tipos de dispositivos definidos na ONVIF são os Network Video Display (NVD) [2], Network Video Analytics (NVA) [3], Network Video Transmitter (NVT) [4] e Network Video Storage (NVS) [5]. Nestes tipos de dispositivos facilmente integramos as câmaras nos NVT, e para este caso podemos trabalhar com especificações de serviço que englobam para além da base, Media, PTZ (Pan-Tilt and Zoom), Imaging, Analytics, Streaming e Device IO.

Existem atualmente algumas aplicações Android que suportam ONVIF. Estas aplicações cliente comunicam diretamente com as câmaras utilizando o protocolo SOAP, o que torna o processamento pesado, sobretudo devido à sobrecarga causada pela serialização de dados XML. Uma dessas aplicações recorre a uma biblioteca em C que é utilizada via JNI para aliviar a referida carga de processamento inerente às comunicações. A biblioteca possui uma dimensão

considerável (9 Mb de código objeto), visto que estamos a falar de um ambiente móvel (Android), e não suporta a configuração de perfis de Media.

Por outro lado, existe também uma aplicação web que integra um serviço que fornece uma API REST [6] para as funcionalidades NVT. Esta API simplifica as operações ONVIF e permite dado o nível de abstração conseguido com o servidor REST automatizar algumas operações como por exemplo a gestão de perfis Media. A API REST utiliza um formato de dados mais simples que o XML, o JSON.

1.2. Motivação

Tanto a biblioteca C e respetiva camada JNI como o servidor REST constituem oportunidades para desenvolver uma aplicação com um peso computacional mais reduzido na comunicação com câmaras ONVIF. Note-se que a utilização de JNI prevê–se vantajosa, pois a maior parte do processamento de comunicação é realizado por código nativo C em vez de Java. No entanto, existe *overhead* na camada JNI e essa previsível vantagem será apenas computacional, pois não reduz o volume de tráfego de rede.

A aplicação que utiliza a biblioteca C não inclui a importante funcionalidade de configuração de perfis de Media. Os perfis de Media controlam os dados multimédia que as câmaras disponibilizam como o *streaming* de vídeo, e por isso são centrais para qualquer aplicação ou sistema baseado em câmaras de vídeo. Além disso, esta funcionalidade é de longe a mais complexa nos dispositivos NVT, quer pela quantidade de entidades e parâmetros envolvidos, quer pelos conceitos e relações com outros serviços como Events, PTZ e Analytics.

O servidor REST NVT dá a oportunidade de libertar a aplicação de todo o processamento da comunicação SOAP e do processamento direto dos dados ONVIF e assim tornar a aplicação Android mais eficiente. O servidor está implementado como um FastCGI [7], que utiliza a biblioteca C e, por isso, oferece eficiência no processamento da comunicação. A utilização de JSON em vez do XML do protocolo SOAP/ONVIF permite que haja uma maior simplicidade na interpretação dos dados, bem como, um *overhead* significativamente mais baixo. Assim, é possível que este modo

de comunicação tenha um menor consumo energético, que é importante para dispositivos móveis com recursos energéticos limitados.

1.3. Objetivos

Esta dissertação tem como objetivo principal desenvolver uma solução que permita fazer a gestão de perfis de Media ONVIF. Além disso, pretende-se fazer uma análise aos diferentes modos de comunicação com as câmaras, quer utilizando um servidor REST a intermediar as comunicações quer utilizando bibliotecas JAVA ou C (via JNI) que permitem facilitar a comunicação com as câmaras. Esta análise vai ser feita através de testes de desempenho computacional e de consumo de bateria.

Deste modo, poderemos apresentar resultados que vão permitir fazer uma decisão mais acertada sobre qual o modo de comunicação que apresenta mais benefícios, e utilizar essa arquitetura no desenvolvimento da aplicação final. Pretende-se que ofereça uma solução prática e funcional para gestão de perfis de Media, abrangendo todos os componentes que fazem parte do complexo serviço de Media incluindo o PTZ.

1.4. Estrutura da dissertação

Neste capítulo 1 foi realizado um enquadramento ao tema abordado na dissertação, passando pela motivação para a sua realização e a descrição dos objetivos a atingir no final deste trabalho.

No capítulo 2 é feita uma abordagem sobre as tecnologias que foram utilizadas ao longo da realização deste trabalho, sendo explicados também conceitos importantes que são extensamente abordados ao longo da dissertação. Também é realizada uma revisão sobre soluções semelhantes à que se pretende realizar, existentes no mercado. As ferramentas que foram utilizadas quer no âmbito de implementação quer no âmbito de teste são listadas neste capítulo.

No capítulo 3 (Comparação dos modos de comunicação), é apresentado o ambiente de testes, as diferentes soluções existentes para a comunicação com as câmaras ONVIF. São explicados os testes realizados e como foram implementados, e por fim, são expostos os respetivos resultados.

No capítulo 4 (Configuração de Perfis de Media), inicialmente é feita uma exposição ao conceito perfil de Media no contexto ONVIF. Na segunda parte do capítulo é descrito o trabalho realizado para a estruturação e implementação de uma aplicação Android para a gestão de perfis de Media.

Ao longo do capítulo 5 são analisados os resultados que foram obtidos pelos testes realizados na parte inicial do trabalho, é também feita uma apresentação gráfica da aplicação que foi contruída para a gestão dos perfis de Media.

No capítulo 6 são feitas conclusões retiradas da realização deste trabalho, é discutido o produto final da realização deste trabalho, bem como, problemas que foram encontrados durante a implementação ou a realização dos testes. São lançadas também algumas propostas para trabalhos a realizar futuramente.

2. Fundamentos

Neste capítulo são analisados conteúdos que foram importantes entender antes e durante a realização desta dissertação. São identificadas algumas aplicações que utilizam também o protocolo ONVIF na troca de dados, e por fim vão ser expostas algumas ferramentas que foram utilizadas em procedimentos, quer de estudo quer de implementação, ao longo desta dissertação.

2.1. Protocolo ONVIF

O surgir de câmaras e outros dispositivos de gestão a operar sobre o protocolo IP foi um enorme passo para esta área de desenvolvimento, no entanto, numa fase inicial todos os grandes fabricantes estavam a utilizar interfaces proprietárias para a transmissão de dados entre este tipo de dispositivos, o que cria uma enorme fragmentação neste mercado e inviabiliza a interação entre dispositivos que não são do mesmo fabricante. O ONVIF constitui uma solução para este problema e foi desenvolvido inicialmente por três grandes fabricantes a operar neste mercado e agora é suportado por mais de quatrocentas companhias que atuam neste mercado [8].

O protocolo ONVIF [1] encontra-se dividido em vários serviços que os dispositivos podem ou não disponibilizar, nomeadamente:

- Core
- Action Engine
- Device IO
- Display
- Imaging
- Media
- PTZ
- Receiver
- Recording Control
- Recording Search
- Replay Control
- Video Analytics
- Video Analytics Device

Como o protocolo define também diferentes tipos de dispositivos, nem todos estes serviços necessitam de estar implementados em cada equipamento. No entanto, qualquer dispositivo que queiria estar de acordo com a norma ONVIF necessita de implementar as funcionalidades contempladas pela "Core Specification", que engloba funções de descoberta, gestão do dispositivo e um serviço de eventos.

Os diferentes dispositivos que se encontram definidos na norma ONVIF são:

- NVD – Network Video Display [2], dispositivo capaz de receber conteúdo pela rede IP e expô-lo, por exemplo, um monitor que recebe conteúdo através da rede;
- NVS – Network Video Storage [5], dispositivo capaz de receber conteúdo de uma câmara por exemplo e gravar num espaço de armazenamento;
- NVA – Network Video Analytics [3], dispositivo capaz de analisar os dados e metadados recebidos de uma câmara IP;
- NVT – Network Video Transmitter [4], dispositivo capaz de enviar conteúdos media através da rede IP, nomeadamente uma câmara.

Para cada um destes dispositivos o protocolo define os serviços que são mandatórios (M), Opcionais (O), e os que são obrigatórios por relação com uma determinada característica (C). Para os dispositivos NVT (que são os abordados neste trabalho), temos, M = {Device Management, Events, Media [9], Streaming, Device IO}, C = {PTZ} e O = {Imaging,Video Analytics}.

Para a comunicação com outros dispositivos o ONVIF define a utilização de SOAP, um protocolo que utiliza para a transmissão de mensagens o HTTP(S), e o formato de dados utilizado é o XML. Cada pedido SOAP [10] é constituído por um envelope que possui um cabeçalho e um corpo. É no cabeçalho que é indicada a ação sobre o dispositivo (i.e. a operação a realizar) e as credenciais do utilizador. No corpo do envelope, em pedidos do tipo SET são enviados os dados no formato pré-definido para cada operação ONVIF, e em pedidos do tipo GET o corpo não tem qualquer informação.

Todas as operações de um serviço que os dispositivos ONVIF disponibilizam são definidas num ficheiros WSDL que estabelecem o formato dos dados que deve ser utilizado.

2.2. Biblioteca UMOC

A biblioteca UMOC [11] foi criada para facilitar o desenvolvimento de clientes NVT. Esta biblioteca proporciona não só os procedimentos de rede para a comunicação com as câmaras, uma abstração ao processamento XML inerente às comunicações SOAP do protocolo ONVIF, e a gestão da memória de todas as estruturas que alocam a informação relativa às operações com os dispositivos, Permite assim um desenvolvimento mais rápido e focado na implementação de aplicações-cliente e não no processo inerente à interação com os dispositivos ONVIF.

Como a biblioteca UMOC é implementada em C portável entre as principais plataformas (Win32, POSIX, Android, etc.) tem a vantagem de poder ser executada em quase todos os sistemas computacionais, permitindo assim a criação de aplicações variadas. Além disso, como foi implementada numa linguagem não-interpretada permite um bom desempenho e uma gestão de memória muito eficiente.

2.3. Servidor Web Apache

Como a realização deste trabalho envolve a utilização de um serviço REST implementado por um FastCGI, para que a API esteja acessível pela rede é necessário que seja exposta através de um servidor Web. Para resolver este problema existem inúmeras soluções no mercado, no entanto, a escolha recaiu sobre o servidor Web mais utilizado no mundo, o Apache [12].

Antes de ser feito o *deployment* do serviço REST com o Apache é necessário conhecer como este servidor funciona e como esta estruturado.

A Figura 1 é uma representação da estrutura de ficheiros do Apache e de como se encontra organizado [13].

No ficheiro ports.conf, encontram-se as configurações das portas TCP em que o Apache se encontra à escuta de mensagens HTTP(S). Este ficheiro permite que sejam habilitadas e configuradas as portas de modo a que os pedidos recebidos sejam direcionados para os sites corretos.

O ficheiro httpd.conf costumava ser o principal ficheiro de configuração do apache2, no entanto, atualmente foi desabilitado e todas as principais configurações foram movidas para os diretórios

e ficheiros abordados nesta secção. Atualmente o referido ficheiro pode existir mas usualmente não contem qualquer informação.

O ficheiro *apache2.conf* é o principal ficheiro de configuração do apache, aonde são definidos parâmetros como o *timeout* de um *request* (configuração global do Servidor Web), o número de clientes que o servidor pode aceitar em simultâneo. Neste ficheiro são incluídos todos os ficheiros de configuração de sites e módulos que estão nos diretórios sites-enabled e mods-enabled. Portanto, só os sites e módulos colocados nesses diretórios é que são carregados e ficam ativos.

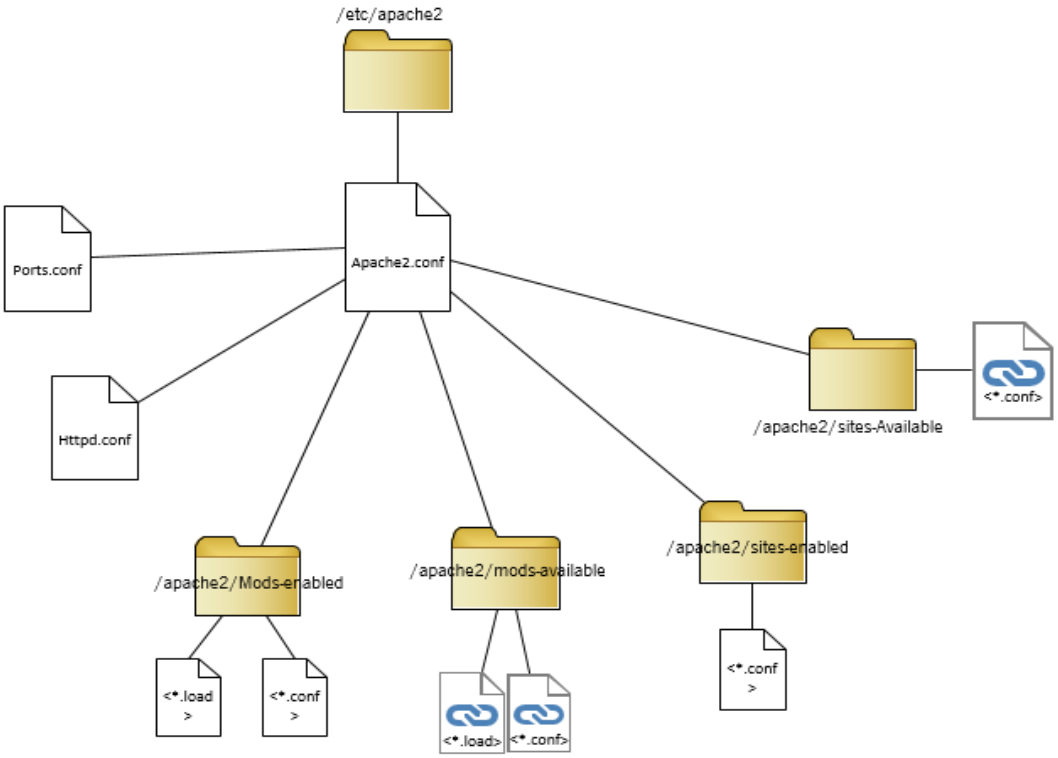


Figura 1 – Estrutura de ficheiros e diretórios do Apache2

As diretorias *mods-enabled* e *mods-available*, contêm os módulos e *add-ons* para o servidor Apache. Todos os módulos que vêm instalados por defeito e são posteriormente adicionados encontram-se no *mods-available*, mas isso não significa que são utilizados. Para que o Apache carregue um módulo, tem de ser executada o comando a2enmod para que o Apache o execute como um módulo. O comando cria um *symlink* na diretoria mods-enabled para os ficheiros de configuração (*<nome_do_mod>.conf* e *<nome_do_mod>.load*) do módulo em questão.

Em *sites-available*, encontram-se todos os ficheiros de configuração de sites disponíveis, mas não activos. É necessário executar o comando a2ensite para criar *symlinks* para os ficheiros de

configuração. Depois, sempre que o Apache é reiniciado, todos os ficheiros para os quais os *symlinks* apontam, são interpretados por pré-definição, e os sites ficam ativos (i.e., são disponibilizados).

2.4. Servidor REST ONVIF

Com base na biblioteca UMOC, foi desenvolvida uma aplicação-servidora que oferece uma API REST para a comunicação com dispositivos ONVIF. O serviço REST [14] é implementado através da utilização de uma Fast Common Gateway Interface (FastCGI) [7] à qual o Apache delega os pedidos HTTP com determinada URI. O tipo de dados utilizado na comunicação entre o cliente e o servidor é o JSON, caracterizado por ser um formato que exige, em termos computacionais, muito menos do que o XML/SOAP no processo de (des)serialização dos dados. Dado a este facto, pode afirmar-se que é mais vantajoso utilizar este tipo de interface em implementações a realizar em dispositivos móveis que dispõem de recursos energéticos e computacionais limitados.

Utilizar um servidor REST para a comunicação com câmaras ONVIF veio trazer um outro nível de abstracção que não é possível apenas com a utilização da biblioteca UMOC. A utilização de um servidor deste tipo permite que qualquer dispositivo capaz de realizar pedidos HTTP, seja capaz de interagir com dispositivos ONVIF. Com a utilização deste servidor é retirada uma grande carga de processamento inerente às operações ONVIF, visto que este utiliza um formato de dados JSON em detrimento do XML associado ao protocolo SOAP.

Para entender a realização de uma operação ONVIF utilizando a API REST é necessário conhecer um dos princípios básicos da filosofia REST. Esta define uma sintaxe universal para identificação de recursos, isto é, cada recurso disponibilizado pelo servidor é unicamente identificado através da sua URI [6]. Adicionando ao recurso a informação do método HTTP utilizado no pedido, que corresponde ao verbo REST, o servidor consegue então fazer corresponder o pedido a uma única operação ONVIF e, consequentemente, identifica a função da biblioteca UMOC que tem que invocar.

O início de uma operação ONVIF através de interface REST (Figura 2) é sempre despoletado por um cliente que faz um pedido HTTP, utilizando o método+URI correspondente à operação a realizar. No cabeçalho dos pedidos são enviadas as credenciais de acesso à câmara

correspondente. A URI contém também o IP da câmara à qual se destina o pedido. Estes dados são argumentos das funções da biblioteca UMOC, que por sua vez realizam o processo de comunicação com o dispositivo. Recebendo a resposta da biblioteca (que por sua vez recebeu a resposta da câmara) o servidor traduz a informação para JSON e envia-a para o cliente na resposta HTTP.

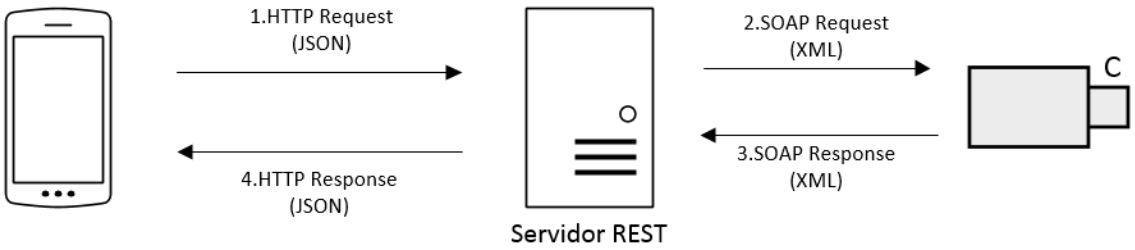


Figura 2 – Diagrama Temporal de uma operação ONVIF via REST

2.5. Interface JNI para UMOC

Com o crescimento rápido do número de dispositivos com o sistema operativo Android, surgiu a necessidade de desenvolver aplicações ONVIF para essa plataforma. Aproveitando mais uma vez a biblioteca UMOC, foi desenvolvida uma interface JNI que permitisse correr código nativo em aplicações Java.

Esta interface permite assim que sejam utilizadas todas as funcionalidades que a biblioteca UMOC disponibiliza utilizando uma API que foi gerada através de uma ferramenta chamada SWIG [15], que gera o código responsável pela tradução do código interpretado Java em código nativo.

Como é apenas uma interface de comunicação com a biblioteca (Figura 3), todas as estruturas de dados implementadas no C são mantidas e apenas acedidas por classes que foram geradas pela ferramenta SWIG e simulam a sua existência em Java. Essas classes são apenas proxies para as estruturas existentes na biblioteca C que armazenam os dados recolhidos dos dispositivos ONVIF em determinada operação. Ou seja, os dados permanecem do lado C.

2.6. Biblioteca ksoap2-Android

Para ser utilizado o protocolo SOAP no Android apenas temos duas opções, visto que a Google nunca implementou nenhuma biblioteca para este protocolo (por questões de eficiência e de incentivo ao uso de serviços baseados em REST), uma delas é implementar um cliente SOAP

próprio que gera manualmente os envelopes SOAP através da (des)serialização de XML e os envia através do cliente HTTP nativo do Android, ou então utilizar uma biblioteca que internamente realize este processo de geração do envelope, e forneça uma API mais intuitiva.

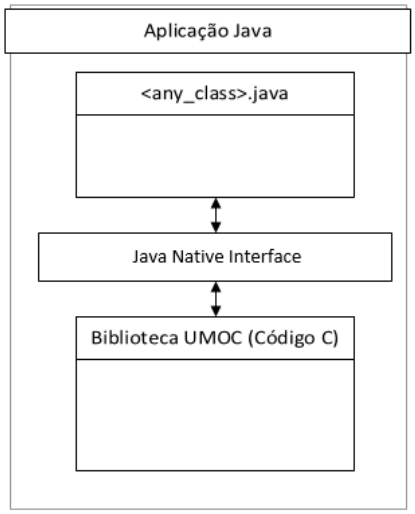


Figura 3 – Enquadramento da JNI entre aplicação java e biblioteca C

Apesar de, através da biblioteca ksoap2-Android [16], não ser necessário fazer a implementação manual do XML, é necessário conhecer bem todos os elementos (e a sua hierarquia) que tem de ser incluídos no envelope para que o servidor consiga interpretar corretamente os pedidos, visto que esta ferramenta não faz geração de *stubs* WSDL, apenas facilita a criação/interpretação de envelopes SOAP e resolve a comunicação entre as partes.

2.7. Ferramentas Utilizadas no Desenvolvimento

Nesta secção são descritas as ferramentas utilizadas na fase de desenvolvimento deste projeto.

2.7.1. Postman

Postman [17] é um excelente cliente para serviços WEB, que funciona como uma extensão para o navegador web Chrome. O Postman permite todos os tipos de autenticação que se utilizam atualmente em servidores WEB (Basic Auth, Digest Auth e OAuth1.0 [18]), e permite configurar os parâmetros dos cabeçalhos dos pedidos e utilizar todos os tipos de métodos HTTP.

Quando se implementa aplicações que consomem serviços REST, principalmente num ambiente de desenvolvimento, é muito importante que se tenham ferramentas que nos permitam fazer a triagem dos erros do lado do servidor ou nas comunicações entre as partes. Através desta

ferramenta é possível recriar os pedidos exatamente como eles serão feitos através duma aplicação Android ou outro cliente qualquer.

Durante o *deploy* do servidor e a implementação da aplicação, esta foi uma ferramenta indispensável não só para efeitos de *troubleshooting*, bem como, para se obter rapidamente os *dumps* das várias respostas do servidor em JSON para a implementação do parsing.

2.7.2. Online JSON Viewer

Para facilitar a interpretação dos dados das respostas do servidor durante o processo de implementação, foi muito utilizada a aplicação web "ONLINE JSON Viewer" [19]. Esta interpreta *strings* JSON e coloca-as numa forma mais legível para um humano, fazendo, por exemplo, a organização hierárquica dos vários componentes de uma estrutura JSON.

2.7.3. Ambiente de Desenvolvimento

Para o desenvolvimento de aplicações Android é necessário um *IDE* que integre o *Android Software Development Kit* (SDK). Numa fase inicial do sistema operativo isto era conseguido com a integração deste pacote no Eclipse. Contudo no final de 2014 a *Google* lançou o ambiente de desenvolvimento integrado oficial do sistema operativo *Android*, com o nome *Android Studio* [20].

Este *IDE* é muito completo e possui já todas as bibliotecas, compiladores e emuladores necessários para o desenvolvimento de aplicações para este SO, e como é oficialmente suportado pela empresa detentora do sistema operativo alvo é rapidamente atualizado com todas as últimas (versões) bibliotecas lançadas.

2.8. Aplicações Existentes

Durante este trabalho foram pesquisadas, principalmente através da loja oficial de aplicações Google Play, aplicações para Android que permitam interagir com dispositivos ONVIF. Nesta secção apresentam-se duas, uma por ser mais relevante para o contexto deste trabalho e outra a título de exemplo.

2.8.1. ONVIF Client for Android

Através do website da aplicação é possível observar que a aplicação [21] utiliza uma biblioteca C na implementação das operações ONVIF o que pode indicar uma performance interessante. No entanto, como não temos acesso ao código é impossível realizar testes de eficiência/performance e comparar com as soluções que temos implementado para esta dissertação.

Como se pode ver na Figura 4, no ecrã inicial da aplicação, nota-se logo á partida uma interface com o utilizador que não é adequada para dispositivos móveis dado o tamanho pequeno de todos os componentes gráficos, tornando a utilização da aplicação difícil para selecionar determinadas caixas de texto e outros componentes.

Para além disso, esta aplicação não implementa nenhuma funcionalidade de persistência de dados, logo sempre que a aplicação é terminada todos os dados inseridos são perdidos e é necessário introduzi-los novamente a cada utilização. Outro aspeto relevante de implementação é o facto de esta aplicação utilizar em todas as operações de Media o primeiro perfil que encontra na câmara, sem possibilitar que as alterações realizadas incidam sobre outro perfil. Em termos de gestão de perfis, a aplicação apenas permite alterar os parâmetros do *Video Encoder* que se podem ver na Figura 5.

2.8.2. Ocular IP Camera

A aplicação *Ocular IP Camera* [22] apresenta uma particularidade interessante, em vez de um cliente ONVIF esta aplicação implementa o protocolo para câmara do telemóvel e transforma-a num NVT, permitindo que a câmara do telemóvel seja acedida por IP, como se de uma câmara ONVIF se tratasse. No entanto, depois de alguns testes a aplicação mostrou-se muito difícil de conseguir para aceder via rede diretamente ao IP e reproduzir o stream RTSP que disponibiliza. A única vez que foi conseguido estabelecer a ligação á camara do SmartPhone foi através do serviço de *cloud* que o programa disponibiliza, ao qual acedemos através de um browser e conseguimos visualizar o *stream* através de uma autenticação com utilizador e password.

Na Figura 6 podemos observar a única interface que a aplicação disponibiliza.

Em termos de configurações ONVIF, a única possibilidade disponível através da aplicação é alterar a resolução, qualidade e os frames por segundo (fps) do video.

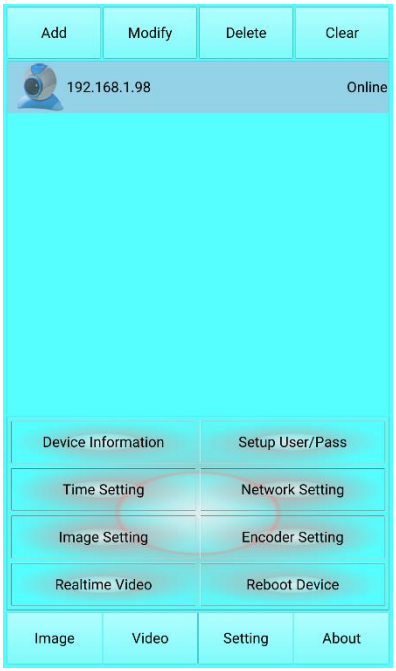


Figura 4 – Menu Principal da Aplicação ONVIF Client

Figura 5 – Menu de Configuração do Encoder da câmara

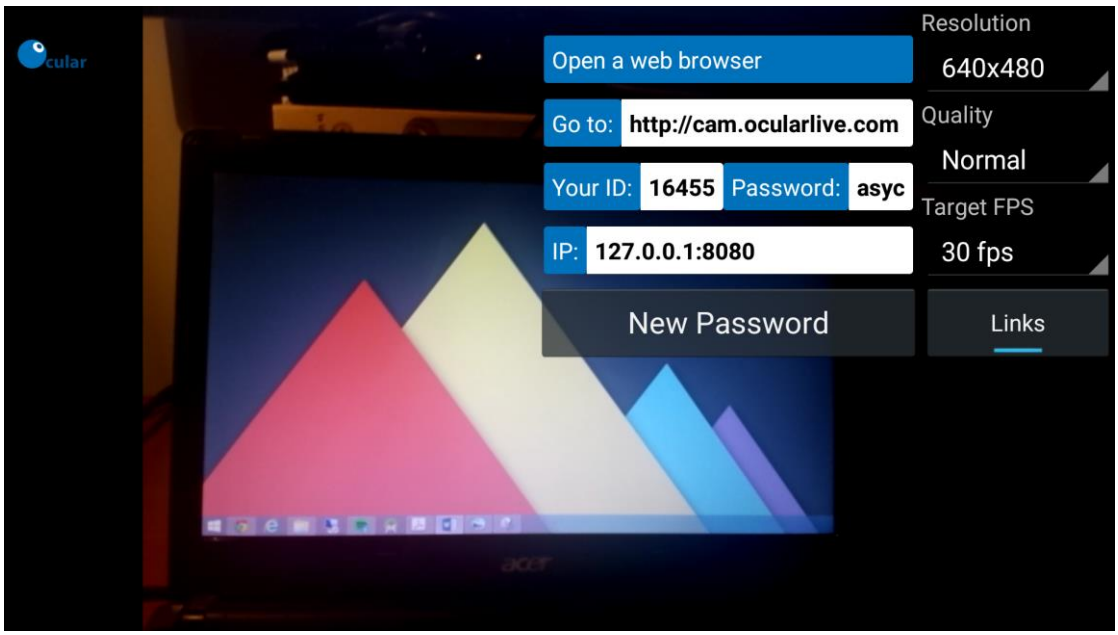


Figura 6 – Screenshot Ocular IP Camera

3. Comparação de Modos de Comunicação

Neste capítulo vão ser testados e avaliados os diferentes modos de comunicação que são passiveis de ser utilizados para comunicar com câmaras ONVIF através de um dispositivo Android, durante este capítulo vão ser então analisados os seguintes modos de comunicação:

1. Utilizar um Servidor REST [14] a intermediar a comunicação com as câmaras
2. Comunicação através de uma JNI para a biblioteca UMOC [11]
3. Utilizar uma biblioteca SOAP para Android [16]

Para perceber qual a melhor destas soluções, vão ser implementados ao longo do capítulo diferentes testes de eficiência energética e de performance para que no final se consiga fazer a escolha mais acertada no modo a utilizar na implementação de uma aplicação de gestão de perfis de Media.

3.1. Análise de Requisitos

A realização dos testes tem como objetivo analisar a performance e eficiência energética de cada modo de comunicação, para isso vão ser realizados em cada modo duas operações ONVIF diferentes para cada um dos testes, *setSystemDateAndTime*() e *getProfiles*(). Foram escolhidos estas duas operações pois permite avaliarmos o comportamento em duas operações que obrigam a um processamento distinto visto que uma é uma operação do tipo "set" e outra do tipo "get". Tendo em conta os vários modos de comunicação em teste e escolhidas as operações que vão servir de teste é necessário definir como se vai aferir a performance e eficiência energética deles. Para aferir a eficiência energética o teste passa por definir um intervalo de queda de carga da bateria (neste caso 3%), período este durante o qual a aplicação de testes se vai encontrar continuamente a realizar uma determinada operação ONVIF, no final é avaliado qual o modo de comunicação que conseguiu realizar mais operações neste período de quebra da carga da bateria, para a mesma operação ONVIF. Para avaliar a performance de um modo de comunicação é definido para todos os testes um número fixo de operações que vão ser realizadas durante o teste (neste caso 5000 operações) e no final aferir qual dos modos precisou de menos tempo para realizar na mesma operação ONVIF este número fixo de operações.

Para que os testes sejam feitos dentro das mesmas condições a implementação dos diferentes testes e dos diferentes modos de comunicação foram todos realizados dentro da mesma aplicação que possui uma interface gráfica básica, para cada modo de comunicação foi implementado um método para cada operação ONVIF, que conforme o teste a realizar é alternada de forma "hardcoded".

Os testes foram realizados utilizando um Samsung Galaxy S4 i9505, utilizando o sistema operativo Android 5.0.1 lollipop, que utiliza a nova máquina virtual Android Runtime (ART) [23], em detrimento da máquina virtual Dalvik [23] utilizada em versões anteriores do sistema operativo.

Para que os testes garantissem credibilidade nos resultados teriam de ser realizados num ambiente de rede complemente controlado para que estes não sejam influenciados por cargas de rede não provocadas pela aplicação de teste, servidor ou câmara utilizada nos testes. Com isto garantimos uma maior fiabilidade nos testes, e conseguimos assegurar que as diferenças entre os vários modos de comunicação não são dependentes da qualidade/tipo de rede utilizada, mas sim apenas das diferenças estruturais do modo de comunicação. Para a realização do teste utilizamos uma LAN criada através de um Router. No entanto, inerente aos diferentes modos de comunicação temos duas configurações de rede relativamente distintas, uma configuração para a utilização do servidor REST e outra configuração que foi utilizada na comunicação direta com as câmaras, quer utilizando o protocolo SOAP previsto pelo ONVIF diretamente pela aplicação quer pela utilização da API JNI da biblioteca C que dispomos

3.2. Configurações de Rede

3.2.1. Comunicação através de um servidor REST

Comparativamente à comunicação direta com a câmara, utilizando um servidor REST [6], acrescenta mais quatro saltos de rede (*hops*) (Figura 7) comparativamente com os outros modos de comunicação. Para compensar isto a configuração de rede utilizada nos outros dois modos foi ajustada (ver como na secção 3.2.2), para que esses quatro saltos a mais não influenciem muito os resultados e a fiabilidade dos mesmos. Isto porque a configuração de rede para os testes visa uma quantidade de saltos relativamente reduzida, em comparação a uma utilização mais realista.

Num cenário de utilização realista por exemplo, utilizador encontra-se noutra cidade para aceder á sua câmara de casa, o número de saltos iria ser tão elevado que não se iriam notar os dois saltos a mais de uma configuração para a outra.

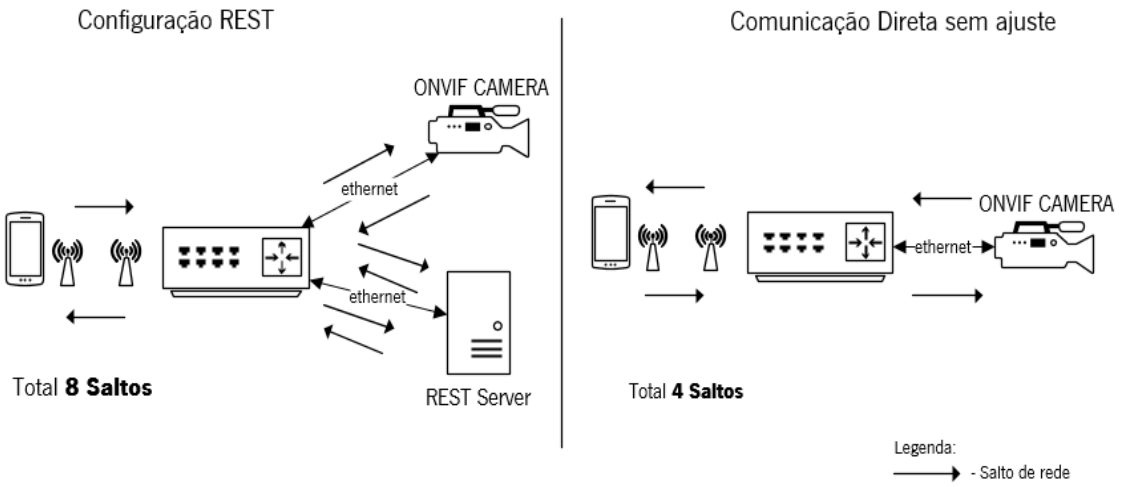


Figura 7 – Diferenças de saltos entre configuração REST e comunicação Direta

3.2.2. Comunicação direta com a câmara

Na Figura 7 pode ser visto a solução mais óbvia para a arquitetura de rede utilizada pelos modos de comunicação que não utilizam um servidor REST a intermediar as comunicações, no entanto, esta solução para o ambiente de testes que se pretende recriar pode provocar uma enorme desvantagem para a solução REST. Este problema numa utilização real não existe porque um utilizador vai utilizar o seu dispositivo através de uma Wide Area Network (WAN) em que os pedidos

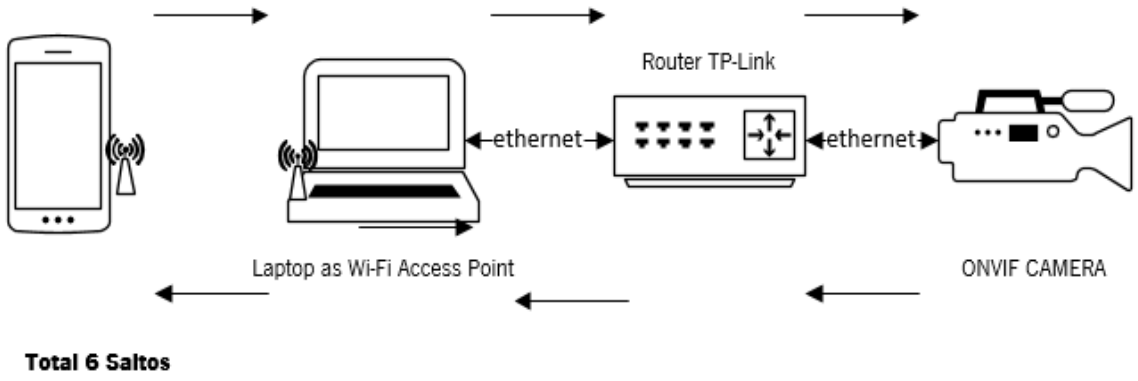


Figura 8 – Configuração de rede ajustada para modos de comunicação direta

vão realizar inúmeros saltos de rede e logo não irá ser relevante a diferença de 4 saltos entre as duas soluções de rede (REST e comunicação direta). No ambiente de testes, para equilibrar este

problema foi pensada uma solução (Figura 8) que permite adicionar dois saltos de rede e assim tornar a diferença de saltos entre as duas arquiteturas menos penosa para o REST (passamos a realizar 6 saltos na comunicação direta, contra os 8 da solução REST), isto foi conseguido deixando de conectar o dispositivo móvel ao *access point wireless* (AP) do router e adicionando um PC com Wi-Fi ligado por ethernet ao router e, com a ajuda de software existente para o efeito, o computador passa a funcionar este como o AP e conseguimos assim um maior equilibro de rede entre as duas arquiteturas.

É importante salientar também que em qualquer dos testes não existia mais nenhum dispositivo, que não os representados nas figuras, conectados ao *router*. Por sua vez, o *router* não estava conectado á Internet, sendo que a LAN era apenas composta pelos elementos ilustrados, sendo que todo o tráfego produzido na rede pertence aos dispositivos envolvidos nos testes. Assim, garante-se que não há nenhuma interferência/congestionamento na rede provocado por terceiros que pudessem afetar os resultados.

3.3. *Deployment* do Servidor

O servidor que fornece a interface REST utilizada nos testes, foi implementado através de uma aplicação FastCGI [7] em C++ que redireciona os pedidos recebidos por um servidor WEB, para a biblioteca UMOC responsável pela comunicação, serialização e (des)serialização XML dos dados e do parsing e validação das mensagens SOAP. O servidor Web escolhido para realizar o *deploy* da interface REST (como foi dito na secção 2.3) foi o Apache na sua versão 2.2.

A instalação do Apache foi realizada num computador portátil com uma instalação de raiz duma distribuição *lightweight* do ubuntu 12.04 LTS, denominada Lubuntu. Como foi explicado na secção 2.3, existem módulos que já vem instalados, no entanto, necessitam de ser ativados e outros módulos que é preciso proceder á sua instalação (como qualquer software Linux).

Para instalar o Apache no sistema operativo Linux existem várias maneiras, mas a mais simples e eficaz é recorrendo ao comando

```
sudo apt-get install apache2
```

Como a comunicação a ser utilizada entre o dispositivo móvel e o servidor será encriptada por SSL é necessário proceder á ativação do módulo do SSL para o apache através do comando

sudo a2enmod ssl

que cria o *symlink* na diretoria */mods-enabled/* para o modulo SSL que se encontra na diretoria /mods-available.

Para criarmos um certificado que vai ser utilizado pelo servidor na comunicação com os clientes foi utilizado o seguinte comando

sudo make-ssl-cert /usr/share/ssl-cert/ssleay.cnf /etc/ssl/private/restonvif.crt

Este comando vai gerar na diretoria colocada no último parâmetro um certificado do tipo *self-signed*.

Para podermos executar um *FastCGI* no servidor é necessário instalar o módulo que suporta a execução desses programas, fazendo a interligação com o Apache. Para o fazermos, executamos os seguintes comandos:

sudo apt-get install mod_fastcgi

e depois para o módulo ficar indexado na diretoria mods_enabled é necessário executar a rotina:

sudo a2enmod fastcgi

Qualquer configuração que posteriormente se queira realizar sobre o módulo do FastCGI apenas é necessário ir ao ficheiro de configuração que pode ser acedido em /etc/apache2/mods-available.

O formato de dados suportado pelo servidor é o JSON, e o FastCGI depende da biblioteca libjsoncpp. Para instalar a biblioteca utiliza-se comando:

apt-get install libjsoncpp-dev

para poderem ser utilizadas as diretivas de DELETE e PUT num script personalizado é necessário instalar um módulo que vai permitir direcionar esse tipo de operações:

Sudo apt-get install mod_actions

Por fim, é necessário configurar o "site" que vai receber os pedidos HTTP e direcioná-los para o fastcgi que gera a API REST com a ajuda da biblioteca C UMOC. Isso é feitono ficheiro umoc.conf, que é colocado na diretoria /etc/apache2/mods-available/. Os principais aspectos da configuração são: a activação do SSL e respectivo certificado, através das linhas:

```
SSLProtocol all
SSLEngine on
SSLCertificateFile /etc/ssl/private/restonvif.crt
```

indicar o script que vai escutar os pedidos direcionados pelo módulo mod_actions, com a seguinte linha:

```
Action fastcgi-script /home/helder/Servidor/umoc_rest_32bit/umoc_rest_32bit.fcgi
```

criar um *alias* que permita ao Apache redirecionar os pedidos cuja URI comece por /onvif para o script do fastcgi e indicar os tipos de pedido HTTP suportado pelo FastCGI:

```
ScriptAlias /onvif /home/helder/Servidor/umoc_rest_32bit/umoc_rest_32bit.fcgi
<Directory "/home/helder/Servidor/umoc_rest_32bit">
        AllowOverride ALL
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        #Require all granted verificar esta situação
        Script GET umoc_rest_32bit.fcgi
        Script POST umoc_rest_32bit.fcgi
        Script PUT umoc_rest_32bit.fcgi
        Script DELETE umoc_rest_32bit.fcgi
</Directory>
```

Por último é necessário executar a rotina que cria o symlink (na diretoria /etc/apache2/sites-enabled) para o ficheiro de configuração criado nos passos anteriores, através do seguinte comando

```
sudo a2ensite umoc.conf
```

Com todas as configurações realizadas, pode-se reinicializar o servidor Apache, de modo a produzirem efeito.

```
sudo service apache2 restart
```

Posto isto, podemos começar a utilizar o nosso servidor, utilizando qualquer tipo de dispositivo e software capazes de realizar pedidos HTTP.

3.4. Aplicação de Teste

Os serviços de dispositivos NVT suportam operações cujo volume de informação trocado é muito diferente, indo desde alguns kilobytes até algumas dezenas de kilobytes. Naturalmente, esta gama de volumes de informação transportada nas mensagens HTTP tem um impacto na comparação. As mensagens menores realçam as latências da comunicação e as maiores são mais imunes. Além disso, as mensagens podem distinguir-se em 2 grandes grupos: as que modificam o estado do dispositivo (i.e., realizam operações de escrita) e as que apenas consultam informação. Também estas 2 realidades devem ser objecto de comparação de forma a melhor espelharem a realidade ONVIF. Por estas razões, foram escolhidas duas operações: *SetSystemDateAndTime* e o *GetProfiles*.

Para realizar uma operação *setSystemDateAndTime* independentemente do modo comunicação usado e dado a sua natureza (operação de escrita) é necessário construir os dados que vão ser enviados para a câmara processar, obrigando a processamento "pré-request" para a construção do JSON ou XML a enviar, mas em termos de "pós-request" não obriga a esforço nenhum por parte do cliente, já no *GetProfiles* a filosofia é um pouco diferente e obriga não a um processamento antes do request a câmara, mas sim ao processamento e instanciamento "pós-request" que é realizado quando este é executado, logo assim conseguimos ter uma noção mais assertiva dos resultados e despistando resultados que possam ser influenciados não pela arquitetura de comunicação mas sim pelo processamento no cliente e no servidor.

O objetivo da implementação de uma aplicação de testes é comparar os diferentes modos de comunicação, avaliando o consumo de bateria (eficiência energética) e o desempenho de processamento entre os vários modos de comunicação. Para se fazer uma análise quer da performance quer da eficiência energética dos três modos de comunicação em teste, foram idealizados 2 tipos de teste. Para avaliar a performance o teste a realizar passa por colocar um cliente a realizar continuamente operações ONVIF até um número pré-definido de operações comum a todos os modos (neste caso 5000 operações), e no final comparar o tempo necessário para cada modo atingir esse número de operações; para avaliar a eficiência energética o teste que

se pretende realizar passa por avaliar durante variação negativa de 3% da carga da bateria quantas operações ONVIF é possível realizar e o tempo necessário para acontecer a quebra de bateria estabelecida, através dos diferentes modos de comunicação.

Dois requisitos importantes para essa aplicação são: ser minimalista, de modo a minimizar o processamento adicional ao que se pretende comparar; ter uma estrutura comum para todos os modos de comunicação, para que a comparação seja válida (difira apenas no modo de comunicação).

Tendo em conta estes requisitos, optou-se por construir apenas uma aplicação para todos os testes: 3 modos de comunicação vezes 2 duas operações, perfazendo um total de 6 cenários de teste. Cada um destes cenários constitui um método que implementa o processamento específico. Esta abordagem tem uma vantagem prática, que é permitir facilmente alternar entre os vários cenários de teste, trocando apenas o método que é executado.

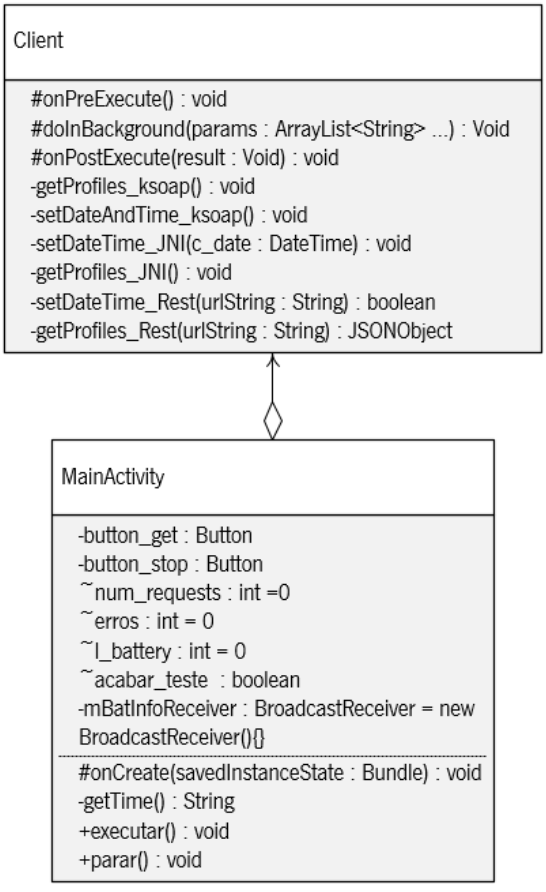


Figura 9 – Diagrama de Classes da aplicação de Testes

3.4.1. Estrutura da aplicação

A aplicação que foi implementada para a realização dos testes, para além de visualmente simples, é apenas composta por duas classes, a MainActivity e Client (Figura 9). A classe *Client* é uma *AsyncTask* [24], necessária na programação Android para realizar pedidos através da rede numa *thread* independente da *thread* de UI.

Para minimizar o *overhead* do processamento utilizando um if dentro do ciclo de teste comandado por uma opção da UI, a alternância entre os diferentes métodos de teste é realizada de forma *hardcoded*, isto é, na classe doInBackground() existem as invocações para os seis métodos de execução de um teste (seis últimos métodos da classe Client, Figura 9), por defeito todos as

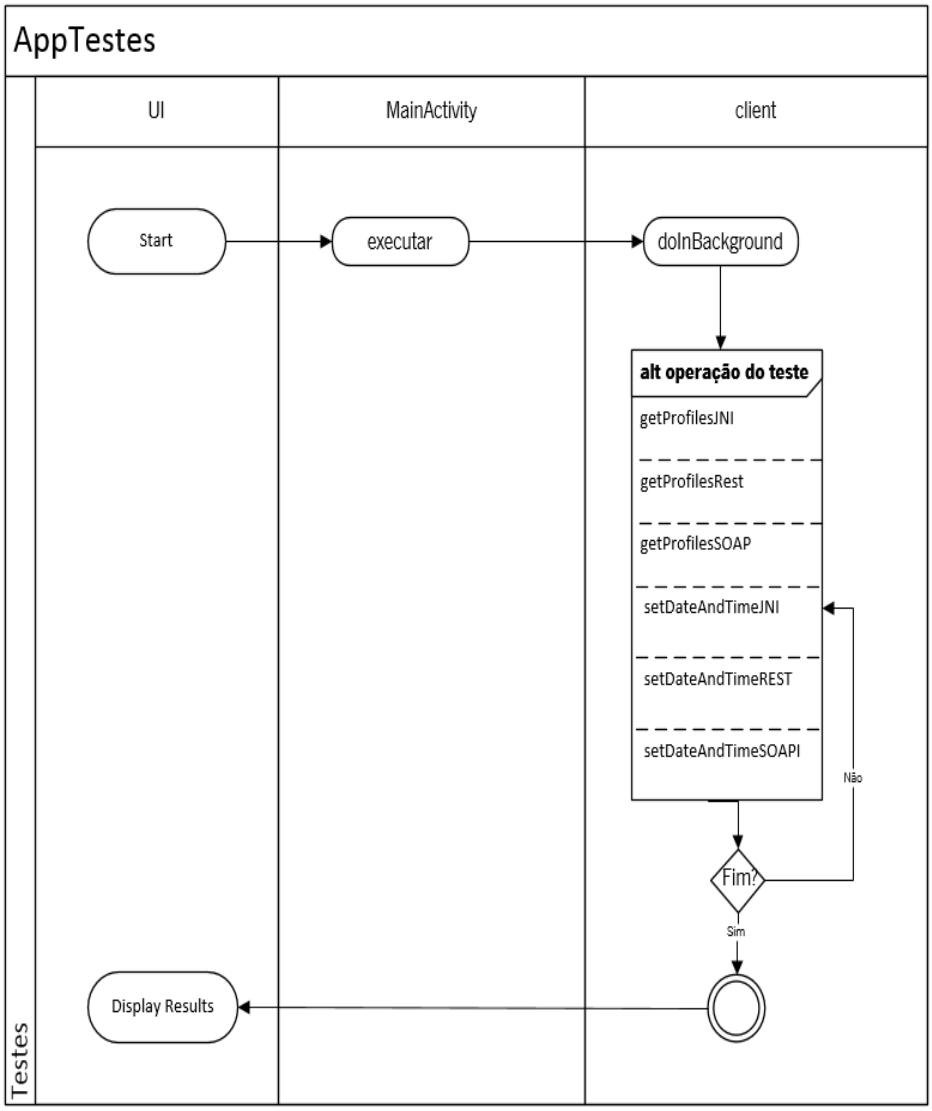


Figura 10 – Plano de execução simplificado da aplicação de testes

invocações encontram-se comentadas; conforme o teste que se pretende realizar a invocação do método correspondente é retirada de comentário e é executada no doInBackGround(). Isto permite que a aplicação se torne simples e a UI não seja um elemento influente em nenhum teste.

Como podemos ver na Figura 10, existe uma UI que apenas possui um botão que permite iniciar o teste, ao ser selecionado é executado o método executar(), este por sua vez instancia uma nova classe Client() que é uma AsyncTask e possui no método doInBackground() no qual existem várias invocações para métodos que representam diferentes testes; conforme o teste a realizar é necessário comentar todas as outras invocações e deixar a invocação do método referente ao teste que se pretende realizar "executável", no método doInbackground() vai ser continuamente invocada (visto existir um ciclo neste método) até ser atingido o valor de pedidos estabelecido ou a diferença na carga da bateria ter sido alcançada, no final do teste é enviado para a UI os resultados do teste.

3.4.2. Descrição das funções de teste

Para explicar processamento de cada teste vão ser explicadas as seis funções executadas conforme o teste a realizar.

3.4.2.1. Operação de SetSystemDateAndTime através do servidor REST

A realização de um pedido SetSystemDateAndTime para um dispositivo ONVIF utilizando a API REST é um processo bastante simples, visto que a única necessidade que temos é de construir uma estrutura de dados JSON com os dados relativos à data, hora e fuso horário. Conforme a especificação da API REST do servidor [14], esse formato é apresentado na Figura 11. O processo de construção dos dados com esse formato é listado na Figura 12.

```
No corpo do pedido:
{
  "Type": "[0/1]",
  "DayLightSaving": [0/1],
  "TimeZone": "[timezone]",
    "Day": day,
    "Month": month,
    "Year": year,
    "Hour": hour,
    "Minute": minutes,
    "Second": seconds
}
Type: (0 - Manual ; 1 - NTP)
```

Figura 11 – Formato do JSON Abstrato

```
setDate.put("Type",0);
setDate.put("DayLightSaving",1);
setDate.put("TimeZone","CST-8:00:00");
setDate.put("TimeFormat","Local Time");
setDate.put("Day",22);
setDate.put("Month",4);
setDate.put("Year",2014);
setDate.put("Hour",23);
setDate.put("Minute",44);
setDate.put("Second",23);
```

Figura 12 – Construção JSON para setDateTime através API REST

Após a construção dos dados no formato desejado é criada uma conexão HTTPS, sobre a qual são enviados os dados para a realização da operação (Figura 14). Utilizando a URI correspondente ao serviço que desejamos utilizar que neste caso é o setSystemDateAndTime() definido na variável "urlString", o servidor por sua vez vai interpretar os dados e realizar a comunicação necessária com o dispositivo ONVIF.

Para realizar a autenticação na câmara são enviadas as credenciais de acesso á câmara através do cabeçalho do pedido HTTPS, o servidor interpreta os dados recebidos no cabeçalho e quando faz a comunicação com a câmara utiliza estas credenciais para se autenticar. O cabeçalho de um pedido HTTP aceita valores em formato chave-valor, que são adicionados através da função *addRequestProperty*().

3.4.2.2. Operação de GetProfiles através do servidor REST

Este tipo de operação, dada a sua natureza de leitura, envolve um pedido através de uma conexão HTTPS, utilizando a URI especificada na API, quando o pedido é sucedido vai receber uma resposta no formato de dados JSON, que contém todas as informações que permitem preencher as classes java para guardar os respetivos dados, no entanto, é necessário realizar o "parsing" de

```
URL url = new URL(urlString);
HttpsURLConnection urlConnection = (HttpsURLConnection) url.openConnection();
urlConnection.setDoOutput(true);
urlConnection.setRequestMethod("PUT");
urlConnection.addRequestProperty("usr", "admin");
urlConnection.addRequestProperty("pwd","umoc15");
OutputStreamWriter out = new OutputStreamWriter(urlConnection.getOutputStream());
out.write(builderJSON().toString());
out.close();
```

Figura 14 – Estabelecer Conexão HTTPS com Servidor REST

```
InputStream inStream = null;
inStream = urlConnection.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(inStream, "UTF-8"));

StringBuilder builder = new StringBuilder();
for (String line = null; (line = reader.readLine()) != null;) {
    builder.append(line).append("\n");
}

JSONTokener tokener = new JSONTokener(builder.toString());
finalResult = new JSONObject(tokener);
```

Figura 13 – Receção Resposta do Servidor REST

todos os campos do JSON, realizar as atribuições e fazer o instanciamento destas classes de armazenamento.

Na Figura 13 pode-se analisar a receção da resposta do servidor. Dada a extensão do processamento apenas é ilustrada a parte inicial do parsing do JSON (Figura 15).

Durante o processamento do JSON vão sendo feitas as atribuições a variáveis, sempre que as variáveis processadas permitam instanciar um objeto JAVA correspondente aos dados em questão, permitindo as variáveis serem descartadas ao longo do processamento, visto que os seus dados encontram-se já alocados em classes JAVA criadas para o armazenamento dos dados, e que no final vão representar os *n* perfis de Media obtidos da operação *getProfiles*.

```
JSONObject multicast = aencoder.getJSONObject("Multicast");
boolean autoStart = Boolean.parseBoolean(multicast.getString("AutoStart"));
int port = multicast.getInt("Port");
int ttl = multicast.getInt("TTL");
MulticastConfig multicastConf = new MulticastConfig(ipAddrInfo, port, ttl, autoStart);
```

Figura 15 – Parsing do Objecto Multicast JSON e criação de objeto JAVA Multicast

3.4.2.3. Operação SetDateAndTime utilizando a API JNI

Tal como utilizando o REST, para realizar desta operação é necessário construir a estrutura de dados que vai ser enviada para a câmara. No entanto, neste caso a comunicação é implementada pela biblioteca C UMOC [11] acessível através de JNI. Como essa interface JNI possui as suas próprias estruturas, estas tem de ser preenchidas e instanciadas para serem utilizadas pela biblioteca.

```
DateTime c_date = new DateTime(ip, username, password);
num_requests++;
// preencher data
c_date.setDayLightSaving(true);
c_date.setTimeZone("CST-8:00:00");
c_date.setType(DateTimeType.NTP);

c_date.getLocal().setDay(5);
c_date.getLocal().setHour(12);
c_date.getLocal().setMinute(34);
c_date.getLocal().setSecond(12);
c_date.getLocal().setYear(2005);
c_date.getLocal().setMonth(3);
```

Figura 16 – Instanciar objeto "datetime" e envio para câmara através da JNI

Na Figura 16, podemos ver como podemos ver como é instanciada a classe "datetime" que é um *proxy* para uma estrutura C na biblioteca.

Quando se encontra preenchida esta classe é invocado o método JAVA que permite que o método nativo setDateAndTime (da biblioteca UMOC) seja executado. O método (Figura 17) que permite fazer esta operação é constituído por quatro parâmetros de *input*, daí a necessidade de ter sido criado o objeto "datetime", os outros parâmetros são os dados de autenticação na câmara (username e password), e o endereço de rede da câmara, quando executado o método é devolvido o resultado da operação através de um inteiro (caso sucesso é devolvido o valor 0).

```
int r= umoc_jni.set_DateTime_LL(ip, username, password, c_date);
```

Figura 17 – Método JNI para operação setDateAndTime

3.4.2.4. Operação GetProfiles utilizando a API JNI

Para realização desta operação temos de utilizar o método *getProfiles*()disponibilizado pela API JNI(Figura 18), este método é constituído por quatro parâmetros.

O ultimo parâmetro é um parâmetro de saída no qual vão ser gravados os dados recolhidos da operação, o espaço de memória para alocar estes dados é reservado através da declaração de um novo objeto do tipo "MediaProfile". A estrutura vazia é então passada na invocação do método Java para a operação getProfiles via JNI que ao ser executado o método a biblioteca UMOC vai realizar todas as operações de *parsing* e comunicação com a câmara, depois disso vai nos ser devolvida uma estrutura (através do objeto JAVA MediaProfile) aonde estão todos os valores recolhidos pela realização da operação.

```
ArrayList<org.uminho.onvif.MediaProfile> profiles;
profiles = new ArrayList<org.uminho.onvif.MediaProfile>();
int r = umoc_jni.getProfiles_LL(ip, username, password, profiles);
```

Figura 18 – Declaração de Media Profile (JNI) e Método Java para a operação getProfiles através da JNI

Acedendo aos dados armazenados via JNI (Figura 19) podemos fazer as atribuições para as classes JAVA criadas para o armazenamento dos dados e instanciar os objetos que vão representar um media profile em JAVA.

```
boolean autoStart = profiles.get(i).getAe().getMulticast().getAutoStart();
int port = profiles.get(i).getAe().getMulticast().getPort();
int ttl = profiles.get(i).getAe().getMulticast().getTTL();
MulticastConfig multicastConf = new MulticastConfig(ipAddrInfo, port, ttl, autoStart);
```

Figura 19 – Aceder aos dados guardados em UMOC via JNI e criar objeto JAVA correspondente

3.4.2.5. Operação SetSystemDateAndTime utilizando a biblioteca SOAP

SOAP é o protocolo nativo de comunicação com os dispositivos ONVIF. Nos processos anteriores foi utilizada uma biblioteca que gera as mensagens SOAP e encobre também a comunicação HTTP, e um servidor que traduz REST/JSON em HTTP/SOAP. A utilização de uma biblioteca SOAP obriga a descer um pouco o nível de abstração e lidar com os detalhes da comunicação ONVIF. Concretamente, é necessário conhecer como é construído um envelope SOAP, quais os *namespaces* utilizados pela câmara, construir o envelope com os dados em formato XML, enviar o envelope e interpretar a resposta.

Para construir um pedido correto, é necessário conhecer todos os componentes do XML que compõem o envelope SOAP da operação ONVIF SetSystemDateAndTime. A forma mais direta foi realizar o pedido através do servidor REST e intercetar a mensagem que este envia para a câmara. Para isso, foi utilizada a ferramenta WireShark. Tendo a mensagem obtida por referência e conhecendo a API da biblioteca Ksoap2-Android, foi então escrito o código JAVA que constrói o XML do envelope com as mesmas características da mensagem enviada pelo servidor REST.

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

```
Element username = new Element().createElement(WSSE_NAMESPACE, "Username");
username.addChild(Node.IGNORABLE_WHITESPACE, "admin");
usernametoken.addChild(Node.ELEMENT, username);
```

Figura 20 – Definição de envelope SOAP e Construção de um dos elementos do cabeçalho do pedido

Na Figura 20 pode ser visto como é construído um dos elementos do cabeçalho do envelope SOAP, utilizando o *namespace* correspondente para aquele campo, e depois é adicionado ao XML através do método *addChild()*.

O corpo do pedido (denominado *"envelope body")* inclui as informações necessárias para a realização do pedido *setSystemDateAndTime*, todos os dados são enviados dentro do corpo do envelope SOAP, na Figura 21 podemos visualizar o exemplo da construção e inclusão no body de um dos campos necessários para a construção do pedido SOAP.

```
PropertyInfo year = new PropertyInfo();
year.setNamespace(ONVIF_SCHEMA_NAMESPACE);
year.setName("Year");
year.setValue("2014");
date.addProperty(year);
```

Figura 21 – Construção do Nó "year" e inclusão no envelope SOAP

3.4.2.6. Operação GetProfiles utilizando a biblioteca SOAP

Para se implementar desta operação, tal como no SetSystemDateAndTime, é necessário realizar um dump da comunicação entre o servidor REST e a câmara para que se conheçam os elementos que constituem o envelope SOAP, como é uma operação do tipo Set o envelope apenas vai ter o header com dados, sendo que o body não contem qualquer informação no momento do pedido, no entanto quando é recebida a resposta da câmara é necessário fazer o parsing do XML, o interpretador incorporado na biblioteca, fazer as atribuições e instanciar as classes que permitem alocar os dados em classes Java.

```
SoapObject multicast = (SoapObject) aencoder.getProperty("Multicast");
SoapObject address = (SoapObject) multicast.getProperty("Address");
boolean autoStart = Boolean.parseBoolean(multicast.getPropertyAsString("AutoStart"));
int port = Integer.parseInt(multicast.getPropertyAsString("Port"));
int ttl = Integer.parseInt(multicast.getPropertyAsString("TTL"));
MulticastConfig multicastConf = new MulticastConfig(ipAddrInfo, port, ttl, autoStart);
```

Figura 22 – Parsing XML recebido na operação ONVIF GetProfiles

3.5. Realização dos Testes e Resultados

Ao longo deste capítulo foi abordado, como foi construído o ambiente de testes, desde o ambiente de rede até às diferenças entre os vários modos de comunicação. Como foi abordado na secção 3.4, o ambiente de testes que consiste em:

- Duas Configurações de Rede, uma configuração via servidor e outra com comunicação direta com a câmara;
- Três modos de comunicação:
    - Servidor REST com formato de dados JSON;
    - Através de uma JNI para a biblioteca UMOC;
    - Utilizando uma biblioteca SOAP para JAVA;
- Em cada modo de comunicação realizar testes realizando duas operações diferentes:
    - SetSystemDateAndTime;
    - GetProfiles;
- Dois limites diferentes:
    - Quebra de 3% de Bateria;
    - Limite de 5000 operações;

Os testes visam avaliar a eficiência e desempenho dos vários modos de comunicação e também orientar a escolha do modo de comunicação a utilizar para desenvolver a aplicação final que faz a gestão de perfis de Media ONVIF.

É de salientar que todos os testes foram realizados sobre as mesmas condições, isto é, antes da realização dos testes foram encerradas todas as aplicações em *background*. No entanto, os serviços que as aplicações (e.g. aplicações de mensagens) mantêm em segundo plano, não foram encerradas dado que isso iria obrigar a um encerramento bruto dessas aplicações e poderia provocar instabilidade do sistema operativo. Outro fator importante diz respeito à carga da bateria. Como não temos a informação às décimas ou centésimas da carga de uma bateria do *smartphone* (tal não seria de qualquer forma exato), realizar os testes de qualquer forma poderia conduzir a resultados em que o consumo de carga da bateria real se situaria entre os 2,5% e os 3,5%, ou seja, com uma variação/erro de pelo menos cerca de 30%. Para evitar isso, foi tido o cuidado de iniciar o teste assim que se verificou a queda de um ponto percentual da carga da bateria. Assim garante-se que os testes utilizaram o mais aproximadamente possível os 3% de carga, tendo assim a sua validade maximizada.

Para ser de mais fácil interpretação, os resultados dos testes foram organizados nas tabelas 1 e 2, que incluem todos os testes realizados para cada modo de comunicação nos dois limites de duração do teste estabelecidos previamente.

Teste Queda Carga Bateria (3 %)	Nº de Pedidos Realizados	Duração do Teste	Operações/%bat	Operações/Seg
getProfiles via REST	3168	00:24:52	1056	2,12
setDateAndTime via REST	5353	00:23:48	1784	3,75
getProfiles via JNI	8881	01:43:40	2960	1,43
setDateAndTime via JNI	16926	00:57:56	5642	4,87
getProfiles via SOAP	2709	01:41:36	903	0,44
setDateAndTime via SOAP	3508	00:50:20	1169	1,16

Tabela 1 – Resultados dos testes com limite pela diferença da carga da bateria

Como podemos observar pelas tabelas, os testes apresentam uma desvantagem muito grande no modo SOAP e REST comparativamente ao JNI. É facilmente percetível, que o modo de

comunicação, quer mais eficiente, quer mais rápido é o que utiliza a API JNI. Como é possível ver, este modo permite que sejam realizados numa determinada descarga de bateria um maior número de pedidos, e também é capaz de realizar um número fixo de pedidos num intervalo de tempo bem mais curto.

Teste 5000 Requests	Queda Carga Bateria (%)	Duração do Teste	Operações/%bat	Operações/Seg
getProfiles via REST	5	00:48:04	1000	1,73
setDateAndTime via REST	2	00:32:03	2500	2,60
getProfiles via JNI	2	00:25:12	2500	3,30
setDateAndTime via JNI	1	00:11:06	5000	7,50
getProfiles via SOAP	7	01:51:18	714,2	0,74
setDateAndTime via SOAP	7	00:36:48	714,2	2,26

Tabela 2 – Resultados dos testes com limite de 5000 operações

4. Configuração de Perfis de Media

O ONVIF disponibiliza vários serviços, como foi explicado na secção 2.1, e como a aplicação final tem como objetivo a gestão do serviço de Media [9] através de um cliente para *Android*, é necessário perceber a estrutura e o papel de um perfil de Media, e as potencialidades que o serviço disponibiliza.

4.1. Perfil de Media ONVIF

Um perfil de Media é nada mais que um conjunto de entidades de configuração que controlam funcionalidades como por exemplo o vídeo e áudio que é disponibilizado pelo dispositivo NVT. Dentro de um NVT podem existir, ou serem criados, inúmeros perfis de Media, que permitem gerar *streams* de vídeo e áudio de acordo com os parâmetros especificados em cada uma dessas entidades.

A identificação dos perfis de Media é realizada através de um *token* único associado a cada perfil. Para além do *token,* existe outra propriedade intrínseca a cada perfil de Media que é o *"fixed".* Este permite identificar se um determinado perfil pode ou não ser apagado. No entanto, nada tem a ver com o facto de um perfil ser imutável ou não. Em todo o caso é possível adicionar, retirar e modificar configurações a um perfil, seja ele *"fixed"* ou não.

4.1.1. Entidades de um perfil de Media

As entidades que podem constar dentro de um perfil são:

- Video Source Configuration
- Video Encoder Configuration
- Audio Source Configuration
- Audio Encoder Configuration
- Audio Output Configuration
- Audio Decoder Configuration
- PTZ Configuration
- Video Analytics Configuration
- Metadata Configuration

Estas entidades, podem ou não, ter dependências entre si, sendo que por exemplo o *video encoder* utilizado depende das capacidades de um *video source* utilizado num determinado perfil. Assim um perfil de Media pode ser composto por todas ou uma parte destas entidades, que dependem também das capacidades de um determinado dispositivo ONVIF. Por exemplo, só existirá uma configuração de PTZ caso esse dispositivo tenha recursos físicos para tal.

Na Figura 23 é mostrado como é constituído um perfil de Media que aborda todas as entidades passiveis de pertencer a um perfil.

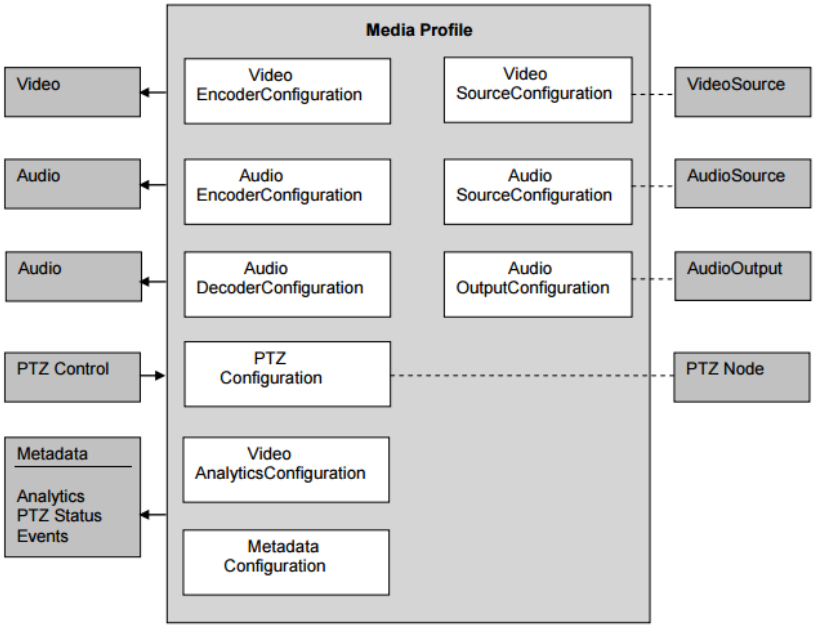


Figura 23 – Perfil de Media Completo (fonte: http://www.onvif.org/)

Através da utilização de perfis é possível configurar como um determinado *stream* de media é apresentado a um cliente, e também gerir as entradas de PTZ e Analytics. Podemos com isto criar diferentes tipos de conteúdo media para diferentes clientes e criar diferentes hierarquias de utilizadores (associados a um determinado perfil) a aceder a serviços de media disponibilizados por um determinado dispositivo. Uma adaptação a um caso real, seria por exemplo criar um perfil para uma utilização numa rede sem limitações na largura de banda e obter um *stream* com uma resolução mais alta, um *encoder* que forneça uma melhor qualidade de imagem e por outro lado criar um perfil que utilize uma resolução mais baixa, e utilizar um *encoder* com uma taxa compressão de dados mais alta para que seja utilizado por exemplo, num dispositivo móvel com limitações de largura de banda e restrições de consumo de dados móveis.

4.1.2. Serviço Media

O serviço de Media ONVIF pode ser dividido em dois grupos de operações:

- Configuração – constituído pelas operações de gestão de perfis e todas as respetivas entidades e seus parâmetros, do tipo: CreateProfile, Add<Entidade>Configuration, Get<Entidade>(Options), DeleteProfile, Remove<Entidade>, GetProfile(s), Set<Entidade>Configuration;
- Streaming – constituídos pelas operações que acedem aos streams multimédia e respectivo e controlo: GetStreamUri, GetSnapshotURI, <Start,Stop>MulticastStreaming e SetSynchronizationPoint.

Uma característica importante no serviço de Media é a existência de uma operação que permite obter os valores possíveis para cada configuração/parâmetro. Por exemplo, no caso de querermos saber que *encoders* determinada câmara tem disponíveis é possível invocar um comando que vai devolver uma lista. Esta lista pode então ser depois utilizada, por exemplo, para construir um menu "drop-down" que vai devolver ao utilizador as suas possibilidades de escolha para o dispositivo em questão. Esta característica é muito importante na eficiência da programação do lado do cliente, permitindo desenvolver uma GUI que apresenta apenas valores de escolha válidos resultando portanto em alterações sempre com sucesso. Ao ser conhecido à partida todas as configurações possíveis no serviço Media de um dispositivo evitamos a realização de pedidos cuja resposta vai resultar em erro e não ser realizada qualquer alteração no dispositivo.

Em algumas entidades existem inúmeras dependências, quer entre os seus parâmetros internos, quer com outras entidades. As entidades para além de dependerem de outras entidades, dependem também do hardware de cada dispositivo. Se uma câmara não tiver uma lente capaz de realizar zoom, ou os motores de movimento pan e tilt, não pode existir a entidade responsável pela configuração do PTZ nos seus perfis de Media. As dependências entre parâmetros de uma determinada entidade acontece, por exemplo, quando se escolhe um determinado VideoEncoder que disponibiliza certas resoluções de vídeo. Ao trocar de VideoEncoder algumas dessas resoluções podem não estar disponíveis.

4.2. Cliente Android para Gestão de Perfis de Media

4.2.1. Introdução

Ao iniciar o desenvolvimento da aplicação foram previamente definidas as principais funcionalidades que vão ser implementadas, este processo permite facilitar o processo de desenho da arquitetura e definir algumas abordagens a seguir no desenho gráfico da User Interface.

Em termos de funcionalidades, a aplicação tem como objetivo permitir:

- Criar, apagar, alterar e visualizar perfis de Media de dispositivos NVT;
- Facilitar a organização de um conjunto de câmaras que um utilizador utiliza e identifica-las facilmente através de uma designação à escolha;
- Permitir que o utilizador consiga visualizar os vários *streams* gerados pelos vários perfis;
- Implementar persistência para armazenar todos os dados inseridos durante a utilização.

No capítulo 3 desta dissertação já foi abordado, mas fez também parte deste processo de decisão, qual a API que seria utilizada para o processamento/comunicação com os dispositivos ONVIF.

4.2.2. Prototipagem da User Interface

Numa fase inicial o Android sofreu muito de segmentação e não concordância de design gráfico entre as várias aplicações. O design gráfico de uma aplicação é composto pelos vários elementos de UI, desde as animações, tamanho e tipo de letra, botões, ícones, tipos de menus, até aos espaçamentos entre elementos de uma lista "drop-down", entre outros inúmeros elementos gráficos e *widgets*.

Um dos problemas que surgiram à partida foi a escolha de um modelo para o design gráfico. A solução foi utilizar o Material Design [25] que a *Google* criou, na expectativa de facilitar este tipo de decisão aos programadores, tornar as aplicações *Android* com um design mais sólido e unificado, fornecendo uma linguagem gráfica consistente em diferentes dispositivos.

Para facilitar a criação dos *layouts* e estruturar a aplicação foi utilizado um software de prototipagem de layouts, especifico para a realização de protótipos de software para ambiente

móvel. Este software consiste numa Web App, tem por nome "proto.io" [26] e é largamente utilizado pelos principais criadores de aplicações móveis.

Na Figura 24 pode ser visto um esboço do menu inicial sobre é mostrada uma lista com as várias câmaras que o utilizador tem sob gestão. Quando é selecionada uma das câmaras é exposto um segundo menu (Figura 25), num formato de grelha, que permite o utilizador aceder às entidades de um perfil de Media, de acordo com o perfil selecionado através da parte superior do ecrã que apresenta todos os perfis disponíveis na câmara selecionada. Ao selecionar uma câmara é aberto o primeiro perfil recebido.

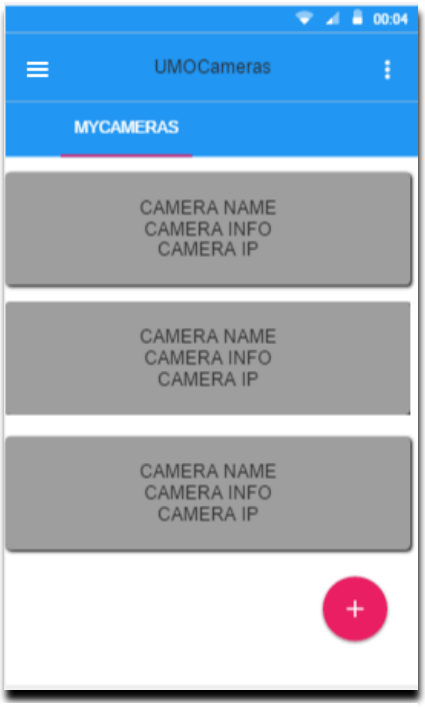


Figura 24 – Protótipo Menu Inicial

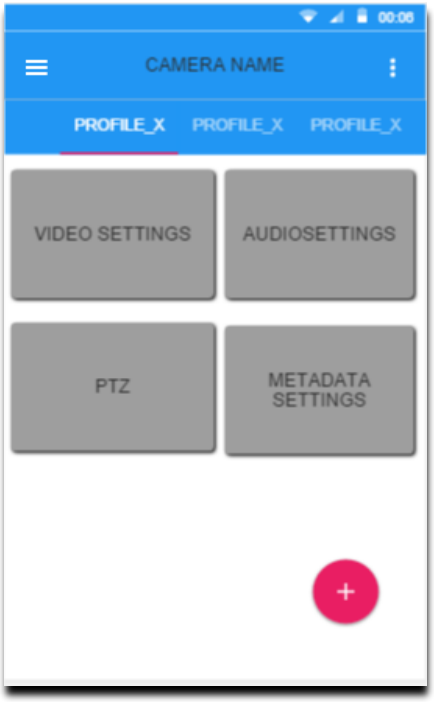


Figura 25 – Protótipo Menu Configurações Perfil de Media

Quando o utilizador seleciona uma câmara a aplicação apresenta perspetiva da Figura 25. Uma câmara pode ter inúmeros perfis e a navegação entre eles deve ser facilitada. Para isso, optou-se pela utilização de "Swipable Tabs" que permitem visualizar diretamente no ecrã as configurações do perfil atualmente selecionado, ao mesmo tempo que mostra as designações de outros perfis existentes. Estes perfis podem ser acedidos diretamente e com uma barra de suporte das tabs "Scrollable" a navegação para outros perfis faz-se através de gestos ("swipes").

Para permitir uma maior facilidade na transição entre câmaras e colocar funcionalidades como as opções de Login/Logout, adicionar nova câmara, ou outras funções que mais tarde possam interessar, foi criado um "Navigation Drawer" com essas mesmas opções (Figura 27). O Navigation Drawer é um painel que se encontra escondido e pode ser aberto através de um *swipe* na parte esquerda do ecrã para dentro, ou carregando no botão que se encontra na "*toolbar*" do lado esquerdo. Outro propósito do "*Navigation Drawer*" [27] é mostrar na parte superior a fotografia, correio eletrónico e nome do utilizador atual. Estes dados são obtidos através da integração da API de login do Google+ [28] (que todos os utilizadores de Android, são obrigados a ter).

Na Figura 26 é mostrado o protótipo a da caixa de diálogo de inserção de uma nova câmara. Este tipo de caixas são bastante utilizadas para suportar a introdução de dados em formulários ou para fornecer alertas durante a utilização da aplicação.

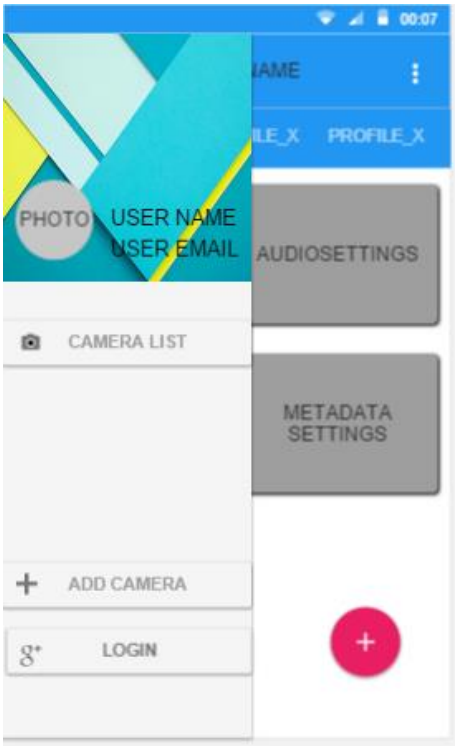


Figura 27 – Navigation Drawer

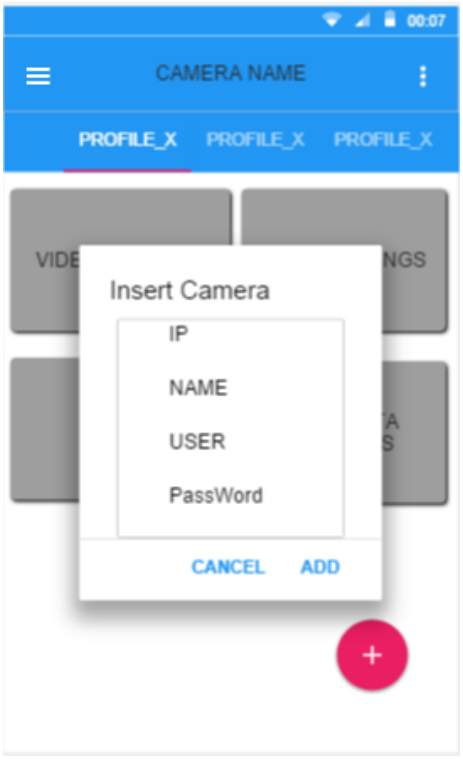


Figura 26 – Protótipo dos Dialogs

Para a exibição ou edição de entidades do perfil de Media selecionado, foi criado um protótipo (Figura 28) aonde se inclui os vários tipos de "widgets" adequados às opções de configuração das entidades, nomeadamente *seekbar, radio button, spinner, toogle button, etc..*). Estes *widgets* suportam respetivamente a escolha de valores numa gama, a selecção de um conjunto de opções,

alterar valor dum booleano, entre outros. Os *layouts* concretos serão definidos aquando da construção dos ecrãs das entidades distinguidas/agrupadas na Figura 25, conforme os parâmetros que estas incluírem.

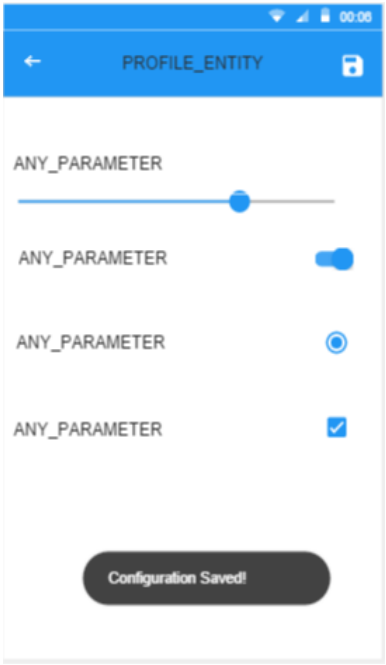


Figura 28 – Protótipo Menu Configuração de uma entidade de um Perfil

4.2.3. Descrição dos Componentes da Aplicação

4.2.3.1. Autenticação

Para permitir a utilização de diferentes utilizadores na aplicação, garantir a separação dos dados entre eles, foi utilizado um sistema de autenticação que utiliza uma API que a Google disponibiliza, que se chama *Google Sign-In API* [28].

Existem quatro requisitos [29] para a utilização desta API, que são:

- A criação de um KeyStore (componente do Android Studio) para armazenar o certificado que é associado á aplicação, isto para o cliente OAuth2 [18] integrado na API autenticar a aplicação nos servidores da Google;
- Criar uma conta na Console Developers da Google e um novo projecto, ao qual se associa a chave SHA-1 do certificado da aplicação; a chave pode ser obtida através da ferramenta da linha de comandos Keytool, utilizando o seguinte comando:

  keytool -exportcert -alias <your-key-name> -keystore <path-to-production-keystore> -list –v

- Incluir a biblioteca do *Google Play Services* na aplicação, o que pode ser realizado colocando uma nova dependência no ficheiro build.gradle com o seguinte formato:

  compile 'com.google.android.gms:play-services:7.5.0'

- Adicionar as seguintes permissões no ficheiro AndroidManifest.xml

  <uses-permission android:name="android.permission.INTERNET" />

  <uses-permission android:name="android.permission.GET_ACCOUNTS" />

  <uses-permission android:name="android.permission.USE_CREDENTIALS" />

A implementação desta biblioteca no código desenvolvido é muito simples, só sendo necessário iniciar um novo objeto GoogleApiClient no método *Oncreate()* da *MainActivity*, e de seguida invocar o método *connect()* desse objeto. A autenticação do utilizador no servidor da Google é feito de forma assíncrona, o que implica a utilização de *callbacks*. Sempre que o login é realizado com sucesso, a função *OnConnected()* é invocada e basicamente a aplicação é iniciada a partir desse método, garantindo a integridade da aplicação. No entanto, só durante o primeiro login é necessário ter ligação à internet para gerar os *Tokens* do utilizador. Depois disso, estes ficam armazenados e o utilizador pode utilizar uma rede local sem conectividade ao exterior para utilizar a aplicação.

4.2.3.2. Persistência

Para garantir que os utilizadores da aplicação não têm de inserir constantemente os dados das câmaras, estes são armazenados numa base de dados *SQLite* [30] [31], nativa do *Android*. Apesar de existirem mais maneiras de implementar persistência, uma base de dados permite maior escalabilidade e segmentação, dada a facilidade em alterar e reestruturar tabelas ou mesmo acrescentar às existentes. As possibilidades alternativas incluem as "*Shared Preferences*" [32] que o SO oferece, no entanto, este mecanismo apenas permite guardar tipos de dados primitivos no formato chave-valor, o que limita muito a utilização. Outra possibilidade seria recorrer ao armazenamento interno, e criar uma estrutura de dados pré-definida num ficheiro, o que traria um desempenho muito pior, e iria obrigar a uma implementação mais difícil e sujeita a erros (envolvendo a serialização e deserialização dos dados em cada leitura/escrita e a gestão de

ficheiros), iria limitar também o crescimento porque torna difícil uma alteração ao modelo inicial que se define na leitura ou escrita dos dados.

Para garantir a persistência foram então utilizadas duas tabelas, uma tabela que agrupa os utilizadores e outra que agrupa as câmaras. Na Figura 29 podemos ver as duas tabelas.

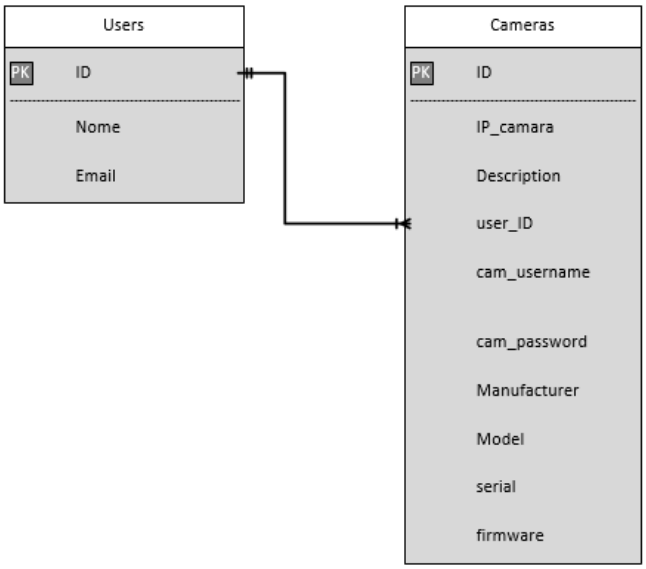


Figura 29 – Modelo de Base de Dados utilizado na aplicação

É importante salientar a relação que existe entre o "ID" de um utilizador e do atributo "user_ID" na tabela das câmaras, que estabelece uma relação de um para muitos, visto que um utilizador pode ter várias câmaras associadas a si. Assim garantimos sempre que todas as câmaras têm um utilizador único associado a si.

A implementação e gestão da base de dados é realizada a partir da classe *DBcore.java* do projeto. Nesta classe encontram-se implementadas as funções de criação da base de dados, operações CRUD (Create, Read, Update, Delete), assim como também as *queries* utilizadas durante o funcionamento da aplicação.

O processo inicial que a aplicação realiza depois da autenticação e permite recuperar um estado anterior (caso o utilizador já esteja registado da base de dados e já tenha câmaras adicionadas) pode ser visto na Figura 30. Este processo acontece sempre que a aplicação é executada, depois de esta ter sido "morta" quer por intuito do utilizador, ou por operações de gestão de memória realizada pelo sistema operativo.

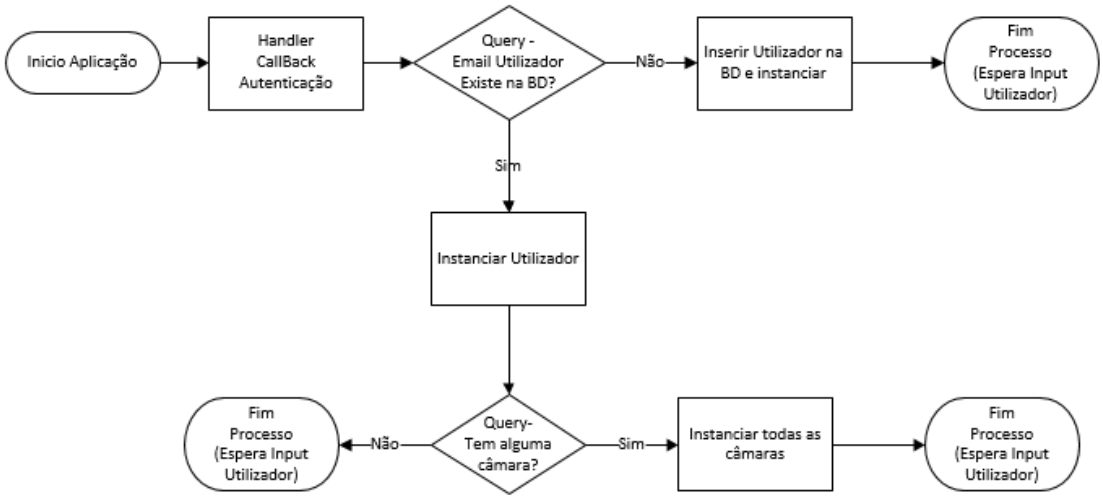


Figura 30 – Fluxograma processo de recuperação do estado da aplicação

4.2.3.3. Correspondência entre as Activities/fragments e a UI

Os componentes Activities [33] e Fragments [34] são os mais importantes da interface gráfica da aplicação pois é neles que todos os outros componentes se vão enquadrar, e porque é entre eles que se processam as transições entre os vários ecrãs da UI.

Com o surgimento da versão 3.0 do sistema operativo Android, surgiram também os fragments. Este componente permite que sejam desenvolvidas interfaces gráficas com maior flexibilidade e modularidade. Os Fragments permitem que as aplicações se adaptem melhor aos ecrãs de menores dimensões (i.e. os programadores podem criar uma UI em que só utiliza um fragment para ecrãs menores, e num tablet por exemplo permitir exibir dois fragments em simultâneo). Um Fragment também pode ser reutilizado em várias Activities, pois implementa o seu próprio layout.

A aplicação que foi desenvolvida utiliza *fragments* para a implementação dos menus principais da aplicação. Foram utilizados principalmente não com o intuito de reutilização de layouts mas sim pela maior eficiência nas transições, visto que a utilização de *activities* torna a aplicação mais pesada. No entanto, as interfaces que permitem a gestão e configuração das entidades dos perfis de Media foram isoladas em *activities*, permitindo evitar a execução de *widgets* relativos a entidades que não são utilizadas ou que já foram utilizadas e não há mais necessidade de manter em memória. Na Figura 31, pode-se ver um pequeno esquema no qual se consegue ver como as *activities* e os *fragments* encaixam entre si e os menus da aplicação, na *MainActivity* existem dois *fragments* que integram os layouts dos menus correspondentes, ao ser selecionada uma câmara

é exposto o *fragment* com as entidades de configuração, que apresenta em separadores os vários perfis, ao ser selecionada uma das entidades de configuração desse menu é aberta uma nova *activity*, que aloja os layouts e código relativo às configurações dessa entidade. Sempre que o utilizador prime o botão *back* é levado para o ecrã anterior.

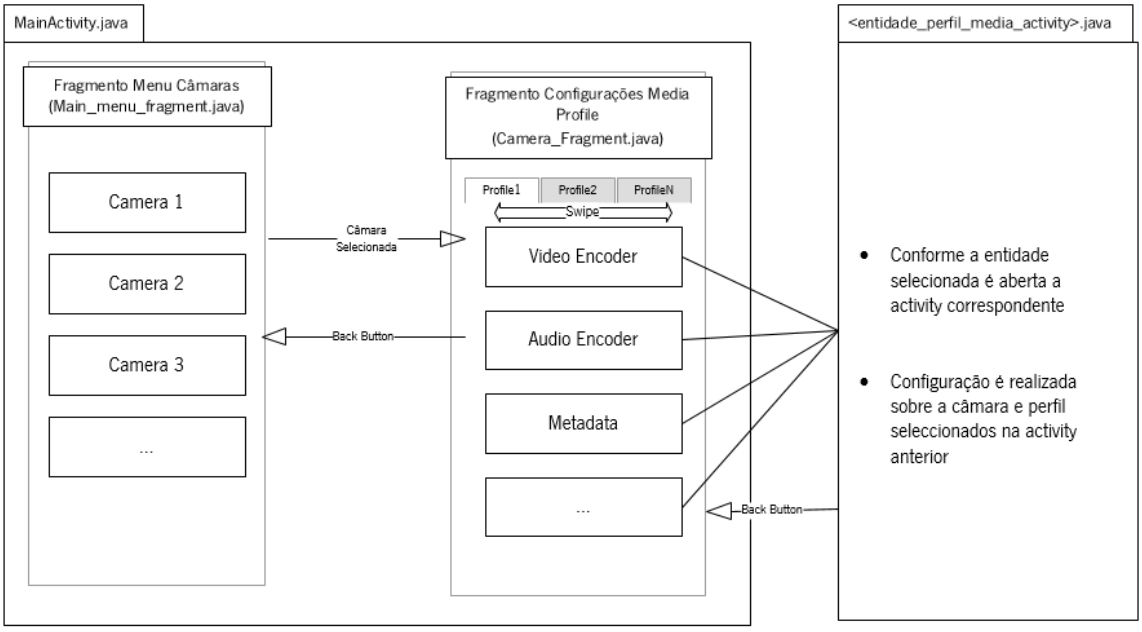


Figura 31 – Integração entre Activities/Fragments na Interface Gráfica

Todas as Activities possuem na sua parte superior uma *toolbar* que apresenta um pequeno menu de contexto, que permite executar algumas funcionalidades conforme o menu em questão. Os fragments apresentam conteúdo que se encontra organizado através de um elemento que permite exibir listas ou grelhas, chamado *RecyclerView* [35], que surgiu em detrimento do rudimentar *ListView* [36]. Este *widget* que foi recentemente integrado no Android consegue gerir melhor grandes *Datasets* e lidar mais eficientemente com a alteração dinâmica do conteúdo, quer por ação do utilizador por exemplo adicionar/remover uma câmara nova à lista do utilizador, quer por eventos de rede. No entanto, a utilização deste elemento obriga a utilização de um *RecyclerView Adapter* [37] que é o responsável pela monitorização dos conjuntos de dados dinâmicos que são apresentados no RecyclerView. Como o RecyclerView utiliza um LayoutManager para fazer a disposição dos vários elementos que compõem um conjunto de dados, permite alternar facilmente entre uma utilização em grelha ou em lista, dependendo do tipo de layout Manager (GridLayoutManager ou LinearLayoutManager) que é instanciado e passado para o RecyclerView.

Uma vantagem inerente à utilização dos fragments é a possibilidade de os dados serem partilhados entre eles através de variáveis declaradas na MainActivity. Isto permite manter sempre uma indexação da câmara selecionada a partir de qualquer *fragment* e depois, no menu seguinte, manter também indexado numa variável o perfil que está a ser consultado/editado. Evita-se assim a passagem de várias variáveis, necessárias para a integridade da aplicação, de uma *activity* para outra, que é um processo pesado para o sistema operativo.

No segundo menu, os "Swipe Tabs", contêm a designação dos perfis de uma câmara, bem como a indicação de que o respetivo perfil é do tipo "fixed" ou não, através da colocação de um ícone junto ao nome do perfil. O elemento que permite a implementação deste tipo de separadores, chama-se *TabLayout*. Este, tal como no *RecyclerView*, utiliza um *Adapter* para monitorizar os dados dinâmicos, visto que este elemento também permite serem adicionados elementos *on-the-fly*. Como o *TabLayout* implementa um *TabListener*, torna-se fácil controlar que perfil o utilizador se encontra a visualizar no momento: foi criada uma variável que guarda o respetivo índice e que vai sendo atualizada, através do *listener*, sempre que existe um *swipe ,*no menu que expõe as várias entidades de um perfil (Figura 25), e o utilizador alterna de um perfil para outro.

As *Activities* que implementam a configuração de cada entidade possuem todas o mesmo layout base, isto é, uma *toolbar* na parte superior, que possui apenas um botão para voltar ao menu anterior, e um botão que permite realizar o *commit* na câmara de todas as alterações que foram realizadas na entidade do perfil em questão e a parte principal do layout que é populada em cada activity de acordo com a entidade a que diz respeito. Caso o botão "save" não seja pressionado todas as alterações realizadas na UI são ignoradas. Desta forma, evita-se multiplicar o número de operações realizadas sobre a câmara e minimiza-se o volume do trafego de rede.

Na parte principal do layout destas activites, são utilizados widgets "clássicos" que permitem alterar os parâmetros de uma determinada entidade. Os widgets mais utilizados foram:

- Spinners – para parâmetros que são valores de um conjunto variável de strings obtidas através da classe MediaOptions da API JNI que ao ser instanciada realiza a operação getMediaOptions() ONVIF;

- SeekBar – para selecionar valores inteiros num intervalo, ou uma RangeSeekBar para parâmetros que são um par de valores que define uma gama de valores dentro de um intervalo;
- Switches – para parâmetros do tipo booleano;
- EditText – para parâmetros que aceitem texto plano
- RadioButton – para os parâmetros que são valores de um conjunto predeterminado de opções (Ex: tipo de ip, IPv4 ou IPv6)

Para utilizar estes *widgets* no layout das *activities* das entidades, não basta colocar o *widget* dentro do ficheiro XML do layout. É necessário colocar um pequeno texto a identificar a que parâmetro cada um se refere, identificar no caso das *seekbars* os seus intervalos e repetir este tipo de layout consecutivamente em cada entidade (Figura 32). Mesmo dentro de uma entidade podemos ter várias vezes o mesmo tipo de *widget* para diferentes parâmetros. Então para facilitar a organização dos *layouts* e da programação foram realizados *layouts-template* para cada um desses tipos de *widget*.

Tendo esses *templates* sido realizados em diferentes ficheiros .xml, utiliza-se a tag <include> dentro do layout da *activity* aonde se pretende incluir esses *templates* colocando um ID diferente para cada inclusão. Assim, no Java é possível utilizar o método que obtém o objeto View, nomeadamente *findViewById*() para se aceder aos elementos internos a cada *template*.

Como ter vários ID's dentro de um layout pode tornar a programação java confusa devido à necessidade de utilizar sempre ID's distintos, a utilização de *layout's-template* permite que apenas se tenha que distinguir os ID's de cada <include>. Os vários objetos internos a esse *include* usam

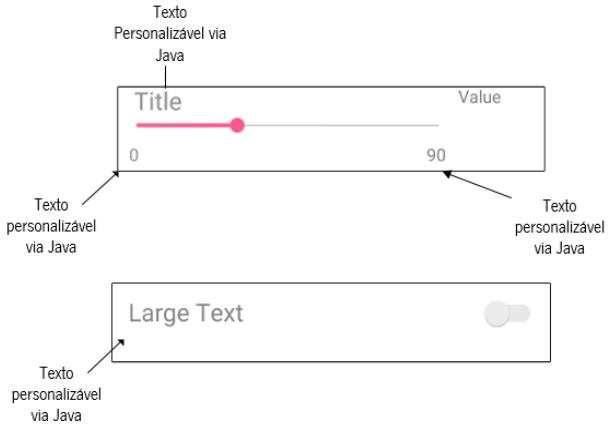


Figura 32 – Exemplos Layout's template utilizados nas Activities das entidades Media Profile

sempre o mesmo ID, podendo/devendo ser utilizados ID's com significado (dentro do template) para facilitar e clarificar a programação que envolve inúmeros *widgets* dentro da mesma *activity*.

4.2.3.4. API JNI UMOC

Como abordado no capítulo 3, existiam vários modos de comunicação mas depois dos testes realizados e de analisados os resultados, decidiu-se utilizar a API JNI.

Esta API utiliza estruturas C para armazenar os dados que vão sendo trocados com as câmaras, o que complica a gestão de memória da aplicação. Como a alocação da memória é realizada pela biblioteca C, por vezes surgem alguns problemas na gestão que é realizada Por exemplo, para se criar um novo perfil é necessário realizar uma cópia de um objeto que representa um perfil, já existente, para que o C faça a alocação de memória da respetiva estrutura e depois trabalhar sobre esse novo objeto-cópia de forma a obtermos o perfil desejado (Figura 33).

Para além da gestão da memória em C, a API JNI (por via da biblioteca C) encobre a comunicação de rede. Isto abre a possibilidade de cometer o erro de invocar as funções de comunicação na *thread* da UI, o que é altamente desaconselhado pelos padrões de boas práticas da programação para Android. De facto, o próprio compilador Android não permite que sejam realizados pedidos de rede na *thread* principal, mas não consegue forçar essa prática no caso da API JNI. Todos os métodos da API JNI que envolvem comunicação com as câmaras foram colocados dentro de *AsyncTask's*, de forma a serem executados de modo assíncrono á *User Interface* numa thread separada.

```
String profile_string = profile_name.getText().toString();
boolean fixed = fixed_value.isChecked();
MediaProfile newprofile = profiles.get(current_profile_index);
newprofile.setToken(profile_string);
newprofile.setFixed(fixed);
umoc_jni.setProfile_LL(ip_task,current_camera.camera_username,
current_camera.camera_password,newprofile);
```

Figura 33 – Criar um perfil novo através da cópia de um existente

Apesar do objetivo principal na utilização desta biblioteca ser para facilitar a comunicação com os dispositivos ONVIF, os dados trocados com as câmaras têm uma representação Java e estão guardados em memória C que depois consoante a necessidade na aplicação são ou não utilizados. Isto faz com que se crie uma camada de memória C intermédia entre a aplicação Android e os dispositivos NVT. Esta camada é dinamicamente alterada sempre que os parâmetros são alterados

na UI da aplicação, este processo é explicado na Figura 34, aonde se pode ver como vão sendo movimentados/armazenados os valores dos parâmetros de um perfil de Media que está a ser alterado pelo utilizador da aplicação, desde a UI da aplicação Android, passando pela biblioteca C, até ao dispositivo ONVIF.

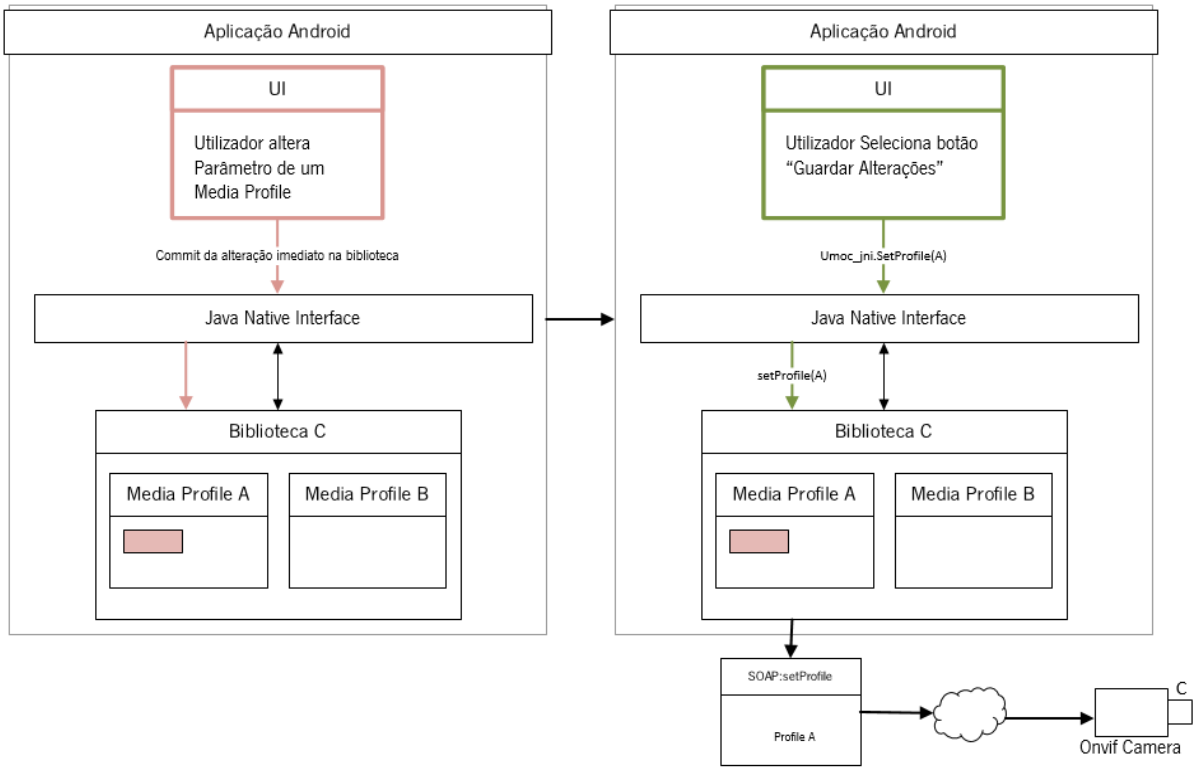


Figura 34 – Diagrama do processo de alteração de um perfil de media

No painel esquerdo da Figura 34, enquadrando com a utilização da aplicação é o momento em que um utilizador se encontra a fazer as alterações através da UI nos parâmetros de configuração de uma entidade, quando um valor é alterado é feita a alteração nas estruturas de dados do C por via da JNI, passando para o painel esquerdo, neste momento o utilizador prime o botão de salvar as alterações despoletando a invocação do método setProfile() da JNI, que faz com que seja realizada a operação de setProfile ONVIF, fazendo com que a biblioteca faça o envio do perfil ,enviando todas as estruturas de dados que representam um perfil e não apenas os parâmetros alterados, para a câmara.

Esta API também é utilizada em outras circunstâncias durante a utilização da aplicação, nomeadamente:

- Para a aplicação obter as características da câmara (fabricante, serial number, modelo, firmware) quando esta é introduzida a primeira vez na aplicação, através da classe DevInfo que quando é instanciada faz o *fetch* dessa informação;
- Para obter os endereços absolutos para os serviços ONVIF através do método getCapabilities() da classe Capabilities;
- Para obter os URL's para os Streams de determinado perfil utilizando o método JNI do serviço de Media, getStreamUri_LL();
- Para criar novos perfis de Media, da forma explicada no início desta secção.

5. Resultados e Discussão

Neste capítulo são discutidos e interpretados os resultados dos testes de comparação dos vários modos de comunicação com as câmaras. É também descrita do ponto de vista do utilizador a aplicação Android que foi desenvolvida.

5.1. Resultados dos Testes

Para avaliar a eficiência energética dos vários modos de comunicação foi calculado o número de operações realizadas por percentagem de bateria consumida, nos testes de diminuição de 3% de bateria.

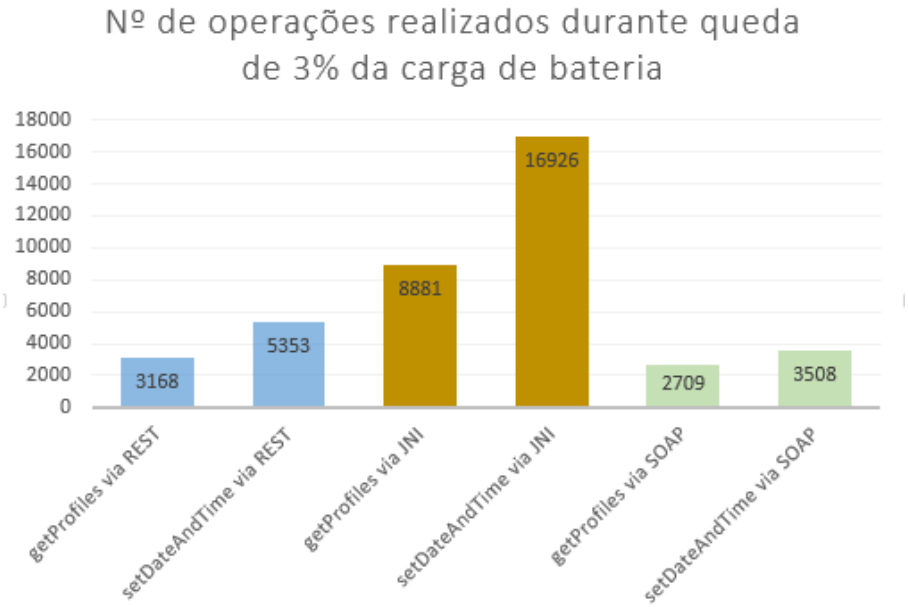


Figura 35 – Resultados do teste de eficiência

Como se pode observar no gráfico (Figura 35) com os resultados do teste de eficiência, os melhores resultados dão vantagem à utilização da JNI. Interpretando estes resultados percebe-se que para uma mesma diminuição de carga da bateria se consegue realizar mais do dobro das operações bem-sucedidas do que o segundo melhor resultado (obtido pela comunicação através de um servidor REST). Mesmo introduzindo um fator de correção devido menor número de saltos de rede, a vantagem da JNI mantém-se. Como o número de saltos do modo JNI é 6 em vez dos 8 do REST, pode aplicar-se uma diminuição de 25% no número operações, resultando 6661 e 12695 respetivamente para o GetProfiles e SetSystemDateAndTime.

No teste de performance foi estabelecido um limite de 5000 pedidos medindo o tempo necessário para serem realizados, como é óbvio, os resultados melhores são os que apresentam um valor (de tempo) menor.

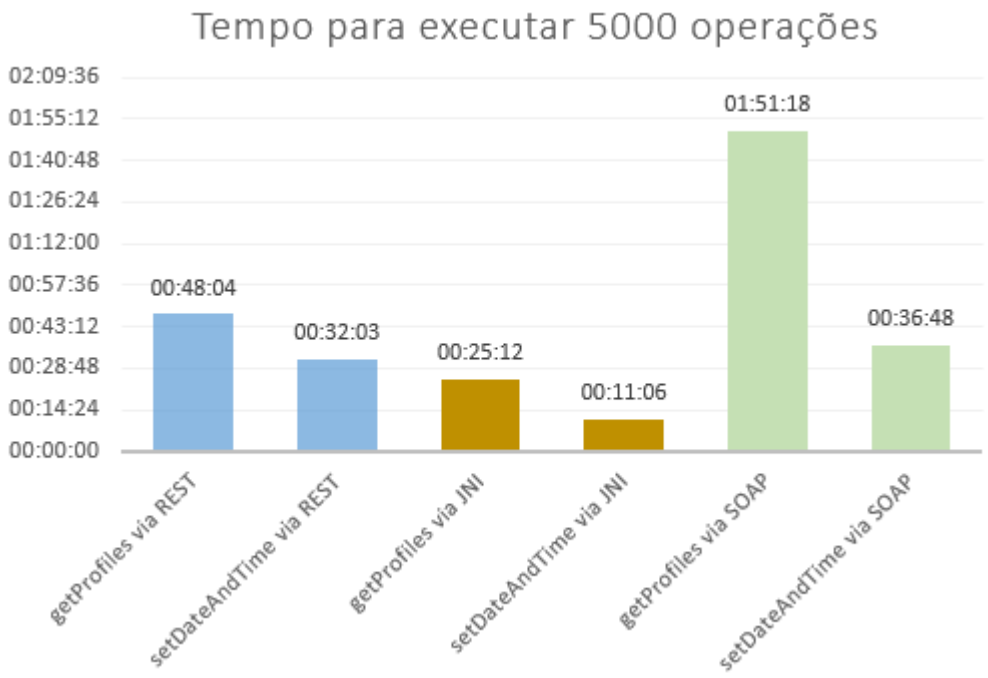


Figura 36 – Resultados do teste de Performance

No teste de desempenho (Figura 36) é perfeitamente visível também que existe uma vantagem muito grande para o JNI. É de notar que para a mesma operação entre o JNI e o REST existe uma diferença de quase o dobro do tempo para realizar o mesmo número de operações.

Seguindo o raciocínio do teste anterior, aplicando o fator de correção de 25% obtém-se ainda assim resultados mais favoráveis ao JNI, com 31 minutos e 30 segundos para o GetProfiles e 13 minutos e 52 segundos para o SetSystemDateAndTime.

Apesar do servidor REST não ter que realizar o parsing de SOAP/XML e utilizar HTTP/JSON, que é um formato de dados que envolve menos dados e menor processamento de (des)serialização, os testes mostram que isso não consegue superar a eficiência e desempenho de uma biblioteca escrita em C. O destaque vai para a eficiência energética que a execução de código nativo permite, mesmo tendo quer processar XML.

No entanto a utilização da JNI traz algumas limitações nomeadamente em termos de portabilidade, visto que a programação C requer pelo menos a recompilação da biblioteca UMOC para cada arquitetura do processador.

Os resultados obtidos utilizando a biblioteca Java que permite construir os envelopes SOAP e comunicar diretamente com a câmara foram os piores. Isto deve-se ao facto do processamento do XML que constitui os pacotes SOAP ser muito pesado para um dispositivo móvel. A própria Google desaconselha o consumo de serviços web SOAP em Android, incentivando a utilização de serviços REST de tal modo que nunca incluiu no seu sistema operativo uma biblioteca para fazer o *parsing* de pacotes SOAP.

5.2. Aplicação Final

Esta secção apresenta a aplicação final que foi desenvolvida e que permite utilizar e gerir o serviço Media disponibilizado por dispositivos ONVIF NVT. A aplicação foi desenvolvida integralmente em Java para Android e os layouts foram realizados de acordo com o vocabulário XML que este sistema operativo disponibiliza. Para comunicar com as câmaras ONVIF foi utilizada uma API JNI que expõe a biblioteca UMOC em que faz a gestão não só da Java.

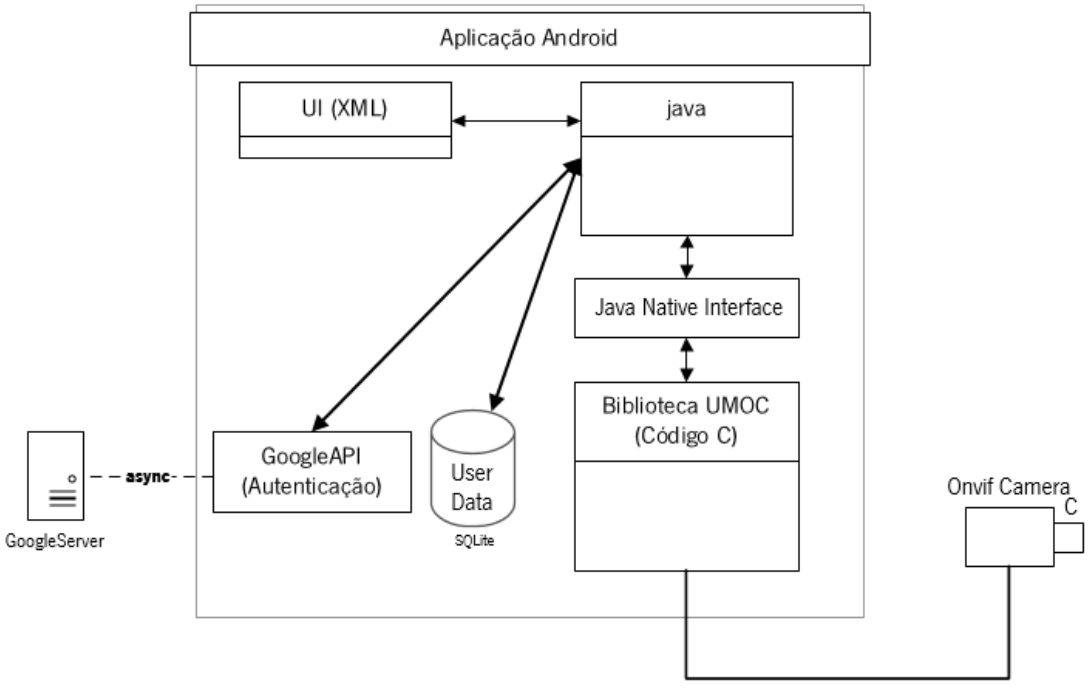


Figura 37 – Arquitetura de Software da Aplicação

5.2.1. Menu Inicial e Painel de navegação lateral

Ao ser iniciada a aplicação é apresentado um menu (Figura 38) que apresenta todas as câmaras que o teu utilizador tem na sua base de dados. No menu de contexto (ícone com 3 pontos), o utilizador tem a possibilidade de apagar as câmaras que se encontram inseridas na base de dados. Ao ser introduzida uma nova câmara (Figura 39), a aplicação carrega os dados que apresenta nos itens do menu principal (Fabricante, modelo) e guarda na base de dados para que não seja necessário realizar este processo novamente.

Sempre que uma câmara existente na base de dados é selecionada e por algum motivo a aplicação não consegue comunicar com esta, é lançado um aviso numa pequena janela pop-up que aparece temporáriamente no formato de Toast (componente de UI para Android).

O botão com o simbolo (+) que se encontra na parte inferior é conhecido como (Float Action Button) e permite realizar ações como adicionar uma câmara ou um perfil de Media numa câmara à escolha.

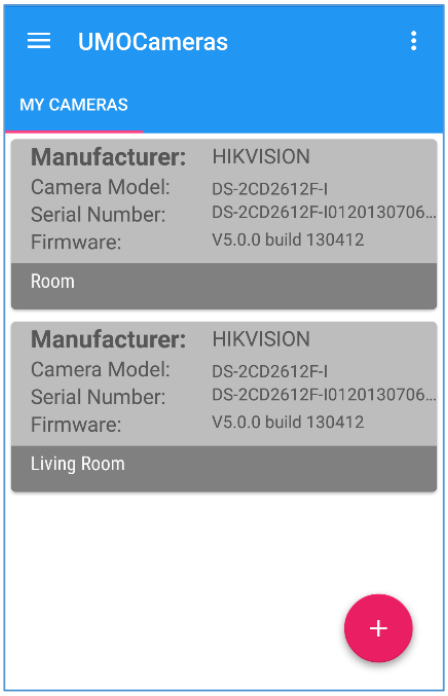


Figura 38 – Menu Inicial da Aplicação

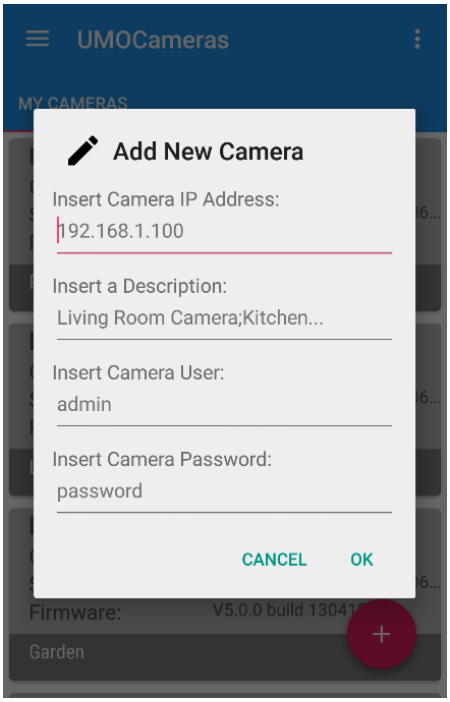


Figura 39 – Dialog de introdução de nova câmara

No painel de navegação (Figura 40), na parte superior são exibidos os dados do utilizador recolhidos da conta da Google (email, foto e nome), e por baixo está uma lista de câmaras. Esta lista permite que a partir de qualquer ecrã da aplicação se possa ir diretamente para a câmara

desejada, oferecendo assim um atalho para troca de câmaras que está sempre disponível e minimiza o tempo dispendido na navegação entre câmaras. Na parte inferior do painel existe um botão para adicionar uma câmara e outro para realizar o *logoff* da aplicação. Neste ultimo caso, os dados do utilizador são apagados.

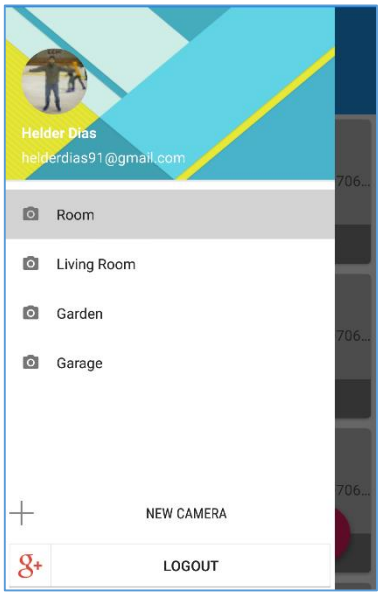


Figura 40 – Painel de Navegação Lateral

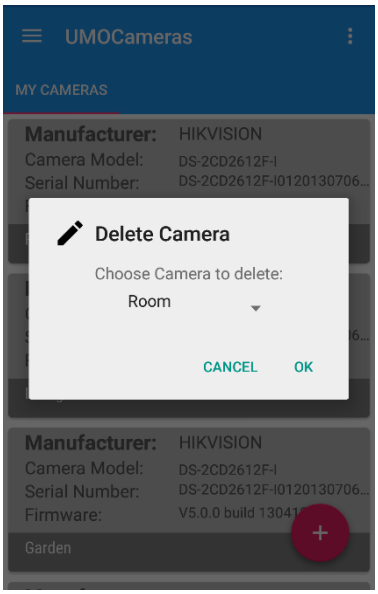


Figura 41 – Dialog para apagar Câmara

5.2.2. Menu de Configuração dos Media Profiles de uma Câmara

No menu de configuração das câmaras (Figura 42), na sua parte superior estão posicionadas as *tabs* correspondentes a cada perfil de uma câmara. Podemos fazer *swipe* para os lados para ir até outro perfil ou então selecionar diretamente carregando em cima do nome do perfil. Conforme o perfil seja "fixed" ou não aparece um ícone para alertar a esta característica do perfil (no caso da imagem são os dois perfis desse tipo).

Na parte principal do menu são expostas as várias entidades de um Perfil de Media. Como foi explicado na secção 4.1.2, podem haver dispositivos que não disponham de certas entidades. Então, quando uma câmara não dispõe dessa funcionalidade o botão que permite selecionar essa funcionalidade é pintado de outra cor, como podemos ver na Figura 42 no botão "*Metadata Configuration*". Por uma questão de facilidade de utilização e de apresentação, as entidades *Video Source* e *Video Encoder* foram fundidas num só menu *Video Configuration*, procedendo-se da mesma maneira nas respetivas configurações de *Áudio*.

5.2.3. Activities de Configuração das entidades

Quando um utilizador seleciona algum dos itens apresentados na Figura 42 despoleta o início de uma *activity* correspondente à entidade selecionada.

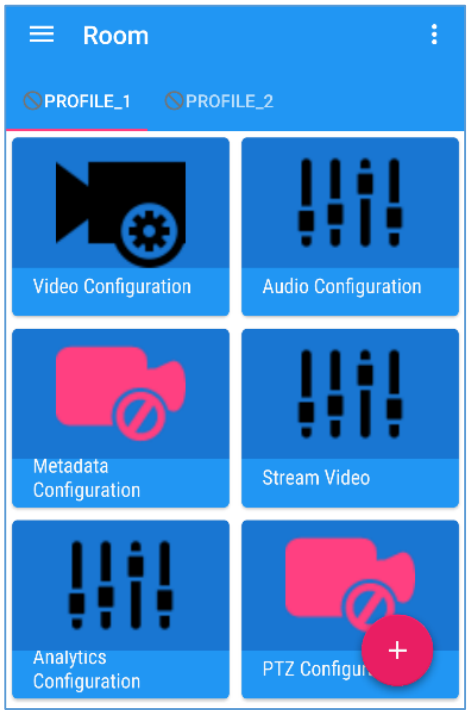


Figura 42 – Menu de Configuração dos Media Profiles

Em todas as *activities* de configuração, existe na *toolbar* superior um ícone que permite fazer o envio das alterações para a câmara, caso contrário todas as alterações realizadas serão descartadas.

5.2.3.1. Video Configurations

Nesta Activity (Figura 43) são apresentados todos os parâmetros que o serviço de Media disponibiliza na entidade de Video Encoder e Video Source, permitindo a configurar:

- Selecionar a fonte de Vídeo;
- Parâmetros da Fonte de Vídeo (X,Y,Width,Height);
- Alterar o Encoder (MPEG-4,H.264,JPEG);
- Parâmetros do Encoder
    - Resolution
    - Quality

    - Frame Rate Limit
    - Encoding Interval
    - Bit Rate Limit
    - Session Timeout
    - Encoder Profile
    - GovLength
- Configuração do Multicast
    - Porta
    - TTL
    - AutoStart
    - IP e tipo de IP (v4 ou v6)

5.2.3.2. Audio Configurations

Na activity Audio Configurations(Figura 44), tal como nas configurações de Video é possível alterar todo o tipo de configurações de áudio permitidas pelo serviço de Media, nas entidades Audio Encoder e Audio Source, nomeadamente:

- Alterar a fonte de Audio;
- Alterar o Encoder (G711,G726,AAC)
- Sample Rate
- Bit Rate
- Session timeout
- Configuração do Multicast
    - Porta
    - TTL
    - AutoStart
    - IP e tipo de IP (v4 ou v6)

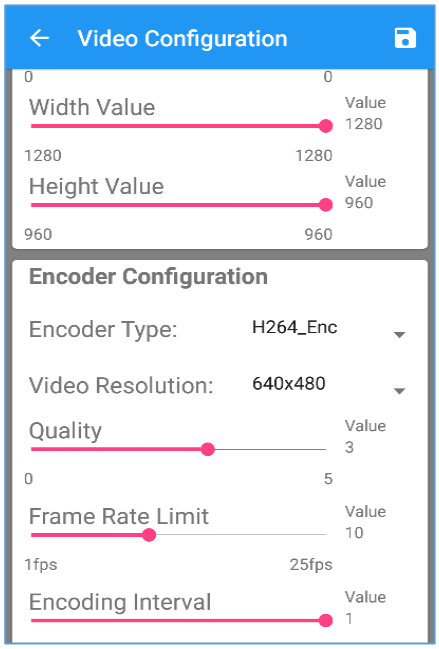


Figura 43 – Activity Video Configuration

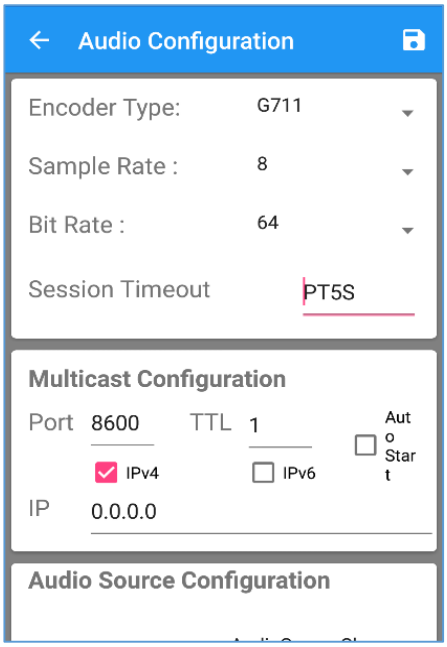


Figura 44 – Activity Audio Configuration

5.2.3.3. Metadata Configurations

O serviço Media permite a configuração de metadados, neste menu pode-se proceder então à realização de alterações nesta entidade, tais como:

- Event Subscription
  - Attributtes
  - Filter Type
  - Subscription Policy
- PTZ
  - Status
  - Position
- Analytics
- Configuração do Multicast
  - Porta
  - TTL
  - AutoStart
  - IP e tipo de IP (v4 ou v6)

5.2.3.4. PTZ Configurations

Em dispositivos que possuam o mecanismo de PTZ é possível alterar as suas configurações através desta activity (Figura 46), podem ser alterados parâmetros como:

- Zoom Limits (X = [Min,Max])
- Pan Tilt Limits (X = [Min,Max] e Y = [Min,Max])
- Default Speed
    - PanTilt (X e Y)
    - Zoom (X)
- Session Timeout

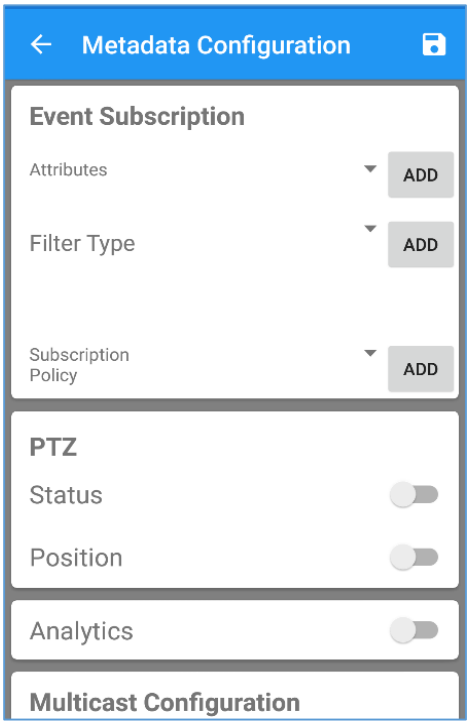


Figura 45 – Activity Metadata Configurations

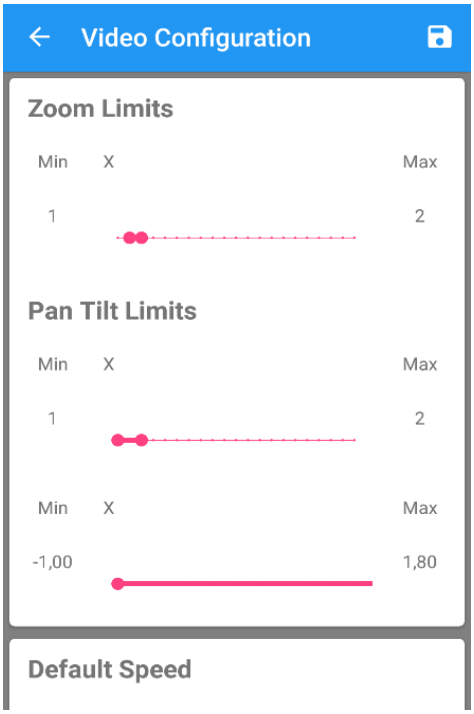


Figura 46 – Activity PTZ Configuration

5.2.3.5. Stream Video

A *activity* de *Stream* de *Video* (Figura 47),não permite qualquer tipo de configuração, foi colocada para o utilizador poder assistir aos *streams* gerados por cada perfil e verificar que as alterações realmente tiveram efeito ou assistir normalmente à transmissão em tempo real fornecida pelas câmaras. O stream é fornecido através do protocolo Real Time Streaming Protocol.

Figura 47 – Activity de Stream Video

5.2.4. Notas sobre a aplicação final

Na fase final do desenvolvimento, foram realizados vários testes de funcionamento da aplicação. Como a câmara utilizada durante o desenvolvimento não dispunha de algumas funcionalidades, nomeadamente PTZ, foram utilizadas câmaras disponíveis online. Todas as funcionalidades de configuração do protocolo ONVIF incluídas na aplicação final foram exaustivamente testadas, em vários ambientes de rede, isto é, utilizando não só utilizando o Wi-Fi dentro de uma rede doméstica mas também em redes institucionais como a *eduroam.* Para além disso foram realizados também os mesmos testes através de redes móveis celulares comuns. Nestas últimas nota-se que o tempo de realização das operações ONVIF é superior. Este notório atraso está relacionado com o débito da rede e o volume de informação das mensagens SOAP.

6. Conclusão

Este trabalho teve como objetivos comparar vários modos de comunicação com os dispositivos ONVIF NVT através de um dispositivo móvel com sistema operativo Android, desenvolvendo uma aplicação de testes que permitisse medir a eficiência energética e o desempenho. Um desses modos de comunicação envolve a utilização de um servidor REST. Por isso, foi necessário fazer o *deploy* de um servidor HTTP Apache e configurar o mesmo para direcionar pedidos para a aplicação servidora REST implementada como um FastCGI. Os outros modos foram uma API JNI para a biblioteca UMOC e um cliente SOAP em Java. Nestes casos foi necessário estudar as respetivas APIs, não sendo necessária nenhuma infraestrutura especifica de rede para além do smartphone e de câmaras ONVIF. Foi desenvolvida uma aplicação de testes que suporta dois tipos de operações ONVIF distintas, nos três modos de comunicação e com dois tipos de duração dos testes.

6.1. Conclusões

Ao desenvolver a aplicação de testes rapidamente se percebe uma grande vantagem em termos de facilidade de implementação para os dois modos de comunicação JNI e REST, que utilizam a biblioteca C UMOC (no caso do REST indiretamente através do servidor). Ao contrário do cliente SOAP Java que obriga a um esforço de implementação muito maior dado que para a mesma operação obriga a escrever muitas mais linhas de código para a comunicação SOAP, que nos outros modos é abstraída pela biblioteca.

Depois de realizados os testes comparativos em condições de rede controladas, foi possível afirmar que o modo de comunicação mais vantajoso é a JNI, tanto em termos de desempenho como em eficiência energética. Por isso, foi tomada a decisão de desenvolver a aplicação de gestão de perfis de Media com a JNI.

A informação obtida nos testes serve de guia para implementações semelhantes em trabalhos futuros envolvendo não só o protocolo ONVIF, mas também outros protocolos que utilizem web services com modos de comunicação e tipos de dados semelhantes aos utilizados neste projeto.

Para desenvolver a aplicação Android para a gestão dos perfis de media foi necessário compreender o funcionamento do serviço de media que o ONVIF disponibiliza. Este serviço é

bastante complexo e representa uma parte muito importante do protocolo visto que afinal é através deste serviço que são configurados os parâmetros que vão influenciar direta ou indiretamente o stream de vídeo disponibilizado, que é a característica principal da utilização de uma câmara de vigilância IP.

Para garantir uma boa experiência de utilizador foram integradas na aplicação funcionalidades que permitem facilitar e agilizar a utilização pelos utilizadores, e que nas aplicações gratuitas existentes no mercado não existe. A utilização de uma autenticação rápida através da API Google sign-in para além de facilitar a autenticação do utilizador, permite também ao programador, no caso de publicar a aplicação numa loja de aplicações, ter acesso a uma consola via web e a dados importantes como o número de registos e acessos em tempo-real. Outra funcionalidade importante foi a implementação da persistência através de uma base de dados permitindo ao utilizador manter todas as informações sempre guardadas no dispositivo para utilização futura, garantindo também uma maior rapidez na utilização.

Uma das maiores dificuldades durante a implementação foram a integração entre os componentes de UI e os *adapters* que fazem a monitorização dos *datasets* a apresentar em listas nesses componentes, visto que por exemplo o componente *RecyclerView* apenas surgiu na penúltima API do Android e é um componente ainda pouco divulgado.

Ao ser utilizada a JNI para a biblioteca UMOC, um dos desafios encontrados foram a alocação de memória para a estruturas que fazem a organização dos dados recebidos pelas câmaras e que obrigam a utilizar métodos para ultrapassar este problema como por exemplo para criar um perfil de media novo é necessário fazer uma cópia de um perfil existente para que a biblioteca faça a alocação desse espaço em memória e trabalhar o novo a partir dessa cópia.

6.2. Trabalho Futuro

O desenvolvimento da aplicação estava apenas destinado à gestão de perfis de Media, no entanto existem muitos serviços ONVIF que podem ser incluídos na aplicação realizada, e que vão poder tirar partido da base de interface de utilizador e das funcionalidades aplicacionais implementadas.

Criar uma solução de organização de câmaras em grupos ou em árvore na interface de utilizador para utilizadores com grande número de câmaras, implementar solução de split-screen com vários *streams* em simultâneo.

Existem também melhorias a realizar na JNI, como por exemplo, processo de alocação de memória, permitindo a uniformização da API JNI e minimizar a exposição à gestão de memória C.

No servidor REST, tentar realizar otimizações no código para que se melhore o desempenho. Desenvolver uma biblioteca JAVA para a comunicação REST que abstraia o processamento da comunicação e a serialização e des-serialização JSON.

7. Bibliografia

[1] ONVIF™, "ONVIF Protocol Specifications," 2011. [Online]. Available: http://www.onvif.org/specs/DocMap.html. [Acesso em 22-03-2015].

[2] ONVIF™, "Network Video Display Device Definition," Março 2011. [Online]. Available: http://www.onvif.org/specs/td/nvd/ONVIF-NVD-Definition-v210.pdf. [Acesso em 01-02-2015].

[3] ONVIF™, "Network Video Analytics Device Definition," [Online]. [Acesso em 01-02-2015].

[4] ONVIF™, "Network Video Transmitter Device Definition," Junho 2011. [Online]. Available: http://www.onvif.org/specs/td/nvt/ONVIF-NVT-Definition-v210.pdf. [Acesso em 01-02-2015].

[5] ONVIF™, "Network Video Storage Device Definition," Junho 2011. [Online]. Available: http://www.onvif.org/specs/td/nvs/ONVIF-NVS-Definition-v210.pdf. [Acesso em 01-02-2015].

[6] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", University of California, 2000.

[7] "Interface FastCGI," [Online]. Available: http://www.fastcgi.com/drupal/. [Acesso em 2015-03-14].

[8] "ONVIF™ Member List," [Online]. Available: http://www.onvif.org/About/MemberList.aspx. [Acesso em 23-01-2015].

[9] ONVIF™, “Media Service Specification,” [Online]. Available: http://www.onvif.org/specs/srv/media/ONVIF-Media-Service-Spec-v221.pdf. [Acesso em 07-01-2015].

[10] J. Snell, D. Tidwell e P. Kulchenko, "Programming Web Services with SOAP", O’Reilly, December 2001.

[11] S. Lopes, S. Silva, J. Mendes, J. Metrolho e D. Duque, “Development of a library for clients of ONVIF video cameras : challenges and solutions”, Industrial Technology (ICIT), 2013 IEEE International Conference, 2013.

[12] “Apache Web Server,” [Online]. Available: http://www.apache.org/. [Acesso em 14-02-2015].

[13] “Ubuntu Documentation (Apache2 Web Server),” Canonical, [Online]. Available: https://help.ubuntu.com/lts/serverguide/httpd.html. [Acesso em 11-03-2015].

[14] J. Valverde, "Aplicação web para configuração e acesso a câmaras ONVIF", Dissertação: Universidade do Minho, 2013.

[15] “SWIG compiler,” [Online]. Available: http://www.swig.org/exec.html. [Acesso em 18-07-2015].

[16] “Ksoap2-Android,” [Online]. Available: http://simpligility.github.io/ksoap2-android/index.html. [Acesso em 11-4-2015].

[17] “Postman,” Postdot Technologies Pvt. Ltd., [Online]. Available: https://www.getpostman.com/. [Acesso em 24-01-2015].

[18] R. Boyd, "Getting Started with OAuth 2.0", O'Reilly Media, 2012.

[19] G. Turi, “Online JSON Viewer,” [Online]. Available: http://jsonviewer.stack.hu/. [Acesso em 28-03-2015].

[20] “Android Studio IDE,” [Online]. Available: http://developer.android.com/sdk/index.html. [Acesso em 15-02-2015].

[21] “Onvif Client for Android,” Happytimesoft, [Online]. Available: http://www.happytimesoft.com/products/onvif-client-for-android/index.html. [Acesso em 11-07-2015].

[22] “Ocular IP Camera,” [Online]. Available: https://play.google.com/store/apps/details?id=com.emilianoschmid.ocular&hl=pt_PT. [Acesso em 10-05-2015].

[23] “ART and Dalvik,” [Online]. Available: https://source.android.com/devices/tech/dalvik/. [Acesso em 12-02-2015].

[24] “Android AsyncTask,” [Online]. Available: http://developer.android.com/reference/android/os/AsyncTask.html. [Acesso em 10-02-2015].

[25] “Material Design,” Google, [Online]. Available: https://www.google.com/design/spec/what-is-material/environment.html. [Acesso em 14-05-2015].

[26] “Ferramenta de Prototipagem,” Proto.io, [Online]. Available: https://proto.io/. [Acesso em 11-05-2015].

[27] “Android Navigation Drawer,” [Online]. Available: https://developer.android.com/training/implementing-navigation/nav-drawer.html. [Acesso em 12-05-2015].

[28] “Google Sign-In API,” [Online]. Available: https://developers.google.com/identity/sign-in/android/sign-in. [Acesso em 18-07-2015].

[29] “Start Integrating Google Sign-In into Your Android App,” [Online]. Available: https://developers.google.com/identity/sign-in/android/start-integrating. [Acesso em 10-06-2015].

[30] S. K. Aditya e V. K. Karn, "Android SQLite Essentials", PACKT, 2014.

[31] “Android SQLite,” [Online]. Available: http://developer.android.com/reference/android/database/sqlite/package-summary.html. [Acesso em 17-05-2015].

[32] “Android SharedPreferences,” [Online]. Available: http://developer.android.com/reference/android/content/SharedPreferences.html. [Acesso em 19-06-2015].

[33] “Android Activities,” [Online]. Available: http://developer.android.com/guide/components/activities.html. [Acesso em 11-02-2015].

[34] “Android Fragments Guide,” [Online]. Available: http://developer.android.com/guide/components/fragments.html. [Acesso em 4-06-2015].

[35] “Android RecyclerView,” [Online]. Available: https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html. [Acesso em 16-06-2015].

[36] “Android ListView,” [Online]. Available: http://developer.android.com/guide/topics/ui/layout/listview.html.

[Acesso em 20-05-2015].

[37] “Android RecyclerView Adapter,” [Online]. Available: https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html. [Acesso em 27-06-2015].

[38] “Android UI elements,” [Online]. Available: http://developer.android.com/reference/android/widget/package-summary.html. [Acesso em 21-04-2015].

[39] “Android TabLayout,” [Online]. Available: https://developer.android.com/reference/android/support/design/widget/TabLayout.html. [Acesso em 29-06-2015].

[40] “Android Storage Options,” [Online]. Available: http://developer.android.com/guide/topics/data/data-storage.html. [Acesso em 17-5-2015].

[41] J. Knutsen, "Web Service Clients on Mobile Android-A Study on Architectural Alternatives and Client Performance", Norwegian University of Science and Technology: Master Thesis, 2009.

[42] P. K. Potti, “"On the Design of Web Services: SOAP vs. REST",” University of North Florida, 2011.

[43] Z. Mednieks, L. D. Meike e M. Nakamura, "Java Programming for the New Generation of Mobile Devices", 2011.

[44] S. Liang, "Java Native Interface: Programmer's Guide and Specification", Prentice Hall , 1999.

[45] D. Sillars, "High Performance Android Apps", O'Reilly Media, 2015.

[46] N. Nurseitov, M. Paulson, R. Reynolds e C. Izurieta, "Comparison of JSON and XML Data Interchange Formats : A Case Study", CAINE, 2009.